(NASA-CR-184685) PROCEEDINGS OF THE WORKSHOP ON SPACE TELEROBOTICS, VOLUME 1 (Jet Propulsion Lab.) 365 p CSCL 09B

N89-26454
--THRU--
N89-26491
Unclas
G3/63 0188082

This publication was prepared by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

ABSTRACT

These proceedings report the results of a workshop on space telerobotics, which was held at the Jet Propulsion Laboratory, January 20-22, 1987. Sponsored by the NASA Office of Aeronautics and Space Technology (OAST), the workshop reflected NASA's interest in developing new telerobotics technology for automating the space systems planned for the 1990s and beyond. The workshop provided a window into NASA telerobotics research, allowing leading researchers in telerobotics to exchange ideas on manipulation, control, system architectures, artificial intelligence, and machine sensing. One of the objectives was to identify important unsolved problems of current interest. The workshop consisted of surveys, tutorials, and contributed papers of both theoretical and pratical interest. Several sessions were held with the themes of sensing and perception, control execution, operator interface, planning and reasoning, and system architecture. Discussion periods were also held in each of these major topics.

ACKNOWLEDGMENT

Thanks are due to D.R. Meldrum, R.S. Doshi, K. Kreutz, H. Seraji, M.H. Milman, and S.K. Owen for significant contributions to the organization of the workshop.

CONTENTS

Volume I

PLENARY SESSION 1

Program Objectives and Technology Outreach
P.S. Schenker 3

Building Intelligent Systems: Artificial Intelligence Research at NASA Ames Research Center
P. Friedland and H. Lum 19 S1

SYSTEM CONCEPTS 27

Space Telerobotic Systems: Applications and Concepts
L. Jenkins 29 S2

Laboratory Testing of Candidate Robotic Applications for Space
R.B. Purves 35 S3

Telerobotic Assembly of Space Station Truss Structure
G. Fischer 45 S4

Robotic Mobile Servicing Platform for Space Station
S.H. Lowenthal and L. Van Erden 55 S5

System Engineering Techniques for Establishing Balanced Design and Performance Guidelines for the Advanced Telerobotic Testbed
W.F. Zimmerman and J.R. Matijevic 67 S6

Dedicated Robotic Servicing for the Space Station
R.F. Thompson, G. Arnold, and D. Gutow 75 S7

Shuttle Bay Telerobotics Demonstration
W. Chun and P. Cogeos 83 S8

Telerobotics: Research Needs for Evolving Space Stations
L. Stark 91 S9

Robot Design for a Vacuum Environment
S. Belinski, W. Trento, R. Imani-Shikhabadi, and S. Hackwood . . . 95 S10

Multi-Limbed Locomotion Systems for Space Construction and Maintenance
K.J. Waldron and C.A. Klein 105 S11

Commercial Users Panel 113 S12

SYSTEM ARCHITECTURE . . . . 125 omit

Task Allocation Among Multiple Intelligent Robots
L. Gasser and G. Bekey 127 S13

Execution Environment for Intelligent Real-Time Control Systems
J. Sztipanovits 131 S14

Distributed Intelligence for Supervisory Control
W.J. Wolfe and S.D. Raney 139 S15

An Architecture for Heuristic Control of Real-Time Processes
P. Raulefs and P.W. Thorndyke 149 S16

Hierarchical Control of Intelligent Machines
Applied to Space Station Telerobots
J.S. Albus, R. Lumia, and H. McCain 155 S17

Programming Methodology for a General Purpose Automation Controller
M.C. Sturzenbecker, J.U. Korein, and R.H. Taylor 167 S18

Real-Time Hierarchically Distributed Processing Network
Interaction Simulation
W.F. Zimmerman and C. Wu 173 S19

Metalevel Programming in Robotics: Some Issues
A. Kumar N. and N. Parameswaran 181 S20

An Approach to Distributed Execution of ADA Programs
R.A. Volz, P. Krishnan, and R. Theriault 187 S21

Computational Structures for Robotic Computations
C.S.G. Lee and P.R. Chang 199 S22

A Run-Time Control Architecture for the JPL Telerobot
J. Balaram, A. Lokshin, K. Kreutz, and J. Beahan 211 S23

Distributed Control Architecture for Real-Time Telerobotic Operation
H.L. Martin, P.E. Satterlee, Jr., and R.F. Spille 223 S24

High Performance Architecture for Robot Control
E. Byler and J. Peterson 233 S25

Software and Hardware for Intelligent Robots
G.N. Saridis and K.P. Valavanis 241 S26

Nested Hierarchical Controller With Partial Autonomy
A. Meystel 251 S27

AUTONOMOUS SYSTEMS 271

3-D World Modeling With Updating Capability Based on Combinatorial Geometry
M. Goldstein, F.G. Pin, G. de Saussure, and C.R. Weisbin 273 S28

On Autonomous Terrain Model Acquisition by a Mobile Robot
N.S.V. Rao, S.S. Iyengar, and C.R. Weisbin 283 S29

SOLON: An Autonomous Vehicle Mission Planner
M.J. Dudziak 289 S30

Mission Planning for Autonomous Systems
G. Pearson 303 S31

Certainty Grids for Mobile Robots
H.P. Moravec 307 S32

Environmental Modeling and Recognition for an Autonomous Land Vehicle
D.T. Lawton, T.S. Levitt, C.C. McConnell, and P.C. Nelson 313 S33

GEOMETRIC REASONING . . . . 337 omit

Telerobot Task Planning and Reasoning: Introduction to JPL Artificial Intelligence Research
D.J. Atkinson 339 S34

Geometric Reasoning
R.F. Woodbury and I.J. Oppenheim 351 S35

A Qualitative Approach for Recovering Relative Depths in Dynamic Scenes
S.M. Haynes and R. Jain 361 S36

A Framework for Qualitative Reasoning About Solid Objects
E. Davis 369 S37

Volume II

SENSING AND PERCEPTION 1

The Sensing and Perception Subsystem of the NASA Research Telerobot
B. Wilcox, D.B. Gennery, B. Bon, and T. Litwin 3

Knowledge-Based Vision for Space Station Object Motion Detection, Recognition, and Tracking
P. Symosek, D. Panda, S. Yalamanchili, and W. Wehner III 9

Sensor Systems Testbed for Telerobotic Navigation
A.W. Thiele, D.E. Gjellum, R.H. Rattner, and D. Manouchehri 19

Monovision Techniques for Telerobots
P.W. Goode and K. Cornils 23

Tracking 3-D Body Motion for Docking and Robot Control
M. Donath, B. Sorensen, G.B. Yang, and R. Starr 31

Object Apprehension Using Vision and Touch
R. Bajcsy and S.A. Stansfield 45

Sensory Substitution for Space Gloves and for Space Robots
P. Bach-y-Rita, J.G. Webster, W.J. Tompkins, and T. Crabb 51

Electronic Prototyping
J. Hopcroft 59

Multiple Degree of Freedom Optical Pattern Recognition
D. Casasent 65

Maximum Likelihood Estimation of Parameterized 3-D Surfaces
Using a Moving Camera
Y. Hung, B. Cernuschi-Frias, and D.B. Cooper 71

The Architecture of a Video Image Processor for the Space Station
S. Yalamanchili, D. Lee, K. Fritze, T. Carpenter, K. Hoyme,
and N. Murray 85

An Optical Processor for Object Recognition and Tracking
J. Sloan and S. Udomkesmalee 95

Improved Shape-Signature and Matching Methods for
Model-Based Robotic Vision
J.T. Schwartz and H.J. Wolfson 103

A Technique for 3-D Robot Vision for Space Applications
V. Markandey, H. Tagare, and R.J.P. deFigueiredo 111

Differential Surface Models for Tactile Perception of Shape
and On-Line Tracking of Features
H. Hemami 125

Constraint-Based Stereo Matching
D.T. Kuan 131

A Database/Knowledge Structure for a Robotics Vision System
D.W. Dearholt and N.N. Gonzales 135

Sensing and Perception: Connectionist Approaches to
"Subcognitive" Computing
J. Skrzypek 145

Parallel Processing for Digital Picture Comparison
H.D. Cheng and L.T. Kou 155

ROBOT SIMULATION AND CONTROL 163

Inverse Kinematic-Based Robot Control
W.A. Wolovich and K.F. Flueckiger 165

Spatially Random Models, Estimation Theory, and Robot Arm Dynamics
G. Rodriguez 177

Task Oriented Nonlinear Control Laws for Telerobotic Assembly Operations
R.A. Walker, L.S. Ward, and C.F. Elia 185

A Universal Six-Joint Robot Controller
D.G. Bihn and T.C. Hsia 197

Real-Time Graphic Simulation for Space Telerobotics Applications
E.W. Baumann 207

Manipulator Control and Mechanization: A Telerobot Subsystem
S. Hayati and B. Wilcox 219

Chaos Motion in Robot Manipulators
A. Lokshin and M. Zak 229

Effect of Control Sampling Rates on Model-Based Manipulator Control Schemes
P.K. Khosla 235

Kinematically Redundant Robot Manipulators
J. Baillieul, R. Brockett, J. Hollerbach, D. Martin, R. Percy, and R. Thomas 245

Control Strategy for a Dual-Arm Maneuverable Space Robot
P.K.C. Wang 257

A Space Systems Perspective of Graphics Simulation Integration
R. Brown, C. Gott, G. Sabionski, and D. Bochsler. 267

A Learning Controller for Nonrepetitive Robotic Operation
W.T. Miller, III. 273

TELEROBOTS 283

A Flexible Telerobotic System for Space Operations
N.O. Sliwa and R.W. Will. 285

Systems Simulations Supporting NASA Telerobotics
F.W. Harrison, Jr. and J.E. Pennington 293

Coordination of Multiple Robot Arms
L.K. Barker and D. Soloway 301

Development of a Semi-Autonomous Service Robot With Telerobotic Capabilities
J.E. Jones and D.R. White 307

A Task-Based Metric for Telerobotic Performance Assessment
J.F. Barnes 317

Motion Coordination and Programmable Teleoperation Between Two Industrial Robots
J.Y.S. Luh and Y.F. Zheng 325

Use of Control Umbilicals as a Deployment Mode for Free Flying Telerobotic Work Systems
J.S. Kuehn and E.D. Selle 335

The Design of a Nine-String Six-Degree-of-Freedom Force-Feedback Joystick for Telemanipulation
M.L. Agronin 341

MAN MACHINE INTERFACE 349

Issues in Human/Computer Control of Dexterous Remote Hands
K. Salisbury 351

Man-Machine Interface Issues in Space Telerobotics: A JPL Research and Development Program
A.K. Bejczy 361

Task-Level Testing of the JPL-OMV Smart End Effector
B. Hannaford 371

A Natural-Language Interface to a Mobile Robot
S. Michalowski, C. Crangle, and L. Liang 381

Types of Verbal Interaction With Instructable Robots
C. Crangle, P. Suppes, and S. Michalowski 393

MIT Research in Telerobotics
T.B. Sheridan and Staff 403

Telerobotics: A Simulation Facility for University Research
L. Stark et al. 413

Mixed-Initiative Control of Intelligent Systems
G.C. Borchardt 423

Report on the Stanford/Ames Direct-Link Space Suit Prehensor
J.W. Jameson 433

Volume III

.PLANNING AND SCHEDULING: BASIC RESEARCH AND TOOLS 1

Planning and Scheduling: Is there a Difference?
K.G. Kempf 3

The Mechanisms of Temporal Inference
B.R. Fox and S.R. Green 9

Contingent Plan Structures for Spacecraft
M. Drummond, K. Currie, and A. Tate 17

Reasoning and Planning in Dynamic Domains:
An Experiment With a Mobile Robot
M.P. Georgeff, A.L. Lansky, and M.J. Schoppers 27

Route Planning in a Four-Dimensional Environment
M.G. Slack and D.P. Miller 41

Prediction and Causal Reasoning in Planning
T. Dean and M. Boddy 49

The Generic Task Toolset: High Level Languages for the
Construction of Planning and Problem Solving Systems
B. Chandrasekaran, J. Josephson, and D. Herman 59

Monitoring Robot Actions for Error Detection and Recovery
M. Gini and R. Smith 67

Recovering From Execution Errors in SIPE
D.E. Wilkins 79

Incremental Planning to Control a Blackboard-Based Problem Solver
E.H. Durfee and V.R. Lesser 91

Knowledge Representation System for Assembly Using Robots
A. Jain and M. Donath 101

Real Artificial Intelligence for Real Problems
R.S. Doshi 115

TRAJECTORY PLANNING FOR MANIPULATORS 117

The Use of 3-D Sensing Techniques for On-Line
Collision-Free Path Planning
V. Hayward, S. Aubry, and Z. Jasiukajc 119

Collision-Free Trajectory Planning Algorithm for Manipulators
F. Pourboghrat and J.Y. Han 125

Task Planning and Control Synthesis for Robotic Manipulation in Space Applications
A.C. Sanderson, M.A. Peshkin, and L.S. Homem-de-Mello 129

Using Automatic Robot Programming for Space Telerobotics
E. Mazer, J. Jones, A. Lanusse, T. Lozano-Perez, P. O'Donnell, and P. Tournassoud 139

MANIPULATOR CONTROL 151

Manipulator Control: An Overview
H. Seraji 153

Adaptive Force-Position Control for Teleoperated Manipulators
A.J. Koivo 155

Adaptive Control of Dual-Arm Robots
H. Seraji 159

Design of a Reconfigurable Modular Manipulator System
D. Schmitz and T. Kanade 171

Nonlinear Feedback Control of Multiple Robot Arms
T.J. Tarn, X. Yun, and A.K. Bejczy 179

Dynamics and Control of Coordinated Multiple Manipulators
S.A. Hayati 193

A Survey of Adaptive Control Technology in Robotics
S. Tosunoglu and D. Tesar 205

Simple Robust Control Laws for Robot Manipulators, Part I: Non-adaptive Case
J.T. Wen and D.S. Bayard 215

Simple Robust Control Laws for Robot Manipulators, Part II: Adaptive Case
D.S. Bayard and J.T. Wen 231

Algorithms for Adaptive Control of Two-Arm Flexible Manipulators Under Uncertainty
J.M. Skowronski 245

Problems and Research Issues Associated With the Hybrid Control of Force and Displacement
R.P. Paul 255

Adaptive Hybrid Control of Manipulators
H. Seraji 261

Adaptive Hybrid Position/Force Control of Robotic Manipulators
F. Pourboghrat 273

Flexible Manipulator Control Experiments and Analysis
S. Yurkovich, Ü. Özgüner, A. Tzes, and P.T. Kotnik 279

Dual Arm Robotic System With Sensory Input
Ü. Özgüner 289

Geometric Foundations of the Theory of Feedback Equivalence
R. Hermann 299

Reducing Model Uncertainty Effects in Flexible Manipulators Through the Addition of Passive Damping
T.E. Alberts 307

Parallel Processing Architecture for Computing Inverse Differential Kinematic Equations of the PUMA Arm
T.C. Hsia, G.Z. Lu, and W.H. Han 317

Manipulator Control by Exact Linearization
K. Kreutz 325

A Virtual Manipulator Model for Space Robotic Systems
S. Dubowsky and Z. Vafa 335

Model Reduction for Discrete Bilinear Systems
A.M. King and R.E. Skelton 345

High Gain Feedback and Telerobotic Tracking
D.E. Koditschek 355

Concept Development of a Tendon Arm Manipulator and Anthropomorphic Robotic Hand
C.T. Tolman 365

Future Research Directions
Moderator: G.A. Bekey

Panel Members: R. Bajcsy, J.Y.S. Luh, A. Sanderson, G. Saridis, T.B. Sheridan, C. Weisbin, ... 373

APPENDIX A

Program Schedule 391

APPENDIX B

Index by Author 405

APPENDIX C

Attendees/Participants 409

PLENARY SESSION

OMIT TO
P. 19

Program Objectives and Technology Outreach

P.S. Schenker
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

INTRODUCTION

~~This paper presents~~ an overview of the research that NASA is conducting in telerobotics ~~, how NASA is going about this work, and why we have undertaken it.~~ is given. ~~It also briefly outlines~~ the content and objectives of the workshop proceedings ~~that follow~~. are outlined.

The topics ~~of the workshop are many. These~~ topics range from basic telerobotics research issues to design and integration of a near-term flight initial operational capability space telerobot. Among the areas to be covered are: machine vision and tactile sensing, manipulator control, AI-based task planning and error diagnosis, teleoperation and human-robot interface, and telerobot system architecture and design concepts. The central technology theme is the supervised automation of space operations, with emphasis on scenarios involving platform or spacecraft servicing, assembly, and repair.

organizers

The objectives of the workshop are threefold: First, NASA would like to convey the spirit of the telerobotics research we are doing and the programmatic directions we expect it to take in the future. Second, organizers we hope through the technical sessions to stimulate topically focused discussions on contrasting approaches to telerobot design and implementation. Finally, and foremost, we seek to identify some of the important unsolved and perhaps yet unaddressed problems confronting the development of a viable space telerobotic capability. ~~Toward the last objective, the workshop will conclude with an invited panel of distinguished speakers; the panelists offer a perspective on telerobotics research issues requiring increased emphasis and juxtapose their viewpoints with the workshop activities that preceded the panel.~~

NASA Activities in Telerobotics

NASA at this time has three principal areas of activity in telerobotics. The first is the Office of Aeronautics and Space Technology (OAST - Code R) Telerobotics Research Program. The second activity is the Office of Space Station (OSS - Code S) Flight Telerobot Servicer Engineering Development Program. The third activity, supported by many NASA Offices and Centers, is various problem-specific telerobot design studies and technology assessments. This last category includes applications such as the Shuttle, planetary rovers, orbital maneuver vehicles. The common goals of these activities is the time-staged development of increased space mission productivity, expanded mission capability, and ultimately extended mission autonomy. Enhanced safety of space operations is a unifying theme across these goals. The planned discussions and presentations of this workshop largely focus on activity of the Code R NASA Telerobotics Program. This program has its 1985 origin in a set of derived system requirements summarized in Figure 1. In essence the program is a "technology push" effort: believing that technology exists or can be defined in the near-to-mid term to impact the above stated mission goals, NASA Code R is investing in developing that technology to ground laboratory breadboard proof-of-concept. (By memorandum of agreement between Codes R and S, elements of this technology are to be transferred to the Flight Telerobot Servicer Progam for subsequent evaluation and advanced development.)

*Dr. Schenker is Chief Technologist of the NASA/JPL Telerobotics Project. His mailing address is Jet Propulsion Laboratory, 4800 Oak Grove Drive, M/S 198-330, Pasadena, CA 91109.

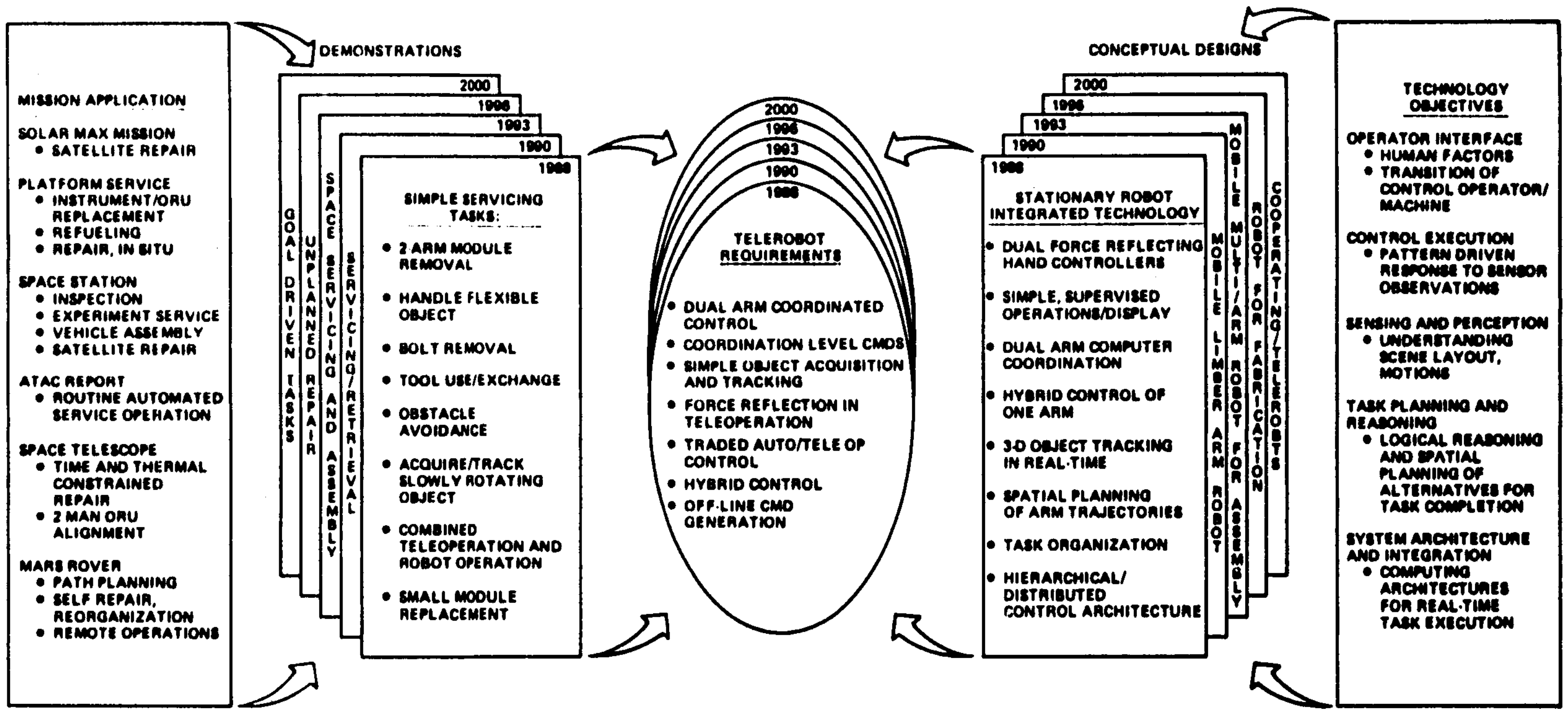

MISSION APPLICATION
SOLAR MAX MISSION
• SATELLITE REPAIR
PLATFORM SERVICE
• INSTRUMENT/ORU REPLACEMENT
• REFUELING
• REPAIR, IN SITU
SPACE STATION
• INSPECTION
• EXPERIMENT SERVICE
• VEHICLE ASSEMBLY
• SATELLITE REPAIR
ATAC REPORT
• ROUTINE AUTOMATED SERVICE OPERATION
SPACE TELESCOPE
• TIME AND THERMAL CONSTRAINED REPAIR
• 2 MAN ORU ALIGNMENT
MARS ROVER
• PATH PLANNING
• SELF REPAIR, REORGANIZATION
• REMOTE OPERATIONS
DEMONSTRATIONS
2000
1996
1993
1990
1988
GOAL DRIVEN TASKS
UNPLANNED REPAIR
SPACE SERVICING AND ASSEMBLY
SERVICING/RETRIEVAL
SIMPLE SERVICING TASKS:
• 2 ARM MODULE REMOVAL
• HANDLE FLEXIBLE OBJECT
• BOLT REMOVAL
• TOOL USE/EXCHANGE
• OBSTACLE AVOIDANCE
• ACQUIRE/TRACK SLOWLY ROTATING OBJECT
• COMBINED TELEOPERATION AND ROBOT OPERATION
• SMALL MODULE REPLACEMENT
2000
1996
1993
1990
1988
TELEROBOT REQUIREMENTS
• DUAL ARM COORDINATED CONTROL
• COORDINATION LEVEL CMDS
• SIMPLE OBJECT ACQUISITION AND TRACKING
• FORCE REFLECTION IN TELEOPERATION
• TRADED AUTO/TELE OP CONTROL
• HYBRID CONTROL
• OFF-LINE CMD GENERATION
CONCEPTUAL DESIGNS
2000
1996
1993
1990
1988
STATIONARY ROBOT INTEGRATED TECHNOLOGY
• DUAL FORCE REFLECTING HAND CONTROLLERS
• SIMPLE, SUPERVISED OPERATIONS/DISPLAY
• DUAL ARM COMPUTER COORDINATION
• HYBRID CONTROL OF ONE ARM
• 3-D OBJECT TRACKING IN REAL-TIME
• SPATIAL PLANNING OF ARM TRAJECTORIES
• TASK ORGANIZATION
• HIERARCHICAL/ DISTRIBUTED CONTROL ARCHITECTURE
MOBILE LIMBER ARM ROBOT
MOBILE MULTI/ARM ROBOT FOR ASSEMBLY
ROBOT FOR FABRICATION
COOPERATING/TELEROBOTS
TECHNOLOGY OBJECTIVES
OPERATOR INTERFACE
• HUMAN FACTORS
• TRANSITION OF CONTROL OPERATOR/ MACHINE
CONTROL EXECUTION
• PATTERN DRIVEN RESPONSE TO SENSOR OBSERVATIONS
SENSING AND PERCEPTION
• UNDERSTANDING SCENE LAYOUT, MOTIONS
TASK PLANNING AND REASONING
• LOGICAL REASONING AND SPATIAL PLANNING OF ALTERNATIVES FOR TASK COMPLETION
SYSTEM ARCHITECTURE AND INTEGRATION
• COMPUTING ARCHITECTURES FOR REAL-TIME TASK EXECUTION

Figure 1. Telerobot Demonstrator Requirements Development

The task domain of the space telerobot has a number of defining characteristics:

a) Workpieces:	Complex man-made
b) Illumination:	Highly variable
c) Observation:	Indirect, video or tele-present
d) Complexity:	Structured to highly unstructured
e) Uncertainty:	Missing data, anomalies likely
f) Manipulation:	Extended and/or flexible structures
g) Mobility:	May be required for traverse or full workspace
h) Communications:	Possible out-of loop delay re: run-time control

Relative to these characteristics, we preview two extreme telerobot system scenarios:

(1) Initial Operational Capability (IOC):"....Stationary two-arm telerobot performs known simple tasks on cooperative spacecraft using hand and power tools ... limited autonomy..."

(a) Stationary telerobot, stabilized task frame, fixed workspace
(b) Stereo presence and force feedback
(c) Detailed CAD workspace reference
(d) Controlled and predictable illumination
(e) Automated workpiece acquisition and verification
(f) Coordinated handling of extended objects which have well-defined grasp points
(g) Operator-designated or control-driven automation/teleoperation trades
(h) Sequential, highly structured task schedule

(2) Technology Outreach: "... Cooperating mobile telerobots perform complex temporary and permanent repairs of damaged elements using auxiliary supports, guides, and power tools... Periods of autonomy measured in moments..."

(a) Free-flying robots, moving task frames, variable workspace
(b) Natural language discourse, single operator supervision
(c) Mixed ambient illumination and active sensing
(d) Automated acquisition, tracking, and 3-D interpretation of workspace
(e) Inter-robot, cooperative handling of generic objects
(f) Cooperative workspace traverse as required
(g) Automated load-sharing/shedding between operator and robot
(h) Parallel, dynamically allocated task schedule
(i) Event-driven, automated replanning of task in run-time

The IOC scenario suggests the laboratory floor telerobot capability that we can and should try to establish as a technology baseline. The NASA Code R program has planned such an IOC demonstration which will be discussed in detail later in the paper. The second scenario of "technology outreach" has very advanced technology features, some or all of which one might expect to characterize a highly evolved telerobot capability. Such an evolutionary telerobot would far outstrip the IOC system in task scope, capability, duration of autonomous operation, and richness of human-machine interaction. The primary reason for introducing these scenarios at this time is to motivate understanding of how and why the NASA Code R Telerobotics Program has been formulated, which is the next topic.

NASA Telerobotics Research Program (Code R)

For many years NASA has conducted and supported research efforts in robotics related areas such as manipulator design and control, teleoperations, human factors and human-machine interface, and AI planning. Concomitant with initiation of the Space Station Program and related recommendations of the Advanced Technology Advisory Committee, NASA OAST sought to unify these research efforts into a focused Automation and Robotics technology development. To this end, two programs were initiated, Systems Automation and Telerobotics. The NASA lead centers for these programs are respectively Ames Research Center and Jet Propulsion Laboratory. The two programs have a common base in five sustaining core research thrusts and the goal of providing ground laboratory demonstrations of system-level breadboard technology readiness. Both programs are under the NASA headquarters supervision of Lee B. Holcomb, Director of Code RC, Information Sciences and Human Factors.

The major elements of the NASA Telerobotics Research Program are as follows:

a) A five-year R&D plan derived by a NASA-wide Telerobotics Working Group
b) A cooperative NASA center/university/industrial effort
c) A system integration testbed maintained at JPL
d) Planned periodic demonstrations of an evolving system capability
e) NASA technology transfer via the Flight Telerobot Servicer
f) Periodic program review by an external Telerobotics Technology Advisor Committee.

In addition to core research activities, which will be described in greater detail shortly, the Code R Telerobotics Program funds the development of a system integration testbed. The main theme of the testbed is that the definition and integration of system-level control for a telerobot is a very challenging research problem in itself: to meaningfully address this problem, both in concept formulation and evaluation, we believe such a real development environment is essential.

Core Research

The five NASA core Automation and Robotics (A&R) research thrusts are summarized in Figure 2. These five research thrusts, as interpreted for the Telerobotics Program, are intended to provide the technology base from which to integrate an evolving telerobot testbed. Figure 3 summarizes the participation of the various NASA centers in the telerobotics core research; Figure 3 is illustrative only - it is neither comprehensive nor does it address research specific to the Systems Automation program.

Sensing and perception deals with acquiring a perception of the environment, specifically determining the geometry, dynamics, identity and relationships of objects. The emphasis is dynamic work piece acquisition and manipulation, i.e., to acquire, track, physically stabilize, and verify objects of interest.

Task planning and reasoning is the area of logical planning and spatial reasoning of activity sequences together with a strong concern for the robustness of performance and the efficient use of human-machine resources. A rather challenging problem in this regard concerns integrating sequential (or even parallel) activity planning with spatial reasoning and coordination. Another important problem involves integrating into the overall system architecture an execution sequence monitoring strategy which goes beyond simple error diagnostics. Error correction may require physical reasoning and backtracking of error in a dynamic replan mode.

The area of operator interface has to do with the interaction between the human supervisor and the autonomous entity, the robot. The emphasis is on cognitive and physical work load reduction, and deals with not only teleoperated or robotic modes, but also shared task modes. Task state presentation is thematic in this development. Dexterous hand control is a major program focus; telepresent feedback through stereo vision force, tactility and various aspects of cross modal and virtual display are also active areas of research and development.

Control execution encompasses all sensor-referenced elements of both the manipulator and platform control. Control schemes here may include not only the classical position and force/torque control, but also more sophisticated adaptive and actively compliant control. There is a strong focus on the synergistic development of advanced control laws and their real-time implementation. This effort includes integrated smart end effectors and various aspects of high-speed, six degree of freedom teleoperations. We are pursuing elements of both hybrid and adaptive control, including management of manipulator redundant degrees of freedom, and elements of multiple manipulator equal-status control.

The fifth core area, system architecture and integration addresses: a) system-scale functional allocation and coordination of automated task sequences, b) run-time exchanges between automated and teleoperative control modes, c) more advanced concepts for shared functions between operator and robot, and d) hierarchical and distributed computer implementations of the above.

Telerobot Testbed

As the coordinating center for the NASA Code R Telerobotics Program, JPL is developing a host-site test-bed. The objectives, approach and significance of the system integration testbed are as follows:

(1) OBJECTIVE: Define and develop telerobot system-level arcitecture for variably structured space servicing, assembly, repair operations

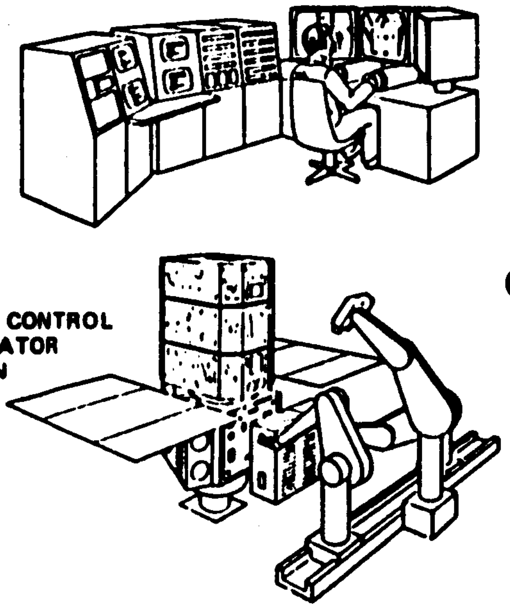
- (a) End-to-end hierarchically integrated run-time control
- (b) Run-time exchange of automation/teleoperations
- (c) Knowledge base generation and automated activity planning for unstructured workspace
- (d) Dynamic sensor allocation strategy for management of workspace uncertainty, execution errors, and anomalous conditions

TASK PLANNING & REASONING

LOGICAL PLANNING AND SPATIAL REASONING OF ACTIVITY SEQUENCES LEADING TO ROBUST AND EFFICIENT TASK COMPLETION

SYSTEM ARCHITECTURE & INTEGRATION

COMPUTING ARCHITECTURES ENABLING RUN-TIME EXECUTION OF HIERARCHICALLY STRUCTURED ROBOT CONTROL FUNCTIONS, INCLUDING THE SMOOTH TRADE OR SHARING OF THESE FUNCTIONS WITH TELEOPERATION



OPERATOR INTERFACE

HUMAN-MACHINE COMMUNICATION/ CONTROL INTERFACES WHICH MINIMIZE OPERATOR COGNITIVE/PHYSICAL WORKLOAD IN TELEOPERATOR, ROBOT, OR SHARED TASK MODES

CONTROL EXECUTION

SENSOR REFERENCED ROBOT MANIPULATOR AND PLATFORM CONTROL MECHANIZATION, CHARACTERIZED BY RULE-BASED AS WELL AS MODEL-BASED/PARAMETER—ADAPTIVE CONTROL LAWS

SENSING & PERCEPTION

SENSOR-DERIVED UNDERSTANDING OF THE TASK ENVIRONMENT - SPECIFICALLY, THE DETERMINATION/ VERIFICATION OF OBJECT GEOMETRIES, DYNAMICS, IDENTITIES, AND INTERRELATIONSHIPS

Figure 2. NASA Core Research Supporting Telerobot Development

ELEMENT \ CENTER	ARC	GSFC	JPL	JSC	LARC	MSFC
SENSING & PERCEPTION	• OPTICAL PROCESSOR FRONT END FOR SYMBOLIC PROCESSOR		• COMPUTER VISION FOR DYNAMIC SCENE UNDERSTANDING • SENSOR FUSION	• LASER 3 D SENSING	• LASER 3-D SENSING • SMART (FOCAL PLANE) SENSOR DESIGN	
TASK PLANNING & REASONING		• CAD FOR ROBOT ASSEMBLY DIS ASSEMBLY TASKS • SPATIAL PLANNING	• TASK SEQUENCE PLANNING • SPATIAL PLANNING • AUTOMATED ERROR DIAGNOSIS			
OPERATOR INTERFACE	• WORK STATION DESIGN • 3 D DISPLAYS		• NONREPLICATED FULL DOF TELEOP-ERATIONS • SUPERVISORY CON-TROL LANGUAGE • TELEROBOT EVA INTERACTION/PERF EVALUATION (MIT)	• REDUNDANT WORK-STATION MANAGE-MENT		• HUMAN-MACHINE INTERFACE RE-QUIREMENTS FOR OPTIMIZED TELE-OPERATIONS (DIS-PLAY/COMMUNI-CATION/TASK DESIGN)
CONTROL EXECUTION	• FLEXIBLE MANIPULA TOR CONTROL		• UNIVERSAL MANIPU-LATOR CONTROL INTERFACE DESIGN • MULTI-ARM COOP-ERATIVE MANIPULA-TION • GENERIC SMART END-EFFECTOR	• MULTI-ARM DYNAMICAL CONTROL • APPLICATION SPECIFIC END-EFFECTOR DESIGN	• ADAPTIVE/OPTIMAL MANIPU-LATOR CONTROL • DUAL ARM MANI-PULATOR CON-TROL • FLIGHT-QUALI-FIABLE MANIPU-PULATOR DESIGN	
SYSTEM ARCHITECTURE & INTEGRATION		• TASK/PAYLOAD DESIGN • TASKBOARDS	• RUN TIME HIERAR-CHICAL CONTROL • TRADED-SHARED TELEOPERATION/AUTOMATION TASK EXECUTION • INTEGRATED SENS-ING/CONTROL FOR UNCERTAIN WORK-SPACE	• FLIGHT EXPERI-MENT CONCEPTS & DESIGN • TELEROBOT PER-FORMANCE EVAL-UATION	• TELEOPERATIONS 7-DOF CONTROL ARCHITECTURE	

Figure 3. Telerobotics Research Thrusts of NASA Centers

(2) APPROACH: Pursue through center-of-excellence based on host-site system integration testbed

(a) Breadboard level technology development and integration
(b) Integration products derived from NASA center core research contracts
(c) Peer-evaluated core research plan and progress
(d) Testbed development plan for evolving system capability
(e) Code R/Code S technology transfer plan to guide and insure product assurance
(f) Industrial contracts to drive system engineering approach
(g) On-site collaboration of contractors and university PI's/research teams

(3) SIGNIFICANCE: Systematic evolution and verification of a telerobot technology concept which directly addresses the future domain, application, and operational reliability requirements of NASA

(a) Fully identifies the research needed to target component/core technologies
(b) Stimulates contributions and involvement of peer-research community
(c) Makes explicit contribution to terrestrial A&R effort
(d) Provides technology feedthrough Mars Rover Mission
(e) Insures that OAST A&R Research effort directly addresses requirements and design issues of flight/applications centers

The aim of the testbed is to define an evolving telerobot system level architecture for space servicing, assembly and repair operations. Enabling components for this capability include, for example, a hierarchical integrated run-time control architecture, and an interactively automated generation of knowledge base and activity plans for tasking in an unstructured workspace. As stated earlier, the approach uses NASA centers of excellence for penetrating selected research areas that support the integration testbed. These technologies will be developed to the breadboard level. An important element here is the collaboration of NASA centers with industry and university contractors.

The planned capability of the 1987-1990 telerobot testbed is summarized in Figure 4 and elaborated in Figures 5 and 6. In 1987, we plan the development of a task automation capability in which trajectory control and elementary off-line task planning are integrated and verified for basic execution sequences on a simple taskboard. In 1988, a full 6-DOF bilateral force-reflecting teleoperations capability will be integrated with an upgraded version of the automation sub-architecture. This initial capability telerobot will be exercised in a satellite repair servicing demonstration derived from the Solar Maximum Mission repair scenario - see Figures 7 and 8. Work leading to the 1987 automation capability is largely a JPL in-house effort. The 1988 telerobot design, implementation and integration work is collaborative with external systems contractors. In 1989, we expect to substantially increase performance of the 1988 telerobot through functional augmentations in most telerobot subsystem areas. This effort will be leveraged by initial technology deliveries from other participating NASA centers and from JPL contractors. Deliveries planned to have a major functional impact include strategies for cooperative (equal-status) dual arm manipulator control, active compliance manipulation strategies, AI-planning capability allowing some degree of on-line error recovery, and an expanded machine vision database. In 1990, the testbed is planned for development to its second major demonstration. This evolution demonstration is planned to incorporate mobility of both the taskboard and robot servicing platform.

To give the reader a better sense of the design approach to the testbed, we now present a brief overview of the testbed architecture. These following remarks are directed to the 1988 telerobot design goal but are conceptually valid for the overall 1987-1990 testbed development exercise. The reader may find it helpful to refer to Figure 6 during the various subsystem discussions:

Now we turn to a more detailed view of the 1988 demonstration and control structure (see Figure 7). The objective in 1988 is the integration of the telerobot. Figure 8 depicts the task board for the demonstration. It is a simulation of the Solar Max satellite. The satellite rotates slowly. The full unlabelled object is tracked, acquired, and stabilized. Then, as necessary, it is rotated for workpiece verification and acquisition. Dual arm coordinated control is required for implementing many of the tasks defined for the demonstration. Also, the capability for a run-time traded autonomous/teleoperative control will be demonstrated.

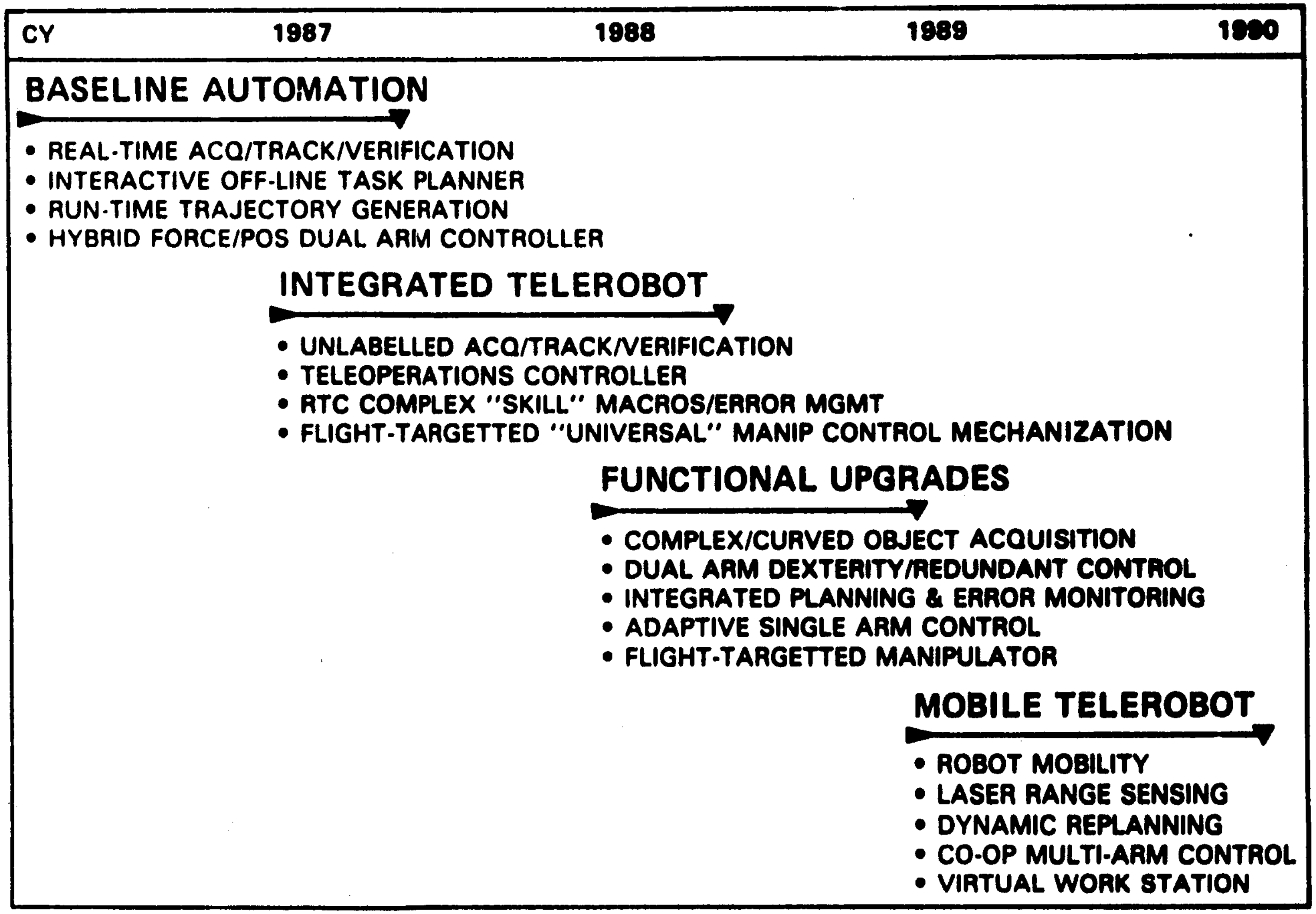

CY
1987
1988
1989
1990
BASELINE AUTOMATION
• REAL-TIME ACQ/TRACK/VERIFICATION
• INTERACTIVE OFF-LINE TASK PLANNER
• RUN-TIME TRAJECTORY GENERATION
• HYBRID FORCE/POS DUAL ARM CONTROLLER
INTEGRATED TELEROBOT
• UNLABELLED ACQ/TRACK/VERIFICATION
• TELEOPERATIONS CONTROLLER
• RTC COMPLEX "SKILL" MACROS/ERROR MGMT
• FLIGHT-TARGETTED "UNIVERSAL" MANIP CONTROL MECHANIZATION
FUNCTIONAL UPGRADES
• COMPLEX/CURVED OBJECT ACQUISITION
• DUAL ARM DEXTERITY/REDUNDANT CONTROL
• INTEGRATED PLANNING & ERROR MONITORING
• ADAPTIVE SINGLE ARM CONTROL
• FLIGHT-TARGETTED MANIPULATOR
MOBILE TELEROBOT
• ROBOT MOBILITY
• LASER RANGE SENSING
• DYNAMIC REPLANNING
• CO-OP MULTI-ARM CONTROL
• VIRTUAL WORK STATION

Figure 4. Telerobot Capability Integration Plan

YEAR	TECHNOLOGY SYSTEM TARGETS	DEMONSTRATION SCENARIO	PRODUCTS/ DELIVERABLES
1987	• INTEGRATE ROBOT AUTOMATION ELEMENTS • VALIDATE SYSTEM DESIGN CONCEPT, FUNCTIONAL PARTITIONING, COMMAND HIERARCHY AND DATA TRANSFER	• AIP (OFF-LINE) GENERATES A TASK SEQUENCE • VISION SUBSYSTEM DETERMINES COORD-INATES OF SPECIFIC OBJECT • RTC CREATES MANIPULATOR TRAJECTORY PLAN • M&CM TRANSFORMS DIRECTIONS TO SERVO COMMANDS. ARMS MOVE AND GRASP MANIPULATE OBJECT	• HIERARCHICALLY INTEGRATED AI-RTC-MCM ROBOT CONTROLLER • SYSTEM VALIDATED – PIFEX VISION SYSTEM – AI PLANNER S/W – RUN-TIME CONTROLLER S/W – RCCL HYBRID F/P S/W – DUAL ARM CONTROL S/W
1988	• INTEGRATED TELEROBOT SYSTEM • HUMAN FACTORS ENGINEERED OPERATOR CONTROL INTERFACE • CONTROL TRANSFER, TELEOP-AUTONOMOUS • INTEGRATION OF UPGRADED AND ADVANCED SUBSYSTEMS	• ACQUIRE AND GRASP SLOWLY ROTATING SATELLITE SIMULATOR - VISION SUB-SYSTEM SENDS HIGH SPEED POSITION DATA TO MCM, TWO ARMS GRASP MOVING SATEL-LITE AND BRING TO A HALT • PLAN AND EXECUTE AN AUTONOMOUS TASK WHICH INVOLVES A SEQUENCE OF TASKS • PERFORM TELEOPERATION TASKS WITH FORCE REFLECTING DUAL ARMS • TRANSFER CONTROL BETWEEN TELEOP-ERATION AND AUTONOMOUS COMPUTER CONTROL	• VISION ALGORITHMS/PIFEX UPGRADE • ENHANCED AI PLANNER & RTC S/W • INTEGRATED FRHC/UCCS/RCCL HARDWARE AND SOFTWARE • INTEGRATED OPERATOR CONTROL STATION • VALIDATED CONTROL SOFTWARE FOR TELEOP-AUTONOMOUS CONTROL TRANSFER
1990	• ROBOT MOBILITY • DIRECT RANGE SENSING • EQUAL AND COOPERATIVE CONTROL OF 7 DOF FLIGHT TARGETED MANIPULATOR ARMS • TELEOPERATIONS CONTROL OF 7 DOF ARMS • COGNITIVE/VIRTUAL DISPLAYS	• A MOBILE ROBOT ACQUIRES TRACKS AND GRASPS A SATELLITE SIMULATOR • THE ROBOT SECURES ITSELF TO THE SATELLITE IN PREPARATION FOR PER-FORMING MAINTENANCE AND REPAIR SERVICING FUNCTIONS • THE AI PLANNER AND RTC PLAN AND EXECUTE A GREATER VARIETY OF TASKS THAN PERFORMED IN 1988. MORE INFOR-MATION IN THE FORM OF RANGE AND TACTILE DATA WILL BE AVAILABLE AND THE MANIPULATOR ARMS WILL HAVE MOTION REDUNDANCY SO THAT INTRI-CATE TASKS CAN BE PERFORMED IN HARD TO REACH PLACES • ERROR DETECTION AND RECOVERY WILL BE FUNCTIONS PERFORMED AT EACH CONTROL LEVEL. ALL ACTION WILL BE COORDINATED THROUGH THE CONTROL HIERARCHY	• PLATFORM/CONTROL MECH. AND ALGORITHMS • LASER FUSED SENSING • SYSTEM VALIDATED – 7-DOF ARMS – CONTROL OF 7-DOF ARMS – FUSED SENSOR DATA – ENHANCED PLANNING, EXECUTION, ERROR DETECTION AND RECOVERY SOFTWARE • TELEOP CONTROL OF 7-DOF ARMS • VIRTUAL WORKSTATION
1990-2000	• FULL AUTONOMOUS NAVIGATION IN A COMPLEX ENVIRONMENT • REAL TIME MONITORING AND RE PLANNING TO DEAL WITH UNEXPECTED OCCURRANCES • EXPANDED SENSING SUITE • ENHANCED AI PLANNING AND COOPERATIVE INTERACTION OF MULTIPLE ROBOTS	• OBJECT RECOGNITION AND ACQUISITION IN A CLUTTERED ENVIRONMENT • REAL TIME REPLANNING AND IMMEDIATE IMPLEMENTATION OF CORRECTIVE ACTION TO AVOID UNFORESEEN HAZARDS OR OVERCOME PROBLEMS • THE HUMAN OPERATOR FUNCTIONS WILL SHIFT TOWARD MONITORING AND SUPERVISION • TWO OR MORE MOBILE ROBOTS WILL BE USED TO COOPERATIVELY PERFORM CONSTRUCTION AND/OR SERVICE TASKS	• A COMPLEX OF SENSORS WITH SOFTWARE FOR PROCESSING, CON-DENSING AND INTERPRETING THE DATA • DEXTEROUS AND SMART END-EFFECTORS • ENHANCED AI PLANNING AND RTC SOFTWARE TO PERFORM ERROR DETECTION AND REPLANNING • PLANNING AND MONITORING SOFT-WARE FOR CONTROLLING AND INTE-GRATING MULTIPLE ACTIVE ROBOTS

Figure 5. System Integration Testbed

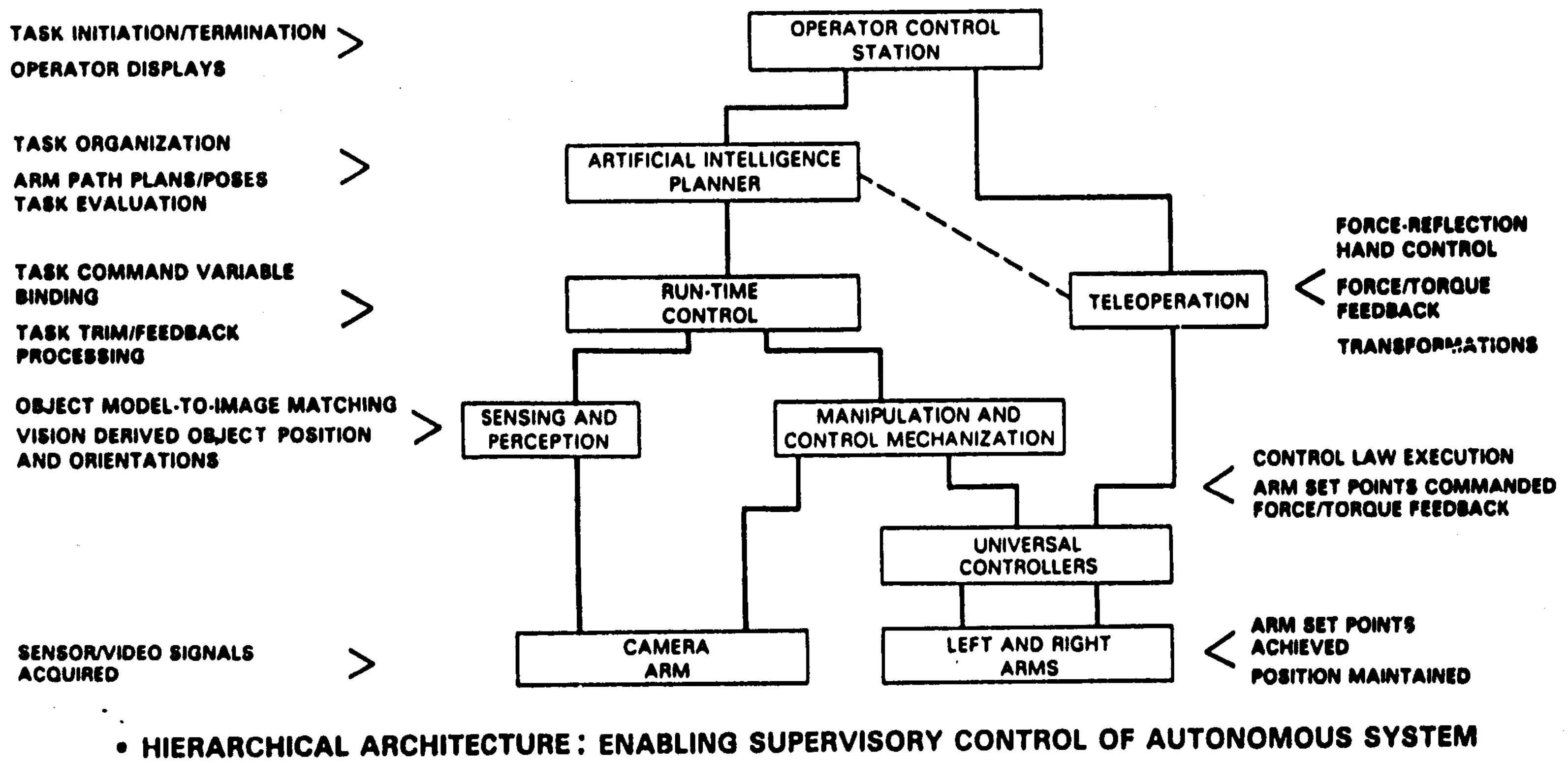
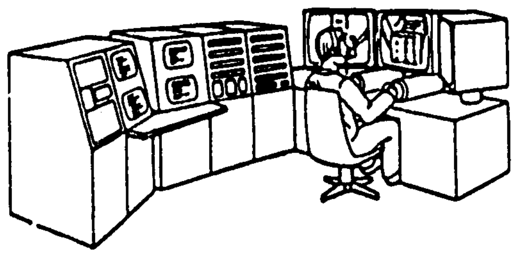

TASK INITIATION/TERMINATION
OPERATOR DISPLAYS
TASK ORGANIZATION
ARM PATH PLANS/POSES
TASK EVALUATION
TASK COMMAND VARIABLE BINDING
TASK TRIM/FEEDBACK PROCESSING
OBJECT MODEL-TO-IMAGE MATCHING
VISION DERIVED OBJECT POSITION AND ORIENTATIONS
SENSOR/VIDEO SIGNALS ACQUIRED
OPERATOR CONTROL STATION
ARTIFICIAL INTELLIGENCE PLANNER
RUN-TIME CONTROL
SENSING AND PERCEPTION
MANIPULATION AND CONTROL MECHANIZATION
TELEOPERATION
UNIVERSAL CONTROLLERS
CAMERA ARM
LEFT AND RIGHT ARMS
FORCE-REFLECTION HAND CONTROL
FORCE/TORQUE FEEDBACK
TRANSFORMATIONS
CONTROL LAW EXECUTION
ARM SET POINTS COMMANDED
FORCE/TORQUE FEEDBACK
ARM SET POINTS ACHIEVED
POSITION MAINTAINED
• HIERARCHICAL ARCHITECTURE: ENABLING SUPERVISORY CONTROL OF AUTONOMOUS SYSTEM

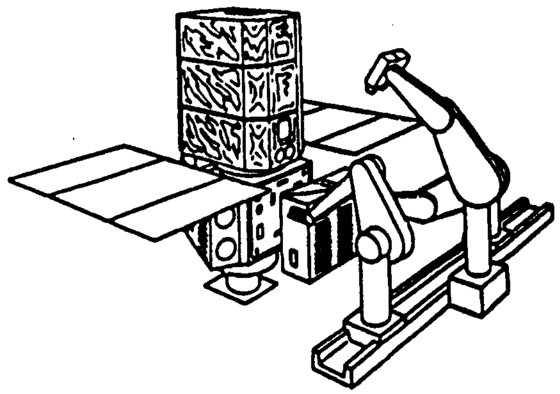
Figure 6. Functional Architecture

STATIONARY ROBOT, SIMPLE SPACECRAFT SERVICING TASKS, SUPERVISORY CONTROL

TECHNICAL ADVANCES

- SPACE SERVICING PRODUCTIVITY IMPROVEMENT
- DUAL-ARM COOPERATION
- TOOL GRASP/APPLICATION





CONTROL STATION

- STEREO DISPLAYS
- TWO-ARM BILATERAL FORCE-POSITION CONTROL
- VOICE RECOGNITION/SYNTHESIS
- OFF-LINE INTERACTIVE PLANNING

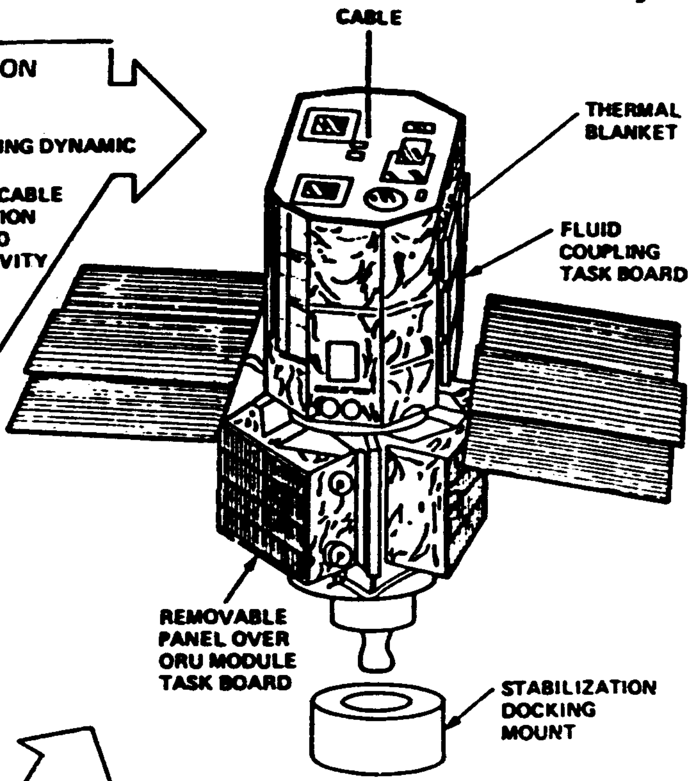
RUN TIME CONTROL/PERCEPTION SYSTEM

- AUTOMATED TASK FRAME ACQUISITION, TRACKING, AND WORKPIECE VERIFICATION
- AUTOMATED RUN-TIME CONTROL SYNTHESIS AND SEQUENCING
- RUN-TIME TRADED AUTONOMOUS/TELEOPERATIVE CONTROL

Figure 7. 1988 Telerobot Demonstration

TASKBOARD DESCRIPTION

- SIMULATE SATELLITE (i.e. SOLAR MAX)
- MULTIPLE DEGREES OF FREEDOM DURING DYNAMIC ACQUISITION AND GRAPPLING
  - COUNTER WEIGHTED SUSPENSION CABLE ALLOWS 3 DEGREES OF TRANSLATION
  - SUSPENSION CABLE ATTATCHED TO GIMBAL JOINT AT CENTER OF GRAVITY ALLOWS 3 DEGREES OF ROTATION
- ACCESS PANELS TO ORU CARD CAGES, FUEL INPUT, ETC.
- GRAPPLING FIXTURES
- THERMAL BLANKET
- DOCKING MOUNT TO PROVIDE STABILIZATION DURING SERVICING SEQUENCES


CABLE
THERMAL BLANKET
FLUID COUPLING TASK BOARD
REMOVABLE PANEL OVER ORU MODULE TASK BOARD
STABILIZATION DOCKING MOUNT

MODULE CHANGEOUT PROTOTASK SEQUENCE

- ACQUIRE, TRACK, AND STABILIZE TASK COORDINATE FRAME
- OPEN THERMAL BLANKET, SECURE
- ACQUIRE POWER DRIVER
- LOOSEN CAPTIVE SCREWS WITH POWER DRIVER WHILE HOLDING COVER
- STOW POWER DRIVER
- REMOVE AND STOW COVER
- RELEASE MODULE RETENTION CLAMPS
- WITHDRAW AND STOW MODULE
- ACQUIRE AND INSERT REPLACEMENT MODULE
- TIGHTEN MODULE RETENTION CLAMPS
- ACQUIRE COVER
- ALIGN AND HOLD COVER
- ACQUIRE POWER DRIVER
- TIGHTEN CAPTIVE SCREWS WITH POWER DRIVER
- TEST SCREWS
- STOW POWER DRIVER
- RELEASE, GRASP THERMAL BLANKET
- DRAPE BLANKET
- FASTEN BLANKET
- INSPECT

Figure 8. 1988 Telerobot Demonstration Taskboard

The telerobot allows the supervising operator two basic modes of control: teleoperation and robot automation. Further, the operator has the ability to smoothly switch between these modes during run-time task execution. The normal task execution scenario is:

a) Operator chooses reference task.
b) Operator specifies automated vs. teleoperative task sequence.
c) Telerobot plans automated task sequences.
d) Telerobot previews automation task sequences.
e) Telerobot identifies and binds teleoperation/automation control trade points.
f) Task instantiated and run time execution implemented.

Should an off-normal event occur during task execution, the above scenario is modified. Teleoperation may now be invoked on an ad hoc basis at any point in the task execution time line. The most common reason for the operator to designate teleoperative intervention is development of unexpected conditions in the workspace. One example is discovery that the taskboard structure deviates from its CAD reference in an unanticipated way (and cannot be updated through run time sensing). Another example is uncorrectable error in the automated task sequencing, e.g., a tool is grossly misaligned or dropped. If the operator does interrupt automated task sequencing with teleoperation, he may yet still implement subsequent planned automation sequences without off-line reinitialization of the entire task. This is allowed by the "monitor point" feature of the telerobot control architecture. Monitor points are in essence nominal telerobot/workspace configurations known to occur at each preplanned trade of teleoperation-to-automation control. If the operator assures that a monitor point task configuration has been obtained at the conclusion of a teleoperation interrupt, then he can smoothly release control to automation in run time.

(Operator control station - OCS)

The OCS is the physical and cognitive interface with the operator. In a real flight system scenario it represents the "local" or base site for remote operation of the robot. The OCS contains the hand controller interfaces for manual position/rate teleoperation of the robot manipulators. The OCS has voice-interactive and keyboard-entry interfaces for command/verification of automated sequencing of the robot manipulators. Finally, the OCS has numerous textual and graphic displays (including high-fidelity wide-field monochromatic stereo) which provide the operator with both physical and semantic representations of the task state. The displays can be used both for run time task supervision and predictive simulation.

(Infrastructure - IS)

The IS is a distributed hardware/software subsystem which performs three principal functions. First, through a centralized system executive, the IS establishes the system electronic configuration and instantiates the initial task database. Second, the IS enables system-wide control/communication via a wiring harness, hardware/software subsystem driver interfaces, and local area net (LAN) protocol. Finally, the system executive performs system self-test and calibration at start-up, as well as system health monitoring and data collection/monitoring during run time.

(AI planner - AIP)

The AIP, with operator interaction, develops the task script, timeline, and bound global variables of the run time execution sequence. In the initial demonstrator operational capability, the AIP performs off-line planning and on-line task montoring with a limited capability for high-level error diagnosis. Out-year development for the 1990 evolution demonstration targets and AIP with capability for on-line planning and dynamic error recovery. The AIP performs both logical and spatial planning functions. The resulting plan specification is in the form of high-level task move sequences: e.g., move right manipulator arm from task frame effector coordinate A to coordinate B through permissible task space freeway volume V. As part of the plan generation process, the AIP identifies the above-described "monitor points" which are task configurations which guarantee safe re-entry from teleoperation to automated sequencing.

(Run time controller - RTC)

The RTC accepts from the AIP the nominal task plan and sequentially interprets it as a set of run time executable "actions." These actions may be of simple or complex form, the latter being termed a "skill." A simple action will typically result in a single RTC command to the manipulator control or sensing subsystem; a complex action will be interpreted as an integrated sequence of commands to the manipulator control and/or sensing subsystems. The main function of the RTC is computation of detailed physical plans for manipulator motion control. These plans are computed in run time, based on the most currently available task world/state knowledge, and utilize exact geometric models of the work-space objects as well as actual geometric-kinematic-dynamical constraints of the robot manipulators. Thus, when the AIP commands a nominal effector trajectory " A-> B through freeway V ", the RTC nominally does the following:

a) Indexes the appropriate skill/action for the AIP command
b) Binds the AIP command parameters (move sequence/geometry)
c) Determines an appropriate terminal stance/pose of the manipulator(s)/effectors(s) for the task-specific command
d) Computes a collision-free arm trajectory(s) through the joint configuration space
e) establishes a joint configuration space waypoint sequence for subsequent communication to the manipulator(s) controller
f) Parameterizes the manipulator(s) controller commands with appropriate controller mode indices and limits

In addition to action-skill commands, the RTC supports "overrides" and "requests". Overrides are software-driven execution stops which occur either at a designated monitor point, or as an interrupt of autonomous control into a halt state from execution of a pre-planned automation sequence. The request command does not affect run time motion control; rather, it either accesses the RTC data base with a query, or, commands computation supporting a predictive simulation.

(Manipulation and Control Mechanization - M&CM)

The M&CM subsystem is the lowest layer of the three-level automation control hierarchy comprised by the AIP-RTC-M&CM interface. The RTC passes to the M&CM the above described joint configuration waypoint list and control mode/parameter commands. The M&CM subsequently computes a smooth trajectory approximation through the waypoints and implements this solution via an embedded real-time controller. M&CM capability is in some instances expected to provide simple human-equivalent skills, e.g., grasp and handling of geometrically regular and extended objects. To this end, the initial M&CM architecture is being developed for a constrained form of dual-arm cooperative control: The left arm is commanded in position control mode; the right arm is commanded in hybrid (force/position) control. When a dynamical chain is closed between the two arms by their grasp of a common structure, the right arm reacts through force/torque sensor and position encoder feedback to maintain nominal force limits. The underlying approach to a single-arm hybrid control is derived from the Raibert-Craig approach; extension and application of this approach to multi-arm cooperating robots is described elsewhere in these Proceedings.

(Teleoperations - TOP)

The TOP subsystem is a distributed computing architecture providing the sensing, control, and display functions necesary to provide a high-fidelity, dual-arm, six degree-of-freedom (DOF) teleoperator capability. The general TOP design approach is a bilateral force-reflecting hand controller in which the master and slave arms can be both kinematically and dynamically dissimilar. The approach thus allows the independent design of a master (hand controller) which meets the requirements of the operator and local control station, and, slave (robot arm) which meets the needs of the remote application environment. The TOP subsystem initial operational capability has all DOF controlled by the operator; however, the general computer-based approach allows growth to shared modes of control in which some DOF are under operator manual control and other DOF are automated.

(Sensing and Perception - S&P)

The S&P subsystem is a video machine vision architecture which provides the automation controller with the capability to track, acquire, and verify position/orientation of man-made objects in real time. The approach is based on a set of computer vision algorithms and custom real-time video processor which are described elsewhere in these Proceedings by Brian Wilcox. The current approach is limited to simple polyhedral objects and use of multi-camera monochromatic intensity imagery. Out-year development for the evolution demonstration of 1990 targets a S&P subsystem with multi-sensor capability, specifically, integrated use of intensity-range information. A complementary goal is real-time capability for acquisition/verification of objects with complex curved surfaces.

Summary and Conclusions

NASA has undertaken an aggressive program of telerobotics research and technology development. The broad goal of the overall NASA program is technology definition and proof-of-concept in support of space servicing, assembly, and repair operations. The resulting telerobotic capability has potential to significantly impact space mission productivity, capability, and automation across a wide variety of scenarios. The NASA/OAST Telerobotics Program has the objective of providing a research base in advanced automation technologies, and defining how these technologies can be synergistically integrated into a system architecture for evolutionary telerobotics. The program is expected to contribute a variety of generic technology products in the areas of artificial intellig ce, robotics, and human factors and teleoperations. Examples of interest to both the space and terrestrial telerobotics communities include: task planners and expert error diagnosticians; control algorithms and real-time mechanization for multi-arm manipulation, as well as adaptive and redundant manipulation; cooperative articulation of robot mobility and manipulation; robust, full-DOF teleoperations controllers targetable to a wide variety of manipulators; effectors and accompanying modal displays; various command-display devices tailored to cognitive requirements of the human operator; and sensor-processor suites adapted to the particular domain requirements.

The NASA/OAST program has the broader systems objective of defining a cohesive architecture for advanced supervisory automation of space servicing tasks. Success is to be documented through a series of ground-based breadboard concept demonstrations which show an evolving capability for addressing increased task complexity, uncertainty, and automation. This work is hosted in the JPL telerobot testbed and draws upon the program core research for its supporting subsystem technologies. Productive work in this area is expected to have high payoff not only to space telerobotics, but to broader NASA applications, and beyond. Fundamental issues in supervised automation are being addressed: 1) identification of realistic run-time allocation of sensing, control, and performance monitoring functions between a human and machine for complex tasks; 2) definition of a resulting hierarchical shared control-and-communication architecture, particularly in addressing management of uncertainty in sensor data and control strategy; and 3) the longer-term development and evaluation of supervised learning strategies for instantiation of desired task procedures and sequences.

Acknowledgements

The preparation of this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Many people have contributed to the ideas and technology plans outlined in this paper. Major influences include the NASA Telerobot Working Group led by S. Szirmay (JPL), the JPL Telerobot Testbed Design Group led by J. Matijevic (Systems) and A. Sirota (Architecture Implementation), and the continuing advisement of Lee Holcomb (NASA HQ/Code RC). The author thanks M. Milman for his assistance in assembling this final manuscript version of the plenary talk. R. French helped to develop a number of the key figures used both in the original talk and this manuscript.

188083 S1-63
7P.

NC473651

Building Intelligent Systems: Artificial Intelligence Research at NASA Ames Research Center

P. Friedland and H. Lum
NASA Ames Research Center
Moffett Field, CA 94035

1. Introduction

The goal of building autonomous, intelligent systems is the major focus of both basic and applied research in artificial intelligence within NASA. This paper discusses the components that make up that goal and describes ongoing work at the NASA Ames Research Center to achieve the goal.

NASA provides a unique environment for fostering the development of intelligent computational systems; the test domains, especially Space Station systems, represent both a well-defined need, and a marvelous series of increasingly more difficult problems for testing research ideas. One can see a clear progression of systems through research settings (both within and external to NASA) to space station testbeds to systems which actually fly on space station.

2. The Long-Range Goal

As a springboard for discussion, let us create a view of a "truly" autonomous space station intelligent system, responsible for a major portion of space station control (the exact system is unimportant). We will build a view of all of the functions this system should have, which of those functions we can achieve (nearly completely) today, which we can easily see happening in the next few years with some engineering-oriented applied research, and which will be doable only with substantial basic research (over at least the next five years).

Our intelligent system will have full responsibility for a major functional component of space station; examples include power, communications, thermal management, and environmental control. It will be responsible for nominal control, acute and chronic failure discovery, diagnosis, and correction, and communications/cooperation with both interested humans (astronauts, ground controllers, scientists) and other intelligent computational systems. The following specific abilities are needed to be able to completely satisfy those responsibilities:

- Scheduling of System Resources to Meet Utilization Requests--the ability to analyze tasks that involve the space station component being controlled in order to set up targets for resource utilization.
- Real-Time Schedule Execution and Monitoring--the ability to translate task requests into executable system commands and to understand if the tasks have been adequately performed.
- Dynamic Schedule Modification--the ability to change the resource utilization schedule to reflect both internal (poorly performed or understood tasks) and external constraints (new tasks added, conflicts with other space station systems, etc.).
- Heuristic/Experiential Failure Detection and Diagnosis--the ability to utilize "shallow" knowledge based mainly upon prior (human or machine) experience with the space station system in order to notice and diagnosis problems with the system.

PAGE 18 INTENTIONALLY BLANK

- Causal/Model-Based (Both Qualitative and Quantitative) Failure Diagnosis--the ability to utilize deeper, "first-principle" knowledge to diagnose system problems.
- Planning of Failure Corrections--the ability to determine a course of actions to repair a diagnosed failure.
- Realtime Failure Correction and Monitoring--the ability to translate a correction plan into actions and understand if the problem has, indeed, been fixed.
- Long-Term Trend Analysis--the ability to understand slow-to-develop trends either to prevent incipient failure or to adjust nominal control over a long period of time.
- Explanation of Actions--the ability to explain, with clarity and brevity, all system actions, from resource utilization schedules to failure diagnosis to failure correction actions. Clarity and brevity imply different explanations for different classes of humans who wish to interact with the system.
- Cooperation with Other Intelligent Systems--the ability to work in concert with other intelligent entities on space station, both human and computational.

Four other necessary abilities cross many functional lines for our truly autonomous system. These are:

- Reasoning Under Uncertainty--the ability to make sensible judgments and carry out reasonable actions when world knowledge is imprecise or incomplete, heuristics or models have built-in uncertainty, or actions have uncertain effects.
- Learning--the ability to alter and improve all functionalities as conditions change and knowledge is added over time. Starting views of how systems operate in space may be wrong. Fault correction actions may alter system configurations. An autonomous system cannot remain static or performance will at best not improve or (more likely) at worst degrade to completely unacceptable.
- Common Sense Reasoning--the ability to occasionally go beyond specific domain knowledge into broad areas of human experience. This includes the ability to bypass established reasoning mechanisms when unexpected events render them clearly useless.
- Self-Understanding--the ability to understand:
    - When and how to utilize different functional abilities such as heuristic as opposed to model-based diagnostic reasoning.
    - When to act independently as opposed to asking for human or other computational assistance.
    - How to prioritize actions--when are things critical and when can they wait.
    - When a problem is beyond the system's range of understanding.
    - When new knowledge is worth saving either directly or as part of a new plan or piece of a model.

Finally, to build such intelligent, autonomous systems, three pieces of artificial intelligence methodology become critical. These are:

- Better Knowledge Acquisition--great improvement in our abilities to acquire, both initially and dynamically via sensor and other input, the heuristic and causal knowledge our system requires. In other words, tools and techniques for constructing the knowledge base.

- Managing of Large Knowledge Bases--how to manage, optimize, check for internal consistency and completeness, etc., knowledge bases at least a few orders of magnitude larger than those used by current large systems.
- Validation of Intelligent Systems--both practical and philosophical understanding of what it means to validate autonomous, computational systems. For space station systems, a mutual education and satisficing process with Space Station personnel and the aerospace industry will be part of the task.

3. Where We Are

The state of the art is marked by the three orthogonal areas of kinds of knowledge we can represent, tools for knowledge base construction and manipulation, and kinds of problem solving. Using one of the three representation paradigms of rules, frames, and logic, we have engineering solutions for storing a wide variety of heuristic/procedural knowledge and factual/declarative knowledge. We have commercial tools (KEE, S.1, KnowledgeCraft and ART are the outstanding examples) which reduce the knowledge base construction time by at least two orders of magnitude over raw LISP (or FORTRAN, PROLOG, etc. for that matter) and which provide a selection of inference methodologies. We can routinely carry out the entire knowledge-based system building task for structured selection problems using experiential knowledge, component configuration problems, fairly complex scheduling problems, simple planning problems, and "intelligent front-ends" for abstruse modeling and database computer programs.

In addition, we can represent and utilize a very limited amount of probabilistic or conditional knowledge about data and knowledge. Our systems can explain their chain of reasoning in simplistic textual and graphical forms. We can also construct quite sophisticated, personalized interfaces to knowledge-based systems. Finally, we can utilize precisely the same knowledge base for distinct purposes, including diagnosis, simulation, and training.

4. What We Can Almost Do

In several other areas, basic research is beginning to make the transition into engineering utilization. Specifically:

- Causal models are beginning to be used on significant problems, augmenting and/or replacing experiential heuristics for diagnosis and fault correction.
- Systems are moving from "leisurely" offline applications into realtime control systems (where realtime currently means on the order of seconds for evaluation and response time).
- Systems that dynamically reconfigure plans to reflect changing conditions during plan execution are beginning to appear in real applications.
- The blackboard framework for cooperative utilization of different sources of knowledge is becoming part of the inference "toolkit." This represents a "micro" view of distributed control among cooperating knowledge-based systems in most current applications.
- Machine learning, still in very simple forms such as explanation based generalization, or learning by example, is starting to make an impact on fielded knowledge-based systems.

5. What We Need to Do

Clearly there is an enormous amount of work to be done to take us from where we are (or soon will be) in the technology of building autonomous systems to a point where the long-range goal is satisfied. Just as clearly, we at NASA Ames cannot do it all. However, the problem areas discussed in this document are relevant to most current work in artificial intelligence research, so we have a wide community of fellow researchers to draw upon. The following are the scientific and engineering research areas in which we at NASA Ames believe our involvement through internal work or support of external work makes sense over the next several years:

- Reasoning Under Uncertainty--as discussed above, particularly focusing on integration into practical knowledge acquisition and representation frameworks and demonstrations of utility.
- Machine Learning--emphasizing automatic knowledge base expansion and correction as well as learning by discovery (carrying out sensor-based "experiments, etc.).
- Causal Modeling and Simulation--particularly on methodologies for integrating these methods with both heuristic-based problem solving at one end and mathematical model based simulation at the other.
- Next Generation Tools for Knowledge Acquisition, Representation, and Manipulation--developing better, faster, more versatile tools to "routinize" much of the knowledge engineering process. Of particular importance is getting past the knowledge acquisition bottleneck and removing a human knowledge engineer from the loop as much as possible.
- Explanation and Interface Technology--making progress in communicating between knowledge-based systems and human users. We wish to add perspicuity and perspicacity of explanation, and make it much more easy to customize interfaces to individual preferences.
- Constructing and Utilizing Large Knowledge Bases--fundamental experiments in treating very large collections of knowledge.
- Acquisition of Design Knowledge and Data for Complex Systems--ensuring that the vast amount of information used and discovered during the design process not be lost for future knowledge-based systems that need to reason about the artifact that was constructed.
- Advanced Methods for Plan Construction, Monitoring, and Modification--integration of current planning methods to ensure usability and flexibility within practical environments
- Hierarchical Control of Multiple Knowledge-Based Systems--experiments in control of multiple systems by hierarchical levels of increasingly more general knowledge-based systems.
- Distributed Cooperation among Multiple Knowledge-Based Systems--the alternate view to the previous topic: coordinated control by cooperative information sharing through a common database or "blackboard."
- Validation Methodologies for Knowledge-Based Systems--both low-level issues of software verification (eliminating redundancies, inconsistencies, etc.) and more fundamental issues of ensuring desired functionality.

Note that certain topics relevant to the Long-Range Goal, in particular common sense reasoning and self-understanding, are not specifically covered in the above research topics. That is because, in the author's view, meaningful work in those areas awaits fundamental progress in other areas like causal modeling and machine learning. There are several very long-term research projects already underway in the "missing" research areas (Lenat's CYC project at MCC is a good example) and we will keep careful track of those projects as part of our overall understanding of autonomous systems.

6. Getting It Done

The previous section has established ambitious goals for our research in intelligent systems. Over the next several years, we will be attempting to build a highly regarded basic research laboratory within the Artificial Intelligence Research and Applications Branch at Ames. As relevant personnel join the program, research projects in many of the above areas will commence. Indeed, as discussed below, several projects already exist or will begin in fiscal 1987. However, for the next year or so, our greatest leverage will come from sponsoring a modest number of extremely high quality external research projects. The external work, besides serving programmatic research needs, will also, we believe, act to educate Ames personnel through collaborative discussions, produce some short-term technology demonstrations, and provide a stream of motivated students, a significant number of whom we hope can be convinced to join NASA upon receipt of their advanced degrees. In addition, we will provide attractive mechanisms for senior researchers to carry out portions of their NASA-sponsored work at the Ames Research Center through sabbaticals and summer visitations.

The final two sections of this document will briefly describe current work in intelligent systems in progress and sponsored by the Artificial Intelligence Research Branch at NASA Ames Research Center.

6.1. Internal Research in Fiscal 1987

Reasoning Under Uncertainty

- Peter Cheeseman, John Stutz, Mary Duffy, et. al. will continue their work in probabilistic reasoning with test applications in planning and learning. The coming year will see a focus on NASA problem domains and comparisons with other schools of uncertain reasoning. Long term goals revolve around developing engineering methodologies for routinely including uncertain reasoning in relevant applications across many different forms of problem solving.

Machine Learning

- Michael Sims and Peter Friedland will initiate a project on learning by discovery. The starting point will be Sims' work in mathematical discovery applied to the engineered systems such as those on Space Station. The eventual goal is to achieve self-improving knowledge-based control and analysis systems.

Causal Modeling and Simulation

- William Erickson, Mary Schwartz, and Peter Friedland will use the space station thermal system as a testbed for work in integrating causal modeling with experiential heuristics and mathematical models. A short term goal is to demonstrate practical accomplishments as part of the 1988 Systems Autonomy Demonstration Program. A somewhat longer-term goal is to formalize the techniques for determining when each of the different forms of reasoning will be most useful for difficult problems.

Knowledge from Design Through Operations

- Cecelia Sivard and Lilly Spirkovska will continue their work on capturing design knowledge on the solar photometer system. A short-term goal is a proof of utility of the methodology, by showing how knowledge acquired during the design process can be effectively utilized during operations and maintenance of a coplex device. A long-term goal is bulding increasingly more sophisticated design tools that automatically construct knowledge bases useful during the entire design through operations process.

Advanced Planning Work

- Peter Cheeseman, et. al. will continue their work on dynamic scheduling and rescheduling. Peter Friedland will collaborate to see if skeletal planning methodologies can be integrated with the work. The eventual goal is to show realtime utility of the methodologies.

Validation Methodologies

- Peter Friedland and Carla Wong will begin a project in the practical aspects of knowledge-based system validation. This will include intense collaboration with Space Station and the space industry to prevent bottlenecks in the goal of a clear path to operational systems on space station. First results will be seen in the 1988 Systems Autonomy demonstration. A longer-term goal will be NASA-wide accepted methodologies for knowledge-based system validation.

6.2. External Research in Fiscal 1987

Reasoning Under Uncertainty

- Lotfi Zadeh, UC-Berkeley-- Dr. Zadeh is the world leader in the branch of uncertain reasoning known as fuzzy logic. During the coming year, we will begin the process of integrating his work with Ames-internal efforts.

- Don Heckerman and Eric Horvitz, Knowledge Systems Laboratory, Stanford University--Heckerman and Horvitz are students who have become leaders in both theoretical and practical aspects of the uncertain reasoning work that originated in the MYCIN project. During the next year they will begin to determine the relevance of those methodologies to NASA domains.

Machine Learning

- Tom Mitchell and Jaime Carbonell, Carnegie-Mellon University--Mitchell and Carbonell, are among the world's leading authorities on research in machine learning. The work will emphasize explanation based generalization and learning by example and include both theoretical aspects and practical applications to NASA domains. In addition, both Mitchell and Carbonell are likely to spend significant amounts of time at the Ames Research Center helping build our internal strengths in machine learning.

Advanced Planning Work

- David Atkinson, JPL--The AI research group at JPL will be conducting Ames-sponsored research in the integration of sensor-based planning, plan monitoring, and plan modification. This is the first effort in what we hope will be a long-term research collaboration with the JPL group.

Hierachical Control of Multiple Knowledge-Based Systems

- Ron Larsen, University of Maryland--We will continue to support Dr. Larsen's work on developing computational structures for hierarchical control of cooperating knowledge-based systems. His work is attempting to integrate traditional decision theory with heuristic and model-based control methods.

- Richard Volz, University of Michigan--Professor Volz is carrying out a variety of projects in sensor-based learning, manipulation, and scene understanding related to coordinated control of robotic systems. This research is being sponsored in cooperation with the JPL-led Telerobotics Program.

Distributed Cooperation among Multiple Knowledge-Based Systems

- Ed Feigenbaum and Bruce Buchanan, Knowledge Systems Laboratory, Stanford University--The Knowledge Systems Laboratory is conducting a significant new effort centering on methods for cooperative control among distributed expert systems. This work includes both long-term research projects resulting in Ph.D. theses and short term development of blackboard-based techniques for cooperative, distributed control.

- Tom Sheridan, MIT--We have been supporting Professor Sheridan's work in various methodologies for distributed and cooperative control in robotic systems. This work will continue as a joint project with the JPL-led Telerobotics Program.

OM:

SYSTEM CONCEPTS

188084 S2-37
68

ND185000

Space Telerobotic Systems: Applications and Concepts

L. Jenkins
NASA Johnson Space Center
Houston, TX 77058

1. Abstract

The definition of a variety of assembly, servicing, and maintenance missions has led to the generation of a number of space telerobot concepts. The remote operation of a space telerobot is seen as a means to increase astronaut productivity. Dexterous manipulator arms are controlled from the Space Shuttle Orbiter cabin or a Space Station module. Concepts for the telerobotic work system have been developed by the Lyndon B. Johnson Space Center through contracts with the Grumman Aerospace Corporation and Martin Marietta Corporation. These studies defined a concept for a telerobot with extravehicular activity (EVA) astronaut equivalent capability that would be controlled from the Space Shuttle. An evolutionary development of the system is proposed as a means of incorporating technology advances. Early flight testing is seen as needed to address the uncertainties of robotic manipulation in space. Space robotics can be expected to spin off technology to terrestrial robots, particularly in hazardous and unstructured applications.

2. Introduction

Increased operations in space with the Space Station and the Strategic Defense Initiative define a need for remote operating systems to assist the space crews in accomplishing a variety of new functions. The role of the space crew is changing, with more missions recognizing the benefits of servicing and maintenance as a cost-effective mode of operating satellites. The size of the Space Station mandates its assembly in space. Other large space systems will require assembly on orbit. Recent Space Shuttle missions have demonstrated the effectiveness of the extravehicular activity (EVA) crew in many of the tasks needed for future space construction, assembly, and maintenance. As the magnitude of mission requirements grows, the productivity of the astronaut must be increased. The use of EVA is crew time intensive; it requires a buddy system as well as an observer in the cabin. Time spent preparing to leave the cabin, prebreathing oxygen, and maintaining equipment add to nonproductive time. Remote operating systems are a means of amplifying space crew output [1]. One concept for remote operations that has been defined in some detail is the telerobotic work system (TWS).

The basic concept of the telerobotic work system consists of two dexterous manipulator arms controlled from a remote station (Figure 1). The direct control of the arms may be supplemented by interaction with a computer to perform certain tasks or portions of tasks. The tasks to be performed range from changing modules and components in the repair of satellites to the construction of large space systems like the Space Station. The operator is provided with sensory feedback of the environment and conditions at the work site. This approach is reflected in the term "telerobotics," which implies a combination of teleoperating and robotics. An objective of the system development approach is to increase the productivity of the operator through more robotic modes having a higher degree of autonomy [2].

A robot operating in the environment of space has analogies to a robot operating in hazardous or unstructured terrestrial situations. The development of a robot with the capability to operate in space can meet many of the requirements of terrestrial applications. Current NASA activities in telerobotics consist of studies and technology development at all centers. The Jet Propulsion Laboratory has a ground-based demonstrator, and Goddard Space Flight Center is the lead center for the flight telerobotic servicer for Space Station.

3. System Requirements

The functional requirements for the telerobot will be derived from the servicing of satellites; satellite repair, assembly, and construction; payload handling; and contingency repair of spacecraft. These functions may be further broken down into a variety of generic tasks. Examples of the tasks are removing and installing fasteners, connection of umbilicals and fluid lines, module replacement, and adjustment of thermal blankets. An operational consideration in the requirements is EVA equivalency. Space systems are being designed to interface with the proven capability of the astronaut in the extravehicular mobility unit. A telerobot with EVA equivalent capability can interface with the space system and also has an operational backup in the EVA astronaut.

The performance of these tasks will be greatly affected by the environment. The lack of gravity forces is the most significant effect on manipulative functions. Zero-g is beneficial in allowing large masses to be handled. Other zero-g

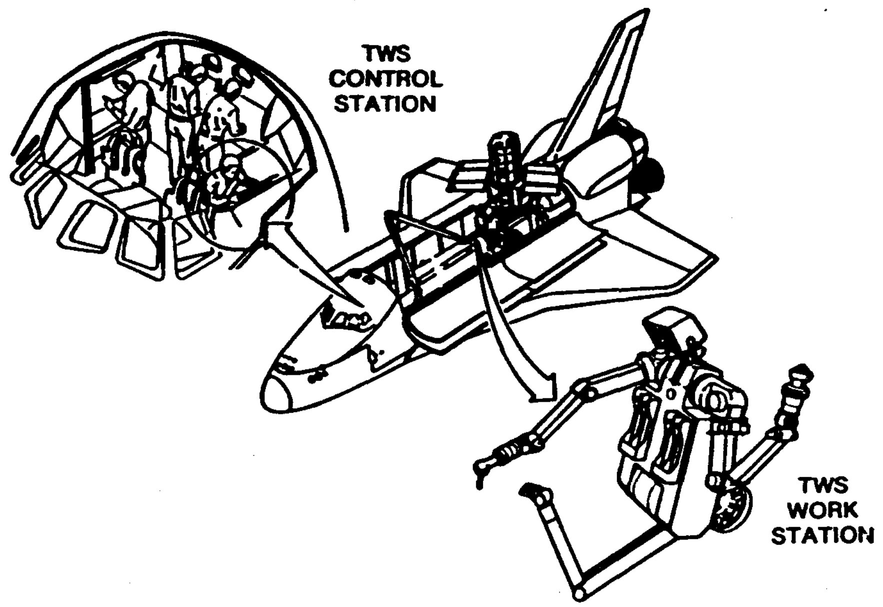


Figure 1. Telerobotic work system

effects are less beneficial. Parts being handled are not positioned and oriented by gravity forces, and the free play in the joints of mechanisms becomes an uncertainty. The human factors that are impacted by the space-flight environment relate to the interaction with displays and controls. Posture is different in zero-g. Restraints will be needed for force-reflecting controllers, and visual perceptions may be distorted.

4. System Architecture

The major elements of the TWS concept are the telerobot, the control station, and the system processor. This configuration corresponds to the architecture for an automated system as defined by Holcomb and others [2] and shown in Figure 2. The robot interfaces with the remote site at which the mission functions or the state changes are to be accomplished. The control station is the operator's interface with the system through controls and displays. The system processor implements the operator's commands and directs feedback of the results. The potential of the architecture is illustrated in Figure 3 as an example of the relationships of functional components of the system. Those relationships may be defined as operator interface, task planning and reasoning, control execution, and sensing and perception. Effectively, there is an operator control loop, an executive control loop, and a local control loop at the remote site. These control loops provide feedback and interaction to enable accomplishing tasks in an effective and productive manner.

5. Conceptual Designs of TWS

The Lyndon B. Johnson Space Center has studied the TWS through contracts with Grumman Aerospace Corporation and Martin Marietta Corporation. Figure 1 illustrates the system arrangement and major components for the initial application on the Space Shuttle Orbiter. The system elements logically divide into the robot work station where the physical tasks are to be accomplished, the control station with the operator's displays and controls, and the system processor that provides the computer power and logic to make the system function. Mobility to reach the work site is achieved with the Shuttle remote manipulator system (SRMS). Stabilizer arms hold the TWS in position at the work site. Later applications of the TWS may achieve greater mobility by using a free-flying module similar to the manned maneuvering unit.

The robot work station has manipulators and end effectors to perform physical tasks. Sensor suites monitor and measure conditions at the work site. Although work-site conditions in space are more structured than in many terrestrial situations, the configuration cannot be as well controlled as in most robotic uses in industry. The ability to determine the state of the task components is critical because of the inaccessibility in space; thus, the need for a preceptive and adaptive system. The concept of EVA equivalency is a strong driver in development of the configuration [3]. The capability of the EVA astronauts to perform dexterous tasks in the servicing and repair of satellites has been well demonstrated in recent Space Shuttle missions. Satellite designs are now being implemented in response to the demonstrated EVA capability. If the TWS can perform tasks equivalent to those of the suited astronaut, there will be satellites to work on.

The EVA equivalency requirement has resulted in strongly anthropomorphic configurations in the contractor concepts [4 and 5]. Grumman has carried the human analogy one step further by using the acronym "SAM" for the Surrogate Astronaut Machine shown in Figure 4. The principal camera location responds to the operator's eye-to-hand relationship. Other cameras on the arms provide additional views of the task. Proximity, force feedback, and tactile sensing supplement the visual aids. A third arm functions to stabilize the TWS at the work site. A dexterous arm similar to the other arms is

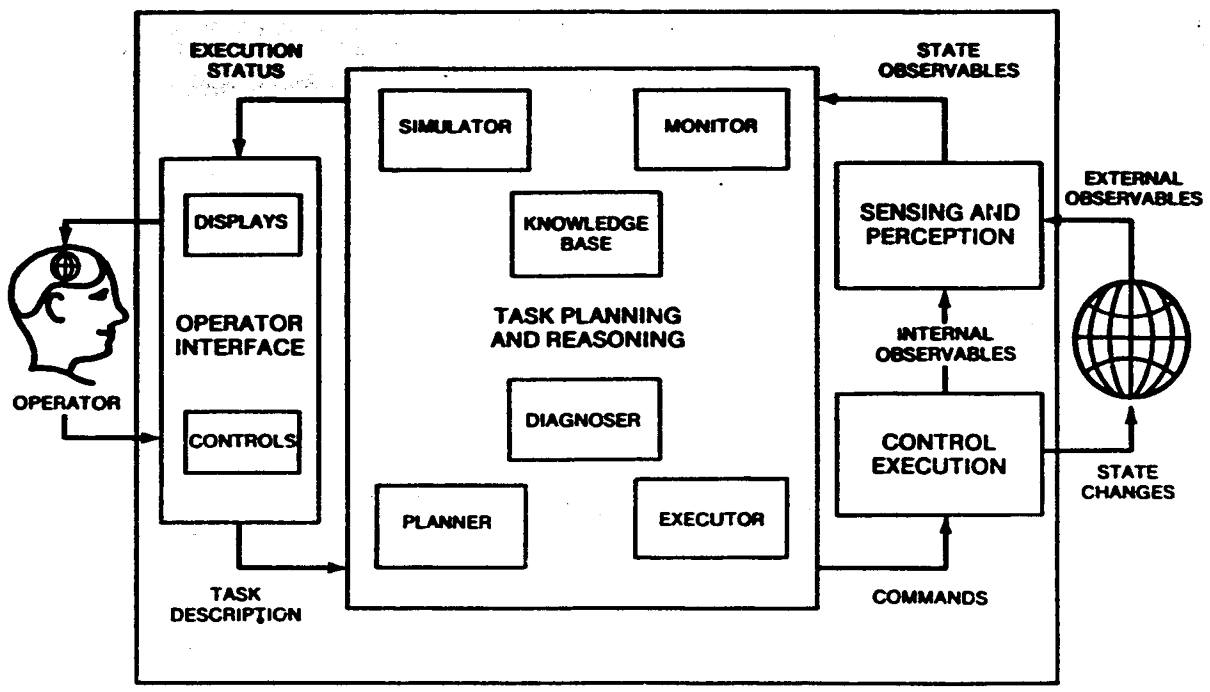


Figure 2. Architecture for an automated system

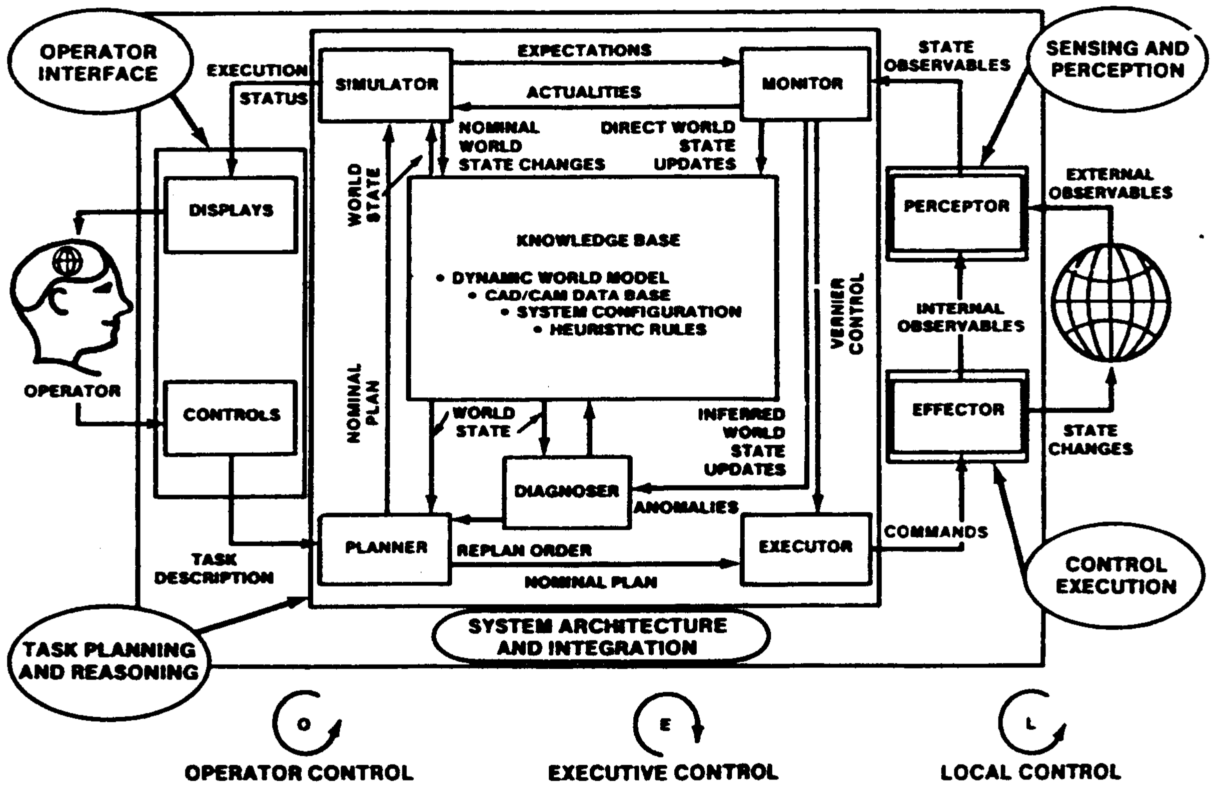


Figure 3. Automated system - control architecture

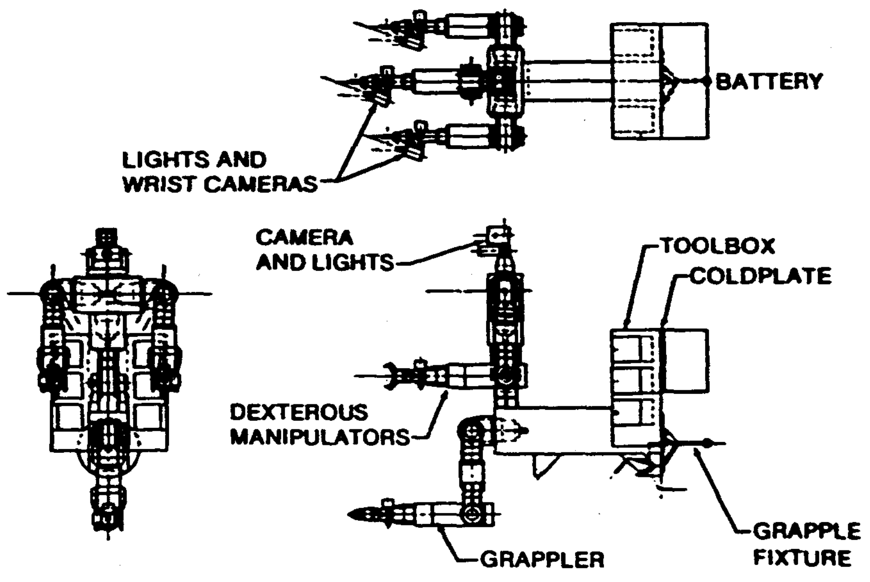


Figure 4. Grumman work station concept

proposed that would give some redundancy. The adequacy of a flexible arm to perform the stabilization function may require flight testing. The most useful approach to end effectors for accomplishing tasks is the attachment of tools to the dexterous arms. Tool stowage is behind "SAM's" torso to reduce the volume of the manipulative system.

Martin's concept particularly differs from Grumman's in the location of the tool stowage in the torso as shown in Figure 5. The dexterous arms are seven-degree-of-freedom (7 DOF) electric drive manipulator arms. The stabilizer arm is proposed to be a stiffer 5-DOF arm. The dexterous arms have a force-sensing wrist with an interchangeable tool device. This configuration allows use of special-purpose tools or a general-purpose gripper. Cameras and lights are mounted in a head with a 3-DOF neck.

The operator interface at the control station is critical for effective interaction with the robot. Interior volume is at a premium in space, particularly on the Space Shuttle. For example, the design of the SRMS was driven to a resolved-rate control system because the swept volume to operate a replica master controller for such a long arm was difficult to accommodate in the Orbiter cabin. Six-DOF rate controllers are proposed by Grumman (Figure 6) and Martin (Figure 7). Technique for controlling a 7-DOF arm is a technology development issue. Martin has suggested a hybrid control system that uses rate or position depending on the task. Replica controllers to position the smaller dexterous arms are a potential tradeoff for TWS application.

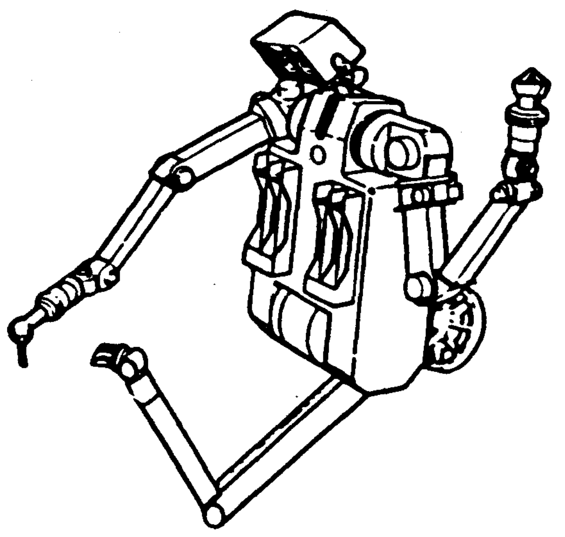


Figure 5. Martin work station concept

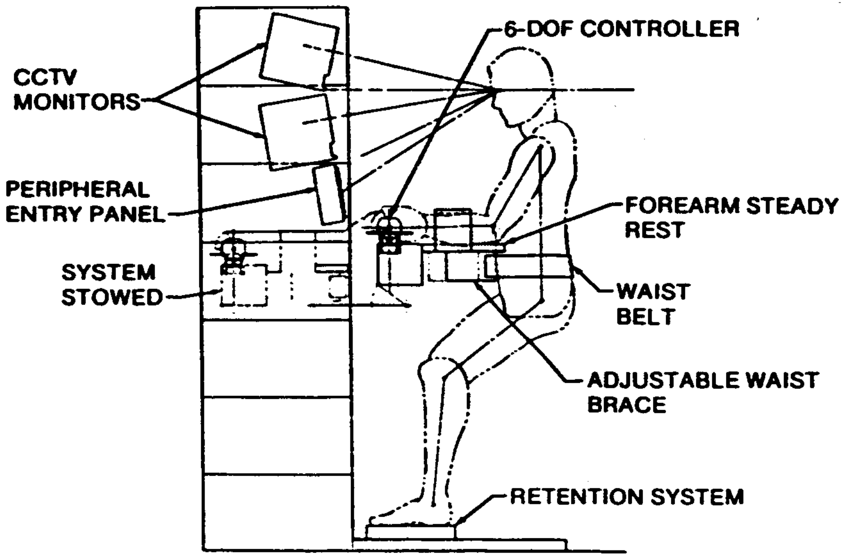


Figure 6. Grumman TWS control station concept

6. Program Development

The development plan for the TWS is predicated on the need to increase the productivity of the crew; therefore, the plan is evolutionary in nature. In this logic, the TWS design must be capable of incorporating technology advances as they become available. This approach will depend on modular subsystems and precise definition of interfaces to enable the adoption of newer innovations. Another feature of the logic is the evolutionary route of teleoperation to telepresence to supervisory control to supervised adaptive robotics [2]. The implications of this approach are evident in the selection of feedback sensors that will be compatible with expert systems and artificial intelligence needed for adaptive robotics. The telerobot technology program of the NASA Office of Aeronautics and Space Technology is consistent with this development approach. In addition to the ground demonstration telerobot at the NASA Jet Propulsion Laboratory, a protoflight testbed has been proposed to support research and technology experiments, to validate ground simulations, and to demonstrate the utility of a dexterous manipulation capability for remote operations in space. The flight telerobotic servicer program for Space Station is currently being defined. The anticipated result is a telerobotic system that will have application on the Space Station, but will have been developed and demonstrated on the Space Shuttle.

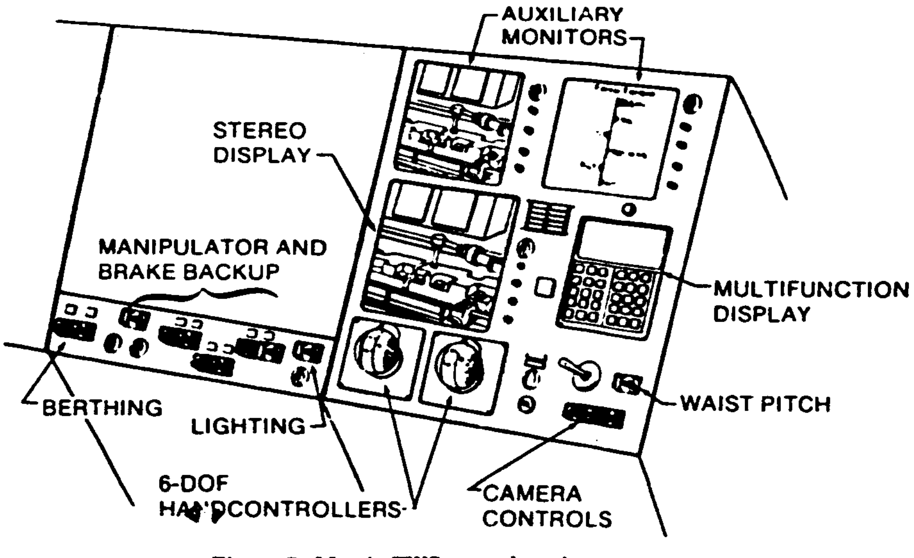


Figure 7. Martin TWS control station concept

7. Summary

The development of a telerobotic work system or a similar concept represents a valuable resource for performing a variety of tasks in the unstructured and hazardous environment of space. Development and demonstration in flight test on the Space Shuttle can lead to applications on the Space Station for the mobile remote manipulator system, the satellite servicer, and the orbital maneuvering vehicle. A system meeting these requirements can be of great use in developing the technology needed for many terrestrial applications of telerobots. Telerobots will find uses in personal service functions for disabled and aged people and in hazardous situations such as are found in construction and agriculture.

8. References

[1] L. Jenkins and R. Olsen, "Remote Operating Systems for Space-Based Servicing," ASME Conference Computers in Engineering, Las Vegas, Nevada, August 12-15, 1984.

[2] L. Holcomb, R. Larson, and M. Montemerlo, "Overview of the NASA Automation and Robotics Research Program," AIAA/NASA Symposium on Automation, Robotics and Advanced Computing for the National Space Program, Washington, D.C., September 4-6, 1985.

[3] L. Jenkins, "Telerobotic Work Systems Concepts," AIAA/NASA Symposium on Automation, Robotics and Advanced Computing for the National Space Program, Washington, D.C., September 4-6, 1985.

[4] "Telepresence Work System Definition Study," Grumman Aerospace Corporation, Contract NAS 9-17229, October 3, 1985.

[5] "Telepresence Work System Definition Study," Martin Marietta Aerospace, Contract NAS 9-17230, October 3, 1985.

183085 S3-37
108

Laboratory Testing of Candidate Robotic Applications for Space

R.B. Purves
Boeing Aerospace Company
Huntsville, AL 35807-3701

BR 112739

Abstract

Robots have potential for increasing the value of man's presence in space. Some categories with potential benefit are: 1) performing extravehicular tasks like satellite and station servicing, 2) supporting the science mission of the station by manipulating experiment tasks, and 3) performing intravehicular activities which would be boring, tedious, exacting, or otherwise unpleasant for astronauts.

An important issue in space robotics is selection of an appropriate level of autonomy. In broad terms ~~we can define~~ three levels of autonomy can be defined: 1) teleoperated - an operator explicitly controls robot movement, 2) telerobotic - an operator controls the robot directly, but by high-level commands, without, for example, detailed control of trajectories, and 3) autonomous - an operator supplies a single high-level command, the robot does all necessary task sequencing and planning to satisfy the command.

~~We~~ Researchers chose three projects for our exploration of technology and implementation issues in space robots, one each of the three application areas, each with a different level of autonomy. The projects were:

1. satellite servicing - teleoperated
2. laboratory assistant - telerobotic
3. on-orbit inventory manager - autonomous

This ~~paper describes~~ these projects are described and ~~summarizes~~ some results of ~~our~~ testing are summarized.

1.0 Introduction

The Space Station is intended primarily to be a facility for the advancement of science. It is not possible to predict precisely the future course of scientific discovery. But, based on our historical experience, we can expect that when new tools, methods, and the proper mind set are combined, then progress and discovery will follow. The Space Station provides such an opportunity. Scientists of the future will make these discoveries to the extent that they have time to immerse themselves in scientific problems. The key to making the Space Station valuable for science is ensuring that those scientists on orbit do not spend all their time in station keeping but on science. We view robotics as a major tool for accomplishing the science mission of the station by making crew time available for science and by automating part of the science task itself.

This paper presents an overview of three robotics system test projects conducted in the Boeing Space Station Robotics Laboratory in Huntsville. Assembly and test were conducted with subcontractor support from Essex Corporation, Georgia Tech Research Institute, Transitions Research Corporation, and Westinghouse Electric Corporation.

2.0 Component Technology and Integration Issues

Our overriding theme in this work was integration of systems having embedded robots for applications useful in space. The idea of focusing on systems may seem paradoxical, but the problems of integrating the robotic system were more important to us than the component technologies. Of course, we were obliged to address the component technologies to make the systems work. One interesting system-level descriptor of a robotic application is its level of autonomy. The level of autonomy of a robot can be thought of in two ways: one, the more autonomous a robot is, the more abstract are the instructions which operate it and, two, the longer it can act without human intervention. For our own convenience we defined three levels on the autonomy continuum:

a. Teleoperated - an operator directly controls each action of the robot. Typical control devices include joystick, teach pendant, and master-slave device.

b. Telerobotic - an operator controls the robot by giving medium level commands like, "move to position A" or "load sample #2."

c. Autonomous - the operator makes a single request for a logically complete service. The robot does whatever planning, obstacle avoidance, etc., which are necessary to complete the service.

We wanted to explore these three levels of autonomy and transition paths for systems with low autonomy to higher autonomy.

The component technologies and issues we wanted to address by incorporation into our systems were: teleoperation time delay, compliance, man-machine interface, sensors, robot control language, multi-arm control, video/lighting, vision, end effectors, and hand controllers.

3.0 Selected Robot Projects

Based on our systems interests and relevant component technologies, we selected three projects. Each project was chosen to be at a different level of autonomy. The component technologies were shared among the projects in such a way as to permit exploration of those technologies but without burdening any one project with too many integration problems. Figure 1 shows the allocation of component technologies and level of automation to our selected projects.

PROJECT / ISSUE	SERVICING TELEOPERATOR	LABORATORY ASSISTANT ROBOT	LOGISTIC ROBOT
TIME DELAY	X	X	
COMPLIANCE	X	X	X
HUMAN INTERFACE	X	X	
SENSORS		X	X
CONTROL LANGUAGE		X	
MULTI-ARM CONTROL			X
VIDEO/LIGHTING	X		
VISION			X
END EFFECTORS	X	X	X
HAND CONTROLLERS	X		
LEVEL OF AUTONOMY	TELEOPERATED	TELEROBOTIC	AUTONOMOUS

FIGURE 1 ALLOCATION OF TECHNOLOGY ISSUES TO PROJECTS

The robotic systems were assembled in our Huntsville Space Station Laboratory. Figure 2 shows the laboratory floor plan. It consists of two major floor areas: a staging area for robot systems and an acoustically isolated control room.

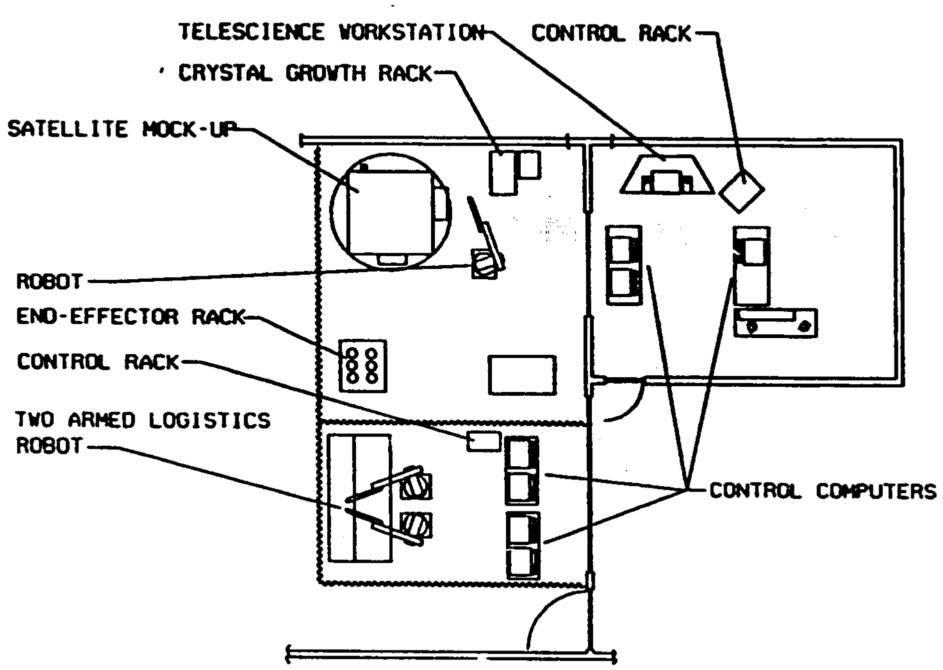


FIGURE 2 BOEING-HUNTSVILLE SPACE STATION ROBOTICS LABORATORY

The staging area is equipped with a variety of special lighting types and black stage curtains. The workstation is located in the control room. A window between the two areas permits viewing during test setup but is equipped with a blind for visual isolation during testing.

3.1 Teleoperator for On-Orbit Servicing

This system provides one-g simulation of a teleoperated robot for characteristic on-orbit servicing tasks. Our objectives in this task were to drive out servicer and human operation design requirements. Figure 3 shows the general test configuration. The servicing tasks were represented on a half-scale satellite mock-up with simulated Orbital Replacement Units (ORU's): Flight Guidance Equipment, Rate Sensing Unit, Battery, and Multimission Modular Spacecraft Module (MMS). These replaceable units are attached to the mock-up by different space-type fasteners and in different orientations. Obstacles can be located to increase difficulty of changeout. The robot arm is a Unimation Puma 560. A black and white CCD camera is attached to the robot wrist (see figure 4). Two other color cameras with remotely operable pan-tilt-zoom-focus are placed for worksite viewing. Several end effectors were developed for this test: MMS tool, gripper, hexdriver, scissors, and insulation holder. The end effectors are electrically driven and mate to a common interface. For this test the workstation is configured with three video monitors, robot control screen, and keyboard voice-driven video controllers and robot controllers.

Six Boeing employees were chosen as subjects in the human factors testing. They had no prior experience with manipulator controls. All had normal or corrected visual acuity as determined by a Class II Flight Visual Examination. Two Skylab astronauts also participated.

The first testing stage was used to develop learning curve data and to compare user preference for teach pendant and joystick. Subjects completed approximately 400 runs with 80 hours of testing. Figure 5 gives an example of learning curve data. The subjects unanimously selected the joysticks over teach pendant as the controller they would choose to do a difficult task. The data showed that subjects learned more and produced faster times with joysticks.

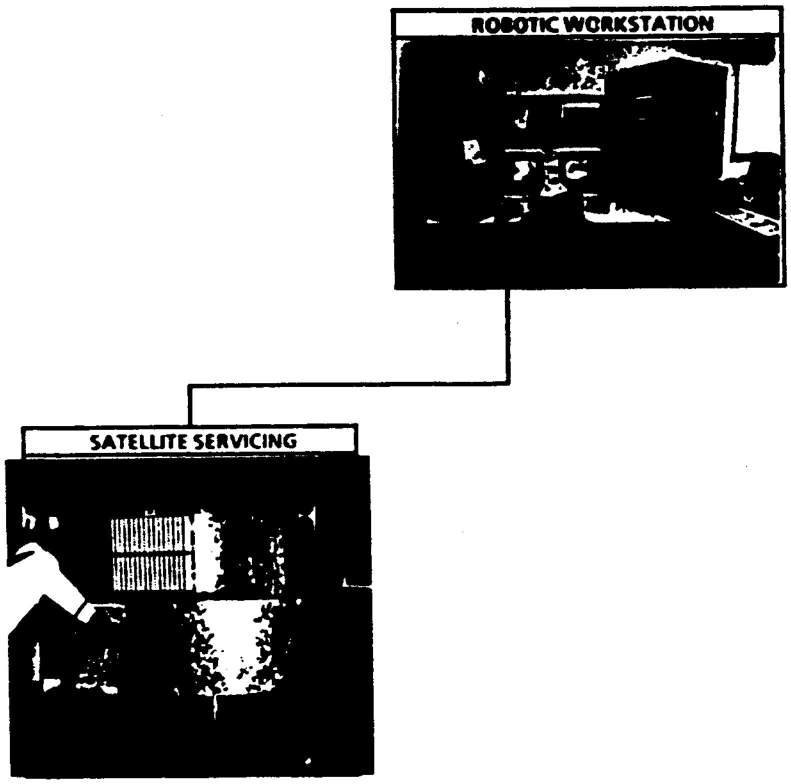


FIGURE 3 CONFIGURATION FOR TELEOPERATOR TESTING

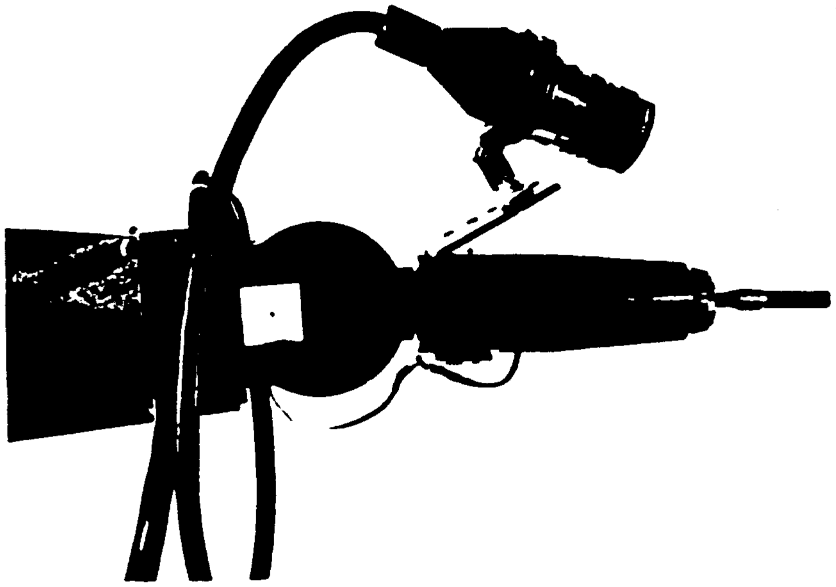


FIGURE 4 HEX DRIVE END EFFECTOR WITH CAMERA

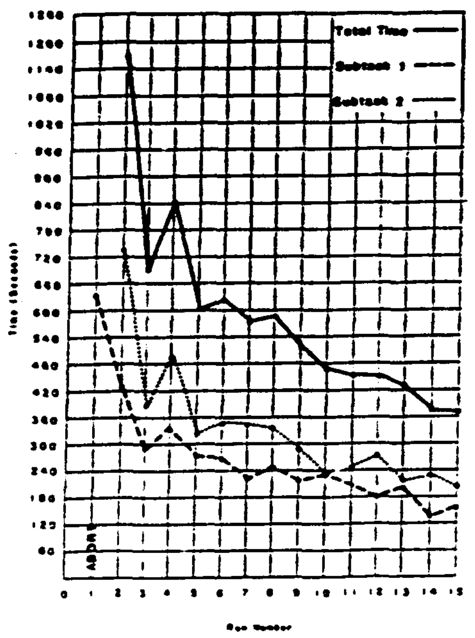


FIGURE 5 EXAMPLE INDIVIDUAL LEARNING CURVE

The second testing stage was used to assess the impact of communication time delay on task performance. By the time we started this testing, the subjects each had a minimum of thirteen hours of manipulator control experience. Only the joysticks were used. Figure 6 lists the tasks and subtasks we used. As an example, Figure 7 shows the J-hook fastener used in one of the tasks. Figure 8 shows how task completion time and arm movement time depended upon time delay.

TASKS

Task	Subtask
J-Hook	o Loosen the hex bolt
	o Unlatch the J-hook
	o Latch the J-hook
	o Tighten the hex bolt
	o Return the arm to the start position
Drive Screw	o Place of the tool on the drive screw
	o Engage the connector
	o Disengage the connector
	o Return the arm to the start position
FGE ORU Removal	o Grasp the EVA Handhold
	o Remove the ORU
	o Return the arm to the start position

FIGURE 6 TASKS USED IN TIME-DELAYED TELEOPERATION

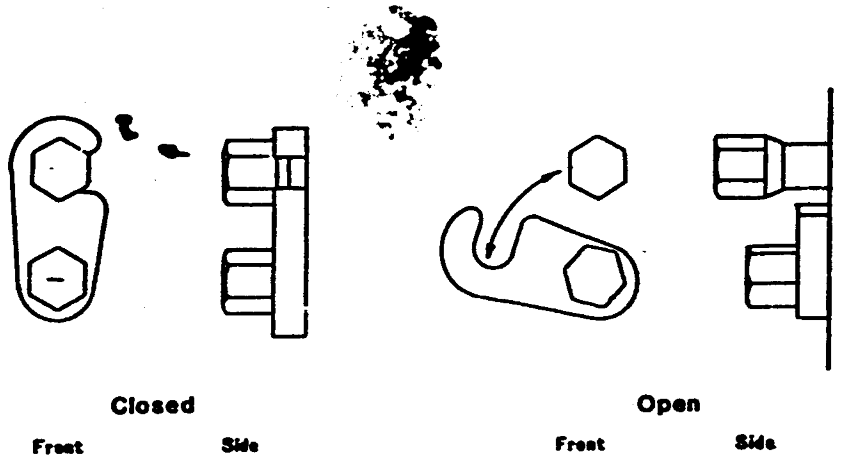


FIGURE 7 EXAMPLE TASK - J-HOOK FASTENER

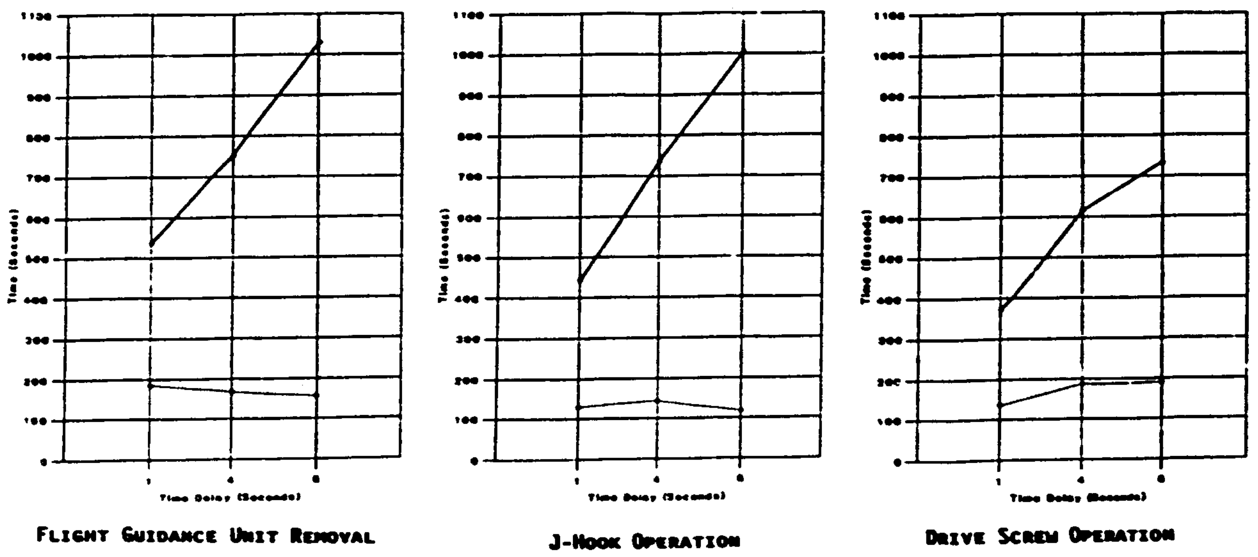


FIGURE 8 EFFECT OF TIME DELAY ON TASK TIME AND ARM MOVEMENT TIME

———————— TASK TIME

.... ARM MOVE TIME

After our teleoperation testing was complete, we were given the opportunity to install the teleoperator on a Space Station Module mock-up. In this case the task was to evaluate control of the teleoperator from an Element Control Workstation in a debris shield inspection mode.

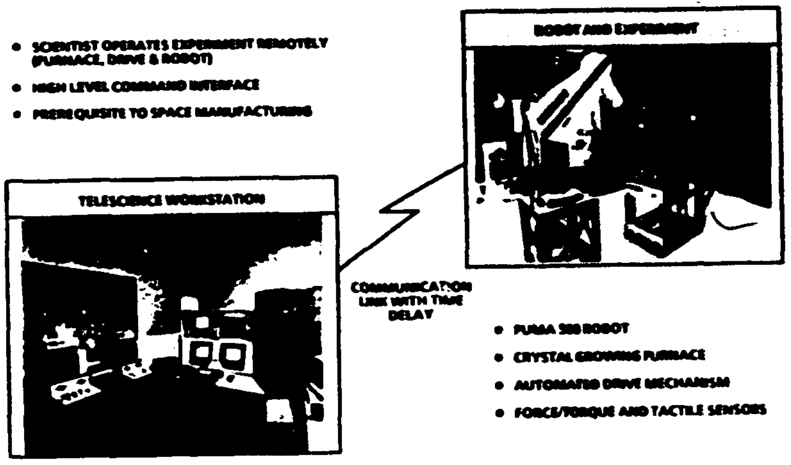

• SCIENTIST OPERATES EXPERIMENT REMOTELY (FURNACE, DRIVE & ROBOT)
• HIGH LEVEL COMMAND INTERFACE
• PREREQUISITE TO SPACE MANUFACTURING
ROBOT AND EXPERIMENT
TELESCIENCE WORKSTATION
COMMUNICATION LINK WITH TIME DELAY
• PUMA 560 ROBOT
• CRYSTAL GROWING FURNACE
• AUTOMATED DRIVE MECHANISM
• FORCE/TORQUE AND TACTILE SENSORS

FIGURE 9 TELEROBOTIC LABORATORY ASSISTANT

3.2 Telerobotic Laboratory Assistant

Our teleoperation work in the previous section showed that many manipulative tasks can be performed under direct control even with a significant time delay. Direct teleoperation should not, however, be considered an acceptable long-term solution. Two important factors score against it: The cumulative risk of damage is high and a highly skilled human operator is required.

This application addressed the use of a robot in telescience. Astronauts will most usually be skilled generalists. We cannot, however, expect them to be experts in all aspects of Space Station sciences. One way to get best use out of the Space Station is to permit a ground-based scientist to conduct his experiment on orbit with the feeling of "being there." This project integrates a telescience workstation with process control of a chemical vapor transport (CTV) furnace, mechanical control of embedded experiment automation, and a laboratory assistant robot

Figure 9 shows the general arrangement for this project. The robot control is at a higher evolutionary level than in our direct teleoperator work. A hierarchical control structure has been built up from very low-level commands. Error conditions are sensed and handled at the lowest practical level. Conditions which cannot be handled by the system are presented to the operator for resolution.

A configuration schematic is shown in Figure 10. The Unimate standard programming language VAL II was used. However, some special-purpose routines were coded in assembly language to facilitate sensor and command port communications. Ultimately, movement of an object was achieved with a VAL MOVE command. We implemented several ways to move an object: normal, shielded, constrained, or compliant.

Shielded motion helps to prevent damage to a robot and its payload. This "force shield" stops motion if forces and torques measured by the wrist sensor exceed expected threshold levels. Threshold levels might be exceeded if, for example, the tool or payload comes in contact with an unexpected surface.

Constrained motion involves shielded motion to the vicinity of a target point and additional movements until expected force and torque values are reached.

Compliant motion uses force/torque feedback to complete an action such as insertion of a tool into a hole. If a threshold is exceeded using complaint motion, but it is not the success constraint value, then attempts are made to comply by altering the robot path based on direction of the experienced forces.

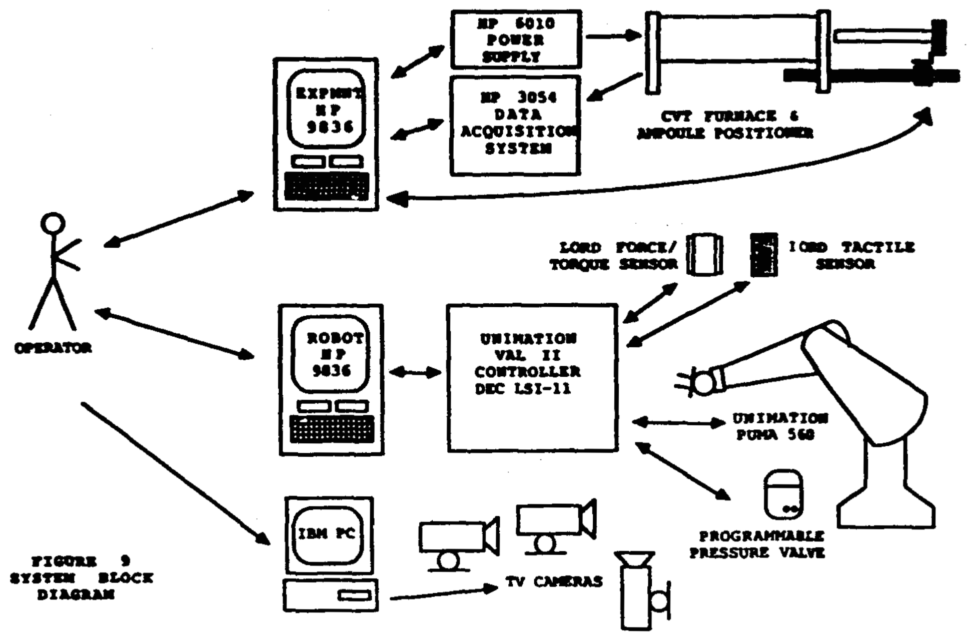

HP 6010 POWER SUPPLY
EXPMNT HP 9836
HP 3054 DATA ACQUISITION SYSTEM
CVT FURNACE & AMPOULE POSITIONER
LORD FORCE/ TORQUE SENSOR
LORD TACTILE SENSOR
OPERATOR
ROBOT HP 9836
UNIMATION VAL II CONTROLLER DEC LSI-11
UNIMATION PUMA 560
IBM PC
TV CAMERAS
PROGRAMMABLE PRESSURE VALVE
FIGURE 9 SYSTEM BLOCK DIAGRAM

FIGURE 10 CONFIGURATION DESCRIPTION FOR LABORATORY ASSISTANT

Additional control primitives allow for object alignment and slip detection using a tactile sensor pad in the gripper. These control primitives were combined in a hierarchical fashion to permit commands like: "grip selected ampoule," "insert ampoule," etc.

Remote operation of a CVT crystal growth experiment was simulated by placing the CVT equipment rack within reach of the robot and by placing control computers and monitors in the adjacent control room. The telescience work-station contained three video monitors: two were normally used to view the work area and one to monitor crystal growth.

The scenario began by the robot attempting to find a reference location on the rack. This was a small post on the rack front surface. The robot grasped the post and used the force/torque sensor to fine-tune the location of the post relative to the robot. The CVT system was initialized by moving the ampoule positioning mechanism to a home location so that an ampoule could be loaded without coming into contact with the furnace. The robot was then commanded to grip a selected ampoule, insert into the positioner, and then to release the ampoule. The positioner was then commanded to insert the ampoule into the furnace. The predefined furnace temperature and ampoule positions were then executed. The telescientist viewed crystal growth with the microscope-video system, changing temperature and position parameters as necessary. When the crystal growth was complete, a sequence of commands was made to remove the ampoule, place in a cooling rack, and stow for shipping.

3.3 Autonomous Logistic Robot

The two projects described thus far each required more or less continuous human supervision. Certainly supervision in the telescience case was at a higher level, but there was no intent to ignore the system until the job completion. In this project our goal was to integrate a system which would complete an entire task without human guidance or intervention. In order to make this possible, we chose a rather structured environment: a mock-up of a Space Station logistic module. We did not fully structure the problem space in that we did not require objects to be precisely located. Figure 11 shows the general system arrangement. Figure 12 shows the equipment configuration.

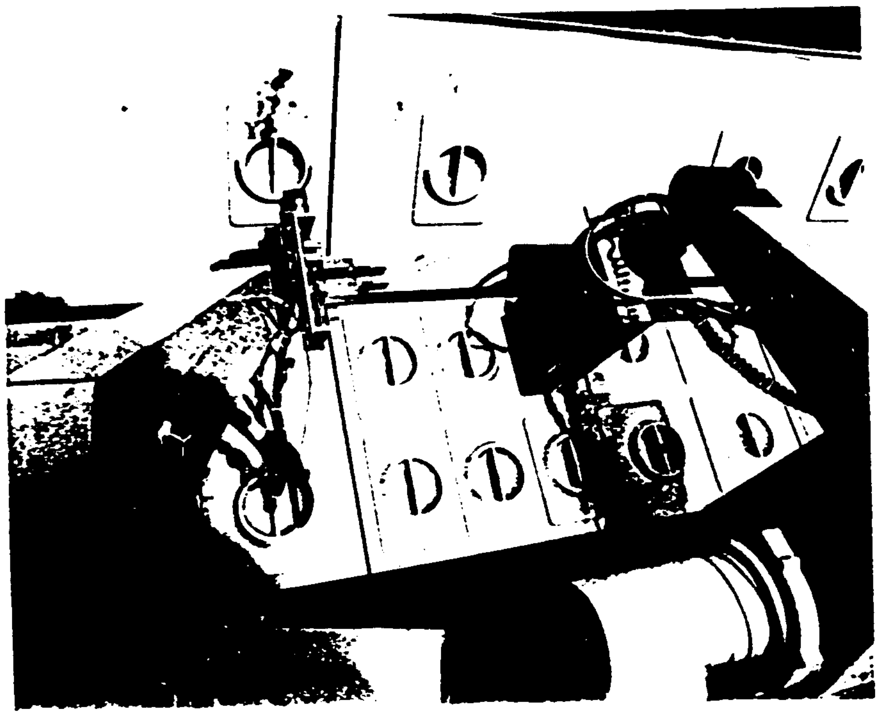


FIGURE 11   TWO-ARMED LOGISTIC ROBOT

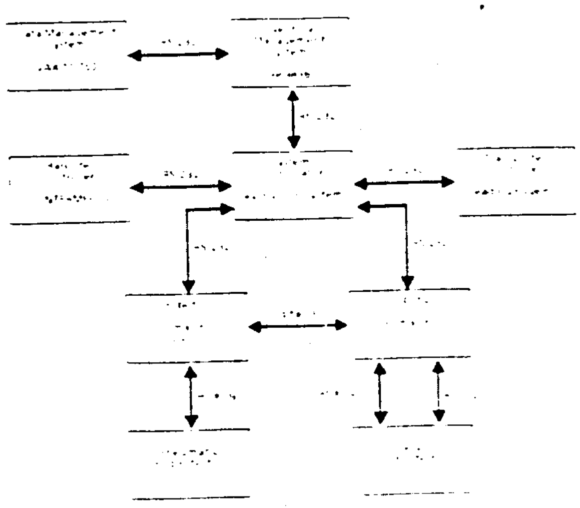


FIGURE 12   LOGISTIC ROBOT EQUIPMENT CONFIGURATION

The logistic module segment mock-up of a radial configuration was approximately half-scale. The drawers and racks were mounted so that articles would fall if not constrained. For test purposes, a set of simple tools was used to represent typical on-orbit inventory items. The purpose of the rail-based transporter was to grossly position the robot in the area of its target and to allow the robots by means of force and vision to accomplish its task.

The robot consisted of two Puma 560 arms mounted on a transporter carriage. The arms shared a common work space. Arm-to-arm messages were used to synchronize arm movements.

Each arm had a different gripper. One had an integral barcode reader. It was primarily for identifying drawers and racks and for grasping handles. The other hand had a servo gripper which was equipped with automatically interchangeable fingers: one set for gripping handles, the other for picking up small objects. The servo feature allowed measurement of position velocity and force parameters.

The Automatix AV-5 vision system was used to identify items in a drawer and to locate open areas where they could be placed. The AV-5 also served as system coordinator.

Test scenarios have allowed us to automatically identify, store, and retrieve inventory items, racks, and drawers. Two-arm activities are coordinated at least to a collision prevention level. Removal of a full rack was a two-armed activity and required fully cooperative two-armed motion.

4.0 General Observations

In order to build systems with embedded robots, we found it necessary to use several different computers, operating systems, and programming languages - not that it was a surprise, but it was a nuisance. If the level to which standards are used in a discipline is one measure of its maturity, then robotics is immature. We need to apply some standards which will permit interconnection and cooperative use of diverse equipment (e.g., vision, arms, grippers, etc.). A layering standard along the lines of the ISO-OSI definitions is essential for design of reusable components and evolving robot autonomy. We don't want to stifle research, but those who must build real systems for space will be greatly aided by standards.

Our teleoperator servicing tests were based on existing satellite design. It didn't take us long to decide that existing satellites would be extremely difficult to service by robot and that two arms would be required. A review of the manipulative difficulties also showed that those tasks could have been designed for relatively simple one-arm operation. The systems we expect to manipulate in space should be designed for robotic attention. Our industrial experience tells us that a task designed for robots can be performed better by humans.

Our time-delay work showed that typical tasks can be done by teleoperation. They just take longer with time delay. The situation is really a bit more complicated. Our test subjects were not repairing a multimillion-dollar satellite. There was a test observer with a panic button carefully watching for the robot to get into trouble. In fact, a significant number of runs were aborted to protect equipment or to release a jammed tool.

188086 S4-18 9P

Telerobotic Assembly of Space Station Truss Structure

G. Fischer
Grumman Space Systems Division
Bethpage, NY 11714

G 6919090

1. Abstract

This paper discusses methods of assembling the Space Station's structure which only utilize telerobotic devices

- an approximately anthropomorphic telerobot with two dextrous arms
- the Shuttle Remote Manipulator System (SRMS)
- various material handling machines.

Timelines and task recommendations for autonomous operations are also included. The paper also describes some experimental results comparing two manipulator control devices.

2. Introduction

Recent studies at Grumman have shown the feasibility of assembling the Space Station truss structure in space using only telerobotic systems. The studies investigated the use of a pair of cooperating dextrous manipulators in an anthropomorphic telerobotic system. Two types of investigations were conducted:

- Computer Aided Design (CAD) studies of the assembly of the Space Station truss structure
- Experimental tests of several subjects and control devices performing some truss assembly tasks using a pair of dextrous manipulators.

These studies were based upon the capabilities of current state-of-the art electro-mechanical devices. Although human directed telepresence control was the baseline of these investigations, the truss assembly activities described here lend themselves very nicely to autonomous (or supervised) robotic operations. This paper will present some of the results from both the CAD studies and the experimental investigations.

The CAD studies addressed the problem of assembling the entire Space Station truss structure from the Cargo Bay of the STS Orbiter, without using Extra Vehicular Activity (EVA). The studies only addressed structural connections and did not consider the installation of utility lines (fluid and electrical). Two different methods of assembly were explored. Both utilized an anthropomorphic machine known as SAM (Surrogate Astronaut Machine) to perform all the dextrous manipulation tasks needed in vacuum to assemble the structure. Figure 1 displays some of SAM's characteristics. The "third arm", which normally functions as a means of attaching SAM to a worksite to allow the SAM mobility aid (SRMS or MSC) to depart and perform other functions, was not utilized in this study. This paper reports the features and significant differences (including task timelines) of the two assembly methods.

A number of experimental results were obtained from a series of test subjects operating a pair of six degree-of-freedom (6DOF) manipulator arms in a telepresence mode. Through the use of voice controlled cameras, the operators relied on video and force feedback to retrieve a "strut" and connect it to a truss node (see Figure 2). Two different control devices were used by each operator:

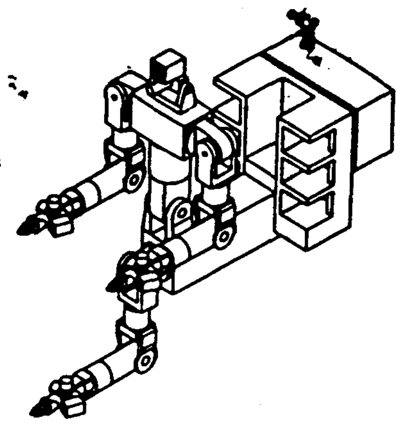


Figure 1. Astronaut Capabilities With SAM

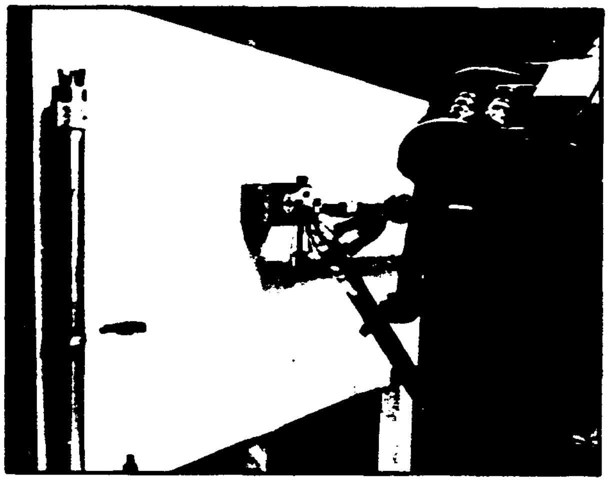


Figure 2. SS Structure Assembly Tests

- A pair of 6 DOF ball type hand controllers utilized a resolved rate control law (see Figure 3). Some force feedback was provided by auditory signals. No force feedback was provided through the ball controller.
- A pair of bilateral force reflecting (BFR) replica master controllers utilized position-position error signals to produce forces in both master and slave arms (see Figure 3).

This paper reports the significant differences found in operating with both of these control systems.

3. Structure Assembly Tests

Three struts were assembled into a node which contained strut termination fittings (see Figure 2). The strut connections, known as the "Wendel-Wendel" joints, require that one manipulator hand hold a strut in position while the other manipulator hand translates a collar over the joint and then locks the joint by rotating the collar about half a revolution. The struts were positioned in the nodes in three orientations: vertical, diagonal and horizontal. In a gravity field, task difficulty was strongly influenced by strut orientation. The vertical strut installation was very easy to do. The horizontal strut installation was quite difficult. Task times were recorded from the start of strut removal from the vertical storage (the left zone of Figure 2) until the visual indicator on the strut locking collar indicated a locked condition.

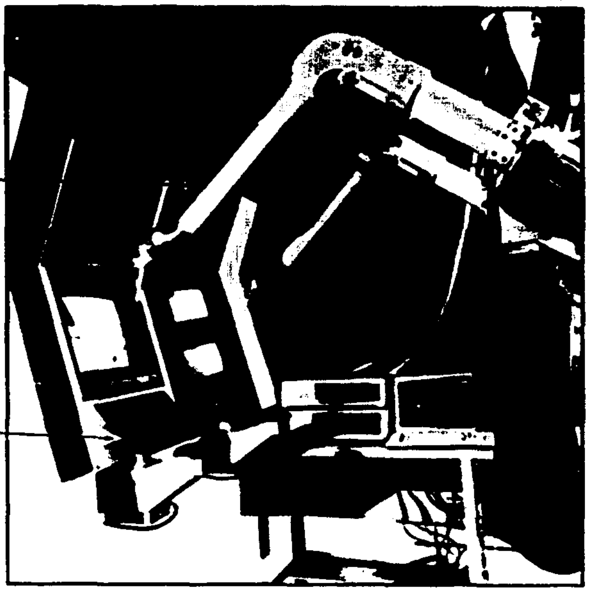


Figure 3. Control Station

Figure 3 shows the two types of control devices which were utilized for these tests. The Master Controllers are 6 DOF BFR arms which have an additional DOF squeeze grip for operating parallel jaw motion end effectors on the slave arm. The master and slave arms have identical structures and kinematics (i.e., a geometry ratio of 1:1). The control laws used by this BFR replica system develop torques, at both the master and slave joints, which are proportional to the position error signal between the corresponding master and slave joints. That is, when the master elbow is displaced 30° with respect to the slave elbow, the operator feels a force at the control handle (which was generated at the master elbow) which tends to drive the master arm to the same position as the slave arm. Simultaneously, the slave arm experiences a torque at its elbow which tends to drive the slave arm to the same position as the master. Thus, high forces at the slave arm are experienced by the operator as high forces on the master arm. This type of control system is known as bilateral force reflection (BFR).

The second type of manipulator controller shown in Figure 3 is a 6 DOF ball gripper type which is under development by CAE Electronics, Ltd. This compact device, and its supporting electronics, translate operator displacement commands (+ - x, y, z, θ, ϕ, ψ) at the ball grip into slave end effector rate commands (+ - $\dot{x}$, $\dot{y}$, $\dot{z}$, $\dot{\theta}$, $\dot{\phi}$, $\dot{\psi}$) in one of several (selectable) coordinate axis systems. For these tests, the selected system fixed the slave hand θ, ϕ $\alpha\nu\delta$ ψ axes to the slave hand, and, when the hand grabbled a strut or collar, to the work object. Thus, ball grip angular input commands were in a work object coordinate system. Translation commands (x, y, z) were always in an inertially fixed reference frame. Thus, plus z was always straight up. Operator forces at the ball grip were very light (with no felt feedback from the slave arm). An auditory system supplied some indication of high forces on the slave arm. This auditory system was not very helpful for manipulator operators of this test series.

Figure 4 shows the laboratory in which these tests were conducted. The worksite region is on the right side. The operator region is on the left side. An opaque curtain was placed between these regions for these tests. The operator received all visual information from three TV monitors (see Figure 3). These monitors received images from three fixed location cameras in the worksite region. The cameras had 3 DOF (scan, tilt, and zoom) and were controlled by manipulator operator voice commands.

One of the major objectives of this test program was to evaluate the effects of the two manipulator control systems (master vs ball controller) on task timelines and to identify benefits and problems associated with them. The results we found were:

- 230% faster strut installation with the master controllers

  -least time difference for the vertical strut, which required the lowest cognitive workload of the ball controller tasks.

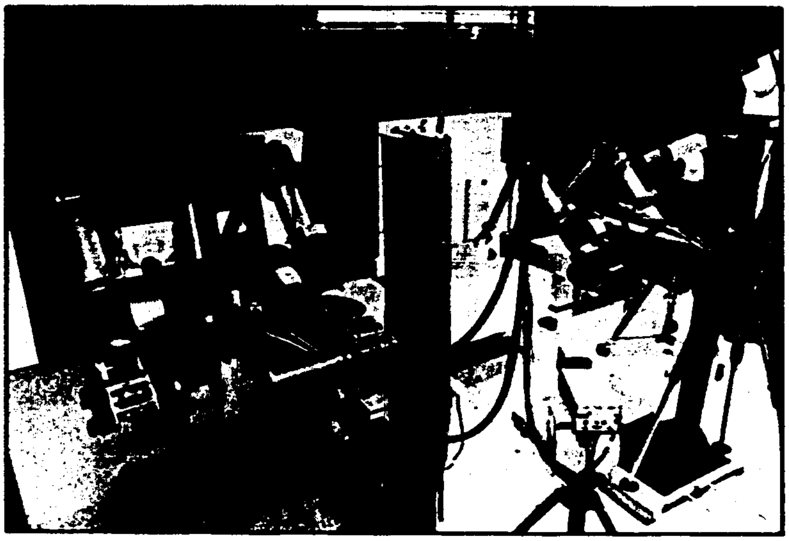


Figure 4. Test Lab Overview

• benefits & problems	BFR-Master	rate-Ball
-speed of movement	good	too slow
-mobility (zone of wk)	good	too restricted
-single axis motion:		
• in coordinates of control system	difficult	excellent
• inclined to control coordinates	difficult	very difficult
-coordinated & constrained 2 arm motions	good	dangerous & very difficult
-operator ability with little practice	good	fair
-ability to join objects without understanding	excellent	almost impossible
-operator fatigue	very tiring	very comfortable
-control over fine (small) motions	poor	excellent

These, and other, experimental results are reported in greater detail in Reference # 1.

4. Space Station Assembly Study

Two different methods of using telerobotic devices (SAM) for assembling the Fall 1986 Langley Task Force Space Station Design were explored. Both methods used SAM in an operator controlled telepresence mode. The major differences between the two methods were in how SAM moved around the worksite and the amount of automation used to enable SAM to obtain supplies for the construction activity.

- Method 1

  -2 Telepresence controlled SAMs
  -Rotating/Translating Fixture Tool for SAM
  -Automatic strut & node dispensers

- Method 2

  -1 telepresence controlled SAM on SRMS
  -Rotating SS assy fixture

Figure 5 depicts a partially completed Space Station truss structure emerging from an assembly fixture at the back of the Orbiter's Cargo Bay. The truss is formed of cubical "bays" which are 5 meters long. Each face of a cube has a diagonal strut in addition to "horizontal" and "vertical" struts. Figure 6 shows more details of the support of the Truss Assembly Fixture within the Orbiter's Cargo Bay. The truss is assembled in the lower bay region by a SAM attached to a "horizontal" beam. The beam is attached to a turntable which provides 2 DOF: translation in the "vertical" (Orbiter z axis) direction and 360° rotation about this "vertical" axis. These 2 DOF, and the 10 DOF within SAM, allow SAM to reach all corners of the lower bay and the storage regions for struts and nodes (which are located outside the truss at the midpoints of the lower horizontal face of the cube). When the truss element feed system canisters become empty, they are replaced with full canisters by a second SAM which is mounted on a Shuttle Remote Manipulator System (SRMS). The canisters contain mechanisms which deliver all truss elements (i.e., struts and nodes) to the same location within a canister to expedite assembly operations.

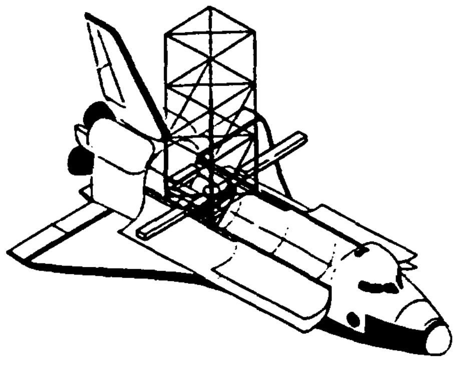
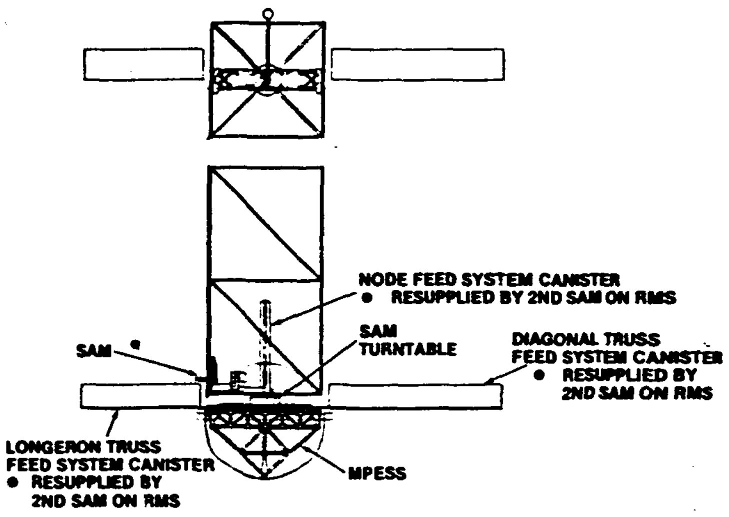


Figure 5. Truss Assembly Fixture, Method 1


NODE FEED SYSTEM CANISTER
● RESUPPLIED BY 2ND SAM ON RMS
SAM
SAM TURNTABLE
DIAGONAL TRUSS FEED SYSTEM CANISTER
● RESUPPLIED BY 2ND SAM ON RMS
LONGERON TRUSS FEED SYSTEM CANISTER
● RESUPPLIED BY 2ND SAM ON RMS
MPESS
Figure 6. Truss Assembly Fixture Details

All Space Station structural assembly sub-tasks have been considered for telerobotic assembly. Reference #2 contains a listing of these sub-tasks and our estimates for the time required for their completion. These sub-task times were grouped into major task activities and summed for the entire operation of assembling the structure of the Space Station's Transverse Boom, which required 70 hours of on-orbit time. These data are displayed on Figure 7.

TASK	TIME (MIN.)	TASK QUANTITY	TOTAL TIME (MINUTES)
DEPLOY ASSEMBLY FIXTURE	100	•	100
BETA JOINT INSTALLATION	115	4	460
SOLAR ARRAY ATTACHMENT	100	4	400
ESM & RADIATORS	305	2	610
ALPHA JOINTS	115	2	300
RCS CLUSTERS	110	2	220
ACA	115	1	115
REMOVE TRUSS AND DEPLOY	45	•	45
RETRACT ASSEMBLY FIXTURE	60	•	60
CLEANUP	60	•	60
			Σ_1 = 2510 MIN
CONSTRUCTION OF BAYS			
FIRST BAY	98	1	98
ADDITIONAL BAYS	68	23	1564
			Σ_2 = 1662 MIN
		ASSEMBLY TIME FOR TRANSVERSE BOOM	4172 MINUTES 70. HRS

Figure 7. Combined SS Assembly Operations Timeline, Method 1

We examined a second method of assembling the Space Station's Transverse Boom structure. This method only used a single SAM which remained attached to the SRMS. The partially constructed SS structure was rotated about the centerline of the assembly fixture (Orbiter z axis) to locate all teleoperations in the same region of the Orbiter. This region allowed easy access to strut and node storage areas within the Cargo Bay and allowed SRMS motion from top to bottom of the assembly fixture. Figure 8 shows part of the fixture assembly operation: the upper portion of one of the four truss supports is about to be inserted by SAM into the previously installed lower two pieces. Figure 9 depicts a node receptacle (which contains 6 nodes) installation into the assembly fixture. Figure 10 shows an assembled truss bay above four previously assembled "horizontal" truss faces within the assembly fixture. SAM is positioned to raise the completed bay to the top of the fixture where it will be held by the fixture. The second bay is completed by SAM attaching vertical and diagonal struts between the completed bay and the next "horizontal" truss face, which has to be raised up to the bottom position. During vertical face assembly, the partially completed structure is rotated 90° to present SAM with a new corner of the truss. During this 90° rotation, angular accelerations have been limited to 1 "g". This limits bending moments on the cantilevered diagonal struts (since they are attached at only one end during rotation) to values which are substantially below their strength. The six timelines associated with assembly Method 2 are contained in Reference #2. The last of these is reproduced here as Figure 11. Method 2 required 72 hours of on-orbit time to assemble the Space Station's Transverse Boom structure (work began after 8 hours on orbit).

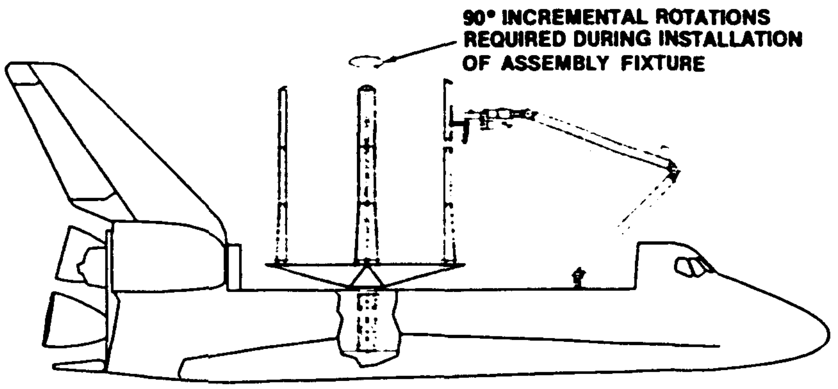

90° INCREMENTAL ROTATIONS REQUIRED DURING INSTALLATION OF ASSEMBLY FIXTURE

Figure 8. Installation of Assembly Fixture, Method 2

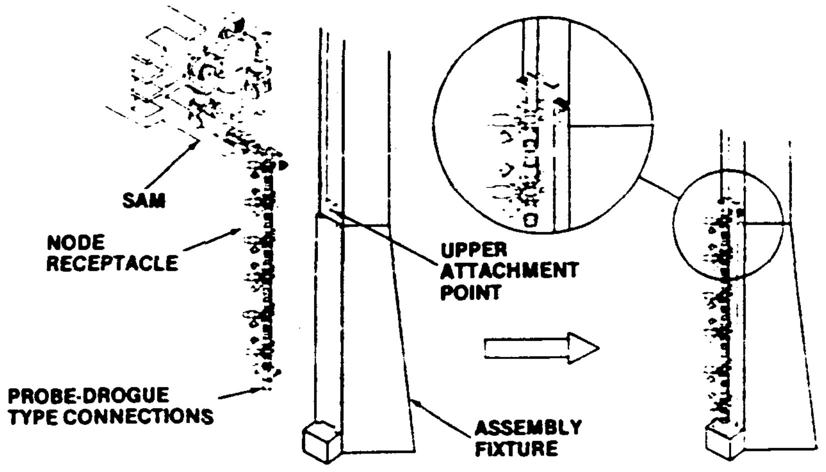


Figure 9. Details of Node Receptacle Installation

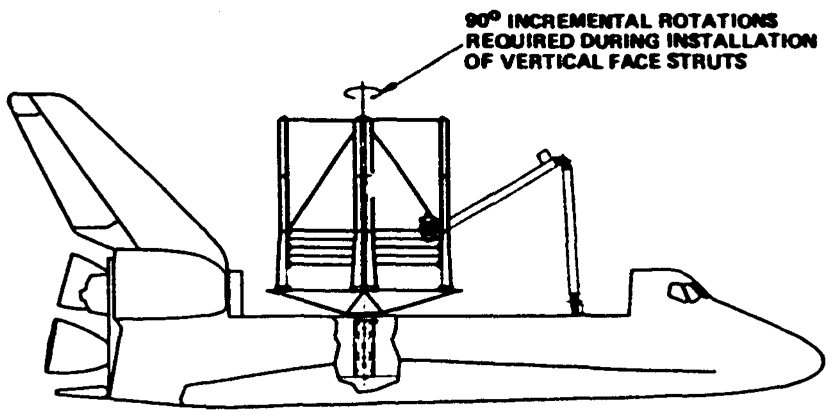

90° INCREMENTAL ROTATIONS REQUIRED DURING INSTALLATION OF VERTICAL FACE STRUTS

Figure 10. Installation of Vertical Face Struts

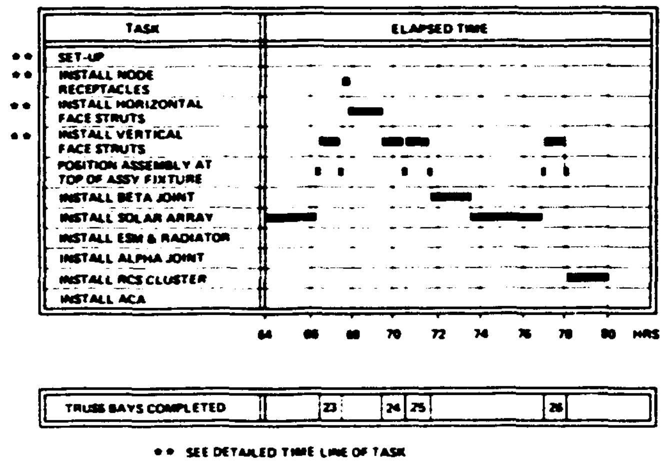


Figure 11. SS Truss Assembly (Day 6) Timeline, Method 2

5. Tasks for Autonomous Telerobotics

Both of the assembly methods which are discussed in this paper have assumed that all motions of the telerobot (SAM) and the SRMS have been commanded by astronauts at control stations within the Orbiter. Since telerobot operations are planned for 16 hours per day, astronauts controlling these devices may experience fatigue, even with frequent shift changes. Consequently, the reduction of astronaut workload is desirable to reduce fatigue and the concomitant probability of errors.

Selective autonomous telerobot actions can reduce astronaut fatigue by occasionally eliminating the need for an astronaut's physical effort and mental attention to details. At the completion of an autonomous robotic task, the astronaut acts like a supervisor and verifies that the autonomous task was performed properly.

Selective autonomy is stressed because of the prohibitive costs of providing a telerobot which is capable of performing all Space Station assembly tasks in an autonomous mode. Also, to perform autonomous tasks in a manner which does not impose a significant weight burden on the Space Station, a high level of machine intelligence is required. At this time, one can not predict that this machine intelligence technology will be available in time to perform the complete assembly of the Space Station.

Figures 12 and 13 list the major tasks which make up the Method 2 assembly activities. Note that some tasks are conducted only once while other tasks are conducted hundreds of times. The three highest repetition tasks use up 20 hours (which is 30%) of dextrous manipulation time. All the tasks selected as candidates for autonomy in Figure 12 represent 1/2 of the dextrous manipulation time for the assembly of Space Station structure. A similar analysis of SRMS tasks (Figure 13) yields 13 hours (90%) of SRMS operations which lend themselves to autonomous operations.

DEXTROUS MANIPULATION TASK	TASK TIME (MIN)	TASK REPETITION	TOTAL TASK TIME (HRS MIN)
INSTALL FIXTURE ELEMENT ON DEPLOYABLE BASE	2.0	12	0 24
INSTALL NODE RECEPTACLE ON ASSEMBLY FIXTURE	2.0	20	0 40
• INSTALL END OF STRUT	2.0	343	11:26
• INSTALL OPPOSITE END OF STRUT	1.0	343	5 43
OBTAIN ASSY FIXTURE ELEMENT FROM CONTAINER	0.5	12	0:06
OBTAIN NODE RECEPTACLE FROM CONTAINER	0.5	20	0:10
• OBTAIN STRUT FROM CONTAINER	0.5	343	2 51
• POSITION ASSEMBLY AT TOP OF ASSY FIXTURE	10.0	33	5.30
REMOVE EMPTY NODE RECEPTACLE FROM ASSY FIXTURE	2.0	4	0:08
STOW EMPTY NODE RECEPTACLE	2.0	4	0:08
INSTALL BETA JOINT	115.0	4	7 40
• INSTALL SOLAR ARRAY	100.0	8	13 20
INSTALL ESW & RADIATOR	305.0	2	10 10
INSTALL ALPHA JOINT	115.0	2	3:50
INSTALL RCS CLUSTER	110.0	2	3 40
INSTALL ACA	115.0	1	1 56

• TASK IS CANDIDATE FOR NEAR TERM AUTONOMOUS OPERATIONS

Σ_{DT} = 67.7 HOURS

Figure 12. Dextrous Manipulation Tasks

NON-DEXTROUS MANIPULATION TASK	TASK TIME (MIN)	TASK REPETITION	TOTAL TASK TIME (HRS MIN)
• <u>RMS TASKS</u>			
TRANSLATE TO ASSEMBLY FIXTURE CONTAINER	1.0	12	0 12
TRANSLATE TO NODE RECEPTACLE CONTAINER	1.0	20	0 20
• TRANSLATE TO DIAGONAL STRUT CONTAINER	1.0	131	2 11
• TRANSLATE TO 5 METER STRUT CONTAINER	1.0	212	3 32
TRANSL TO ASSY FIXTURE BASE WITH FIXT ELEMENT	1.0	12	0 12
TRANSLATE TO ASSY FIXTURE WITH NODE RECEPTACLE	1.0	20	0 20
• TRANSLATE TO ASSEMBLY FIXTURE WITH STRUT	1.0	343	5 43
• TRANSLATE TO OPPOSITE END OF STRUT	0.5	208	1 44
• <u>NON-TELEROBOTIC TASKS</u>			Σ_{RMS} = 14.2 HOURS
CHECK-OUT RMS	10.0	1	0 10
CHECK-OUT SAM	10.0	1	0 10
DEPLOY ASSEMBLY FIXTURE BASE	15.0	1	0 15
OBTAIN SAM FROM STOWAGE USING RMS	10.0	1	0 10
ROTATE ASSEMBLY FIXTURE 90	0.5	212	1 46
			Σ_{NTR} = 2.5 HOURS

• TASK IS CANDIDATE FOR NEAR TERM AUTONOMOUS TELEROBOTIC OPERATIONS

Figure 13. Non-Dextrous Manipulation Tasks

6. Conclusions

Our work has persuaded us that the Space Station structure can be reliably assembled by telerobotic systems. We believe

- that the majority of tasks should be under astronaut control using telepresence technology
- that selective tasks should be performed autonomously by the telerobot
- that the primary telepresence control device should be a bilateral force reflecting replica master
- that the initial Space Station structure can be assembled during 1 STS mission without EVA.

7. Acknowledgements

The work described in this paper was performed under contract to the National Aeronautics and Space Administration, Johnson Space Center. Testing was conducted at Grumman's Manipulator Development Laboratory.

I would like to thank Mr. Erik Eriksen and Dr. John O'Hara for their contributions to our test programs, and, Mr. Dan Guinan and Mr. Kirk Sneddon for their Space Station assembly studies.

8. References

#1: O'Hara, John M., "Telerobotic Work System: Space-Station Truss-Structure Assembly Using a Two-Arm Dextrous Manipulator", Grumman Space Systems Division, Bethpage, NY, November 1986.

#2: Fischer, Grahme, "Telerobotic Work System, System Definition Study, Phase 2, Final Presentation", Grumman Space Systems Division, Bethpage, NY, January 1987.

188091 S5-18 118

Robotic Mobile Servicing Platform for Space Station

S.H. Lowenthal and L. Van Erden
Lockheed Missiles and Space Company
Sunnyvale, CA 94088

L172965

1. Abstract

Semi-autonomous inspection and servicing of Space Station's major thermal, electrical, mechanical subsystems is a critical need for the safe and reliable operation of the Station. A conceptual design is presented of a self-intelligent, small and highly mobile robotic platform. Equipped with suitable inspection sensors (cameras, ammonia detectors, etc.), this system's primary mission is to perform routine, autonomous inspection of the Station's primary subsystems. Typical tasks include detection of leaks from thermal fluid or refueling lines, as well as detection of micro-meteroid damage to the primary structure.

Equipped with stereo cameras and a dexterous manipulator, simple teleoperator repairs and small ORU changeout can also be accomplished. More difficult robotic repairs would be left to the larger, more sophisticated Mobile Remote Manipulator System (MRMS). An ancillary function is to ferry crew members and equipment around the station.

~~The~~ Primary design objectives were to provide a flexible, but uncomplicated robotic platform. One which caused minimal impact to the design of the Station's primary structure but could accept more advanced telerobotic technology as it evolves.

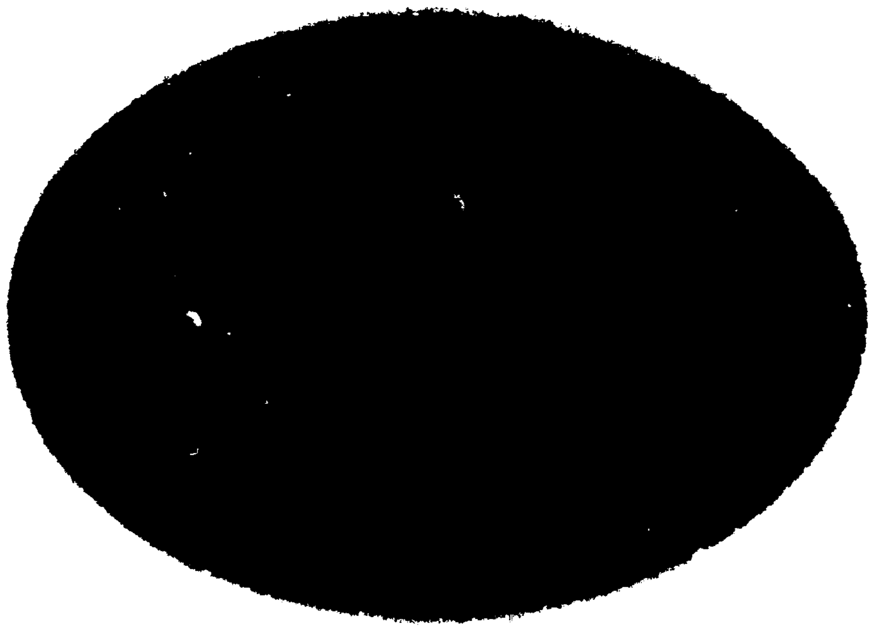


Figure 1. Manned Mobile Serv. .r

2. Introduction

The Space Station will undoubtedly be the largest and one of the most complex "spacecraft" ever launched by man. Bounded by a structural surface of more than 2 acres, the Station will contain many miles of electrical power, thermal fluid and data communication utility lines as well as house dozens of primary and secondary subsystems and components. Many of these elements have design lives of 20 years or more and must function reliably during this period in the hazardous environment of space. In recognition of the need to enhance the operational efficiency and reliability of Space Station, a congressionally appointed Advisory Committee [1], recommended that the initial Space Station should utilize a high degree of automation and robotics (A & R) technology. Among many of the Committee's recommendations was a suggested NASA A & R demonstration to construct "a mobile 'go-fer' robot to assist in crew tasks" [1]. The concept to be discussed in this investigation addresses this important recommendation.

Considerable work has been performed, for example see [2 to 4], in identifying teleoperator/robotic concepts and technology to assist in the on-orbit servicing and repair of spacecraft. It is clear that automated robotic work systems can considerably enhance the productivity of the flight crew. This is true, provided that the servicing tasks are well-defined, and secondly, that the required servicing mechanisms and the equipment to be serviced have been "scarred" to accommodate such automation. Furthermore, hazardous tasks, such as a propellant refueling operation, would obviously be more safely performed from a remote site.

Man's permanent presence onboard Space Station offers new and greater opportunities to repair and service in-orbit spacecraft. Robotic retrieval of satellites via free-flying robots (or robotic Orbiting Maneuverable Vehicles (OMV'S)) assisted by a teleoperated RMS are logical applications of A & R technology. However, considerable advancements in automation technology are still required, ranging from control architecture, task planning and artificial intelligence (AI) to robotic manipulator design and external sensor development [1]. Examples of how future space flight telerobots would differ from those in industry can be found in [5].

3. Space Station Inspection and Servicing

Apart from servicing orbiting payloads and spacecraft, the complexity, size and longevity of the Station warrants extensive application of automation to perform the necessary "housekeeping" and maintenance ~~functions. Representative examples of the station's subsystems which will~~ require periodic inspection and/or servicing are illustrated in Figure 2. Fault detection and isolation will be needed for the electrical power cables, communication and data lines, and those used for the thermal environmental control system. Detection of hazardous leaks from propulsion fuel lines or from the thermal fluid bus carrying anhydrous ammonia will also be a concern.

Micro-meteroid damage to the primary or secondary structure, solar panels, radiators, etc. must also be checked. Environmental damage such as that due to long term exposure to atomic oxygen or UV radiation to structural materials may occur as well. Contamination of optical surfaces, mirrors, and array panel surfaces can be expected. The diagnostic/maintenance list is extensive.

Reference [6] addresses many of those inspection and servicing needs of the Space Station from the standpoint of A & R. In this study, candidate A & R functions were identified, ranked and costed. Weighted assessments were made in terms of safety, productivity, IOC cost, risk, spinoff likelihood, reliability/maintainability and commonality. Table 1 is an example of one of the value ranking tables in [6], showing the priority of the first 29 of the 58 A & R candidate functions evaluated. It becomes apparent, upon reviewing this list that many of the inspection related tasks are not only important to perform but, moreover, have some of the most favorable cost-to-benefit ratios. For example, see utility run, truss/structure and thermal control system inspection items.

Another important conclusion from the study performed in [6], is that a substantial savings in crew time could be realized by automating the inspection process. According to Figure 3, automating inspection activities represent a savings of 90% of the crew's time relative to 10% for those due to repair. This finding is based on the realization that inspection related activities are both more frequent and time consuming. This is not to infer that the inspection process is necessarily more important than repair. A repair to a critical system, while not necessarily time consuming, could be critical for safe operation.

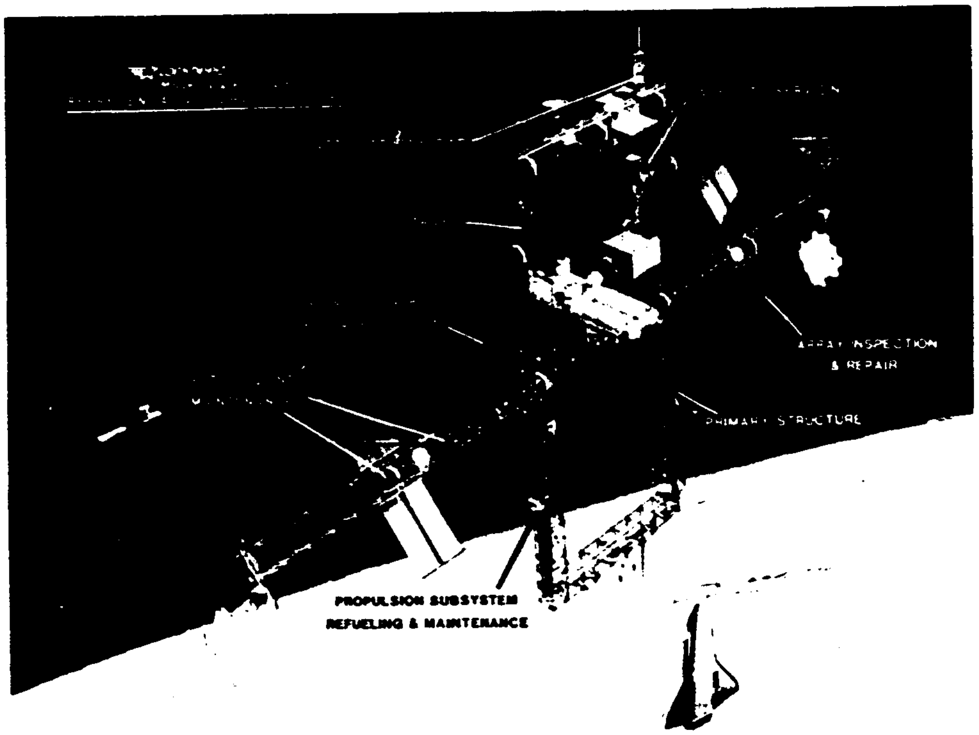

ARRAY INSPECTION & REPAIR
PRIMARY STRUCTURE
PROPULSION SUBSYSTEM REFUELING & MAINTENANCE

Figure 2. Representative Servicing Operations

Table 1. Benefit-to-Cost Ratio

• *FUNCTIONS CAPABLE OF BEING PERFORMED BY GOFIRS*

Rank	Evaluation	Benefit to Cost Ratio	Candidate
1	67.80	3.87	•Payload-Servicing-Robot
2	67.45	5.38	•Utility Run-Inspection-Replacement
3	67.24	5.55	•Inspection-Repair-of-Trusses-and-Structures
4	63.64	0.79	•Distributed-Smart-Camera System
5	60.92	2.69	•Thermal-Control-System-Inspection
6	60.40	2.88	•Mounting Plates-Assembly-Inspection-Repair
7	58.25	2.76	•Robotic EVA-Crew-Assistant
8	57.62	1.92	•Interconnect-Inspection-Repair
9	57.45	1.84	Hazardous Material-Handling-System
10	57.38	2.26	Tunnel Inspection-for-Exterior-Damage
11	56.75	1.48	•Gimbal Maintenance
12	56.59	1.02	•Robotic Inspection-Cleaner
13	56.47	2.05	Berthing-System-Inspection-and-Repair
14	56.23	1.40	Passive Thermal-Control-Monitoring
15	56.00	0.82	Bolt Torque-Preventative-Maintenance
16	54.86	0.68	EVA Task Mission Planning Aid
17	54.55	0.22	Thermal-Control Maintenance
18	54.27	0.03	•Hazardous-Utilities Connection
19	53.63	0.01	Attitude-Determination-from Camera Video
20	53.34	0.00	Inspection-of Pressure-Seals
21	53.33	0.70	•Non Destructive-Testing-of Struts and Mounts
22	52.22	1.20	Active Thermal Control System Assembly and Operations
23	51.49	0.96	Space Station Markings-Inspection Repair
24	49.92	1.24	Voice-Controlled Camera Adjustment
25	49.85	2.37	Thermal Curvature-Control
26	47.99	0.00	Space Station Service-Vehicle
27	47.55	1.61	Space Station Coordinator
28	47.48	0.06	Structure Assembly
29	47.24	0.77	Knowledge Based System for Fault Diagnosis of Communication Systems

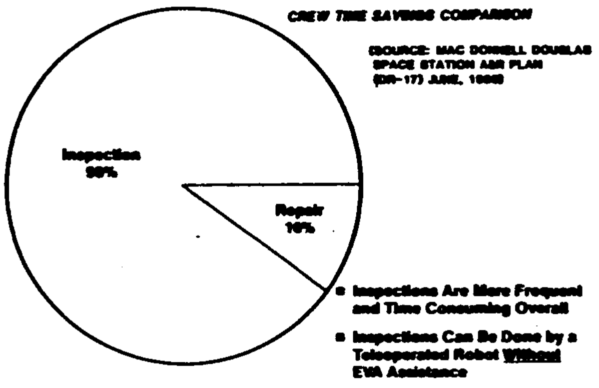


Figure 3. Inspection Versus Repair Robotic Tasks

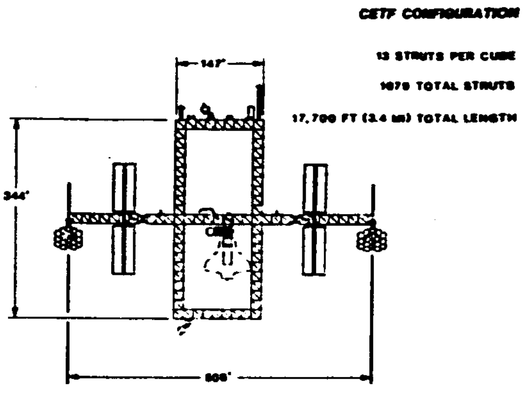


Figure 4. Inspection Travel Distances

Some appreciation of the magnitude of the inspection process can be gained from Figure 4. Considerable crew effort is involved in just examining the 3.4 miles of tubular struts which comprise the Station. Add to this the miles of electrical, thermal and data lines which could develop problems during the Station's 20, 30 or 50 year life. Fortunately, most of the required inspection activities can be performed without the need for extensive crew EVA time by utilizing a special purpose robotic mobile platform in conjunction with internal system sensors. The **Global Operational Flight Inspection/Repair System** or "GOFIRS" can perform (alone or in conjunction with a robotic MRMS) many of the inspection/servicing tasks identified in Reference [6] as listed in Table 1 (see asterisk items).

4. The "GOFIRS" Concept

In establishing the conceptual design of a robotic platform to meet the Station's needs, certain ground rules had to be established. These appear in Table 2.

Table 2. Major Conceptual Ground Rules

- **TASKS**
  - AUTONOMOUS INSPECTION ("SCOUT" FOR MRMS/TRANSPORT)
  - ASSIST CREW EVA (TOOL/ORU RETRIEVAL)
  - TELEOPERATED SERVICE & REPAIR (SMALL ORU MAINTENANCE)
  - CREW TRANSPORT (WHEN REQUIRED)
  - ACCOMODATE FTS (IF COST EFFECTIVE)
- **PERFORMANCE**
  - ACCESS STATION PRIMARY SUBSYSTEMS
  - REQUIRE NO MAJOR MODIFICATIONS TO STATION STRUCTURE
  - ON-BOARD INTELLIGENCE FOR INSITU DIAGNOSTICS OF SENSOR DATA
  - MANIPULATOR WITH ROBOTIC SENSOR (STEREO VISION FOR TELEOPERATOR SERVICING)
  - REACH ANY POINT ON STATION WITHIN 5 MINUTES (RATE=100 FT/MIN)
  - PROVIDE MINIMAL VIBRATION DISTURBANCES TO STATION
  - SATISFY ALL OTHER STATION OPERATIONAL, RELIABILITY, & SAFETY REQUIREMENTS

An important ground rule is that the GOFIRS is not intended to replace the much larger, more sophisticated MRMS/Transporter but to augment its capabilities by detecting possible mission threatening defects or faults. The ability of MRMS to perform routine inspection of the myriad of subsystems on board Station is limited by its large size (spanning more than a 5 meter bay) and its relatively slow speed (less than 2 feet/minute). A small, highly maneuverable platform capable of accessing tight interior spots could be designed to be simple enough, hence affordable, so that several GOFIRS could be on continuous patrol.

Although its primary mission is one of inspection, crew EVA assistance and transport could also be provided. Small scale teleoperated servicing and repair could also be accomplished. This could be particularly valuable if the MRMS was tied up completing an activity on one end of the Station when some system needed immediate servicing on the other end.

Another ground rule is that the GOFIRS should make maximum use of current robotic technology (sensors, computer architecture, manipulators, etc.) and yet have sufficient growth capability to accept more advanced A & R technology as it comes on line. An example of this is the Flight Telerobotic Servicer (FTS) concept to be developed as a "robotic front end" to the MRMS and OMV.

In terms of performance, the GOFIRS should be able to reach all or most of the important subsystems. It would be highly desirable to provide this mobility without disturbing the basic design of the primary structure by using miles of additional track or special cabling. GOFIRS should be equipped with sufficient onboard sensors and intelligence to perform routine analyses of inspection data and report anomalies and their location back to the command module. For example, the location of a leak in the thermal fluid lines would be identified. An onboard microprocessor would make a determination of the extent of this leak. Based on preprogrammed limits, immediate crew attention could be requested, or the anomaly could be simply "logged" for the next schedule maintenance activity.

Clearly, deciding a course of action based on real time sensory inputs would embrace the new and growing technology of Artificial Intelligence or AI. If corrective action is needed, a GOFIRS equipped with the appropriate teleoperator/telepresence sensors, cameras and manipulator, could make the repair under the control of a human operator. If the repair could not be made remotely from the command module, then the GOFIRS could transport a crew member to the site to make the repair. In a more futuristic version of this scenario, the repair could be made autonomously by GOFIRS under the automatic control of an "expert system".

Due to its mobility, GOFIRS would offer a secondary benefit of being able to ferry the crew and needed equipment, tools or ORU's around the Station for EVA. A minimum nominal travel rate of 100 feet/minute (1.1 miles/hr.) would enable any point on Station to be reached within 5 minutes. Of course the mass and acceleration rates must be sufficiently small as not to induce significant vibrational disturbances into the Station.

Table 3. The "GOFIRS" Concept

GLOBAL OPERATIONAL FLIGHT INSPECTION REPAIR SYSTEM

- SELF-INTELLIGENT, SMALL, HIGHLY MOBILE INSPECTION & ROBOTIC REPAIR PLATFORM
  - WHEEL (OR MAST) DRIVEN, TABLE-TOP SIZED VEHICLE WHICH CAN ACCESS MAJOR AND MINOR SPACE STATION SUBSYSTEMS.
  - EQUIPPED WITH ON-BOARD SENSORS & TELEMETRY TO AUTONOMOUSLY PERFORM ROUTINE INSPECTION.
  - EQUIPPED WITH CAMERAS & MULTI-DEGREE OF FREEDOM ROBOTIC MANIPULATOR TO PERFORM EITHER TELEOPERATOR OR TELEROBOTIC SERVICING OR UTILIZE FTS AND ORU'S
  - PROGRAMMABLE MICROPROCESSOR TO PERFORM ASSIGNED TASK, TRACK CURRENT LOCATION AND PERFORM ROUTINE ANALYSIS OF INSPECTION SENSOR DATA.
- MULTIPLE "GOFIRS" CAN CONTINUOUSLY PATROL DESIGNATED SEGMENTS OF SPACE STATION.
- CAN ASSIST MSC/TRANSPORTER AND ASTRONOUTS IN EVA ACTIVITIES.
- CAN FERRY CREW FROM POINT TO POINT
- "LOW LEVEL" GOFIRS CAN BE DEVELOPED IN THE TIME FOR FLIGHT EXPERIMENT DEMONSTRATION.

Features of the GOFIRS concept which meet the above ground rules are summarized in Table 3. It is envisioned that there may be a need for multiple GOFIRS to "patrol" various segments of the Space Station in a relatively slow "inspection" mode. It is also envisioned that one GOFIRS may be assigned to each of the outboard rotating solar wings while one or two more units will patrol the fixed central portion of Station.

This would circumvent the need to devise a means for crossing the alpha rotary joints. For example, 4 such GOFIRS dispersed in this manner could make a complete inspection of the Station's primary structure in approximately 60 hours at an inspection rate of 2 feet/minute.

Another point to be made is that a "low level" GOFIRS type system utilizing current state-of-the-art robotics technology could be developed in time for a shuttle flight demonstration prior to launching the Space Station.

5. Description of Capabilities

A conceptual illustration of a "wheel driven" GOFIRS appears in Figure 5. A description of features and capabilities is summarized in Table 4. On board microprocessor capability will be needed to perform the inspection/repair functions, control the GOFIRS motion, and to perform in situ diagnostic analysis of sensor data. This data can be logged and telemetried back to the crew capsule or ground using data compression techniques at some later time or at once if an emergency requires immediate attention. Location of the defect and information for guidance could be obtained by encoded magnetic strips like "bar codes" circumscribing the struts. Optical sensors could be used in place of these magnetic strips.

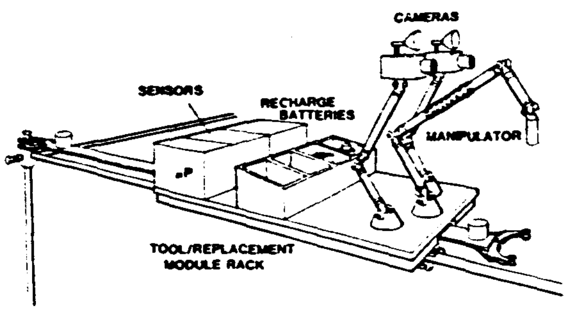


Figure 5. Global Operation Flight Inspection Repair System

Table 4. Capabilities

- SELF-INTELLIGENT MOTORIZED PLATFORM
  - ON-BOARD ROM FOR ITINERARY & INSPECTION/REPAIR FUNCTION
  - ON-BOARD ROM FOR MOTORIZATION, PIVOTING SEQUENCE & SELF-DIAGNOSTIC
  - ON-BOARD INSPECTION/DETECTION SENSOR & DIAGNOSTIC WITH TELEMETRY CAPABILITY (AMMONIA DETECTOR, TUBE DEFECT X-RAY DETECTOR, ETC)
  - ON-BOARD GUIDANCE (MAGNETIC STRIP POSITION LOCATOR)
- TELEROBOTIC FUNCTIONS/ROBOTIC MANIPULATION (8 D.O.F.)
  - SEMI-AUTONOMOUS OR TELE-OPERATOR REPAIR/COMPONET REPLACEMENT (BI-NOCULAR CAMERA/FORCE-FEEDBACK MANIPULATOR)
  - ROUTINE EMERGENCY SERVICING FUNCTIONS (ALPHA BRG PACKAGE, FLUID MODULE, ROLL-RING CHANGEOUT, ETC)
  - ON-BOARD "TOOL CHEST" REPLACEMENT PART CAROUSEL
- ELECTRO-MECHANICAL DESIGN FEATURES
  - "STEERABLE" & "LATCHABLE" DRIVE CASTOR WHEEL BOGIES } OR COLLAPSIBLE
  - POLYMER-COATED WHEELS (LIGHTLY SPRING LOADED) } MAST PROPELLED
  - NODAL PIN PIVOT LATCH MECHANISM
  - TUBE CLAMP MECHANISM FOR LARGE TORQUE REACTION
  - RECHARGABLE BATTERY PACK

Teleoperator or telerobotic repair would be accomplished with one or two dexterous manipulators having the appropriate tactile/force feedback sensors and utilizing multiple cameras (stereo-vision). Tools and replacement parts would be carried to facilitate either crew EVA or telerobotic On-orbit Replaceable Unit (ORU) changeout.

The platform would be motorized being either a wheel driven or propelled by a coilable mast arrangement to be discussed later. In the wheel driven variant, two "steerable" and "latchable" drive wheel bogies (See Figure 6) would be lightly spring loaded against the tubular strut with soft polymer coated wheels. The allowable contact pressures to prevent damage to the struts would be determined by extensive tests. However, in-house tests of a protype, aluminum-clad, carbon-graphite epoxy tube of the required size sustained a point load of over 200 pounds without damage. Anticipated wheel loads for the GOFIRS would be at least one order of magnitude lower than this. A tube clamp mechanism with a large footprint could be incorporated if needed to react large torques during part removal and replacement.

The large MRMS/Transporter will make use of pins attached to the truss nodal connectors to crawl along the station. These same pins, as shown in Figure 7 could be used to pivot the GOFIRS from strut to strut. In one arrangement, shown in Figure 8, a simple jaw type grip, equipped with a gear drive could swing or pivot the platform about the pin. The steerable wheel bogie (Figure 6) can "spiral" the platform to a sideface (see Figure 7) and the pivoting action can then take place. Diagonal members (not shown) can be reached by pivoting 45 degrees. Pivoting 180 degrees will permit continued motion along the same longeron. Thus through various combinations of pivoting and spiraling virtually any strut member can be reached without additional tracks or alterations to the primary structure.

Rechargeable batteries would provide energy for locomotion, microprocessing and robotics functions. A power bus outlet could recharge the batteries after a predetermined tour.

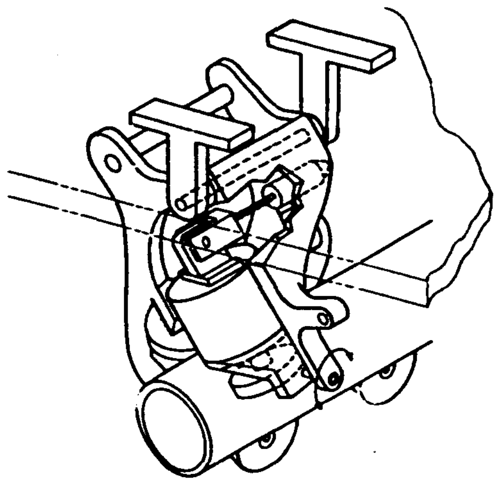


Figure 6. Wheel Bogie

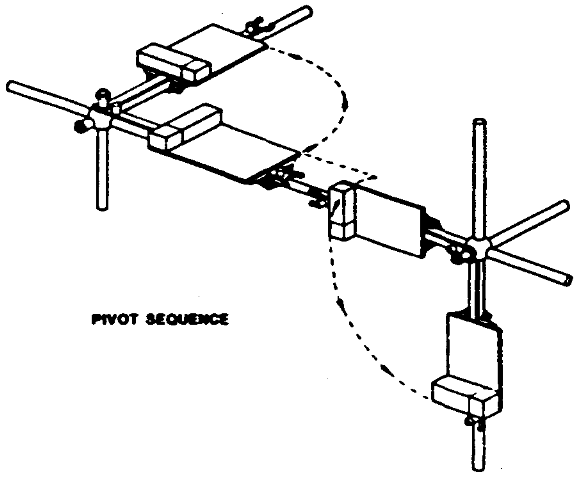


Figure 7. Pivot Sequence

Additional capabilities such as detecting, replacing and/or cleaning solar array panel segments could be achieved with GOFIRS as illustrated in Figure 9. Here the GOFIRS shuttles back and forth on a tubular strut supported by two expandable masts. Scanning across the array, "window washer fashion", the infrared sensors on board the manipulator are mapping temperatures to isolate malfunctioning solar cells.

In the event that the tubular struts themselves cannot be used for support, a twin mast driven platform is envisioned, as illustrated in Figure 10. Appropriately expanding and contracting the masts will provide linear motion. These coilable masts are similar to those conventionally used to deploy flexible solar arrays. A turn-table bearing will permit the platform to assume any planar orientation. The pivoting function will occur in the same manner as before with the exception that the forward mast segment will translate and fold (see Figure 10) to allow motion along the side face. Stiffness and buckling strength of the twin masts are not anticipated to be a problem. Adequate cycle life of the flexible battens must be established.

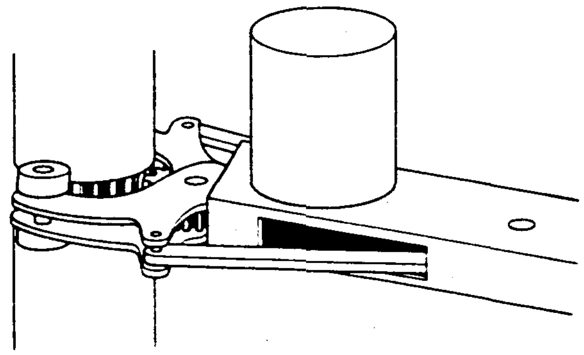


Figure 8. Nodal Pin

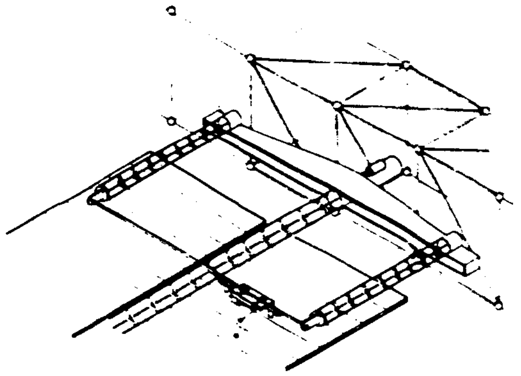


Figure 9. Solar Array

6. Telerobotics

In another variant, as shown in Figure 11, the GOFIRS serves as a combined crew transport and man-controlled servicer. Here the astronaut has a direct visual link with the repair or changeout activity while crew members aboard the Station or on the ground can participate if need be. The system is equipped with one or possibly two dexterous manipulators to facilitate teleoperator repairs.

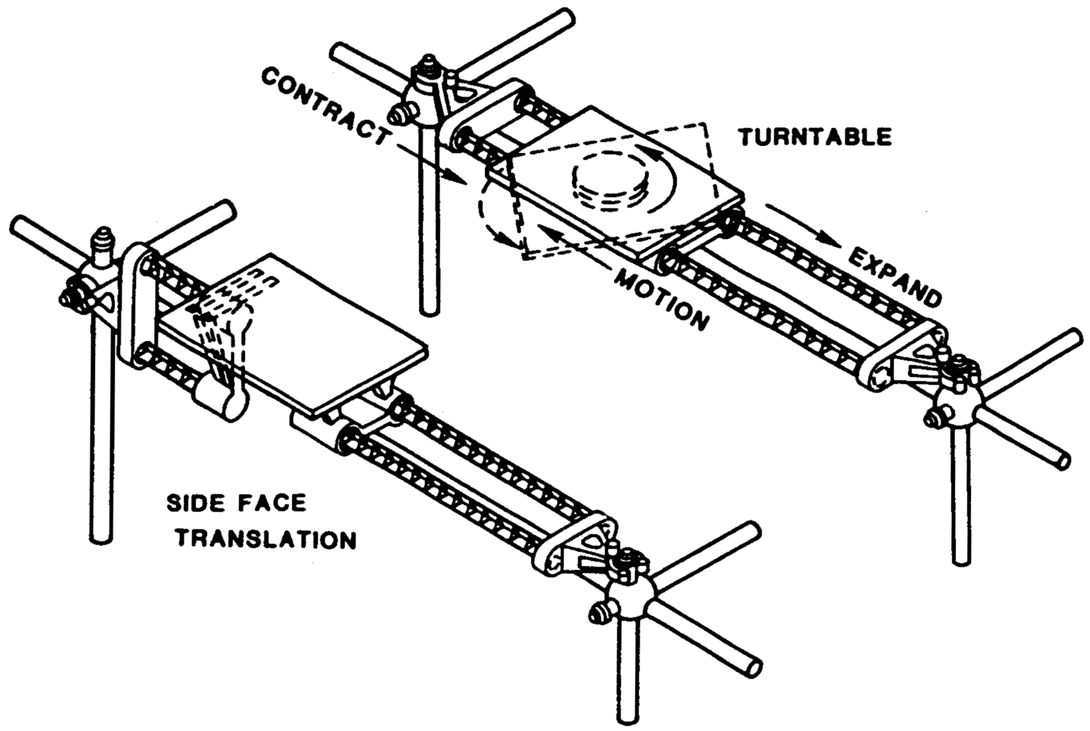


Figure 10. Twin Mast Propelled Platform

Figure 12 illustrates a more advanced telerobotic service configuration for GOFIRS. In principle, the robotic unit pictured here can be the same as that developed for the MRMS or Robotic OMV. The addition of the robot strengthens the GOFIRS capability in performing repairs and making replacements but will undoubtedly add to the cost and size of the platform. A cost-to-benefit assessment of the degree of robotic sophistication will be needed.

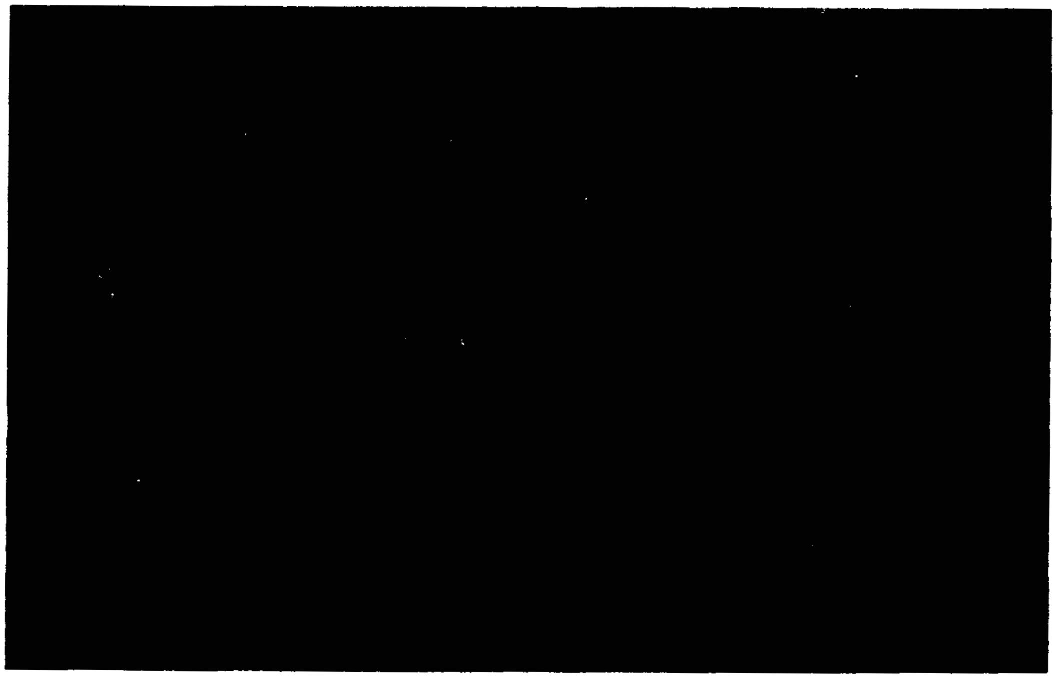


Figure 11. Crew-Controlled Servicer and Transport

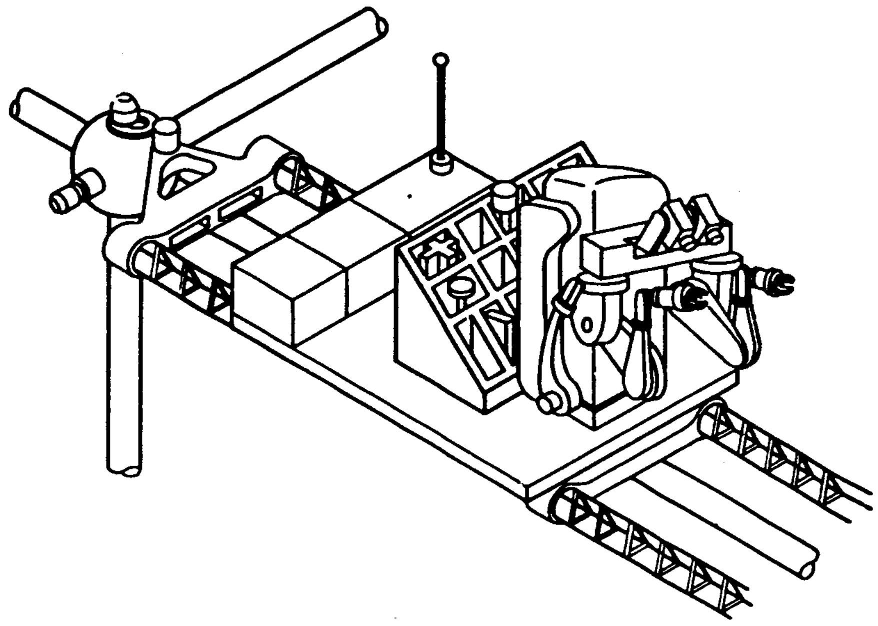


Figure 12. Flight Telerobotic Servicer Mounted Platform

7. A & R Technology For Space

Substantial progress has been made in the development and application of automation and robotic technologies for ground based applications. Industrial robots are now commonplace in factories. However, the repetitive, well structured, highly defined tasks that shop robots are well suited to perform are not commonplace in space. In space applications the tasks are often very diverse, less frequent and highly complex.

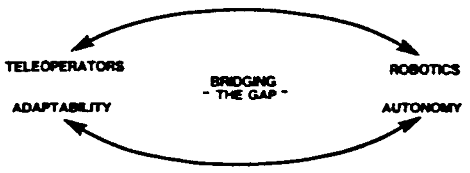


Figure 13. The A & R Challenge!

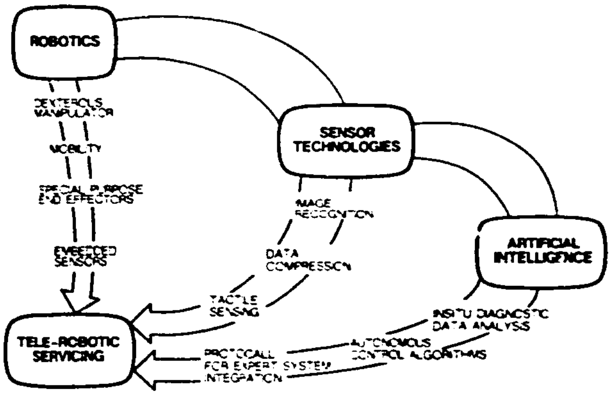


Figure 14. Enabling Technologies

Figure 13 best illustrates this dichotomy of needs. On one hand, robots are ideally suited to *autonomously* perform well-structured tasks. On the other hand, teleoperated manipulators, such as those used in the nuclear industry, can be *adapted* to perform less structured tasks due to their human operators. However, the price for this adaptability is the expenditure of dedicated operator time, a precious commodity aboard the Space Station.

The "challenge" is to bridge the gap between the adaptability afforded by a teleoperator and the autonomy offered by a robot. Some see this bridge as a "tele-robot", one system offering both capabilities, while leaning toward more teleoperation in the early years.

The GOFIRS concept presented here enbodies this duplicity of capabilities. Moreover, it offers the opportunity to have a *distributed* robotic capability about the Space Station, in the same manner as robotic machines are distributed about our factories. Consider the effectiveness and reliability of several smaller machines, simultaneously performing a sequence of simpler tasks in comparison to one super-sophisticated machine required to alone perform the cumulative tasks of the team of smaller robots.

Despite significant on-going progress in many areas of A & R technology, our current level of technology would only support a space based telerobot having relatively low level capabilities. A partial, by no means complete, list of areas where strengthening is warranted appears in Figure 14. The work needed in the robotics area can most quickly be envisioned by the somewhat facetious notion of adapting a 2-ton shop robot to become a space-qualified, flight manipulator. One that has sufficient dexterity to remove a defective circuit board from a delicate instrument if required. Reliabilities associated with today's industrial robots are far from those required for precision space mechanisms. Few, if any, have been designed to operate in a vacuum. Unfortunately, the effort needed to develop and demonstrate the relevant electro-mechanical technologies for a space-worthy, multidegree-of-freedom robot is sometimes under-appreciated.

Other areas requiring continued attention include a range of sensor and detector technologies, with high emphasis on vision related systems for teleoperation (see Figure 14). A whole family of inter-related activities fall under the area of machine intelligence, including task planning and reasoning, control execution, human interfaces and system architecture. The ability to make in-situ, real time, autonomous assessment of sensory inputs will be particularly important in enhancing crew productivity.

8. Conclusion

A concept for a self-intelligent, mobile platform is presented which can perform many of the inspection and maintenance activities envisioned for Space Station. Routine inspection related tasks can represent the single greatest expenditure of crew time given the shear size and complexity of Station's support systems. Several sensor-equipped, mobile platforms or GOFIRS working together with health monitoring sensors internal to these subsystems would be of great value in identifying not only the location but, moreover, the extent, hence urgency, of the defect. In this way, GOFIRS performs as a "scout" for the crew and relieves scheduling of the large teleoperated MRMS.

Features of the concept include the ability to move about Station without the need for special tracks or cables. Virtually all exterior and interior areas within the Station's framework are accessible.

The GOFIRS concept is modular, accomodating more advanced robotic capabilities as they evolve. In its simplest form, GOFIRS is an inspection cart with some crew and tool transport capabilities. Rudimentary teleoperation capability can be added with the addition of a flexible manipulator and vision equipment. Later, a more advanced flight telerobotic servicer unit could be accommodated. In this way a stepping stone approach can be taken. The advantage is the ability to demonstrate the concept at some low level with technologies available today at minimal risk.

The next phase of the work is to establish the performance requirements of a GOFIRS type system in relation to operational inspection and servicing activities to be scheduled on board the Station. This will set the frame work of preliminary design to arrive at the balance between cost, risk and capability. Suitability for early flight demonstration would also be assessed.

REFERENCES

[1] Advanced Technology Advisory Committee, "Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy", Volumes 1 & 2, NASA TM-87566, March, 1985.

[2] A. J. Meintel, Jr. and R. T. Schappell, "Remote Orbital Servicing System Concept, Satellite Services Workshop", NASA Johnson Space Center, June 22-24, 1982.

[3] L. M. Jenkins, "Telerobotic Work System Concepts", Proc. AIAA/NASA Symposium on Automation, Robotics and Advanced Computing for the National Space Program, September 4-6, 1985. Washington, D.C.

[4] R. E. Olsen and A. Quinn, "Advanced Orbital Servicing Capabilities Development", Aerospace Environmental Systems, paper number 860992, Proc of the 16th ICES Conference, July 14-16, 1986, pp. 719-732.

[5] H. L. Martin, D. P. Kuban, D. M. Williams, J. N. Herndoin and W. R. Hamel, "Recommendations for the Next-Generation Space Telerobot System", Oak Ridge National Lab TM-9951, March 1986.

[6] D. R. Flaherty, "Automation and Robotics Plan (DR-17)", Space Station Definitions and Preliminary Design, Work Package Number 2 McDonnell Douglas MDCH2036A, June 1986.

188088 S6-14 18

System Engineering Techniques for Establishing Balanced Design and Performance Guidelines for the Advanced Telerobotic Testbed

W.F. Zimmerman and J.R. Matijevic
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

CB 553097

1. Abstract

Research and development projects have characteristically followed development processes structured around well-defined, but loosely organized, research goals. This particular approach differs from the standard, application-specific, product development found in the private sector. Nevertheless, research and development often follows a less defined application route because of the substantial amount of technical risk associated with its research goals. Novel system engineering techniques have been developed and applied to establishing structured design and performance objectives for the Telerobotics Testbed that reduce technical risk while still allowing the testbed to demonstrate an advancement in state-of-the-art robotic technologies. To establish the appropriate tradeoff structure and balance of technology performance against technical risk, an analytical data base was developed which drew on 1) automation/robot-technology availability projections, 2) typical or potential application mission task sets, 3) performance simulations, 4) project schedule constraints, and 5) project funding constraints. Design tradeoffs and configuration/performance iterations were conducted by comparing feasible technology/task set configurations against schedule/budget constraints as well as original program target technology objectives. The final system configuration, task set, and technology set reflected a balanced advancement in state-of-the-art robotic technologies, while meeting programmatic objectives and schedule/cost constraints.

2. Introduction

Funding limitations in both private and government sectors often make it difficult for research and development environments to operate totally independently of mainstream applications of potential products resulting from the research. Similarly, the Telerobotics Testbed, a research and development effort, is being viewed as a source of advanced seed robotic technology for the Space Station Flight Telerobotic Servicer (FTS). The near horizon for first-element launch (FEL) and initial operational capability (IOC) (i.e., the early to mid-1990's) places some pressure on the testbed breadboard effort to tailor its technology thrusts, and potential applications, towards these near-term developments. One of the challenges associated with defining the testbed breadboard development program is finding the appropriate balance between establishing an aggressive technology development program, yet maintaining a viable application channel with the Space Station FTS environment and development schedule. From an operations research viewpoint, this situation represents the classical problem of satisfying several competing objectives with limited resources. Although it would appear that classical linear programming or "branch and bound" optimization techniques could be applied as solution structures to the competing objectives problem, in fact the introduction of key intangible (i.e., not readily quantifiable) variables made the solution of the problem not immediately amenable to a rigorous mathematical representation. Nevertheless, optimization techniques such as branch and bound provided a structure for obtaining progressive, feasible sets of solutions that could be independently examined until a "reasonable" solution to the performance versus technical risk tradeoff problem was found. The following paragraphs discuss 1) how the overall problem and solution structure was developed, 2) the tradeoff variables (both tangible and intangible), 3) the rationale behind the derivation of feasible solution sets, 4) the selected feasible solution and associated bounds, and 5) supporting data.

3. Problem Definition and Solution Structure

The first step in obtaining a solution to the competing objectives problem was to establish a concise definition of the objectives and constraints. The major variables that needed to be satisfied in the tradeoff process were as follows:

1. Programmatic technology objectives - Addresses the overall approved technology goals jointly agreed to by the research sponsor (NASA Office of Aeronautics and Space Technology [OAST]) and the responsible research organizations (Jet Propulsion

Laboratory, Langley Research Center, Marshall Space Flight Center, Johnson Space Center, Ames Research Center, Goddard Space Flight Center, and Massachusetts Institute of Technology.)

2. Viable mission task set - Refers to the development of an application environment which is both feasible (in terms of technology performance capabilities and constraints) and representative of a real-world use of the technology.
3. Schedule - Addresses the time constraint associated with completing the technology objectives as part of normal programmatic planning/assessment, and meeting other outside schedule needs such as the FTS FEL/IOC development and qualification milestones.
4. Cost - Refers to the budgetary constraint imposed at the programmatic control organization (NASA OAST).
5. Performance - Addresses the capability of the hardware and software to actually execute and successfully complete a selected task set (a measure of technical risk).
6. Technology availability - Refers to the actual state of maturity of a given technology element as measured against state-of-the-art and in the context of the overall system capability to perform a selected task set.

In examining each of the above variables in terms of "objectives" and "constraints" it became clear that the first two variables (technology objectives and mission task set) represented the primary optimization objectives. The ability of the program, and actual system design, to reach these objectives would be subject to the constraints imposed respectively by schedule, cost, hardware and software performance limitations, and the relative states of achievable maturity of the component technologies. Mathematically, the optimization problem could be stated as follows:

$$\max_{T,a} \sum_{t=1}^{n} \sum_{a=1}^{m} T_{t_a} \tag{1}$$

subject to,

$$\sum_{t=1}^{n} \sum_{a=1}^{m} s_{t_a} \leq S_T \tag{2}$$

$$\sum_{t=1}^{n} \sum_{a=1}^{m} c_{t_a} \leq C_T \tag{3}$$

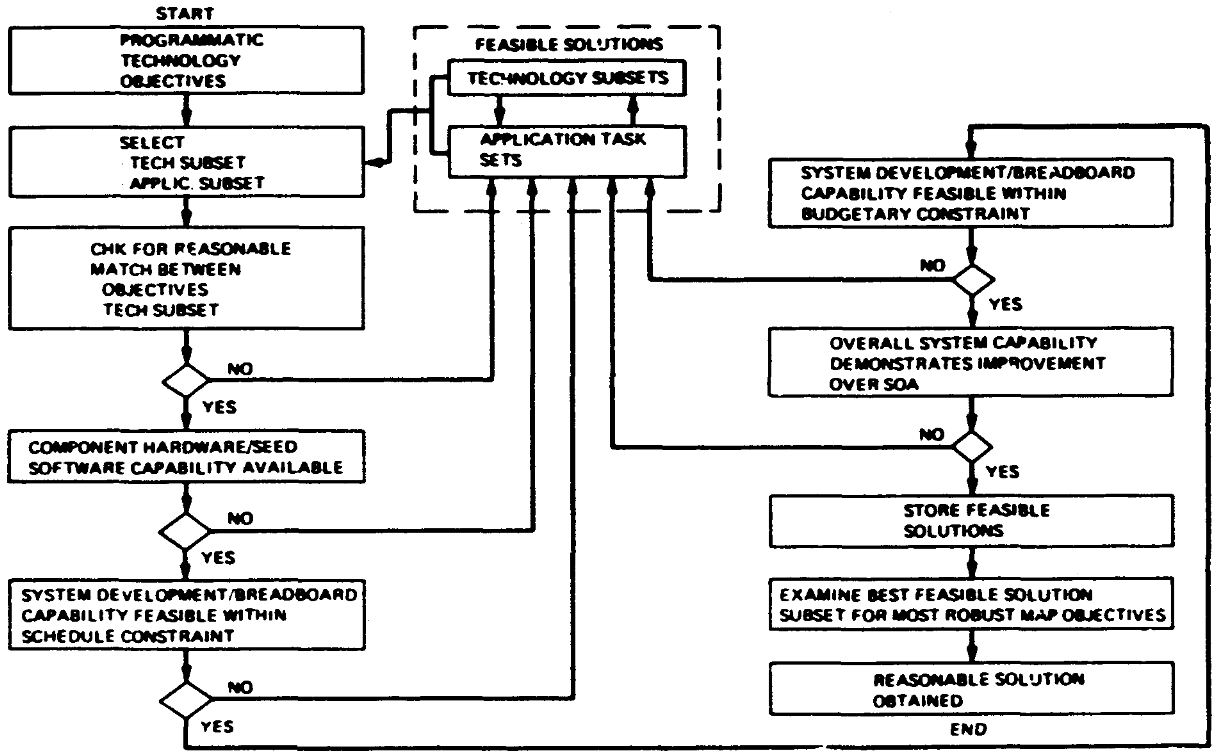
$$\sum_{t=1}^{n} \sum_{a=1}^{m} p_{t_a} = P_T \tag{4}$$

$$\sum_{t=1}^{n} \sum_{a=1}^{m} t'_{t_a} > T'_T \tag{5}$$

The above formulation basically states that it is desirable to maximize the overall targeted technology capability (T) and feasible application task performance capability (a) subject to 1) the respective technology development schedules (s) not exceeding the overall programmatic schedule (S), 2) the respective technology development costs (c) not exceeding the overall programmatic cost ceiling (C), 3) the respective technology performance limitations (p) being commensurate with the overall programmatic technology performance objectives (P), and 4) the aggregate achievable technology maturities (t') being greater than the overall state-of-the-art technology level (T'). The above formulation serves the purpose of providing a clea statement of the competing objectives problem. However, from a practical standpoint it is very difficult to actually measure all of the above variables. For example, the technology objectives and application task set do not lend themselves to quantification in the same sense as cost and schedule. Similarly, setting the state-of-the-art technology baseline and comparing the composite testbed technology maturity level against that baseline is also difficult to quantify. Therefore, these three variables represented *important, but intangible variables*. The remaining variables (schedule, cost, and performance) represented the tangible variables.

In order to cope with the intangible variables, a more empirical approach was taken to structuring the optimization problem. Keeping the objective function and constraints the same, a modified branch and bound technique was formulated that provided a tradeoff structure that could accommodate both quantitative and qualitative representations of the objective and constraint variables.

Therefore, the next step in formulating the solution was to tailor the branch and bound optimization structure to handle both qualitative and quantitative decision data. By definition, the branch and bound optimization technique starts by setting a bound on the objective function (Ref. 1). Next, the technique requires that the set of all feasible solutions (i.e., in this case the technology and application task sets) first be partitioned into several subsets. Because the objective of the exercise is to maximize the chances of meeting the original technology objectives while exercising those technologies in the most robust application environment possible, any subset of alternative technologies and applications that does not meet the original objectives is eliminated. Each subset is evaluated against the objective function and constraints until a solution is found that meets all conditions. In the absence of a clear-cut analytical solution to the competing objectives problem, a decision network was designed that allowed the subset partitioning and evaluation steps to be completed in exactly the same spirit of the branch and bound solution structure outlined earlier. This decision network is shown in Figure 1.



START
PROGRAMMATIC TECHNOLOGY OBJECTIVES
SELECT TECH SUBSET APPLIC. SUBSET
CHK FOR REASONABLE MATCH BETWEEN OBJECTIVES TECH SUBSET
NO
YES
COMPONENT HARDWARE/SEED SOFTWARE CAPABILITY AVAILABLE
NO
YES
SYSTEM DEVELOPMENT/BREADBOARD CAPABILITY FEASIBLE WITHIN SCHEDULE CONSTRAINT
NO
YES
FEASIBLE SOLUTIONS
TECHNOLOGY SUBSETS
APPLICATION TASK SETS
SYSTEM DEVELOPMENT/BREADBOARD CAPABILITY FEASIBLE WITHIN BUDGETARY CONSTRAINT
NO
YES
OVERALL SYSTEM CAPABILITY DEMONSTRATES IMPROVEMENT OVER SOA
NO
YES
STORE FEASIBLE SOLUTIONS
EXAMINE BEST FEASIBLE SOLUTION SUBSET FOR MOST ROBUST MAP OBJECTIVES
REASONABLE SOLUTION OBTAINED
END

Figure 1. Feasible Solution Decision Network

Figure 1 displays the serial decision process that allows the subsets of feasible solutions to be filtered out of a large group of candidate technologies and applications. Note that the decision structure is designed to be an "and" decision gate so that both objectives and constraints must be simultaneously satisfied (as implied in eqs. 1-5) to obtain a "reasonable" solution. It should also be noted that although the above structure provides a reasonable solution, by design, it does not yield the rigorous, analytical numerical solution that linear programming or classical branch and bound optimization techniques yield.

4. Data Base

The above objective and constraint variables were supported by an extensive quantitative and qualitative data base. These various data bases are summarized below:

1. Programmatic technology objectives (qualitative) - The programmatic objectives were established at the onset of the testbed project (Ref. 2). The overall Phase 1 (FY 1987/1988) program objectives were 1) automated object acquisition and tracking, 2) video-based location/orientation of simple objects, 3) off-line coordination-level telerobot activity planning, 4) an architecture for coordinated planning/diagnostics for telerobot command and control, 5) dual-arm coordinated control with hybrid force/torque, position, and rate feedback, 6) dual force reflecting hand controllers, stereo display, and fused force/torque video feedback for teleoperation, 7) an architecture for run-time control of the telerobot with the capability to interpret and execute task primitive commands generated by the activity planner, and 8) a

distributed, multi-processor command and control hierarchy with the capability to be modularly upgraded and provide simple error recovery.

2. Viable mission task set (qualitative) - A fairly extensive literature search was conducted to establish an application task set in which to develop and test the various technologies and overall telerobot system (Refs. 3-13). At the onset of this portion of the analysis it was assumed that the most viable application of the telerobot, in the near term (per the FTS augment to extravehicular activity), would be for on-orbit assembly and servicing. Therefore, the application task set was sought primarily in planned, or historical, on-orbit servicing activities. Skylab and Shuttle historical experiences were most useful. Unfortunately, proposed Space Station-related servicing missions such as Space Telescope were not defined to a level of detail that would facilitate an accurate mapping between servicing functions and needed technologies. Ultimately, the Solar Max repair mission provided a full array of detailed servicing tasks that was sufficiently granular and representative of probable FTS servicing activities so as to provide a good starting application subset.
3. Schedule (quantitative) - The schedule constraints imposed on the project were 1) a demonstration of core technology elements by end of FY 1987, 2) followed by a full integrated demonstration of the complete telerobotic breadboard system by end of FY 1988.
4. Cost (quantitative) - The cost constraint for the project for the three-year effort starting FY 1986 (including funding outside leverage from other NASA centers, industry, and universities) was projected to be approximately $20M.
5. Performance (quantitative) - The performance envelope of the technologies was derived from the actual physical capabilities and constraints of the hardware and software used in the research laboratory. For example, the vision subsystem was able to provide fixture location to within 1 mm and resolve unoccluded fixtures (within the constraints of the internal object model software) such as small panels, handles, or bolt heads. The PUMA 560 arms (typical of nationwide laboratory hardware) used in the control technology development, had specified reach envelopes, joint movement constraints, and load-handling capabilities. Once a task set, object library, and task data base (object locations, forces, torques, etc.) were established, the system performance was simulated on an IRIS dynamic computer display system to obtain a rough estimate of system and application feasibility. A single frame of the dual arm servicing simulation is shown in Figure 2.

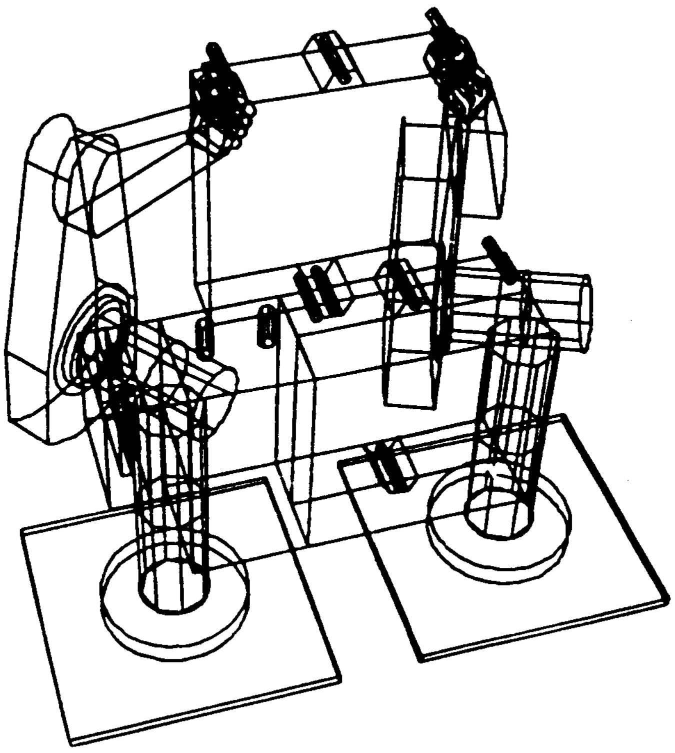
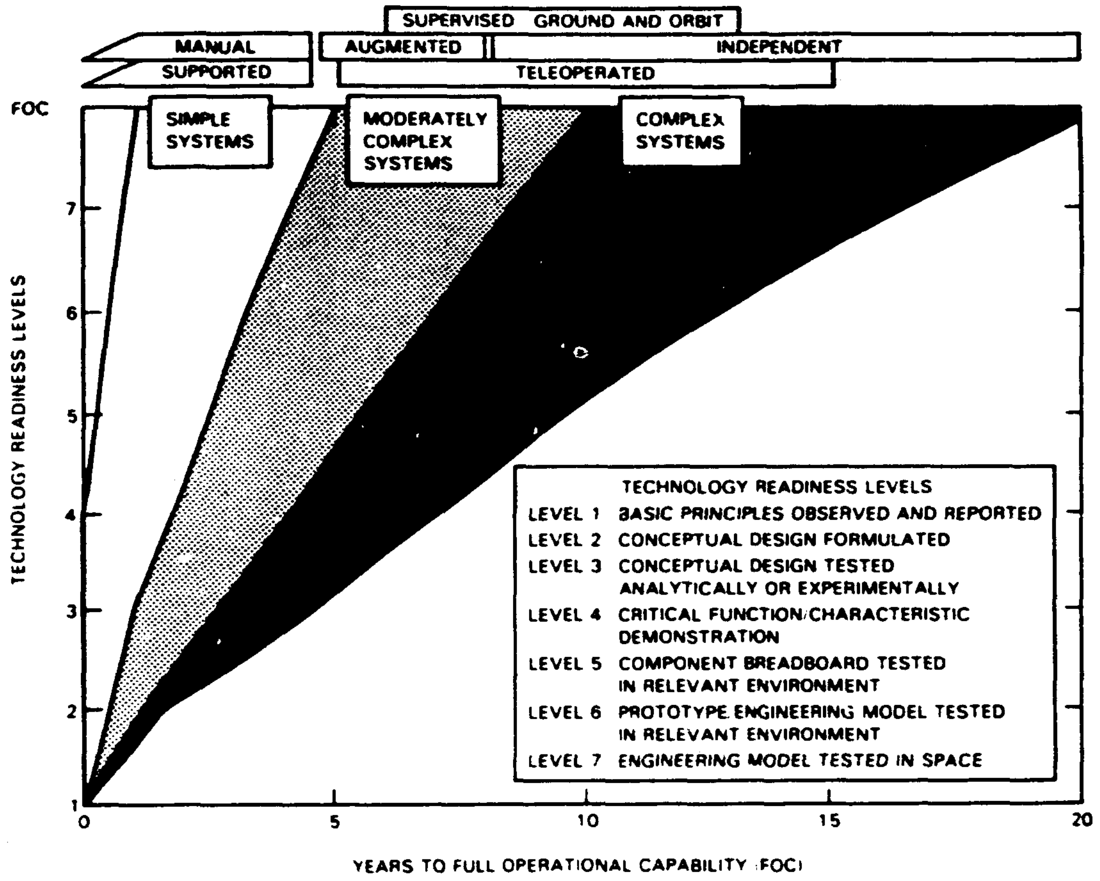


Figure 2. Dual Arm Telerobot Servicing Simulation (IRIS)

6. Technology maturity and availability (qualitative) - When faced with hard schedule and budgetary constraints, projects must set their sights on technology goals which represent both an advancement as well as a realistic, achievable objective. Although some studies have been done which suggest both maturity levels and time frames for the

breadboard and fully operational versions of advanced automation technologies (see Refs. 14, 15, and 16), generally it is extremely difficult to bound, or constrain, a qualitative variable using an upper bound which has a fairly large variance itself. This problem is compounded when considering other constraints such as setting technology goals that enable the breadboard development (i.e., the FY 1988 schedule constraint) to transfer technology in a timely manner to both the FTS brassboard and fully operational configurations. This development constraint implies that a distinct time frame is needed to move through all the development stages as shown in Figure 3.



SIMPLE SYSTEMS: IMPLEMENTATION OF A SINGULAR ACTION; OPERATIONS GENERALLY INDEPENDENT OF OTHER FUNCTIONS. UNIQUE APPLICATIONS ALTHOUGH BASIC PRINCIPLES WELL UNDERSTOOD

MODERATELY COMPLEX SYSTEMS: MULTIPLE INTERACTING FUNCTIONS OR ACTIONS. COMPLEX CONTROL LOGIC OR NETWORKS. BASIC IMPLEMENTATION TECHNIQUES SIMILAR TO PREVIOUSLY DEVELOPED SYSTEMS

COMPLEX SYSTEMS: MULTIPLE INTERACTING FUNCTIONS OR ACTIONS. COMPLEX CONTROL LOGIC OR NETWORKS. REDUCTION TO PRACTICE OF DESIGN CONCEPTS WHEN COMPARABLE SYSTEM HAS NOT BEEN DEVELOPED

Figure 3. Technology Development as a Function of Readiness Levels and Time

Therefore, rather than establishing an upper constraint for the maturity variable, a state-of-the-art baseline was established and used as a known, lower bound. The state-of-the art lower bound then simply had to be exceeded while simultaneously providing a viable breadboard configuration that could appropriately meet FTS schedule constraints. The state-of-the-art baseline was set against available, working engineering models and included 1) sensing and perception - simple labeled and unlabeled object tracking with manual acquisition; 2) task planning/reasoning - off-line sequence generation and no well-structured human-robot cooperative plan generation; 3) operator interface - dual arm teleoperation, limited real-time computer graphic displays, stereo vision, limited external state sensing, limited operator/workstation integration, no traded control between teleoperation and autonomous states; 4) control execution - model-based single arm control or teach pendant, leader-follower dual arm position control, limited hybrid control, 5) control architecture

and integration - limited hierarchical control, centralized processing/memory, coordination level control in structured manufacturing environments, distributed processing architectures, teleoperation and autonomous control not traded, limited hierarchical error management.

5. Tradeoff Results

The last step in the analysis was to execute and re-execute the Figure 1 decision structure until a reasonable solution was obtained which met both the objective function and constraints. The iteration process commenced with publishing an application task set (drawing on the full Solar Max servicing scenario) along with the projected commensurate implementation technologies. Immediate problems were encountered because 1) the real-time reconfiguration task elements associated with main electronics box (MEB) exceeded the task planning capability of the system, 2) object masses and electric socket removal forces exceeded the load characteristics of the PUMA arms, 3) some component disassembly sequences exceeded the hardware and software control characteristics of the PUMA arms and control algorithms, and 4) the large array of geometric shapes associated with the servicing environment exceeded the vision system CAD data base. The servicing scenario was downscaled. The task-related objects were redesigned to accommodate the PUMA constraints and simplified.

In the manner described above, each application task set and corresponding technologies were reviewed with the various subsystem research engineers against schedule constraints, budgetary limitations, hardware/software limitations, and the state-of-the-art baseline until a subset of each was obtained which satisfied all the objectives and the constraints. The corresponding solution set is shown in Table 1 (Ref. 17).

Table 1. Telerobot Application and Technology Solution Subsets

Application Task Set	Technology
1. Capture/dock slowly rotating satellite (1 rpm)	Automated labeled object acquisition, tracking, dual arm servoing
2. Verify initial object in task sequence (MACS)	Automated stationary object verification
3. Remove star tracker covers on MACS	Teleoperation under alignment/ accuracy/force constraints (dual arm)
4. Confirm auto sequence plan	Operator-AI planner interaction (operator can update object location, confirm plan, or update a task monitoring point)
5. Teleop traded off to auto, verify/grasp bolt wrench	Automated object verification, plan execution, hierarchical control with limited error recovery
6. Remove MACS retaining bolts	Automated object verification, hybrid force/position and force/ torque control with trimming
7. Remove/replace MACS	Automated object verification, dual coordinated master/slave arm control, simple collision avoidance, position and rate control
8. Auto traded off to teleop for satellite repositioning	Dual arm teleoperation, position/alignment control (video, stereo, 6 DOF hand control, and voice camera control)
9. Remove MEB thermal blanket	Same as 8 above, handling flexible objects
10. Teleop traded off to auto, hinged panel door opened, simplified MEB electrical connectors removed, MEB removed and replaced	Same as 4 through 7 above, limited automated flexible object handling, precise automated control in simple obstacle field with quarded motion along an arc

The above table is somewhat abbreviated for summary purposes. However, the complete detailed application task set and technology correlation is provided in the Telerobot Testbed functional requirements (Ref. 17). By far, the largest improvements in the respective technologies over state-of-the-art revolved around the vision-based fixture update and its integration with the control execution, the integration of the planner with the control execution, the auto to teleop traded control, the dual arm coordinated control, and the distributed control hierarchical design with on-line (although simple) error management woven throughout the hierarchy. The application task set, although simplified to meet performance and technology constraints, still provided a viable environment reasonably close to projected orbital replacement unit (ORU) removal/replacement FTS tasks. Finally, the selected technology subset was reasonably in-line with schedule/cost constraints; and, although composed of both state-of-the-art technologies and evolutionary (as opposed to revolutionary) improvements over other state-of-the-art technologies, the selected subset appeared achievable in a manner commensurate with supporting the out-year FTS development.

6. Conclusions

The revised branch and bound solution structure augmented with the supporting data bases and system simulation provided an excellent blueprint for obtaining a reasonable solution to an extremely difficult tradeoff problem. This technique has proven very useful for structuring the Telerobot Testbed research and development program to be sensitive to real-world demands and constraints. The technique is presently being employed to start negotiating and planning the 1990 demonstration.

7. Acknowledgments

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the Office of Aeronautics and Space Technology (OAST), National Aeronautics and Space Administration (NASA).

8. References

[1] Hillier, F., and Lieberman, G., Introduction to Operations Research, Third Edition, Holden-Day Inc., Oakland, California, 1980.
[2] Schober, W., "Automation and Robotics," RTOP 506-45, (JPL Internal Document) Jet Propulsion Laboratory, Pasadena, California, December 10, 1985.
[3] NASA Langley, "Langley Workshop: Projected Shuttle, OMV, OTV Missions," updated April 1985.
[4] Hoverkamp, J., "MSFC Skylab Mission Sequence Evaluation," Marshall Space Flight Center, NASA report no. TMX-64816, March 1974.
[5] Freeman, D., "STS Work Day Handbook," Johnson Space Center, Operations Planning, February 1980.
[6] Loftus, J., "An Historical Review of NASA Manned Spacecraft Crew Stations," NASA Johnson NASA Johnson Space Center, undated.
[7] Do Lau Lam, et al., "Study on Manned Versus Automated Space Activities," ESA, December 1980.
[8] Essex Corp., "Analysis of Large Space Structure Assembly: Man-Machine Assembly Analysis," NASA MSFC, Report no. 3751, 1983.
[9] Cepollina, F., Sprott, L., "On-Orbit Maintenance and Repair: New Era in Space Research and Industrialization (Solar Max)," NASA Goddard Space Flight Center, undated.
[10] Cepollina, F., et al., "Extravehicular Activity Annex: Solar Max Repair Mission," NASA JSC doc. no. 14082, Annex 11, Rev. A, March 1984.
[11] Loughead, T., Pruett, E., "EVA Manipulation and Assembly of Space Structure Columns," NASA MSFC, Report no. 3285, 1980.
[12] Zimmerman, W., et al., "Space Station Man-Machine Automation Tradeoff Analysis," JPL Pub. 85-13, Jet Propulsion Laboratory, Pasadena, California, February 15, 1985.
[13] Oberg, J., "Space: Soviet Space Walks," OMNI, Vol. 7, no. 2, November 1984.
[14] NASA, Advanced Technology Advisory Committee (ATAC), "Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy," NASA Technical Memorandum 87566, March 1985.
[15] Firschein, O., et al., "NASA Space Station Automation: AI-Based Technology Review,: SRI International, March 26, 1985.
[16] Zimmerman, W., Marzwell, N., "Space Station Level B Automation Technology Forecasting/Planning Structure," JPL, Space Station Level B White Paper, (NASA internal document), NASA Johnson Space Center (Space Station Office), Houston, Texas, April 30, 1985.
[17] Jet Propulsion Laboratory, "Functional Requirements for the 1988 Telerobotics Testbed" (JPL internal document D-3693), Jet Propulsion Laboratory, Pasadena, California, October 1986.

188089 S7-18

88

Dedicated Robotic Servicing for the Space Station

R.F. Thompson, G. Arnold, and D. Gutow
Rocketdyne Division, Rockwell International Corporation
Canoga Park, CA 91303

[illegible]

1. ABSTRACT

~~This paper presents~~ the concept of a series of dedicated robotics manipulators that would be resident in the subsystems of the Space Station. These would be used to do Orbital Replacement Unit (ORU) exchanges, inspection of the components, and in certain cases subsystem assembly. By performing these well definded tasks automatically, higher crew productivity would be achieved. In order to utilize the robots effectively ORU's must be designed to allow remote release and quick disconnection of the electrical, fluid, and thermal connections. The robot must be of a modular design for ease of maintenance and must have an adaptive control capibility to make-up for slight errors in programming.

2. INTRODUCTION

The construction, operation and maintenance of the Space Station will present many challenges. In the past space based systems required that the components be certified for the life of the mission with little or no opportunity for servicing. Since the Space Station will be a permanent manned platform in space the opportunity exists to not only service and maintain the components but also to update them as improved technology is developed. With this in mind it is important to utilize available resources and techniques to design the station to be as easily assembled, serviced, and maintained as possible. In addition it is important to keep in mind that the purpose of the astronauts presence on the space station is to provide support for the experiments and manufacturing efforts in space and not to be incumbered with the mundane tasks of station maintenance and servicing.

3. METHODS FOR ASSEMBLY, SERVICING, OPERATION, AND MAINTENANCE

Three methods exist for assembly, servicing and maintenance of the Space Station. These are Extra-Vehicular-Activity (EVA), Inter-vehicular Activity (IVA) using remote teleoperated manipulators and automation in the form of fully automatic robotic manipulators.

EVA

EVA provides the greatest flexibility of the three methods since the astronaut can interact directly with the system. However, this presents the highest risk to the astronaut. In addition several other drawbacks exist. This method is expensive in that the life support systems (EVA suit) are expensive to maintain with costs estimated to be $80,000 per EVA hour. Also when comparing the time to perform tasks during EVA to normal activities on the ground it can take up to eight times as long to perform the same task. Additional time must be spent in pre and post EVA activities. With these facts in mind it would thus be important for the astronaut to remain in the station unless absolutely required to leave.

IVA

IVA allows the astronaut to remain in the station and perform tasks outside using a remote teleoperated manipulator. Presently two systems the Flight Telerobitic Servicer (FTS) and the Mobile Service Center (MSC) have been identified to do this. Teleoperation allows a high degree of flexibility in positioning the manipulator to perform the tasks since it is under continuous control of the astronaut. However, it is not a simple task to position the manipulator even using multiple cameras and displays. This method is also time consuming and can take up to sixteen times as long normal ground activities. In addition the astronauts attention is required even for the simplest activities such a moving along the truss. Also the manipulator is making unplanned moves that can impart inertial loads on the station resulting in vibrations or effecting the validity of low g experiments.

ROBOTICS

The last method is to use fully automated devices such as robots. Robotic devices can be preprogrammed to perform tasks that would not require the direct attention of the astronaut. As part of the program the robot could pause at critical points and the allow the astronaut to view the operation and correct the motions, if necessary, before continuing. The ability to control the robot remotely would be provided as a back-up. Because of the relative inflexibility of the robotic devices the tasks would have to be well defined. These tasks would include ORU removal and replacement, component inspection, and, in limited cases, assembly. This would free up the astronaut to do other less defined tasks such as shuttle unloading and satellite capture and servicing using the teleoperated devices. It would be difficult to program a single mobile robot to service all parts of the station. A more viable solution would be to provide dedicated robotic devices as an integral part of the Space Station subsystems.

4. DEDICATED ROBOTICS

This concept provides for modular robotic devices that would be dedicated to the assembly, operation and servicing of particular subsystems on the Space Station. By limiting the tasks required of a particular robot to those for that subsystem the complexity can be significantly reduced.

ORU REPLACEMENT

The primary task of the dedicated robots would consist of removal and replacement of ORUs. The replacement would be performed on a preprogrammed basis and would not require intervention by the astronaut. Because the robot is local to the subsystem, it could aid in the diagnosis of the failure through subsystem testing. If the immediate cause of the failure cannot be identified, or narrowed to a particular ORU the robot would be available to do ORU swapping to determine which component had failed.

ASSEMBLY

Well defined assembly tasks such as first time insertion of ORU's into the subsystems could be accomplished using the robots. With proper considerations given component design such as a common interface and methods of locking the component into place on the structure the robot could perform more complicated assembly operations. These same techniques would be useful in reducing the complexity and time required to do assembly using EVA or IVA. Using multiple robots in the individual subsystems would allow simultaneous operations to proceed.

INSPECTION

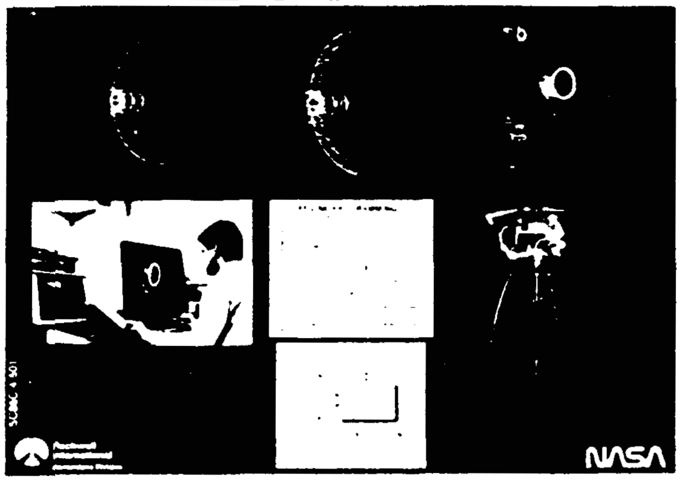
If it is determined that the cause of the failure is external to the ORU, the astronaut could use the robots sensors and vision system to help identify the cause of the failure. The astronaut would either program the robot to make the repair or use the information obtained to help plan an EVA. Preventative inspections and service of the components in the subsystem would help to predict and prevent catastrophic failures. These could be carried out on a regular basis without astronaut attention and be reported directly to the health monitoring system.

PROGRAMMING

A primary concern for use of robots would be the programming of the many and various tasks. This could be accomplished using graphic simulation and Offline Programming (OLP) based on a CAD data base of the Space Station. The majority of tasks can be identified, programmed, and simulated prior to launch. These programs would be stored and executed as required during operation. However, many tasks would require programming during flight. This could be accomplished by ground crews using the OLP and simulation stations, and then uploaded to the station for execution. The astronaut could perform the simulation (and programming if necessary) on board the Space Station to verify that the task will be accomplished to his/her satisfaction. The OLP/simulation system would be provided with a user friendly interface. Specifying the particular subsystem in question would bring a simulated cell onto the display with the robot and all components. By simply indicating the positions to move to or the task to perform the program would be simulated and down loaded to the robot for execution. A similar system to this is being developed at Rocketdyne for welding the Space Shuttle Main Engines as shown in figure 1.

FIGURE 1

OFF LINE PROGRAMMING

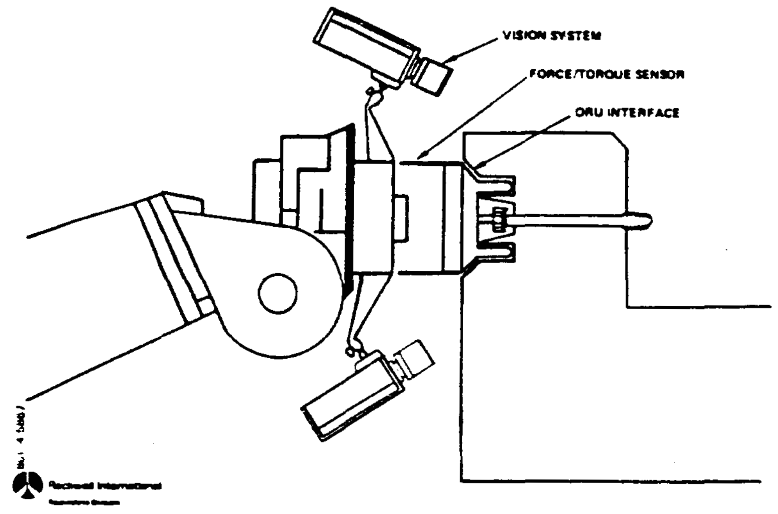


END EFFECTORS

Critical to the operation of the robot, or any remote manipulator would be the ability to interface with the various ORUs. The end effector would be designed to mate with the ORU as shown in figure 2 and include a mechanism for actuating the built in locking and ejection system. This will require the ability to easily change end effectors to accommodate various ORUs within the subsystem. In addition a compact end effector which houses multiple sensors could be provided. This would be used during the regular inspection periods and for trouble shooting and diagnosing problems.

FIGURE 2

TASK SPECIFIC END-EFFECTORS
REMOVE AND REPLACE ORU'S

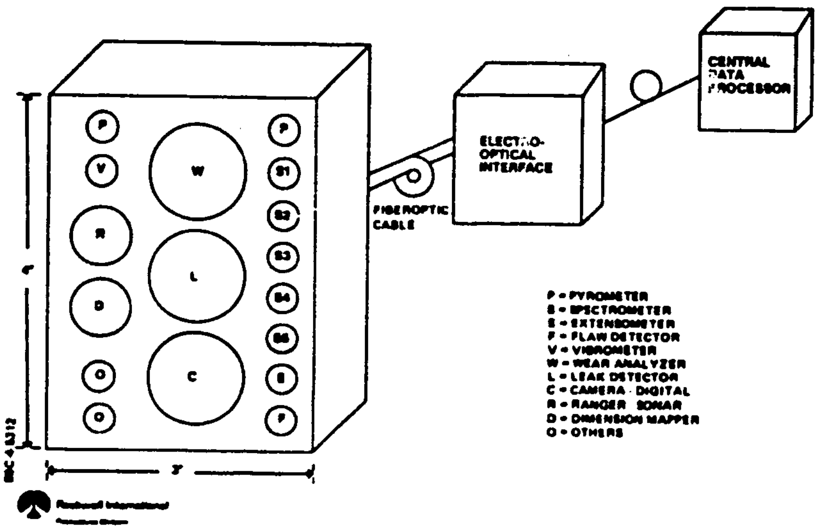


SENSORS

Offline programming and graphic simulation provides a path for the robot which will avoid collisions. To make up for the variation between the programmed path and the actual path both vision and tactile sensors will be required. The vision system as shown in figure 2 will allow the robot to adapt to variation in the location of the ORU interface and position the robot for final docking with the ORU. Additionally the system would include an optical character reader to identify the ORU. A force/torque sensor in the wrist would be used to adaptively position the robot to prevent jamming and provide a smooth, parallel insertion. This is also shown in figure 2. During the inspection task, various types of sensors will be required. Voltage, current, logic, and communication checks can be performed with a "plug in" type connection to ports on the ORUs. Measurement of mechanical properties such as vibration, temperature, wear, torque and surface defects are more difficult. Figure 3 shows a concept for a compact end effector with multiple sensors that are fiberoptically coupled to the control electronics.

FIGURE 3

END EFFECTOR FOR COMPREHENSIVE INSPECTION BY DEDICATED SERVICING ROBOTS

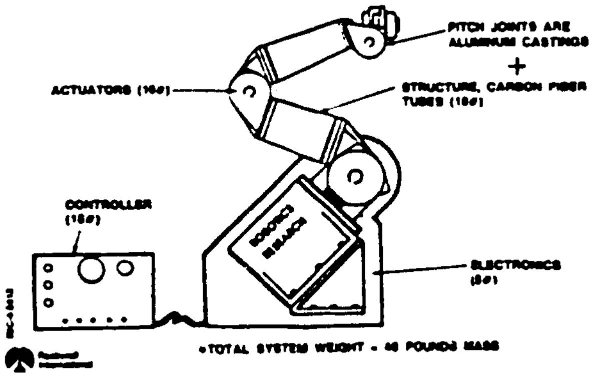


5. ROBOT DESIGN CONSIDERATIONS

By limiting the envelop to a particular subsystem on the station the robot manipulator can be designed as a rigid structure thus simplifying the control. A modular design would allow similar components to be arranged in different configurations to accommodate the variations in the tasks. Due to the lack of gravity the robot need only provide the force necessary to accelerate and decelerate the ORU. By programming very low accelerations and decelerations the axis motors and drives can be made small and compact. This would have the additional advantage of reducing the inertial reaction on the Space Station structure. The use of composites in the design of the manipulator links can provide a light weight and rigid manipulator which is capable of moving large masses. Also use of direct drive motors would reduce the weight. Figure 4 shows the design of redundant 7 axis robot with a 90 inch modular arm whose total weight with control is estimated to be 46 lbs.

FIGURE 4

ROBOTIC SYSTEM WEIGHTS



PITCH JOINTS ARE ALUMINUM CASTINGS
+
STRUCTURE, CARBON FIBER TUBES (10#)
ACTUATORS (16#)
CONTROLLER (15#)
ROBOTICS RESEARCH
ELECTRONICS (5#)
*TOTAL SYSTEM WEIGHT = 46 POUNDS MASS

6. ADVANTAGES TO DEDICATED ROBOTS

There are several advantages to the use of dedicated robotics over other methods discussed. Present plans call for the use of two teleoperated devices, the Mobile Service Center (MSC) and the Flight Telerobotic Servicer (FTS), to perform all remote assembly, service, and maintenance tasks on the Space Station. This would create a problem if more than two tasks were required at the same time especially if the tasks were on opposite ends of the station. Dedicated robots could perform simultaneous tasks on various parts of the station in many cases with-out requiring the direct attention of the astronaut.

REDUCED ORU COUNT

Many critical systems on the Space Station will require double, triple or quadruple redundancy. By utilizing dedicated robots failed components can be replaced immediately rather than waiting for a planned service interval. This would alleviate the need to provide as high a redundancy level as predicted and thus reduce the ORU count and number of spares required.

PREVENTATIVE MAINTENANCE

By providing the local ability to do inspections and subsystems checks with the robot, it will be easier to determine the cause of the failure and to identify the failed component. Also by performing regular inspections with the robot, failures can be predicted and corrective action taken before a catastrophic failure occurs which could damage adjacent components.

DESIGN FOR SERVICE

As mentioned before, space based systems in the past required that the components be certified for the life of the mission. These former systems in comparison to the Space Station were relatively short lived and less complex. Since the mission life of the Space Station is 30 years, components are required to have a mean time between failure (MTBF) of from 10 to 30 years. In order to obtain these high MTBF's, significant development and manufacturing costs will be incurred. With the use of dedicated robots the ability to maintain and service the Space Station would be significantly enhanced. This would provide the opportunity to design the components for a shorter service life of from 1 to 5 years and thus avoid some of the initial development and fabrication costs associated with commissioning of the Space Station. In addition, as new technology is developed obsolete components could be easily replaced so that the Space Station remained at the highest state of the art obtainable.

7. REQUIREMENTS FOR USE OF DEDICATED ROBOTS.

In order to utilize robots consideration will have to be given to the design of the Space Station and its subsystems and components. These same considerations will also provide for ease of service and assembly by EVA and IVA.

ORU DESIGN

The Orbital Replacement Units should be of a modular design and provide for a common interface between the ORU and the manipulator end effector. A range of interface sizes should be provided to accommodate the different size ORU's. This interface should be designed with adequate lead-in so that a slight misalignment of the end effector would not cause jamming. An alignment target should be provided so that the vision system can locate and do final positioning for connecting to the ORU. In addition identifying markings should be provided adjacent to the target so that they may be verified by the optical character reader.

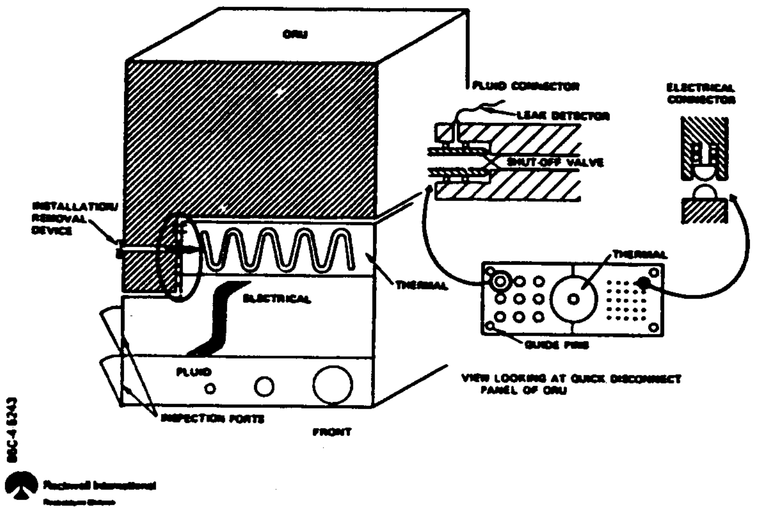
Quick disconnects should be provided for electrical, communication, fluid and thermal connections to the ORU. Fluid connections should contain a check valve shut off and a leak detection device with double seal arrangement to determine if the check valve has sealed. The robot can be programmed to pause for a leak check. if a seal has not been achieved the removal can be aborted.

A method to connect/lock and to unlock/eject the ORU should be provided so the manipulator is not required to push or pull on the ORU. This will prevent uncontrolled motions by the manipulator when the ORU is removed and provide the forces necessary to overcome the required contact pressures. The actuator for this mechanism would be contained in the end effector of the robot.

Many of the initial start-up and servicing problems on complicated systems such as the Space Station are associated with the connections to the individual components. An additional requirement should be to route the electrical cables, fluid lines and connectors in such a manner that they may be easily inspected and repaired remotely from the front panel. A concept of this ORU design is shown in figure 5.

FIGURE 5

ORU SERVICEABLE CONNECTIONS



CAD DATA BASE

In order to provide for offline programming and graphic simulation of the robot tasks an accurate CAD data base of the Space Station will be required. This will assure that the robot path will not interfere with other portions of the station and that the actual robot motions can be executed. In addition this data base will be invaluable in configuration control and redesign during growth of the Space Station.

DELIVERY OF ORUS

Replacement ORU's, components, and end effectors must be delivered from storage to the individual robot system. Failed components and unused end effectors must be returned for storage or repair or delivery to earth. In order to accomplish this an Automated Guided Vehicle (AGV) system would be provided as shown in figure 6. This system would consist of battery operated carts that would be guided by a rail attached to the station structure. The carts would receive control signals via the rail and be directed to the specific location requiring the replacement part. These carts would be loaded and unloaded by a Automatic Retrieval and Storage (ARAS) system as shown in figure 7. This would assure that the prop components were delivered in a timely manner and would also support simultaneous servicing : the subsystems. The AGV would also be useful in delivering equipment and tools to th teleoperated manipulators and astronauts during EVA.

FIGURE 6

AUTOMATIC GUIDED VEHICLE (AGV) FOR DELIVERY OF ORUS

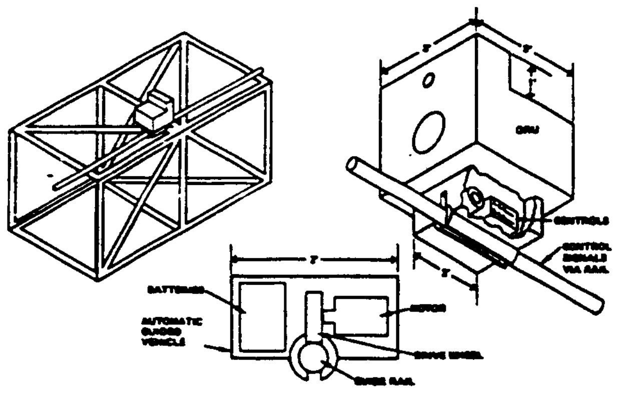
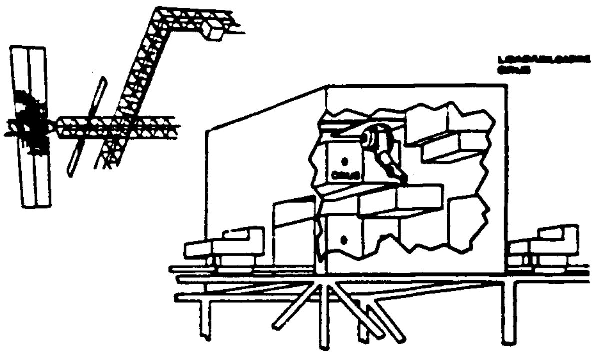


FIGURE 7

AUTOMATIC RETRIEVAL AND STORAGE SYSTEM FOR ORUS

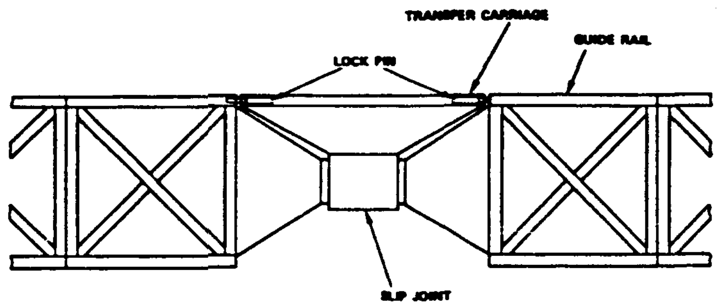


Crossing the alpha joint will be a major problem in servicing components on the power generation booms. One alternative is to stop the rotation of the alpha joint during the time the remote manipulator or AGV is crossing the joint. This, however, requires additional power to stop and start the joint. To overcome this problem a Transfer Carriage could be provided as shown in figure 8. When the AGV arrives at the alpha joint the carriage would be locked onto the Space Station structure. The AGV would move onto the Transfer Carriage. When the power boom rotates into position the Transfer Carriage would lock onto the power boom and disengage from the structure. The AGV would then move onto the power boom.

FIGURE 8

AUTOMATIC GUIDED VEHICLE (AGV)

TRANSFER CARRAGE FOR ALPHA JOINT



8. POTENTIAL APPLICATIONS

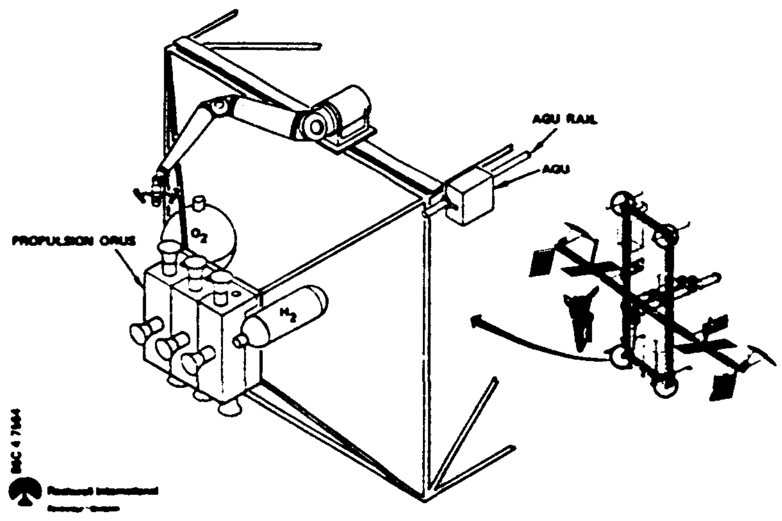
Various subsystems on the Space Station are candidates for dedicated robotics. Two typical applications would be the Laboratory module and the Propulsion unit.

PROPULSION

A typical concept for servicing components on exterior system of the station is shown in figure 9 for the propulsion system. In this case the robot could exchange ORU's consisting of three propulsion units. The robot would also perform regular inspections of the propellant lines and fittings to check for leaks. A modular end effector could also be developed which would fit around a leak in a section of tubing and repair the tube in-situ.

FIGURE 9

PROPULSION SYSTEM SERVICING

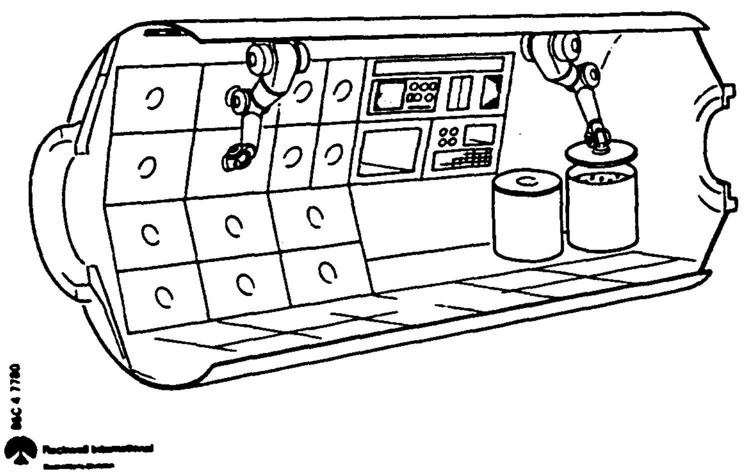


LABORATORY MODULE

As shown in figure 10, several dedicated robots can be provided in the laboratory module for servicing and operation of the experiments. This would reduce the cost to the customers by allowing them to automate their experiments without having to build it into their equipment. Instrumentation could be shared between experiments and customers thus lowering the cost. In addition, customers could be allowed to control their experiments from the ground by using the OLP/simulation facilities. Many experiments will be carried on in a vacuum environment. By servicing these experiments with a robot, the astronaut would not be required to suit-up.

FIGURE 10

DEDICATED ROBOT IN EXPERIMENTAL BAY



9. CONCLUSIONS

Modern factories today rely on multiple dedicated robotic devices to increase the productivity of their workers and remove them from the repetitive and boring manufacturing tasks. The Space Station can also benefit from applying this technology to servicing and maintenance. In addition with the proper thought to component design the possibility exists that the dedicated robot systems could aid in assembly. This would have the added advantage of allowing the assembly of various subsystems to proceed simultaneously and reduce the time to commission the Space Station. This does not suggest that the Space Station can be totally automated at this stage. There will always be tasks that require the direct intervention of the astronaut. However many well defined and repetitive tasks exist that would benefit from the application of a robot requiring a minimum amount of adaptive control. By applying existing technology as well as limiting there use to well defined tasks dedicated robots could be made available for IOC.

10. ACKNOWLEDGEMENTS

Much of the information presented in this paper has been gleaned through numerous briefings and meetings concerning automation and robotics for the Space Station. Thanks to Davoud Manouchehri of Rockwell's Space Station Division for information on teleoperation. Thanks also to Sarkis Barkhoudarian of Rocketdyne's Sensor group for his concepts on multisensor end effector. We would also like to acknowledge the studies conducted by RCA Corp. on ORU design under contract to Goddard Space Flight Center. Acknowledged is technical support of Jim Karlen and Jack Thompson of Robotics Reasearch Corporation in the design of the manipulator and Dr. Mark Hornick of Allen-Bradley Division of Rockwell International Corporation for his support in simulating the mechanics of the redundant manipulator. The offline programming and graphic simulation is being developed for Rocketdyne by the Rockwell Science Center under contract to Marshall Space Flight Center.

11. REFERENCES

[1] Corinne C. Ruokangas, Jim F. Martin, "Off-line Programming: A Robotic Shuttle Welding System," Robots 9, Detroit, Michigan, June 2-6, 1985.

[2] Margaret M. Clarke, Ph.D., "Recent Advances in Teleoperation: Implications for the Space Station," Space Tech, Anaheim, California, September 23-25, 1985.

188090 S8-16 88

Shuttle Bay Telerobotics Demonstration

W. Chun and P. Cogeos
Martin Marietta Aerospace
Denver, CO 80201

Mi 411300

1. Abstract

A demonstration of NASA's robotics capabilities should be a balanced agenda of servicing and assembly tasks combined with selected key technological experiments. The servicing tasks include refueling and module replacement. Refueling involves the mating of special fluid connectors while module replacement requires an array of robotic technologies such as special tools, the arm as a logistics tool, and the precision mating of ORUs to guides. The assembly task involves the construction of a space station node and truss structure. It will highlight the proposed node mechanism. In the process, the servicer will demonstrate a coordinated, dual arm capability.

The technological experiments will focus on a few important issues: the precision manipulation of the arms by a teleoperator, the additional use of several mono camera views in conjunction with the stereo system, the use of a general-purpose end effector versus a caddy of tools, and the dynamics involved with using a robot with a stabilizer.

Proposed industry space manipulators range in lengths from two to fifty feet. If the robot is a free-flier, the length of the arms is a function of its docking location in relationship to its task. As a result, no one set of manipulators can do every job. One solution to the problem is a "reconfigurable arm." This concept requires a modular approach of assorted arm lengths and different size drives including special modular sections. An integrated package of a hand, wrist, and forearm with actuators is an example of a modular section.

Interchangeable end effectors and tools is another facet of this concept. Each configuration is customized to the application. The demonstration will test some aspects of the "reconfigurable arm" during the technological experiments.

The robotic servicer will be mounted on a pallet as an experiment in the shuttle bay. It will be integrated with a minimum of two task panels complete with knobs, connectors, switches, and doors. The pallet will be self-sufficient and able to test all the aforementioned capabilities. A portion of the tasks will involve the Remote Manipulator System (RMS) in picking up the robotic servicer. The demonstration will answer several important questions concerning a robot doing extravehicular activity (EVA) and non-EVA types of work.

2. Introduction

Space is a natural environment for a test bed of advanced technologies such as robotics. The most frequently described missions include repair, housekeeping, or emergency work. Eventually, servicing and assembly missions will be common occurrences. However, the technology is far from mature with several critical questions yet to be answered such as "what are the major components of such a robot and will it be able to accomplish its mission?"

A mature servicing robot cannot be developed without some key intermediate steps. One of these is the subject of this paper, a shuttle bay experiment that merits some attention. This demonstration will be based and operated in the shuttle bay. The robotic servicer is an extension of the Johnson Space Center (JSC) work being conducted on their anthropomorphic-sized robotic servicer, the Telepresence Work Station (TWS) [1].

This paper describes the system, including the robot and pallet structure. The versatile system will be able to test various servicing and assembly scenarios, and will test key technological experiments.

3. Concept

Mounted on a pallet (Figure 1), the telerobotic demonstration experiment will be conducted from the shuttle bay in an actual space environment. The pallet will contain all elements of the experiment with the exception of the operator's console, mounted in the aft flight deck. The pallet includes a dual-arm robot, task panels, equipment rack, support electronics, tether and take-up reel, and robot latch mechanism and launch restraint.

The robot (Figure 2) will be configured with dual arms, each with seven degrees of freedom (DOF). Their basic design will be based on the Protoflight Manipulator Arm (PFMA) with an advanced wrist (3-DOF, concurrent axes) and additional improvements at the component and subsystem level. This design will lower the risk associated with successfully fielding an advanced manipulator in a useful time frame [2]. Each arm will be reconfigurable to allow changes in length, drive strength, and end effector. This capability will provide increased flexibility and opportunity to determine optimum configurations for particular tasks. Additionally, an evaluation of a general-purpose end effector, as opposed to an assortment of specialized tools, can be co::ducted.

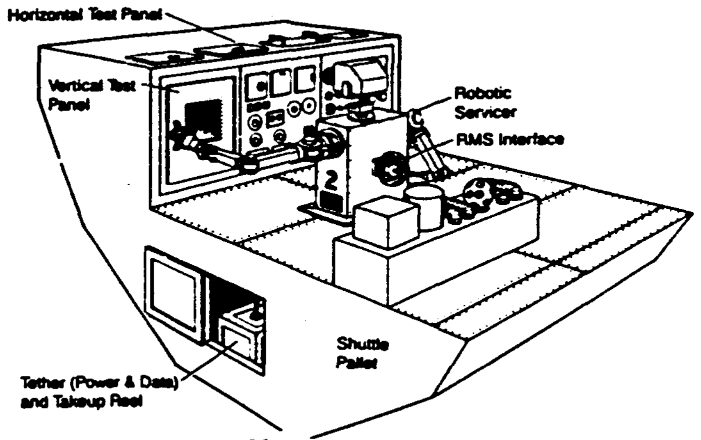


Figure 1. Telerobotic demonstrator on pallet

The robot vision system will include a stereo camera set housed on a 3-DOF mount, above and between the arm shoulder joints. In addition, a single camera will be mounted on each forearm to ensure a closeup view of the workspace if desired. How these mono cameras can be used in conjunction with the stereo system for enhanced task performance will be the focus of several experiments.

A tether and take-up reel will allow the robot to leave the pallet (Figure 3) and perform tasks in conjunction with the shuttle RMS without having to rely on a self-contained power source. This also allows the manipulator control elect:onics to be remotely located, further reducing weight and volume of the actual robot.

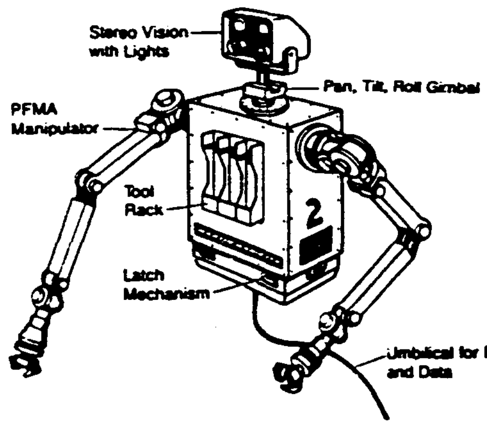


Figure 2. Robotic servicer

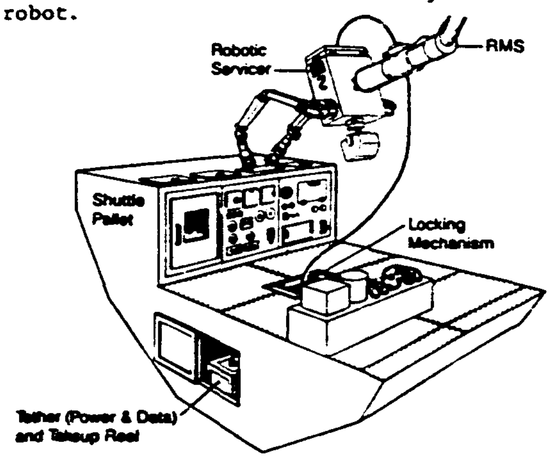


Figure 3. Robotic servicer on a tether

Integral to the pallet will be the task panels, re,resenting a wide range of activities. At least tw.) of these will be mounted askew to permit testing with the robot attacned both to the pallet, or the RMS. These panels can be tailored to almost any task the manipulator would oe required to perform. Their size would permit the simulation of portions of actual flight hardware, ensuring a realistic portrayal of a known servicing t. ., including obstacles to be encountered and avoided.

A nearby equipment rack will contain all elements of the desired assembly tasks. Having both within its reach envelope, the manipulator can use this rack of components in conjunction with the task panels, increasing the number and complexity of tasks that can be performed from a fixed location. The different tasks are systematically assessed for complexity before the demonstration by an index based on motion primitives [3].

4. Servicing

Servicing is a primary issue for any telerobotic system [4]. The failure of a fuse on the Solar Maximum Mission rendered the hardware useless until the unit was serviced. Some key servicing scenarios include module changeout (mating and demating connectors, fastener removal, precision alignment); inspection, checkout, and calibration; manually deploying an appendage; and cable take-up and untangling.

Module changeout is characteristic of any servicing mission. The Hubble Space Telescope has 12 primary orbital replaceable units (ORUs), such as batteries, electronic boxes, and fuse plugs [5]. When replacing an ORU, initial steps include the demating of several electrical connectors (signal and power). This requires a delicate and dexterous series of hand/wrist movements. The ORU can be removed once the fasteners are loosened. The subsequent reversal of this scenario replaces the module. Caution must be taken in precisely aligning the module during replacement. The module changeout can be demonstrated on the pallet system. Behind the door on the task panel would be a generic ORU. The robot would open the door and remove the ORU, then reattach the module to the equipment rack for storage. Conversely, the same ORU will be returned to the task panel. The experiment would be repeated with the robot fixed to its base and attached to the RMS.

Another servicing task involves the deployment of antenna booms with failed actuators. In this event, a jackscrew must be turned for manual deployment. Affixed to the task panel is a simulated boom the robot must deploy by driving the actuation mechanism.

In many situations, parts and tools may be tethered on a line that must be reeled in. A tangled line can cause many complications. In this instance, the tether must be handled meticulously so as not to worsen the situation. In such a case, the manipulator must demonstrate a high degree of dexterity. A tangled tether can be put in one of the sliding drawers of the task panel. The robot must untangle the line, wind it up, and return it to the drawer.

The last servicing scenario is refueling. The critical step in refueling is mating the male fluid connector to its female counterpart, necessitating handling a cumbersome hose. For this demonstration, the female connector would be in the task panel. The male connector is reeled out of the equipment rack and mated to the task panel.

5. Assembly Tasks

Candidate assembly tasks to be performed by the shuttle bay telerobotic demonstrator include those necessary for Space Station deployment and assembly of future orbital platforms. Teleoperated robots could execute boring and repetitive assemblies that would easily fatigue a Space Station EVA crew member. Primary among these is assembly of the truss structure forming the keel of the Space Station. Elements of one design candidate proposed by Lockheed Corporation would be carried on the equipment rack and assembled/disassembled by the manipulator [6]. Figure 4 illustrates this truss construction mechanism and how the two basic elements are assembled. Dual-coordinated arm motions are required as the mechanism is designed to be operated with the gloved hands of an astronaut using no tools. This task would be conducted with the robot secured to the pallet, negating the requirement for an additional stabilizing arm.

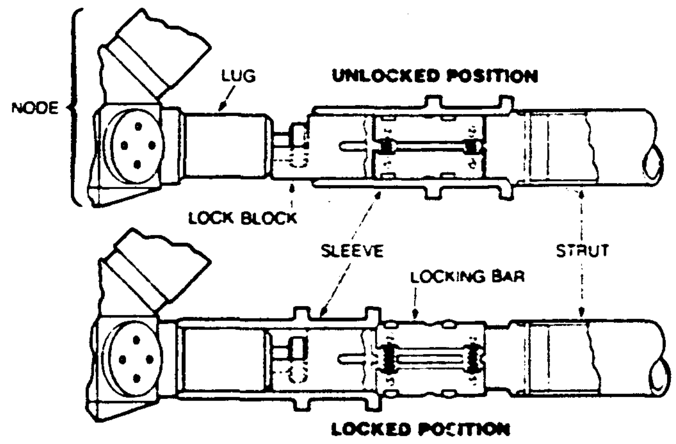


Figure 4. Truss construction mechanism

This experiment could also be repeated with the robot suspended from the RMS as a means of assessing the effectiveness of a single-arm stabilizer. In this position, one arm could act as the stabilizer, while the other performs the assembly of a truss element to a node rigidly attached to the equipment rack.

Additional assembly tasks include construction and installation of radiator panels, installation and connection of utility trays, and installation of payload support equipment.

Sample components from various tasks would be included in the equipment rack. Additionally, several configurations of the same task could be tested to determine the optimum design for a teleoperated environment.

6. Key Technological Experiments

In addition to assembly and servicing, there will be several key issues studied. One of these, the dynamics of a stabilizer, will be demonstrated by the assembly scenario on truss construction already mentioned.

By doing these real tasks, the teleoperator will have the opportunity to evaluate stereo viewing versus several mono views. As previously mentioned, the robot has a stereo vision "head" and supplemental mono cameras mounted on each arm. Optimizing the field of view for each of these is critical to ease task execution. Equally important is lighting. Both wide and spot beams will be evaluated. Understanding the limitations of the vision system is very important, and this experiment will surely point them out.

The robot will negotiate the standard complement of switches, knobs, and doors on the task panel. Also represented on the panel is a selection of fasteners and connectors, including the peg-in-the-hole experiment with the peg tethered. Figure 5 shows an example of a generic task panel. These tasks will test the manipulators' dexterity in space. We will also be able to investigate task sequencing as well as accomplishing a "time study." Each task is repeated with the robot both securely mounted to the pallet as well as being held by the RMS.

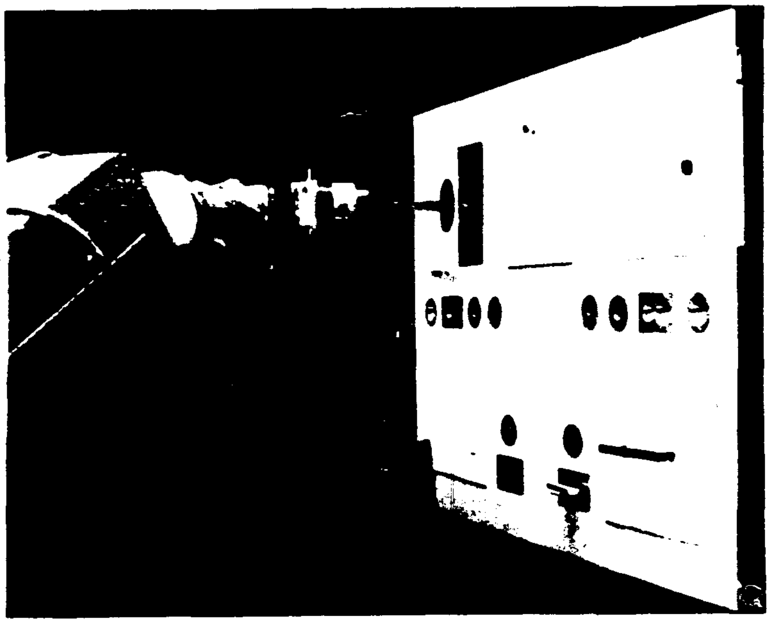


Figure 5. Example of a task panel

A major topic is the issue of modularity. Length is a major consideration in manipulator design. If the arm was anthropomorphic, it would be approximately four feet long, in contrast to the shuttle RMS, which is fifty feet long. With this large disparity, it can be proven that no single manipulator can do every task. The answer comes from the application. The length of the arm is dictated by its location with respect to the task location. For example, if the servicer is attached to a docking port ten feet from the task, then the arm should be at least ten feet long to be able to reach it.

The logical solution is to have more than one size arm. What we are proposing is a "family" of manipulator components, that when assembled, would lend itself to a variety of jobs (Figure 6).

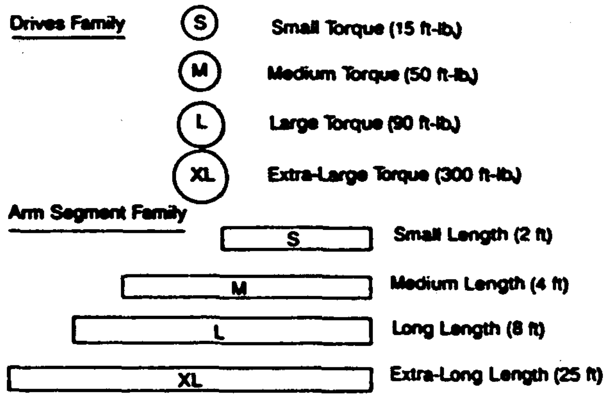


Figure 6. Family of drives and arm segments

By assembling these "families" of components, it is feasible to construct several different arms (Figure 7) to satisfy an assortment of missions. The result is some standardization of the manipulator with no major standardization of the tasks to be performed. As a consequence, the manipulators could be easily upgraded to incorporate new emerging technologies.

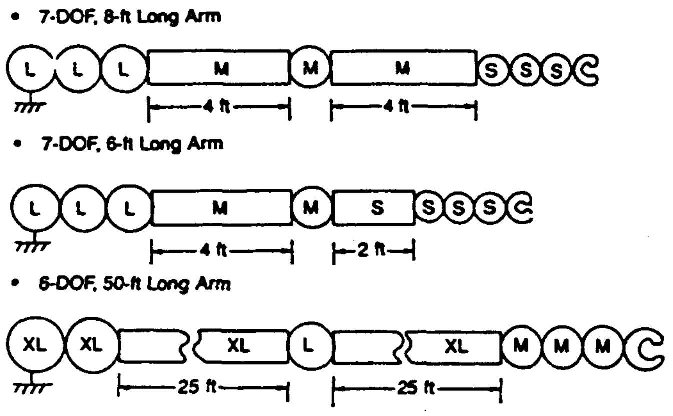


Figure 7. Several different manipulators

This concept is flexible, a compromise between distributed and totally integrated actuation; a hybrid we would like to call "reconfigurable" (Figure 8).

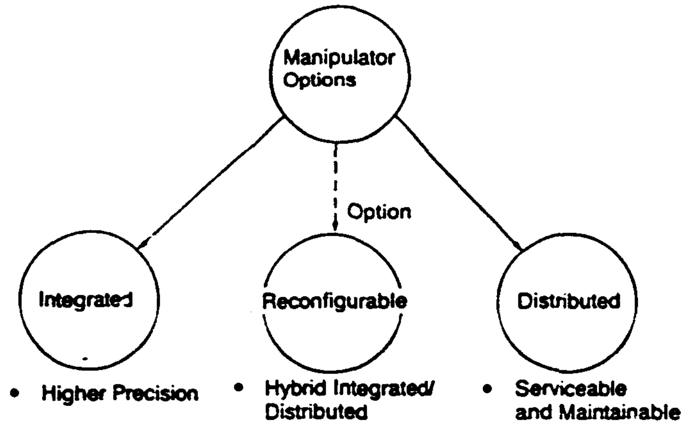


Figure 8. Hybrid manipulator option

The concept optimizes the manipulator by using both integrated and distributed modules. One example would be to use a Protoflight Manipulator Arm (PFMA) at Marshall Space Flight Center with selected upgrades. The existing arm, from the elbow back to the shoulder, has distributed actuators. The lower arm would be an integrated module consisting of the forearm/wrist/end effector. With the actuators built into the forearm, the wrist could be compact and dynamics improved. Figure 9 is an illustration of two possible configurations.

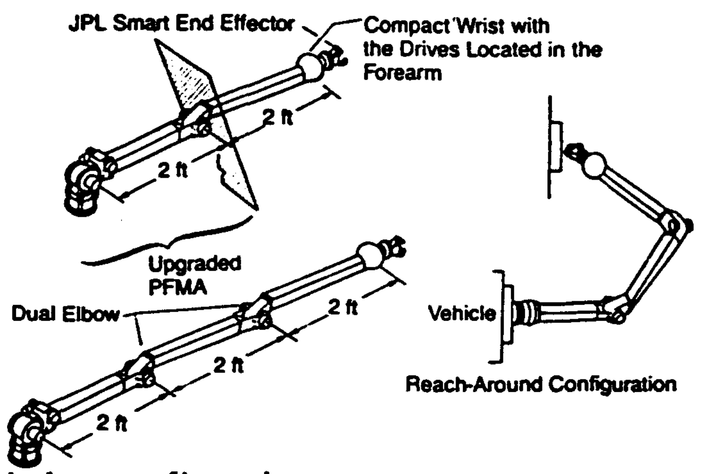

JPL Smart End Effector
Compact Wrist with the Drives Located in the Forearm
2 ft
2 ft
Upgraded PFMA
Dual Elbow
2 ft
2 ft
2 ft
Vehicle
Reach-Around Configuration

Figure 9. Two typical arm configurations

The success of the reconfigurable arm is dependent on an intelligent controller, and a mechanical attachment scheme that is easily connected or disconnected. The controller must be able to handle any arm configuration; from long to short and from simple to multijointed, and adapt to changing inertias, frictions, and other drive parameters.

Our concept will include one reconfigurable arm. The initial demonstration will include reconfiguring a 4-foot arm (two 2-foot segments) into an 8-foot arm (two 4-foot segments). The ability of the adaptive control system to compensate for this increase in length will be tested on the task panel. The attachment scheme should be light but rigid. It will be scaled for the different size interfaces. Next levels of component sizes are compatible (Figure 10).

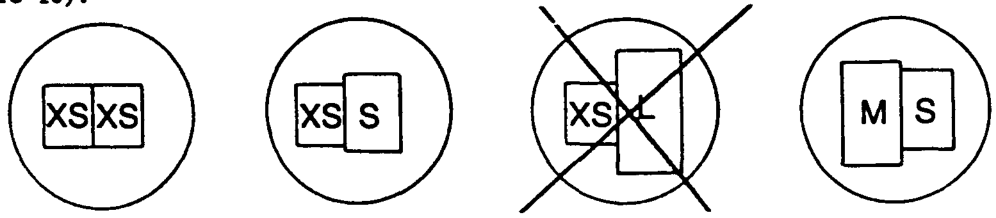


Figure 10. Compatibility of adjoining level component sizes

The modular theme also affects the end effector issue. The majority of servicing and assembly tasks require the use of some tool in conjunction with the human hand. The hand is the most flexible gripper or tool existing. The same is true for the robotic hand. At times, the hand will perform as a tool; for example, unscrewing a loose bolt without a wrench. However, it would not be efficient for our robot to have a tool holding another tool. Figure 11, from the TWS study, shows a mechanism that interfaces to a variety of tools such as a gripper or a ratchet. This mechanism has power and signal channels. There would be a locking interface to the different tools much like a bayonet mount, with a built-in power takeoff. This power takeoff is a single drive that actuates any tool that is compatible with its interface. As a result, the weight for a motor in each tool is eliminated.

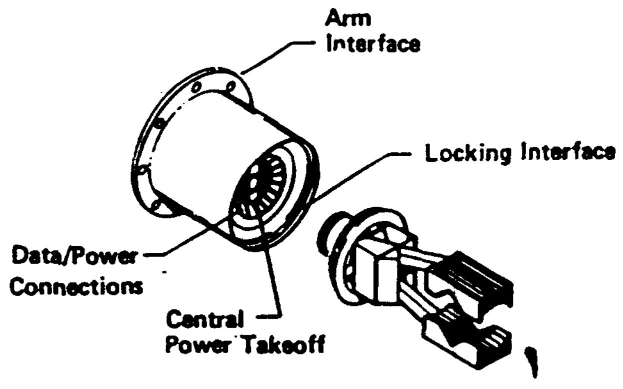


Figure 11. Power takeoff mechanism

Presented is a modular approach to the utilization of a variety of tools. The numerous tools will vary from a parallel gripper to a camera as shown in Figure 12. This flexibility allows this modular end effector to perform in more roles than a single tool, such as a screwdriver, could.

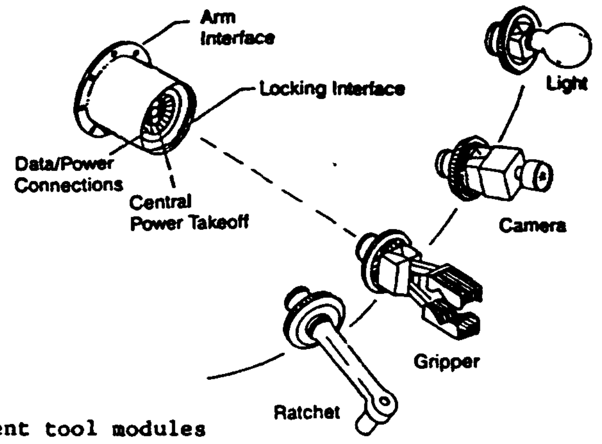


Figure 12. Different tool modules

At this time, it is not conceivable that an articulated hand can be packaged in the same manner as the previously mentioned tools. For this shuttle bay demonstration, there will be two lower arm assemblies. One assembly consists of the lower arm, compact wrist, and a dexterous hand. The other assembly is identical with the exception of the dextrous hand and the addition of a power takeoff mechanism. Both lower arm assemblies will be mounted on the equipment rack. The various tools to be exchanged will be attached to a holder on the same equipment rack.

7. Conclusion

Experiments, such as the shuttle bay telerobotics demonstrator, are necessary for the realization of a mature servicing robot. There are too many unknowns associated with assembly and servicing tasks, and as many as possible must be resolved to successfully transition from a manual environment to teleoperation and automation.

This experiment will provide an ideal environment where several key issues can be explored, including the following:

- Advanced control schemes designed to enhance telerobotic operation. Their higher level control modes can improve manipulator dexterity, and ability to adapt automatically to changing task environments. It is felt these control schemes may help ease the time-delay problems associated with in-orbit activities controlled from ground stations. Simulated random time delays would permit this evaluation and comparison to more conventional schemes such as bilateral force reflection.

- Vision and lighting systems, and how they affect the operators performance. How the mono camera information can be presented and used in conjunction with the stereo system to aid the operator while working in a cluttered or cramped environment. Visual acuity is extremely important when faced with complicated tasks and relatively little or no experience.

- Even with an adequate vision system, the ability to maneuver the manipulator arms around obstacles and perform dextrous tasks in cramped quarters must still be proven. The pallet demonstrator does not have to rely on the RMS to provide this type of challenge, as it is capable of performing many complex tasks with the manipulator fixed in its base.

Individually, many of these issues have been demonstrated or are being developed in a simulated environment. It now must be shown that they all can play together and perform in the place that counts: space.

The different elements of the experiment must be carefully chosen so adequate experience will be gained where most needed. Appropriate hooks and scars will be built in for future growth and to ensure a system with a long, useful life.

The shuttle bay experiment is not mission critical. It is well constrained on the pallet and presents minimum risk to other shuttle payloads and crew members. Its most significant impact will be on the knowledge and experience gained, by industry, in all disciplines of robotics.

8. References

[1] "Telepresence Work System Definition Study," Final Presentation, Martin Marietta Aerospace, October 1985.

[2] P. Brunson, W. Chun, and P. Cogeos, "Next-Generation Space Manipulator," Proceedings of the Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, November 13-14, 1986.

[3] J. Barnes, "A Task-Based Metric for Telerobotic Performance Assessment," to be presented at the NASA Workshop on Space Telerobotics, Pasadena, California, January 20-22, 1987.

[4] "Satellite Services Handbook--Interface Guidelines," LMSC/D931647, Lockheed, December 23, 1983.

[5] "Hubble Space Telescope--Orbital Maintenance," 6630-85, Marshall Space Flight Center.

[6] L. McCarthy (editor), "Space Station Structural Joint Is Simple, Positive." Design News, November 3, 1986, pp 96-97.

59-18
188091 46

Telerobotics: Research Needs for Evolving Space Stations

L. Stark
University of California, Berkeley
Berkeley, CA 94720

[illegible]

1. Introduction

The definition of telerobotics (TR) has not yet stabilized nor made the standard English language dictionary. I tend to use telerobotics as meaning remote control of robots by a human operator using supervisory and some direct control. Thus, this is an important area for the NASA evolving space station. By robot, I mean a manipulator/mobility device with visual or other senses. I do not name manipulators, as in many industrial automation set-ups, robots even if they can be flexibly programmed; rather calling these programmable manipulators. Our own laboratory at the University of California, Berkeley, has been involved in problems in display of information to the human operator, in problems of control of remote manipulators by the human operator, and in communication delays and band-width limitations as influencing both control and the display. A number of recent reviews have appeared with discussions of the history of telerobotics beginning with nuclear plants and undersea oil rigs.

2. Three Simultaneous Research Directions

I believe that we should engage in triplicate or three way planning. It is important to carry out our research to accomplish tasks (i), with man alone, if possible, such as in EVA (extra-vehicular activities), (ii) with autonomous robots (AR), and (iii) with telerobotics. By comparing and contrasting the research necessary to carry out these three approaches, we may clarify our present problems.

There are problems using man alone. The space environment is hazardous. It is very expensive to have a man in space; NASA must have quite adequate cost figures obtained from the demonstration projects that have already been accomplished with the shuttle program. We may also need a higher quality of performance than man alone can provide in terms of strength, resistance to fatigue, vigilance, and in meeting special problems. For example, if the space suit is not of constant volume under flexible changes of the limbs, then a great deal of strength is used up just in maintaining posture.

Problems with autonomous robots lie in our not having mastered the technology to build them and have them perform satisfactorily. They are not yet available! Indeed, designs are not yet fixed and it is not certain how feasible they will be, especially in terms of robustness and reliability.

Therefore, we can see that telerobotics is a viable leading edge technology. However, all three directions should be intensively pursued in research and development, especially for the next stages of the evolving space station planning.

FIGURE 1: TRIPLICATE PLANNING

PROBLEMS WITH MAN ALONE

- HAZARDOUS ENVIRONMENT:
  (SPACE SIMILAR TO NUCLEAR PLANTS, UNDERSEAS)
- EXPENSIVE (I.E. EVA IN SPACE)
- NEED INCREASED QUALITY IN
  - STRENGTH
  - FATIGUE RESISTANCE
  - VIGILANCE
  - PERFORMANCE

PROBLEMS WITH AUTONOMOUS ROBOTS

- NOT YET AVAILABLE
- DESIGN NOT FIXED
- FEASIBILITY NOT CERTAIN
- RELIABILITY NOT TESTED

THEREFORE: TR IS A VIABLE LEADING EDGE TECHNOLOGY

ALL THREE DIRECTIONS SHOULD BE SUPPORTED FOR EVOLVING SPACE STATION PLANNING, RESEARCH, AND DEVELOPMENT.

3. Space Station Tasks

One of the major roles that NASA can play is to hypothesize tasks for the evolving space station. In this way research regarding the design of telerobots to accomplish these tasks can be guided. For a list of seven groups of tasks see Figure 2.

As I will consider later, it is important to distinguish between those tasks unique to the NASA/evolving Space Station and those with "industrial drivers" that will accomplish development of new technologies in hopefully a superior fashion and thus enable conservation of limited NASA resources.

4. Problems in Telerobotics

This next section of my talk, reviews of problems in telerobotics, will be abbreviated. The review is divided into problems in telerobotics concerning displays, vision and other senses (Figure 3) and problems in telerobotics dealing with control and communication (Figure 4).

In each section, I start with basic properties of the human operator and end up with planned capabilities of autonomous robots. In between, I try to cover what knowledge exists now in our field of telerobotics. (See also companion paper by Stark et al in this volume.)

FIGURE 2:
NASA SHOULD HYPOTHESIZE TASKS FOR EVOLVING SPACE STATION

HOUSEKEEPING
- LIFE SUPPORT SYSTEMS
- INVENTORY CONTROL, ACCESS AND STORAGE
- RECORD KEEPING
- GARBAGE DISPOSAL

PROTECTION
- FROM SPACE GARBAGE
- FROM METEORITES
- FROM TRAFFIC FLOW

MAINTENANCE
- SATELLITE
- VEHICLES
- SPACE STATION ITSELF

CONSTRUCTION
- ADDITIONAL SPACE STATION STRUCTURES

MANUFACTURING
- CRYSTAL GROWTH, BIOPHARMACEUTICALS

MOBILITY
- AUTOMATIC PILOTING
- NAVIGATION
- PATH PLANNING

SCIENTIFIC
- LANDSAT TYPE IMAGE PROCESSING FOR AGRICULTURE
- METEOROLOGY
- ASTRONOMY
- HUMAN FACTORS RESEARCH
- SCIENTIFIC RECORD KEEPING

FIGURE 3: DISPLAY PROBLEMS FOR THE HUMAN OPERATOR

HUMAN VISION LLV, MLV, HLV

DISPLAY GRAPHICS (RASTER/VECTOR)
- ON-THE-SCREEN ENHANCEMENTS
- ON-THE-SCENE ENHANCEMENTS
- OTHER SENSES DISPLAYED:
- INPUTS TO OTHER SENSES

PERSPECTIVE AND STEREO DISPLAYS
- TASK PERFORMANCE CRITERIA

HELMET M[illegible]
- [illegible]NCE: SPACE CONSTANCY

HUMAN OPERATOR (H.O.) PERFORMANCE
- FATIGUE, EFFORT, VIGILANCE

ROBOTIC VISION*
- LLV - CHIPS
- MLV - BLOCKWORLD AND HIDDEN LINES
- HLV - ICM, AI

*Note: LLV is lower level vision, MLV, middle level vision, HLV higher level vision, including ICM, image compression by modeling and AI, artificial intelligence.

FIGURE 4: CONTROL AND COMMUNICATION PROBLEMS FOR THE HUMAN OPERATOR

BASIC PROPERTIES OF H.O., ESPECIALLY FOR EVA TASK PERFORMANCE
- NERVE, MUSCLE, AG/AT MODEL+
- SAMPLED-DATA (SD) AND ADAPTIVE CONTROL
- PREDICTION, PREVIEW, OPTIMAL CONTROL--KALMAN FILTER

H.O. CONTROL OF VEHICLES, MANUAL CONTROL

H.O. CONTROL OF TR

H.O. SPECIAL CONTROL:
- PREVIEW, DELAY, BILATERAL, HOMEOMORPHIC CONTROL

LOCOMOTION (HUMAN, ROBOTIC):
- NAVIGATION--PATHWAYS
- POTENTIAL FIELD ALGORITHMS

HLC (HIGH LEVEL CONTROL):
- SUPERVISORY CONTROL
- *MULTIPERSON COOPERATIVE CONTROL; RCCL; FUZZY SETS*

AUTONOMOUS ROBOTIC (AR) CONTROL
- SENSORY FEEDBACK, ADAPTIVE CONTROL, AI

+Note: AG/AT is an agonist/antagonist muscle pair, reciprocally innervated for fast movements and co-contracted for posture and impedance control.

5. Industrial Drivers For Certain Necessary Space Station Technologies.

This next section deals with the future, and especially with "industrial drivers" other than NASA for new technologies which may be required in the evolving Space Station. In Figure 5 I list nine components of a telerobotics system that certainly seem to be driven by important industrial hardware requirements, research and development. Therefore, it seems reasonable for NASA to sit back and wait for and evaluate these developments, saving its resources for those necessary technologies that will not be so driven.

FIGURE 5: DRIVERS OTHER THAN NASA FOR NINE NEEDED TECHNOLOGIES

ROBOTIC MANIPULATOR AND CONTROL SCHEME

- JOYSTICK - AIRCRAFT
- AR MANUFACTURING INDUSTRY, NUCLEAR INDUSTRY, MINING INDUSTRY, SENSORS: FORCE AND TOUCH; COMPLIANT CONTROL

ROV AND MOBILITY

- MILITARY, TANKS AND OTHER VEHICLE PLANS?
- UNDERSEA ROV - OIL AND COMMUNICATIONS INDUSTRY
- LOCOMOTION - UNIVERSITY RESEARCH
- SHIPPING INDUSTRY: SHIPS AT SEA (AR, TR, MAN)

TV CAMERA

- ENTERTAINMENT INDUSTRY - COMMERCIAL DEVICE
- SECURITY INDUSTRY
- NEED MOUNTS, CONTROLS AND MOTORS FOR PAN, TILT AND FOR STEREO VG

GRAPHICS

- ENTERTAINMENT INDUSTRY IS A BETTER DRIVER THAN COMPANIES BUILDING FLIGHT SIMULATORS;
- HMD AS AN EXAMPLE.
- EM SENSORS RESEARCH/HEAD-EYE MOUSE

ICM

- LANDSAT
- SECURITY
- MEDICAL INDUSTRY - CT AND MRI
- INDUSTRIAL PRODUCTION LINES
- TD - IMAGE UNDERSTANDING

COMPUTER

- COMPUTER INDUSTRY
- (HDW) AND (SFW)
- COMPUTER SCIENCE RESEARCH BASE IS NOW VERY BROAD

COMMUNICATION

- COMMUNICATION INDUSTRY IS HUGE
- SHIPS AT SEA
- BW COMPRESSION
- REMOTE OIL RIGS
- ARCTIC STATIONS

PLANS AND PROTOCOLS TO COMBAT H.O. FATIGUE AND TO PROMOTE H.O. VIGILANCE

- OFFICE AUTOMATION FORCES
- AIR TRAFFIC CONTROL NEEDS
- SECURITY INDUSTRY

COOPERATIVE CONTROL

- MILITARY - SUBMARINE CONTROL
- HELICOPTER FLIGHT CONTROL
- AIR TRAFFIC CONTROLLERS
- NUCLEAR INDUSTRY
- CHEMICAL PLANT INDUSTRY

Looking at these figures gives us some concept of how industrial development may provide various types of technologies for the evolving Space Station; indeed, NASA may be able to pick and choose from off-the-shelf items! For example, the most powerful computers on the last space shuttles were the hand-held portable computers that the astronauts brought aboard which contained much greater capability than the on-board computers; those had been frozen in their design ten years ago in the planning stages for the space shuttle.

6. Necessary Telerobotics Technologies to be Sparked by NASA

However, there are several areas in telerobotics that may likely not be driven independently of NASA, or where NASA may have an important role to play. Indeed, the Congress has specifically mandated that 10% of the Space Station budget should be used for Automation and Robotics development, and that this in some sense should spearhead industrial robotics in the United States (Figure 6).

FIGURE 6: AREAS SPARKED BY NASA NOT INDUSTRIALLY DRIVEN

VISUAL ENHANCEMENTS FOR GRAPHIC DISPLAY

TELEPRESENCE WITH STEREO HELMET MOUNTED DISPLAY (HMD)

MULTISENSORY INPUT PORTS:

- WORRY ABOUT H.O. OVERLOAD CONDITION (ESPECIALLY WITH COOPERATIVE CONTROL AND COMMUNICATION)

HIGHER LEVEL ROBOTIC VISION:

- EXAMPLE--IMAGE COMPRESSION BY MODELING (ICM) (TO REQUIRE LESS INFORMATION FLOW AND FASTER UPDATE)

SPECIAL CONTROL MODES FOR H.O.

- HOMEOMORPHIC CONTROL
- BILATERAL CONTROL
- TIME DELAY AND PREVIEW CONTROL FOR TIME DELAY
- COMPLIANT CONTROL

HIGHER LEVEL CONTROL LANGUAGES

- (SUCH AS RCCL; FUZZY CONTROL; PATH PLANNING BY POTENTIAL FIELD CONSTRUCTION)

REMOTE OPERATING VEHICLES (ROV) SPECIAL CONTROL PROBLEMS:

- NAVIGATION, ORIENTATION, OBSTACLE AVOIDANCE FOR ROV

COOPERATIVE CONTROL:

- COOPERATION AMONGST HUMANS, TELEROBOTS, AND AUTONOMOUS ROBOTS

COMPLIANT, FLEXIBLE, HOMEOMORPHIC MANIPULATORS

- GRASP VERSUS TOOL USING

HOMEOMORPHIC DUAL MODE CONTROL

IMPEDANCE CONTROL

7. University NASA Research

I now would like to make a plea that NASA should expand and stimulate telerobotics research conducted within the university environment. Of course, as a professor I may have a bias in this direction and I am willing to listen to contrary arguments! In addition to the benefits of the research accomplished by universities, NASA also gets the education and training of new engineering manpower specifically directed towards telerobotics, and focused on the evolving Space Station.

What kind of university and educational research should be funded in general by NASA. I believe there are two levels of cost (with however three directions) into which these educational research labs should be classified.

(i) First are Simulation Telerobotics Laboratories. Here we need graphics computers, perhaps joysticks, perhaps higher level supervisory control languages, cameras, image compression techniques and communication schemes. I would guess that our country needs at least thirty such systems for education and training. These systems should be very inexpensive, approximately $50,000 each. They need not even be paid for by NASA, since universities can provide such research simulation laboratories out of their educational budgets or from small individual research grants. Our Telerobotics Unit at Berkeley has been thus funded. A good deal of exploratory research can be carried out inexpensively in this manner.

(ii) Second, we need Telerobotic Laboratories with physical manipulators present as important research components. In this way, experiments with various robotic manipulators, especially those with special control characteristics such as flexibility, homeomorphic form, new developments in graspers, and variable impedance control modes, other than are found in standard industrial manipulators, would be possible. I guess that there are about five such laboratories in some stage of development at major universities in the country. I would further estimate that these laboratories could each use an initial development budget of $300,000 to enable them to purchase necessary hardware in addition to software as existent in the Simulated Telerobotics Laboratories.

Another set of costly laboratories would be Telerobotics Laboratories with remote operatoring vehicles (ROV). Here again, we need about five laboratories at universities with first class engineering schools. Again, I estimate about $300,000 each for the initial hardware support of these ROV labs. They could then study transfer vehicles, local Space Station vehicles, Moon/Mars Rovers, and even compare MMU vs. telerobotic controlled vehicles.

The university laboratories would contrast with and serve a different function than ongoing aerospace industrial laboratories, and NASA and other government laboratories. These latter assemble hardware for demonstration and feasibility studies. Then unfortunately they are somehow unable to carry out careful human factors research dealing with the changing design of such pieces of equipment. In the university setting, this apparatus could be taken apart, changed, revitalized, modified and the flexibility would inform our current capability. I would like to contrast the Gossamer Condor and Gossamer Albatross with the NASA program. It was clear that if McCready was ever to be successful, he had to build an experimental plane which was expected to break down each experimental day. But the plane could be repaired in a few minutes! This "laboratory bench" concept is so different from twenty-year-ahead-planning currently controlling our space program that has been effectively eliminated at NASA. I think it is important to reintroduce rough-and-ready field laboratories back into the space program.

8. NASA Prizes

Another role that NASA might play is to offer demonstration contracts or, even better, prizes for accomplishment of specific tasks. Again I turn to the Kremer Prize: here a private individual donated prize money to be awarded to the first to build a man-powered aircraft conforming to certain carefully laid out specifications.

Communication channels for controlling remote vehicles and remote manipulators are already set up. Thus we could have prize contestants demonstrating at differing locations on earth at one "g"; next demonstrations using elements capable of operating in space, or even more stringently, of having that minimum mass capable of being lifted into space; and then we might have true shuttle and space station demonstrations.

9. Intellectual problems in TR for the Space Station

Finally, I would like to leave you with the thought that the list of to-be-sparked-by-NASA problems in Figure 5 contains many important intellectual problems facing the area of telerobotics. Although these areas are being approached in our research community at the present time, it may not be possible to forsee what novel kinds of challenges will face the evolving Space Station in twenty years. Even though I may not predict accurately, I certainly hope I am there in person to watch telerobotics playing a major role in operating the Space Station. Acknowledgement: Support from JPL (956873), Dr. Bejczy; NASA-Ames (NCC-86), Dr. Ellis; and discussion with colleagues Drs. W. S. Kim, F. Tendick and B. Hannaford.

S10-37
188092
98

Robot Design for a Vacuum Environment

S. Belinski, W. Trento, R. Imani-Shikhabadi, and S. Hackwood
University of California, Santa Barbara
Santa Barbara, CA 93116

CD464001

1. Abstract

The cleanliness requirements for many processing and manufacturing tasks are becoming ever stricter, resulting in a greater interest in the vacuum environment. We discuss the importance of this special environment, and the developement of robots which are physically and functionally suited to vacuum processing tasks. Work is in progress at the Center for Robotic Systems in Microelectronics (CRSM[††]) to provide a robot for the manufacture of a revolutionary new gyroscope in high vacuum. The need for vacuum in this and other processes is discussed as well as the requirements for a vacuum-compatible robot. Finally, we present details on work done at the CRSM to modify an existing clean-room compatible robot for use at high vacuum.

2. Introduction

Among the many advantages of robots is their ability to work in harsh environments. Robots are being developed for maintenance of nuclear facilities (for example, the ODEX walking robots by Odetics, Inc) and for high temperature and other harsh environments. The high vacuum environment is now becoming more important in many high technology manufacturing tasks, and as a result the need for vacuum compatible robots is increasing.

Most processes requiring high cleanliness standards are now performed in clean rooms. The principal users of clean rooms are advanced industries making use of thin film technology. Materials manufactured in ultraclean environments include [1]: VLSI semiconductors, compact discs, photographic films, magnetic and video tapes, precision mechanisms and sterile drugs and antibiotics. Today's rigorous cleanliness requirements can be illustrated dramatically with the example of the actual development going on in the VLSI semiconductor field. Table 1 shows that the critical particle diameter, i.e. the maximum size of tolerable contaminant particles, is projected to be 0.05 μm in the near future.

Storage density (kbit/chip)	Line spacing on wafer (μm)	Minimum critical particle diameter (μm)	Market dominance period
16	4.0	0.4	1981 - 1984
64	2.5	0.3	1984 - 1988
256	1.5	0.17	1984 - 1988
1×10^3	0.9	0.09	1988 - 1990+
4×10^3	0.5	0.05	1988 - 1990+

Table 1. Line spacing and critical particle diameters for high density integrated circuits (adapted from [1]).

As of 1985, the most demanding air cleanliness level established in the U.S. Federal Standard 209b is a cleanliness class 100. This refers to a maximum concentration of 100/ft^3 for

* Research Assistant in the Department of Mechanical Engineering, UCSB
† Faculty in the Department of Electrical and Computer Engineering, UCSB
††The CRSM is one of the Engineering Research Centers established by the National Science Foundation

particles of diameter greater than or equal to 0.5 μm. It is proposed in [1] that the present standards be extrapolated as listed in table 2.

Cleanliness class	Particles per ft^3 equal to and greater than: 0.02 μm	0.1 μm	0.5 μm	5 μm
1	10^3	3 x 10	*	*
10	10^3	3 x 10^2	10	*
100	†	3 x 10^3	10^2	*
1000	†	†	10^3	7 x 10^0
10,000	†	†	10^4	7 x 10
100,000	†	†	10^5	7 x 10^2

* Indication not meaningful for statistical reasons.
† Indication not relevant for the definition of cleanliness requirements.

Table 2. Proposal for extrapolating the US Federal cleanliness standards (from [1]).

In the near future, cleanliness class 1 and even stricter environments will be required. The three main sources of particle contamination are: the outside air (10^7 - 10^8 particles > 0.5 μm m^{-3}), equipment, and humans, who give off about 100,000 dust particles > 0.3 μm and more than 1000 bacteria and spores per minute. An increasing number of specially designed robots are being used in clean rooms not only because of their potential efficiency and productivity, but also to replace one of the principal sources of contamination: human beings.

Eventually, a wide range of processing and analysis tasks will be performed in vacuum environments. The reasons for using this special environment are many and include the following [2]:

- To prevent physical or chemical reactions occurring between atmospheric gases and a desired process;
- To disturb an equilibrium condition that exists at room temperature so that absorbed gases or volatile liquids can be removed from the bulk of the material (e.g., degassing of oils and freeze drying), and adsorbed gases from the surface;
- To increase the distance that gas and vapor particles must travel before colliding with one another so that a process particle can reach a solid surface without making a collision (e.g., vacuum coating and the production of high-energy particles);
- To reduce both the number of molecular impacts per second and the contamination times of surfaces prepared in vacuo (e.g., clean surface studies and the preparation of thin films), and
- To reduce the concentration of a component gas below a critical level (e.g., the removal of oxygen, water vapor and hydrocarbons in tungsten filament valves).

Presently, epitaxial growth of semiconductor films takes place in the low vacuum range. Sputtering, plasma etching, plasma deposition, and low-pressure chemical vapor deposition are performed in the medium vacuum range. Pressures in the high vacuum range are required for most thin-film preparation, electron microscopy, mass spectroscopy, crystal growth, x-ray and electron beam lithography, molecular beam epitaxy, and the production of cathode ray and other vacuum tubes. [3] These environments are completely unsuitable for the presence of human beings. The special suits that are worn in outer space, for example, would be much less appropriate in a specialized vacuum chamber used for a critical processing task. The amount of time needed to produce the desired pressure level once the suited worker had entered the chamber would prove to be quite long. The need for robotics and automated systems inside the vacuum chamber may thus be more urgent than in a normal environment or clean room. A principal advantage would be the ability to manipulate objects within the chamber without opening the vessel and subsequently re-evacuating, thus avoiding a very time consuming process. The availability of vacuum-compatible robots will improve the efficiency of many processes now carried out in vacuum and should encourage the use of vacuum processing in new areas.

There are a number of differences between the vacuum environment of space and that produced artificially on earth. Pressures can be achieved in vacuum chambers which are comparable to those in standard space orbits. Pressure levels reach 10^{-6} and 10^{-9} Torr at 200 and 800 km above sea level, respectively. However, in vacuum chambers on earth, the outgassing from components in the chamber works directly against the pumping equipment. In space there is no problem maintaining the pressure, but it is known that the outgassing of the space vehicle causes an expanding gas cloud to surround it. The shape of this depends on the venting paths of the hardware, the outgassing of external surfaces and backscattering by the local atmosphere. Thus, outgassing is also a major concern in space. In addition, spacecraft are exposed to the full solar spectrum as opposed to only a fraction of the spectrum on earth. Possibly the most important difference between vacuum processing facilities on earth and those in space is the difference in the gravitational force. Some processing tasks may be better suited to the weightless environment of space.

3. APPLICATION: Gyroscope Assembly in High Vacuum

Delco Systems Operations in Santa Babara, CA has developed a revolutionary new gyroscope, which is now entering the production stage. For reasons described later, the assembly must take place in a high vacuum environment of 10^{-8} Torr. The CRSM is modifying an existing robot so that it can operate at high vacuum. It will be used by Delco for production of the new gyroscope.

Gyroscope Description

The gyroscope to be assembled is the hemispherical resonator gyro (HRG) which Delco has been developing since 1975. The HRG is not a laser based gyro, yet does not have the rotating parts usually found in mechanical gyroscopes. Edward Loper and David Lynch of Delco list the following attractive characteristics of this gyroscope[4]:

- Extremely low power dissipation, and hence negligible warm-up transient
- Passive mechanical integration of angular rate, but with whole angle readout, making it immune to electrical power interruption, and thus giving it high tolerance to nuclear radiation effects
- Ability to operate at very high angular rates without performance degradation
- Capability to operate over the military temperature range without temperature control.

The HRG consists of three principal parts constructed of fused quartz as shown in Figure 1.

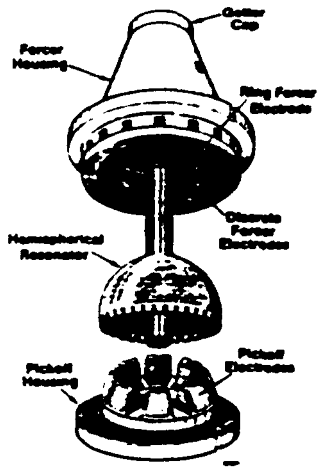


Figure 1. Principal Components of the HRG (from [4]).

These parts are bonded together with indium after being positioned relative to each other with accuracies in the sub-micron range. The principle that the gyro operates under was first described by G.H. Bryan, who did experiments on the vibrational modes of shells of revolution using wine glasses in the late 19th century. The basic operation of the HRG is represented in Figure 2. The "wine glass" in this case is the hemispherical part of the resonator, which is forced to vibrate at 2500 Hz, resulting in a standing wave with a zero-to-peak flexing amplitude of 4μm as indicated in Figure 2. As the entire gyro (all parts are fixed relative to each other) is rotated through an angle, the vibration pattern responds by precessing relative to the resonator through an angle proportional to the resonator's rotation angle. The location of the vibration pattern is sensed by electrodes located in the pickoff housing. By determining the movement of the vibration pattern relative to the case, and knowing the relationship between pattern movement and case rotation, the angle of rotation of the case is found. More technical details and test results may be found in [4], [5] and [6].

Need for a Vacuum Environment

The main components of the gyroscope are constructed of fused quartz, which is an inert, stable material having low thermal sensitivity and extremely low damping. This makes it an attractive material to use in precision mechanical and optical devices. It also exhibits extremely low internal damping, the damping time constant of the resonator being around 900 seconds. This means that very little energy is needed to sustain the vibration and that a power failure will not cause a loss of positional information until approximately 15 minutes have passed. The positive aspects of using fused quartz will deteriorate as the quartz becomes less pure. Even gases which attach themselves to the surface of the HRG components will cause variations from optimal gyroscope performance. The gyroscope thus must be manufactured in a high-vacuum environment and maintained at such throughout its life, a minimum of 20 years.

Gyroscope Production

In order to achieve a relatively high production rate for the gyro, Delco has proposed the use of a large vacuum chamber as shown in Figure 3. There will be a large central chamber surrounded by 12 smaller compartments, each of which may be opened to either the outside or the central chamber. The central chamber will be constantly maintained at high vacuum. Ideally, then, the raw gyro parts would be placed into compartment #1 and be removed sometime later as a completed gyro, with the intervening assembly tasks performed automatically. One alternative to this is a smaller vacuum chamber in which all steps of the assembly would be performed. This would result in an extremely slow assembly task due to the continual and time-consuming actions of venting and re-evacuating the chamber. In order to

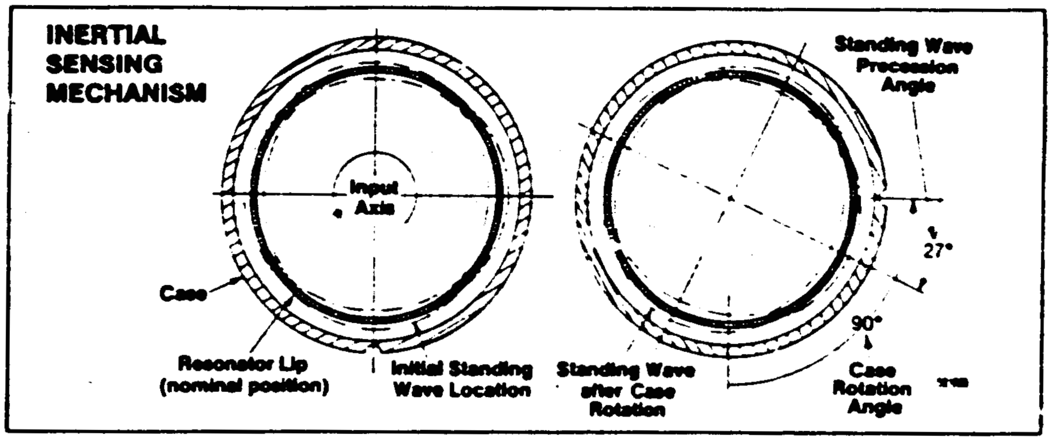

INERTIAL SENSING MECHANISM
Input Axis
Case
Resonator Lip (nominal position)
Initial Standing Wave Location
Standing Wave after Case Rotation
Standing Wave Precession Angle
27°
90°
Case Rotation Angle

Figure 2. Inertial Sensing Mechanism of the HRG

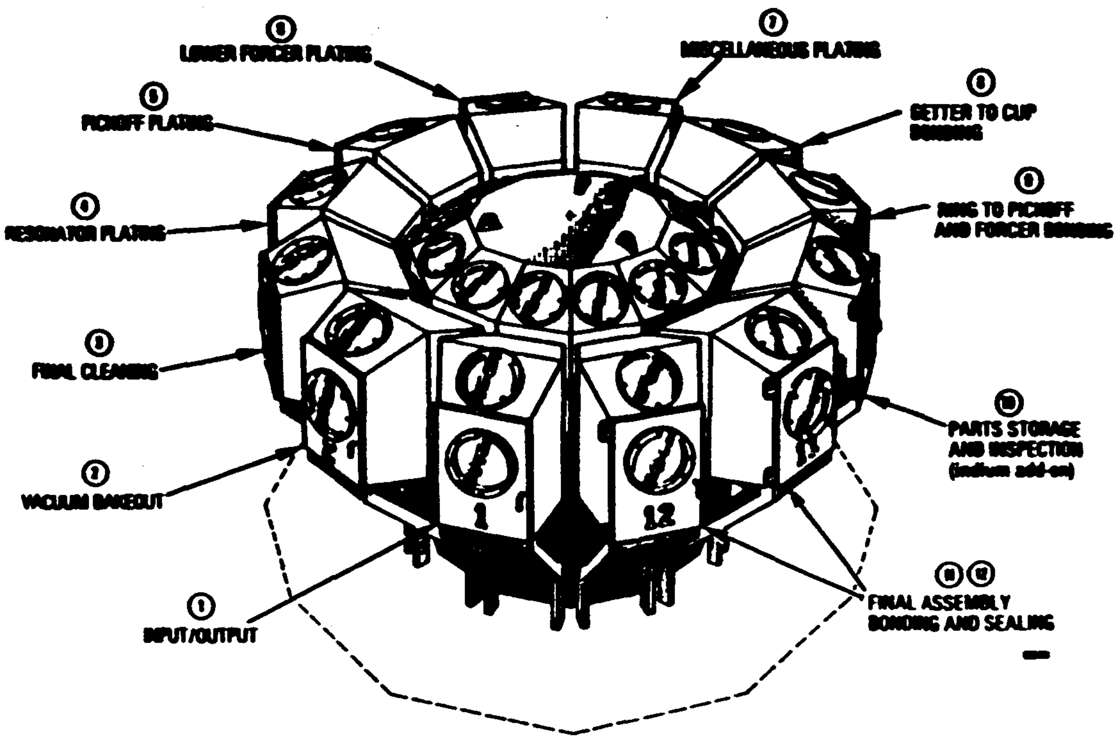


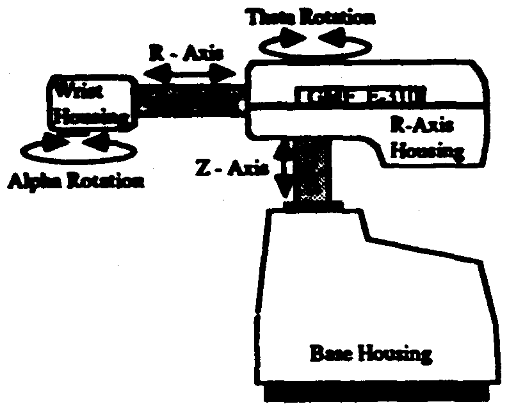
Figure 3. Vacuum Chamber concept for HRG Assembly

take advantage of the increased manufacturing efficiency the larger assembly chamber would bring, a means of transferring gyro parts between various stages of assembly is required. A robot positioned at the center of the assembly chamber is the ideal component.

The availability of vacuum-compatible robots is presently limited, although this is likely to change in the near future. A prototype vacuum-compatible robot has been developed by Yaskawa of Japan, but the small size of this robot eliminates it from consideration for use in the Delco system. As research progresses on the development of vacuum-compatible robots, Delco has decided to have the CRSM modify an existing robot for use in their assembly task. The motivating factors in this decision were *cost* and *time*. Although it is desirable to obtain a robot which was designed and built specifically for the vacuum environment, the first step is to obtain a vacuum-compatible robot. This will occur more rapidly by doing a modification to an existing robot. When a 'ground-up' vacuum-compatible robot becomes available it can then be compared to the modified robot.

4. Description of the GMF E-310 Clean-Room Robot

The robot undergoing modification at the CRSM is a GMF (General Motors - Fanuc) model E-310 cylindrical coordinate robot, originally designed for use in clean rooms to class 10. This robot was chosen for its size and configuration as well as its good accuracy and repeatability for a robot of its size. Tests at the CRSM have shown the repeatability to be ±10μm. The E-310 used for the gyroscope assembly task has four degrees of freedom (see Figure 4) consisting of two linear axes and two rotational axes. The robot has three main housings which are joined by the shafts of two linear axes. The base housing contains the motors for the Z-axis and the Theta-axis. The Z-axis motor, through a belt, drives a lead screw - linear bearing - linear guide assembly which causes the Z-axis shaft to move vertically. An electromagnetic brake is installed on the top end of the lead screw to prevent the robot from dropping when power is not available. The Theta-axis motor is linked to the robot base through an RV gear reducer and a pair of spur gears. A cross roller bearing is used to support the robot and allow for rotation. The R-axis housing sits atop the Z-axis shaft and contains a mechanical configuration very similar to that of the Z-axis. A motor mounted in the housing is coupled to the horizontal R-axis shaft through a lead screw, driving it back and forth. At the end of this shaft is the wrist housing, containing a fourth motor which drives the end-effector mounting plate (Alpha-axis) via a right angle gear set.



AXIS	MOTION RANGE	MOTION SPEED (max)
Z	0 ~ 300 mm	300 mm/s
θ	-150° ~ +150°	90°/s
R	0 ~ 500 mm	900 mm/s
α	-180° ~ +180°	90°/s

Figure 4. GMF E-310 Robot configuration and specifications.

The task for the researchers at the CRSM was then to modify the GMF E-310 so that it not only could operate in a vacuum environment to 10^{-8} Torr, but could also operate over long time periods without degrading its surroundings (outgassing, etc). The design requirements called for a modified robot with capabilities as close as possible to the original robot. The motion range of the two rotational axes would definitely be needed for the assembly task in the vacuum chamber. As for the linear axes, it was felt that their travel range could be restricted if necessary. The dimensions of the final vacuum assembly chamber had not been finalized at the beginning of the project, and could be adjusted based on the final specifications of the modified robot. It was expected that the final assembly chamber would not require the full 300mm of Z-Axis stroke. The stroke of the R-Axis would be related to the horizontal depth of the vacuum sub-chambers surrounding the main chamber. The robot must have enough horizontal stroke to reach into these chambers and perform specific functions. This is also dependent upon the final configuration of the end-effector, which might also extend into the chamber. It was agreed that an R-Axis stroke of at least 300mm would be desirable. An important point, however, is that this stroke must be *useful* when the robot is configured in the vacuum chamber. In other words, when the robot rotates inside the chamber, the wrist should just clear the inner wall. Then when the R-axis is extended, the wrist will move into the compartment with full stroke capability.

The completed robot must be capable of operating in temperatures as high as 100°C. Although some of the compartments will reach temperatures much higher than this, the central chamber will not. An initial bakeout may raise the temperature of the robot to two or three times its maximum operating temperature. This bakeout need not be accomplished in a matter of hours since the robot will remain in a vacuum atmosphere as long as equipment service intervals will allow. A degassing cycle as long as one to three days is thus acceptable.

In summary, the principal design requirements for the modification of the GMF E-310 robot for vacuum compatibility were:

- Modification of axes movement range:
  - Z-axis: maintain 300mm stroke if possible

- R-axis: maintain 500mm stroke if possible; if reduced, resulting stroke must be *useful* in the vacuum chamber
- Theta-axis: maintain ±150° rotation
- Alpha-axis: maintain ±150° rotation

- Limit negative effects on the vacuum environment (outgassing, etc)
- Design for ≤ 100°C operating environment
- Complete the modifications within one year.

5. Modifications for a Vacuum-Compatible Robot

The design of robots for vacuum brings together many disciplines, including the study of kinematics, dynamics, control, sensors, mechanics, materials, and tribology. Tribology, the study of friction, wear and the application of lubricants, is critical when applied to vacuum-compatible robots. Traditional lubricants have vapor pressures which are much too high for vacuum applications, resulting in rapid evaporation and unprotected surfaces. Dry lubricants, especially MoS_2, have been used in many high vacuum applications, however they tend to generate more debris and need replenishing more often than wet lubricants. Advances in both dry and wet lubricants are being made rapidly, helping to make this important part of the robot design less complex.

The first decision in the modification of the GMF E-310 was between two basic philosophies. The robot could either be totally exposed to the vacuum environment or it could be sealed in a type of "suit" which would allow the inside components to operate at atmospheric pressure, as they were originally designed to do. In order to expose the entire robot to a pressure of 10^{-8} Torr, a number of key changes would have to be made. The major ones would be in the lubrication systems, the surface finish and materials, and the motors. After examining this choice, it was concluded that it would entail a substantial amount of redesign work, and that a total exposure robot would be better designed from scratch. The goal then became one of designing a new housing for the robot which would seal it from the vacuum environment, while accomplishing the design goals as set forth in section 4. The sealing "suit" has to be as leaktight as the walls of a high-quality vacuum chamber, yet must also allow the desired movement by sealing two linear (R and Z) and two rotary (Theta and Alpha) motions.

Rotary Sealing

A differentially pumped 360° rotatable platform from Thermionics was chosen as the rotational sealing mechanism. As shown in the cross section of figure 5(a), the platform contains three spring loaded seals which are 80% teflon and 20% graphite. Two chambers are formed which are pumped to different levels of vacuum. For this application, the chamber closest to the atmospheric pressure side is roughed to approximately 10^{-3} Torr, while the chamber closest to the vacuum side is maintained below 10^{-8} Torr using an ion pump. Figure 6 indicates the placement of the rotatable platforms in the design.

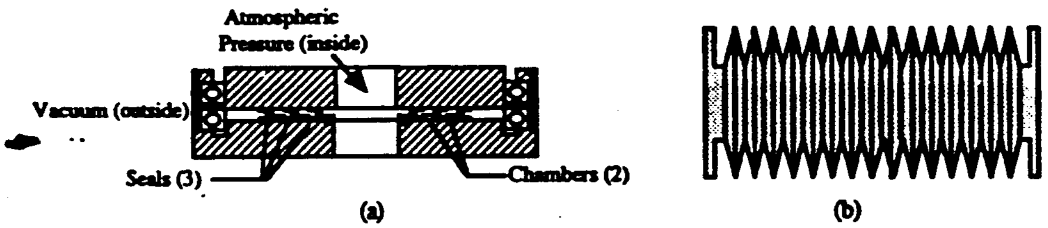


Figure 5. (a) Differentially pumped rotatable platform for sealing Theta and Alpha rotations, (b) Stainless steel bellow for sealing R and Z linear motions.

Linear Sealing

The two linear axes are sealed with vacuum compatible stainless steel bellows (figure 5b). Figure 6 shows how they are configured on the modified robot. Note that an additional bellow has been added to the rear of the R-axis housing. Because the robot internal pressure is essentially 760 Torr above its external environment, large forces exist which tend to push the R-axis forward and the Z-axis upward. For the bellow sizes which were chosen, the upward force due to the pressure differential is 477 lbs and the outward force on the R-axis would be 232 lbs if no compensation were used. The rear bellow shown in figure 6 surrounds a pipe which is attached to the same linear bearing as the R-axis. Thus, a force is produced which directly counteracts the one tending to force the R-axis outward. As for the Z-axis, the weight of the upper components tends to counteract the upward force sufficiently.

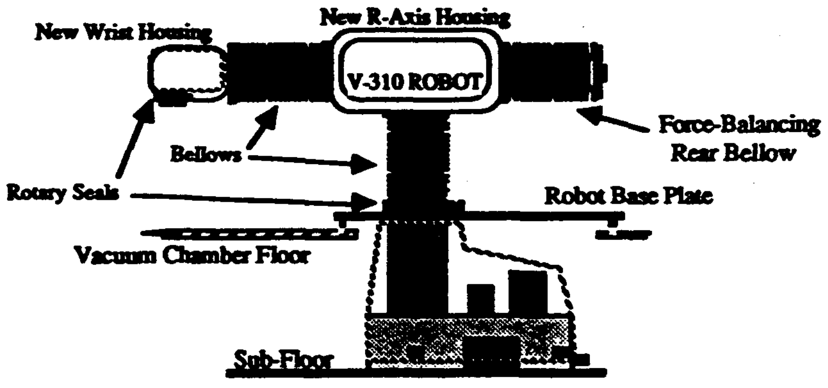


Figure 6. Modified E-310 Robot

Component modifications

The main housings, for the wrist and the R-axis, are replaced with new housings which are designed to be vacuum-compatible and interface with the bellows and the rotary seals. A special feedthrough is designed for interfacing through the wrist to the end-effector. Otherwise, all internal components are left unchanged from the original E-310 robot.

6. Future Work

The vacuum compatible version of the GMF E-310 as modified by the CRSM is scheduled for completion in Spring, 1987. Before then, work will be done to develop appropriate vacuum-compatible end-effectors for use in high vacuum environments. A cooperative venture with Yaskawa of Japan will involve basic research into vacuum robotics and applications. Yaskawa has developed *axial gap pulse motors* especially for use in vacuum. They make use of magnetic stainless steel for the motor body, have a special coating on the coil, and have been specifically designed to prevent heat build up. Research is continuing into *non-contact* drive mechanisms which would allow the elimination of lubricants.

7. Conclusions

The modification of an existing robot for a specific task at high vacuum has been described. Vacuum processing will become increasingly important as the need for ultraclean manufacturing grows. Vacuum-compatible robots will be necessary workers in this harsh environment. The CRSM is developing robot and end-effector technologies to be used for vacuum processing.

8. Acknowledgements

The work described in this document was performed at the Center for Robotic Systems in Microelectronics at the University of California, Santa Barbara, with funding from Delco Systems Operations. We acknowledge the technical support of Edward Loper and Arthur Voros of Delco Systems Operations and Welcome Bender and Joe Nassiri of the CRSM.

9. References

[1] H.H. Schicht, "Clean room technology: the concept of total environmental control for advanced industries", Vacuum, Vol. 35, No. 10/11, pp. 485-491, 1985.

[2] J. Benyon, "What do we mean by Pressure?", Vacuum, Vol. 20, No. 10, pp. 443-444, 1970.

[3] John H. O'Hanlon, "A User's Guide to Vacuum Technology", John Wiley & Sons, Inc., 1980.

[4] E.J. Loper and D.D. Lynch, "The HRG: A new low-noise inertial rotation sensor", 16th Joint Services Data Exchange for Inertial Systems, Los Angeles, Nov. 16, 1982.

[5] E.J. Loper and D.D. Lynch, "Projected System Performance Based on Recent HRG Test Results", Avionics Systems Conference, Oct. 31 - Nov. 3, 1983, pp. 18.1.1 - 18.1.6.

[6] E.J. Loper and D.D. Lynch, "Hemispherical Resonator Gyro: Status Reports and Test Results", National Technical Meeting of the Institute of Navigation, Jan. 17-19, 1984, San Diego, CA.

188093 511-37
7P.

Multi-Limbed Locomotion Systems for Space Construction and Maintenance

K.J. Waldron and C.A. Klein
Ohio State University
Columbus, Ohio 43210

DM 573208

1. Abstract

Multi-limbed locomotion systems operating in a "hand over hand" fashion are attractive for the construction and maintenance of large structures in space. They offer much greater energy efficiency as compared to thruster based mobility systems. They also free the designer of the structure from the necessity to incorporate tracks or other elements necessary for construction robots into his structural design. This type of locomotion system also offers great flexibility for handling unexpected situations such as structural failures.

A well developed technology of coordination of multi-limbed locomotory systems is now available. ~~This presentation will include~~ results from a NASA sponsored study of several years ago. This was a simulation study of a three-limbed locomotion/manipulation system. Each limb had six degrees of freedom and could be used either as a locomotory grasping hand-holds, or as a manipulator. The focus of the study was kinematic coordination algorithms.

The presentation will also include very recent results from the Adaptive Suspension Vehicle Project. The Adaptive Suspension Vehicle (ASV) is a legged locomotion system designed for terrestrial use which is capable of operating in completely unstructured terrain in either a teleoperated or operator-on-board mode. Future development may include autonomous operation. The ASV features a very advanced coordination and control system which could readily be adapted to operation in space. An inertial package with a vertical gyro, and rate gyros and accelerometers on three orthogonal axes provides body position information at high bandwidth. This is compared to the operator's commands, injected via a joystick to provide a commanded force system on the vehicle's body. This system is, in turn, decomposed by a coordination algorithm into force commands to those legs which are in contact with the ground. The individual leg controls are mode switched between a force control mode, when the foot is on the ground, and a position velocity mode when the foot is being returned. This form of control is attractive for space applications of multi-limbed systems, whether for locomotion or manipulation, since the weight appears only as one of the forces acting on the vehicle body and the coordination algorithms are set up to minimize generation of loads by limbs pushing against one another.

2. Introduction

Multi-limbed systems can be used for locomotion over space structures as well as for manipulation. A proven technology of artificial limbed locomotion is now available [1,2]. A configuration which is well adapted both to locomotion and to manipulation is shown in Figure 1 and has been studied in simulation [3,4].

Limbed locomotion has several advantages for use over space structures. First, a limbed system can be wholly electrically actuated using only energy which is renewable via solar panels. The actuation system can be configured for regeneration minimizing energy requirements. Further, limbed locomotion requires only discrete hand-holds and should require no modification of many structures. Finally, limbed systems provide great flexibility permitting adaptation to unexpected situations caused either by failures within the locomotion system itself, or by damage to the structure it is negotiating.

Technologies which provide capability for autonomous identification of hand-holds in real time [5] are now under test at several centers [1,6]. This raises the possibility that a remote operator need only designate a path which would be traversed autonomously, in much the same way as is planned for a Mars rover [7]. In this manner, direct teleoperation would only be necessary when performing manipulative tasks.

In this presentation the status of the technologies needed for realization of a limbed locomotion/manipulation system will be reviewed.

3. A Suggested Configuration

As is shown in Figure 1, the suggested configuration has three limbs. This will allow locomotion with two hands gripping hand-holds at all times. It will also allow use of two manipulation modes: a single armed mode in which two hands grip hand-holds leaving one free for manipulation, and a two armed cooperative mode with one hand gripping a hand-hold. It is expected that manipulation would be performed in a teleoperative mode. Locomotion would be performed in either a teleoperative mode, or autonomously with the remote operator designating path segments, as mentioned above. However, even in the teleoperative mode, the operator would command only direction and rate, individual limb movements would be fully automated.

The optimum geometry of a limb for locomotion [8] is quite compatible with that for manipulation [9]. In fact, if the first joint axis is placed parallel to the longitudinal axis of the body (the preferred locomotion direction), the second axis intersects the first and is normal to it, the third is parallel to the second and is placed exactly half way along the length of the limb, the fourth is also parallel to the second and third, the fifth intersects the fourth normally and the sixth intersects the fifth normally, as shown in Figure 2. The configuration is optimal for both purposes. However, a three limbed system lends itself better to a rotationally symmetric configuration which favors omni-directional motion without a preferred direction. In such a configuration it is preferable to place the first axis of each limb parallel to the axis of symmetry, as in the Odex, for example [2].

A four-legged system would have the attraction of allowing bilateral manipulation from a firmer base with two hands grasping handholds. It could also be configured as a bilaterally symmetric system with a preferred direction of motion, with some gains in locomotion speed and efficiency. This would, however, probably only be an advantage on very large structures. The cost would be the necessity of coordinating 24 degrees of freedom rather than 18, and the corresponding increase in system complexity.

4. Coordination

The primary problem in adaptive limbed locomotion is the automatic coordination of a relatively large number of actuated degrees of freedom to achieve a desired body motion. A great deal is now known about this problem as a result of the Adaptive Suspension Vehicle project [1]. In particular, the problem of using irregular, so-called "free gaits" to accommodate to sparse foot-holds or hand-holds is well understood. The architecture of a suitable free-gait algorithm is shown in Figure 3. With respect to gait, which is the phasing of limb movements, the situation in space is, in fact, much simpler than in terrestrial locomotion since there is no need to maintain stability against gravity. Thus, the sole determinant of the locomotion potential of a limb at a given instant is its kinematic margin [3]. This problem was examined in the simulation study described in references [3] and [4].

The Adaptive Suspension Vehicle [1], shown in Figure 4, uses an inertial local guidance system to detect body motion at high bandwidth. The actual body rate is compared to the operator's commands via a six axis joystick. The resulting error is converted into a commanded acceleration and, hence, a commanded inertia force system. This inertia force system is combined with the weight of the machine and decomposed via a force allocation algorithm into commanded leg forces [8,9]. Finally, these are decomposed via a Jacobian transformation into commanded actuator forces. The use of the legs as force generators in this fashion removes the necessity for sophisticated dynamic modelling of the legs.

The inertial sensing package is a relatively simple unit consisting of a vertical gyroscope, rate gyroscopes on three orthogonal axes, and accelerometers on the same axes. Drift and integration errors are corrected at low bandwidth by use of position information from joint position sensors on the leg joints.

Obviously, a substitute would have to be found for the use of gravity for a vertical reference in this scheme. However, the force control strategy is very applicable. In this manner the problem of limbs working against each other, which is characteristic of cooperative, multi-limbed operations, is avoided.

5. Sensing

Although it can be assumed that the geometry of the structure over which the machine will operate will be known, except for circumstances in which the structure is damaged, it is still necessary to provide on-board sensing to identify hand-holds and other features. This is necessary to correct location of the hand-hold for inertial system drift and other system errors. In fact, location of landmarks, perhaps combined with proprioceptive position sensor information, can be used to provide the necessary low bandwidth position data to update the inertial guidance system. As was mentioned above, technologies which will permit automatic, real-time identification of such features are being developed in the Adaptive Suspension Vehicle (ASV) [5] and Autonomous Land Vehicle (ALV) [6] programs. The scanning rangefinder described in reference [5] is being used to select footholds for the Adaptive Suspension Vehicle. It provides a range image in angle-angle coordinates scanning a 128 X 128 pixel field at two frames per second. In the ASV configuration the field scanned is 60° in the vertical plane and 80° in the horizontal. The ALV uses a scanning rangefinder with a slightly different field. Similar instruments are being used at several other centers.

The necessary technology for conversion of the range image into a continuously updated elevation map in Cartesian coordinates based on the vehicle, and for selection of footholds based on that map, has been developed for the ASV, and would be directly applicable to a limbed locomotion system in space. The scanner, in the ASV configuration, would be too heavy and bulky, and too expensive in power requirements for use in space construction. However, since this was the first operational unit, considerable development is possible. The mechanical scanning system could readily be redesigned into a much more compact, and much lower power system. The need for higher resolution at shorter range would allow use of a lower power laser.

Video based terrain modelling technologies are also being developed [6], particularly in the ALV program. At present, the emphasis is on identifying large features, such as roads. Further development of this approach might ultimately provide models at higher resolution than can be achieved with the laser scanner approach.

Another important feature of the sensing system is the need to sense both position and force at all actuated joints. In fact, joint rate sensing is easily added and is useful in limb control. Position, velocity, and force sensing are used on the ASV actuators [1]. Position/rate sensing is important for guidance of the limb when it is not gripping a hand- hold, and for inference of the position of the system from known hand-hold positions. Force sensing is needed for limbs gripping hand-holds, and for cooperative manipulation.

6. Controls

A number of features of the control system for the proposed machine have already been described. The proposed system has a hierarchical architecture shown in Figure 5. It is based on techniques proven in the ASV project. As is the case with that system, it is intended that the operator could interact with the system at several different levels [1,3]. At the highest level, the operator would designate path segments to be autonomously traversed, as suggested in Section 1. Figure 6 shows a composite photo of a three-legged robot walking over a series of path segments [4]. It might be noted that operation in this mode would require very little hardware or software added to that needed for continuous interaction. This is a result of the need for automated hand-hold selection and coordination for effective locomotion.

The next level of interaction would be one in which the operator continuously commands body velocity and angular velocity during locomotion via a joystick controller similar to that used on the ASV [1]. This is expected to be an effective control mode for relatively short traverses. Note that hand-hold selection and limb coordination would be fully automated in this mode.

At the next lower level of interaction, the operator would directly control the position, relative to the machine body, of one, or two hands. NASA has investigated six-axis controllers suited to this function [12]. This mode would be used both in teleoperated manipulation and in locomotion when the operator finds it necessary to directly control hand-hold selection and limb and body motion. This might happen, for example, when repairing damage.

It should be noted that the proposed system has a great deal of inherent flexibility of operation and redundancy. Locomotion is possible using only two limbs. Loss of function of several actuators could be accommodated to with degradation in performance, but without total loss of function. The use of differing mixes of automated and teleoperated function allows adaptation to a large variety of situations.

7. Conclusion

A technology of coordinating multi-limbed systems for locomotion, cooperative manipulation, and for manipulation with multi-finger, multi-degree of freedom hands is now available. It is here suggested that a manipulation system with at least three dual purpose limbs can provide an attractive mobility capability. Key features of the technology for this purpose have been discussed in this paper.

8. Acknowledgement

Work discussed in this paper was supported by NASA Langley, grant number NAG1-30 and by the Defense Advanced Research Projects Agency, contract number DAAE07-84-K-R001.

9. References

1. Waldron, K.J., and McGhee, R.B., "The Adaptive Suspension Vehicle," IEEE Control Systems Magazine, Vol. 6, No. 6, December 1986, pp. 7-12.

2. Bartholet, T., "Odetics Makes Great Strides," Nuclear Engineering, Vol. 31, No. 381, pp. 43-44, April 1986.

3. Klein, C.A. and Patterson, M.R., "Computer Coordination of Limb Motion for Locomotion of a Multiple-Armed Robot for Space Assembly," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-12, No. 6, pp. 913-919, November/December 1982.

4. Yeh, S.C., "Locomotion of a Three-Legged Robot Over Structural Beams," M.S. Thesis, Department of Electrical Engineering, The Ohio State University, August 1981, Advisor: C.A. Klein.

5. Zuk, D., Pont, F., Franklin, R. and Dell'Eva, M., "A System for Autonomous Land Navigation," Proceedings of Active Systems Workshop, Naval Postgraduate School, Monterey, California, November 6, 1985.

6. Lawton, D.T., Levitt, T.S., McConnell, C., Glicksman, J., "Terrain Models for an Autonomous Land Vehicle," Proceedings of 1986 IEEE International Conference on Robotics and Automation, Vol. 3, pp. 2043-2051, San Francisco, April 7-10, 1986.

7. Klein, G., Waldron, K., and Cooper, B., "Current Status of Mission/System Design for a Mars Rover," Unmanned Systems, Vol. 5, No. 1, pp. 28-39, Summer 1986.

8. Waldron, K.J., Vohnout, V.J., Pery, A., and McGhee, R.B., "Configuration Design of the Adaptive Suspension Vehicle," International Journal of Robotics Research, Vol. 3, No. 2, pp. 37-48, Summer 1984.

9. Vijaykumar, R., Waldron, K.J., and Tsai, M.J., "Geometric Optimization of Serial Chain Manipulator Structures for Working Volume and Dexterity," International Journal of Robotics Research, Vol. 5, No. 2, Summer 1986, pp. 91-103.

10. McGhee, R.B. and Orin, D.E., "A Mathematical Programming Approach to Control of Joint Positions and Torques in Legged Locomotion Systems," Theory and Practice of Robots and Manipulators, Morecki, A., and Kedzior, K., Eds., New York: Elsevier, pp. 225-232, 1977.

11. Waldron, K.J., "Force and Motion Management in Legged Locomotion," IEEE Journal of Robotics and Automation, Vol. RA-2, No. 4, pp. 214-220, December 1986.

12. Bejczy, A.K. and Handlykken, M., "Generalization of Bilateral Force-Reflecting Control of Manipulators," Proceedings of Fourth Symposium on Theory and Practice of Robots and Manipulators, Zaborow, Poland, pp.242-255, September 8-12, 1981.

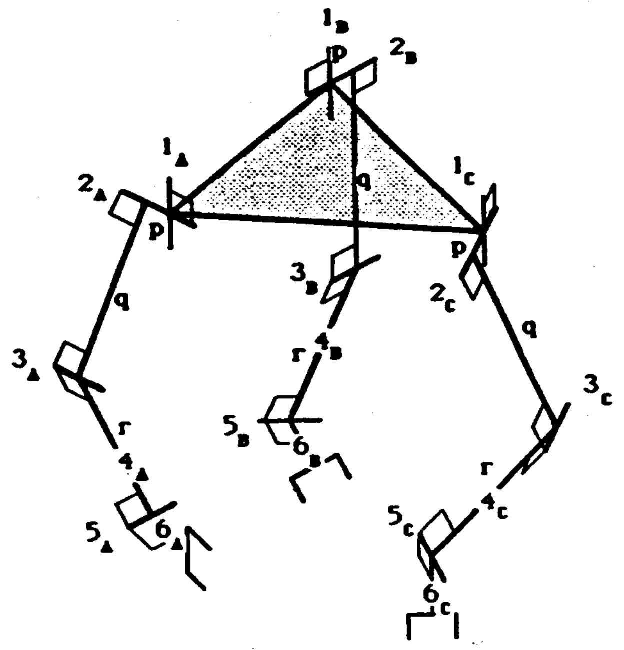


Figure 1. Proposed configuration of a limbed locomotion/manipulation system.

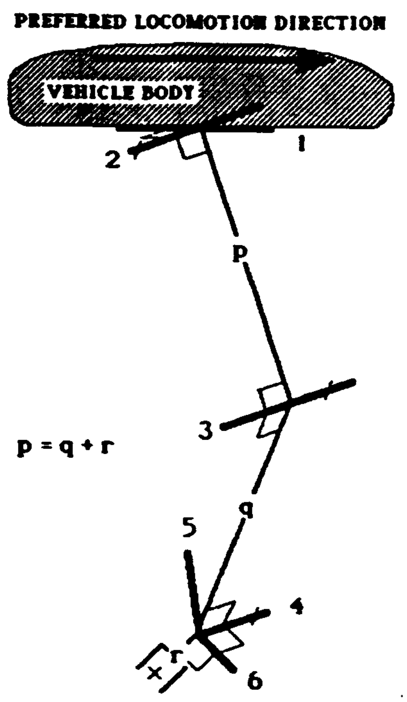


Figure 2. Optimal geometry for a dual purpose limb.

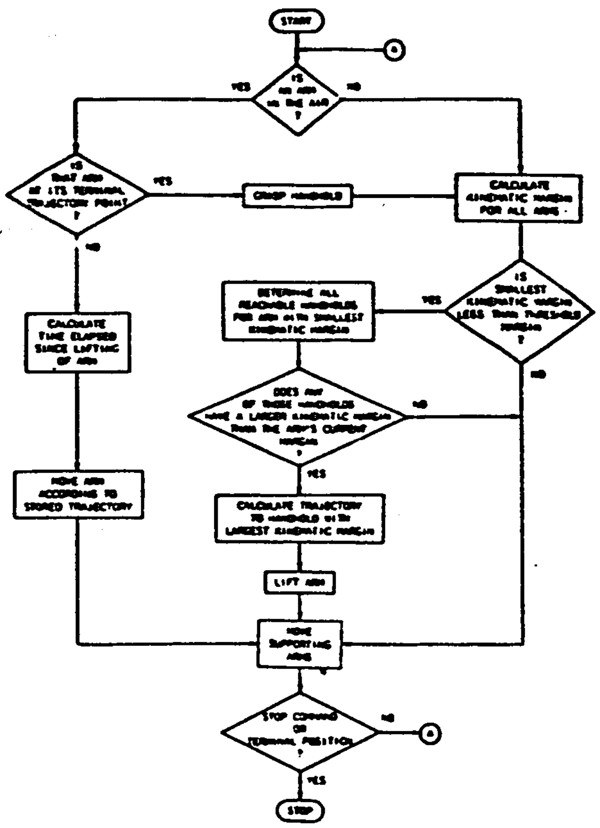

START
A
IS AN ARM IN THE AIR ?
YES
NO
IS THAT ARM AT ITS TERMINAL TRAJECTORY POINT ?
YES
DROP HANDHOLD
CALCULATE KINEMATIC MARGIN FOR ALL ARMS
NO
IS SMALLEST KINEMATIC MARGIN LESS THAN THRESHOLD MARGIN ?
YES
DETERMINE ALL REACHABLE HANDHOLDS FOR ARM WITH SMALLEST KINEMATIC MARGIN
CALCULATE TIME ELAPSED SINCE LIFTING OF ARM
NO
DOES ANY OF THESE HANDHOLDS HAVE A LARGER KINEMATIC MARGIN THAN THE ARM'S CURRENT MARGIN ?
NO
YES
MOVE ARM ACCORDING TO STORED TRAJECTORY
CALCULATE TRAJECTORY TO HANDHOLD WITH LARGEST KINEMATIC MARGIN
LIFT ARM
MOVE SUPPORTING ARMS
STOP COMMAND OR TERMINAL POSITION ?
NO
A
YES
STOP

Figure 3. Architecture of free gait algorithm [3].

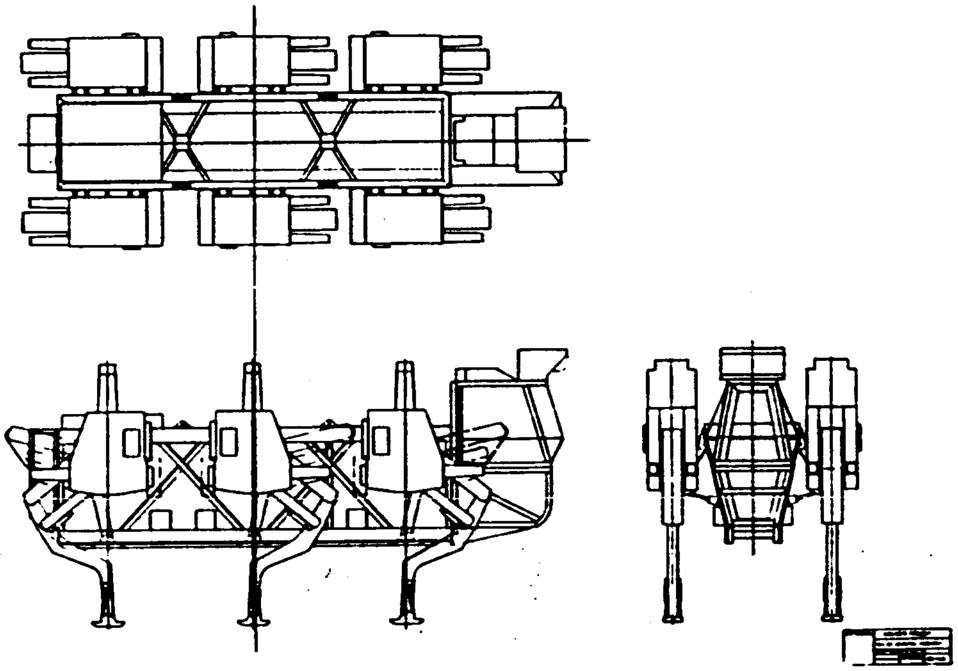


Figure 4. The Adaptive Suspension Vehicle.

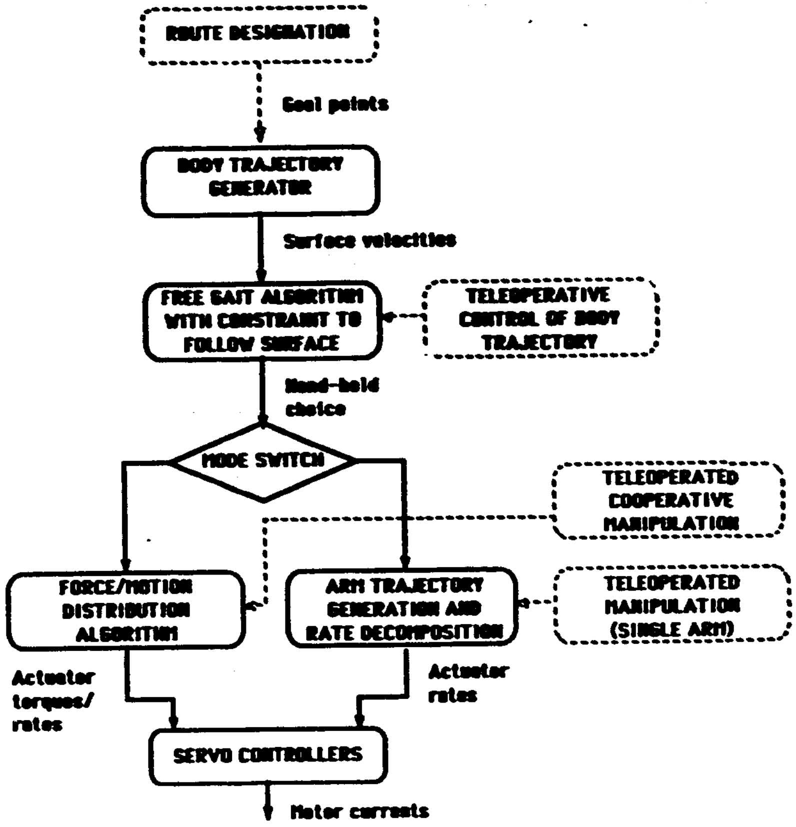

ROUTE DESIGNATION
Goal points
BODY TRAJECTORY GENERATOR
Surface velocities
FREE GAIT ALGORITHM WITH CONSTRAINT TO FOLLOW SURFACE
TELEOPERATIVE CONTROL OF BODY TRAJECTORY
Hand-held choice
MODE SWITCH
TELEOPERATED COOPERATIVE MANIPULATION
FORCE/MOTION DISTRIBUTION ALGORITHM
ARM TRAJECTORY GENERATION AND RATE DECOMPOSITION
TELEOPERATED MANIPULATION (SINGLE ARM)
Actuator torques/ rates
Actuator rates
SERVO CONTROLLERS
Motor currents

Figure 5. Control and coordination software architecture. The dotted boxes indicate operator interactions.

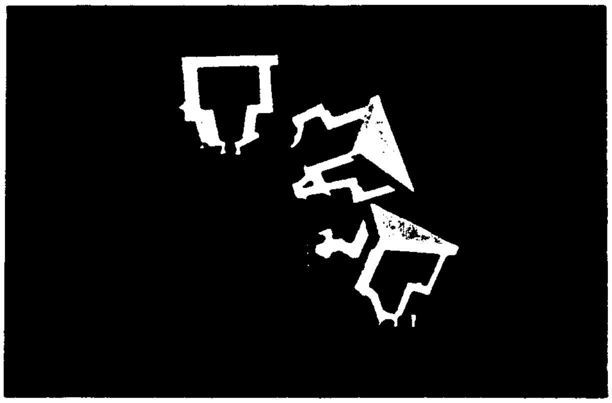


Figure 6. Photo of simulated space vehicle walking over a structural beam.

188094 5/2-85
118.

Commercial Users Panel

Ab [illegible]
[illegible]

INTRODUCTION

The National Aeronautics and Space Administration (NASA) Office of Commercial Programs (OCP) has responsibility for insuring the transfer of NASA-developed and -sponsored technology to the public and private sectors. The OCP program of actively pursuing new projects for terrestrial application of NASA technology is assisted by 1) a Technology Applications Team (TATeam) at the Research Triangle Institute (RTI) and 2) a network of Technology Utilization officers at NASA Field Centers.

NASA technology transfer has been fostered by an original mandate in the Space Act of 1958 which created NASA. This provides for "...the widest practical and appropriate dissemination of information concerning NASA activities and the results thereof." More recent directives include 1) the Stevenson-Wydler Technology Innovation Act of 1980 (PL 96-480), 2) Report 98-867 of the Committee of Conference to Accompany Bill HR 5713, which authorized funding for the Space Station, directing NASA automation and robotics to be "...identified and developed not only to increase the efficiency of the Station itself but also to enhance the nation's technical and scientific base leading to more productive industries here on earth." and 3) the Technology Transfer Act of 1986 which further promotes industry interaction with Federal laboratories.

The Jet Propulsion Laboratory's Space Telerobotics Workshop afforded an opportunity for the non-aerospace community to review NASA-related projects and planning in automation and robotics (A&R), an area of emphasis by OCP. OCP selected a Commercial Users Panel to meet at the workshop. This group, Table 1, represents organizations and industry sectors with the potential for expanding commercial telerobotics.

In contrast to the aerospace and academic makeup of most of the Workshop, the panel primarily represents the commercial sector outside the NASA family. While such industries as automotive or electronics have lesser motivation for man-in-loop approaches, they do represent about 70% of the market for robots in the U. S. It was also kept in mind that single component spinoffs (e.g., sensors, system architecture, manipulators) from NASA telerobotics-related research could provide commercial improvements in any sector. Such representatives as the U.S. Army's Human Factors Laboratory do not directly represent commercial interests, but their decisions and efforts will have important future impacts on automated systems, especially in mobility and manipulation. Such applications of NASA telerobotics as satellite servicing were not represented on the panel since these are so closely linked to NASA programs as to assure natural technology transfers.

TABLE 1. PANEL MEMBERS

D 4267791

Mr. Joseph S. Byrd
Savannah River Laboratories--
DuPont
Aiken, SC

Mr. Carl Flatau ~~AF~~ TM029029
Telerobotics, Inc.
Bohemia, NY

Dr. David C. Hodge
Human Factors Engineering
Laboratory
Aberdeen Proving Ground H6521544
Aberdeen, MD

Dr. Ralph Hollis 1A 056187
IBM Research
Yorktown Heights, NY

Mr. Eugene F. Leach CH685839
Caterpillar, Inc.
Peoria, IL

Mr. Ray Gilbert
NASA-Office of Commercial
Programs NCUS-981
Washington, DC

Dr. John Cleland
NASA Technology Applications Team
Research Triangle Park, NC
Electric Power Research Institute EK106 30[illegible]
Palo Alto, CA

S038[illegible]

Dr. Larry Leifer
Veterans Administration and
Stanford University
Stanford, CA

Dr. Joseph Naser EK 106302
Nuclear Power Division
Electric Power Research Institute
Palo Alto, CA

F0378316

Dr. Samson D. Schmuter
Manufacturing Development Center
Ford Motor Company
Detroit, MI

Dr. George Schnakenberg B 6751547
U.S. Bureau of Mines
Pittsburgh, PA

Mr. Kenneth F. Sebok ~~AO~~ PK907543
Perry Offshore
Riviera Beach, FL

Dr. Thomas Walters JJ574450
Technology Utilization
Jet Propulsion Laboratory
Pasadena, CA

✓
SSC

Discussion by the panel was focused on telerobotics, first because NASA has the opportunity to remain at the leading edge of this science through Space Shuttle and Space Station activities and through other NASA R&D programs. Second, telerobotics represents a limited subset of the overall NASA automation and robotics activity which could be reasonably discussed within a limited timeframe. NASA has decided to utilize telerobotics as a foundation for its A&R activities and to move toward more autonomous systems from this foundation. This approach is also strongly influencing Strategic Defense Initiative (SDI) planning, especially as related to unmanned maintenance of orbiting platforms and satellites.

Commercial User Panel members examined NASA-related telerobotics programs and technology through materials supplied before the Workshop, and through plenary sessions, conversations and other presentations at the Workshop. The panel then met in closed session to discuss how, why, and where NASA developments might be applied.

MOTIVES

The NASA Technology Utilization program is predicated upon a demand side approach, i.e., problems and requirements are requested from companies, reviewed through the Technology Utilization network, and matched where possible with appropriate NASA technology. The approach relieves industry of having to sort out the myriad technology developments over NASA's history. Similarly, the CU Panel was asked to identify needs of industry possibly related to NASA telerobotics developments. On the earth, several motives for teleoperation arise. These can reasonably be grouped into the categories of Safety, Security, and Productivity.

Safety: The primary concept in promoting safety by telerobotics is that of allowing a remotely located person to manipulate, inspect, or perform some other action in a hostile environment. The nuclear industry has long implemented master-slave type operations to allow handling of radioactive materials. Related to this, the Savannah River Laboratory is currently supporting GCA to build a large telerobot to handle contaminated equipment and put it into storage. Both Dr. Naser and Mr. Byrd discussed how telerobotics are becoming an option to improve the safety of 1) nuclear power plant operations and maintenance and 2) response to emergency situations. The essential precept is to minimize possible human exposure to radiation. In emergency situations, a telerobot can move into a hostile environment much more quickly than a person who must wait for radiation levels to subside or at least be specially clothed against radiation and high temperatures (ice water circulation suits). The telerobot may be unable to completely solve a contingency problem, but may perform many simpler preparatory tasks. This allows a person to follow up and complete the more complex parts of a repair. An example is telerobot removal of bolts from a valve flange. The valve is then aligned and replaced by a human who leaves immediately, allowing the remotely controlled robot to complete retightening and initiate leak testing.

Handling of other hazardous materials such as toxic chemicals with teleoperator manipulators has received less attention. Laboratory robotics are becoming more important in the chemical processing industry. Safety motivations for utilizing telerobotics in field operations (for example, the handling of hazardous wastes, from sampling through drum removal) have not resulted in a market large enough to attract equipment manufacturers. Mr. Leach indicated that Caterpillar had discussed such systems with companies responsible for hazardous waste management, but could not redesign equipment for less than a dozen units.

Mr. Sebok of Perry Offshore and Dr. Schnakenberg of USBM said that their organizations' respective ultimate objectives were to remove people from the water and from underground. Oceanographic explorations and off-shore oil rig maintenance have more recently emphasized the utilization of unmanned submersibles, a trend that is aimed finally toward near-autonomous systems. This trend and the advances in undersea machines have been in evidence with the recent explorations of the R.M.S. Titanic by the Woods Hole Oceanographic Insitute's submersible, Alvin, and its tethered telerobotic "eyeball", Jason Jr. (National Geographic, Vol. 170, No. 6, December 1986).

In mining, not all hazards are associated with underground operations, Dr. Schnakenburg stated. For example, there is interest in automating high wall surface mining where the miner drives into a wall with a coal seam exposed and goes as far as possible, e.g., out to 1000 feet. Inexpensive but effective teleoperation is needed to optimize this process.

Robotics have as yet made little inroads into the health and medical field, but considerations for staff safety, said Dr. Leifer, may move teleoperated units into isolation wards. Also, patients who must avoid exposure to any contamination (from other people or through frequent access and egress to their isolation rooms) could be remotely treated and supported.

Dr. Hodge expressed the Army's concern for promoting safety through A&R as one of "soldier survivability". Here telerobots might be used in activities ranging from removing fuses from unexploded armaments to initial engagement of an enemy.

Safety was also considered in an aspect other than human protection. In the case discussed, Dr. Hollis indicated that the one possible application of telerobotics as a system in electronics/semiconductor manufacture would be in clean rooms. Contamination-free environments are preferred and sometimes essential for semiconductor work. Telerobot use would also reduce the size of these expensive enclosures, allow more flexibility than wire guided mobile units, and allow prompt emergency responce.

<u>Security</u>: This category relates to safety but with an important distinction. i.e., rather than a requirement to remove people from hostile environments (safety), there is sometimes a need to maintain a person in the control loop to provide confidence (security) that adequate intelligence and experience is available. This is again perhaps most obvious in the nuclear utility industry. As stated by panelists Naser and Byrd, people are simply not going to be replaced by autonomous machines in nuclear power plants.

NASA considerations in applying telerobotics follow much the same lines. A teleoperated manipulator in orbit is probably handling very expensive equipment in a critical environment under conditions which can be only partially simulated on the ground. Here a "mistake" by a more autonomous machine could mean a failed mission or even a threat to personnel in orbit. At the current stage of robotics development, the level of confidence is greater for the telemanipulator with human backup.

Another area where a high level of confidence is essential is in the health and medical field. Mr. Flatau described a proposed endocorpuscular teleoperator for endo surgery and other emerging options of teleoperator microsurgery were discussed. Stanford and the University of Southern California are pursuing such research, the latter utilizing X-ray tomograms for manipulator orientation in brain surgery. Dr. Hollis also briefly described a project that IBM is pursuing with the University of California--Davis in robotic machining of bone for prosthetic knee or hip joint implantation. More precise routing should promote improved bone ingrowth. In all cases, the surgical physician must constantly observe these operations and maintain ultimate control. A consideration here related to telerobotics is how to provide a capability for failure or fault prediction and to allow near-instantaneous interruption by the human controller.

The panel noted that NASA progress in telerobotics is important to the "security" category of motivation because NASA demonstration of technology can offer confidence to those in the commercial sector who must first minimize risk in applying automated systems. NASA applications should also help identify those parameters of teleoperation entailing higher risk than others.

Productivity: Safety and security considerations have been the most important motivations for the industries (i.e., nuclear and undersea) which have led the way in telerobotics. However, these motivations may be closely related to productivity, and it was interesting to find that the panel emphasized increased productivity in considering telerobotics for commercial applications. This agrees with the needs for automation in space described in the original report by the NASA Advanced Technology Advisory Committee, "Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy" (NASA Technical Memorandum 87566). Here productivity through A&R has been broken out to include -- lowering of operating costs, increasing flexibility (communications, computers, modularity) to support innovation, improving reliability, achieving station autonomy, and performing tasks unsuited to humans alone. "Reducing hazards" constitutes a final category.

As a first example by the panel, John Hodge said that, beside soldier survivability, their lab concentrates on 1) force multiplication, and 2) reducing military operation costs. Related to the first area, the Army operates on no-projected-growth of manpower. High tech is being relied upon to increase productivity or control more weapon systems. Of 24 current related Army projects, six deal with weapon delivery (vehicles, mobility), four with reconnaissance (mobility, sensors), and fourteen with services and support (manipulation and mobility about half and half). He stated that his shop was supporting development of 1) a field material handling robot (FMR), an autonomous robot with a twenty-five foot reach and 4,000 pound payload for handling ammunition, 2) a soldier robot interface project (SRIP), which is a small teleoperated platform with a manipulator having a six foot reach and 150 pound payload, used for ordinance disposal and refueling, and 3) a single manned station for controlling two semi-autonomous vehicles simultaneously. This project is called TEAM. The Army is forecasting AI and robotics developments as they relate to their own projects and training. There is also work underway on a robot sentry vehicle which could patrol nuclear weapon stockpiles.

Teaching robots efficiently requires addressing many of the same human interface problems found in telerobotics. Mr. Schmuter related one aspect of Ford's automated manufacturing to telerobotics developments. To improve efficiency, his group is building a generic robot controller for computer-aided path generation. The idea is to reduce by an order of magnitude the number of points to teach for 3-D complex trajectories. Most pendant buttons and the typical recourse to a special robot language would be eliminated. After thirty minutes of instruction, Ford employees are now teaching 3-D paths in about 15 percent of the time it may have taken before.

Caterpillar is also looking for better man-machine interfacing, said Gene Leach. They will accrue the same advantages as space based teleoperators from improvements in levers, knobs, linkages and such control strategies as going from joint-specific to end-point control. Heavy equipment vendors, both for surface excavation and underground mining, are increasingly seeking "expert system"-type diagnostics for maintenance and fault prediction of machines. The panel experts for both mining and excavation believe their industries will keep the man on the machine for some time to come. The first priority of new systems is to make the job easier for the man, automating more repetitive cycles, within the context of cost effectiveness. John Hodge mentioned that the enthusiastic response to Army solicitations for field materials handlers indicated a market potential.

George Schnakenberg explained that the underground mining environment is one of consistently structured geometry and that automation is most effective in the "long wall" mining prevalent in Europe while the "room-and-pillar" approach common in US underground coal mines introduces more complexities. Also, metals mining is so batch-process oriented as to be non-conducive to automation. Nevertheless, with 82 percent of US recoverable energy reserves being represented by our coal resources, more productive mining should be a national goal, and productivity is an important consideration in USBM automation planning.

Thin seam mining utilizing TV cameras and teleoperators is under study, as well as automation of simpler repetitive tasks on continuous mining machines. Canada is attempting to develop automated drills and load haul dumps (2-8 yard scoops that roll on rubber tires) and Germany is attempting to automate installation and removal of roof supports.

Nuclear and other utility power plants could operate more efficiently with properly designed surveillance, testing and preventative maintenance teleoperators. Such remote controlled duties as checking valves and sensors, looking for leaks and testing for radiation content of any leaks should also be accompanied by some on-board intelligence. As with any teleoperation involving routine tasks, operator strain from constant observation is too high, e.g., long hours using a stereo monitor. Surveillance and scrabbling telerobots have received well-publicized testing at the Three-Mile Island nuclear plant and remote operated cavity cleaners are being used. The three major nuclear power utility vendors have used robots in the steam generator area. Both Germany and France have a brigade of emergency telerobotic vehicles on exhibit. However, Carl Flatau believes this technology lags US potential technological capability. He cited a unit developed mutually with a Belgian company which can climb stairs, clear obstacles 16 inches high, has autonomous capability (backtrack to operator) if radio contact is lost, and has programmable navigation including some obstacle avoidance.

For undersea operation, telerobotic system ruggedness and reliability is especially essential, first because of the difficult logistics of maintenance and secondly, because standby costs at off-shore oil rigs range from $10,000-100,000 per day. Mr. Sebok described their teleoperations (which can be applied down to depths of 20,000 feet or more) by two methods, 1) for simple tasks, multiple degree of freedom manipulators operating off subsea vehicles and 2) for complex tasks, specially designed tool and manipulator work packages as complicated as the vehicles themselves. For example, a package for a large oil company installs 650 pound insert valves with 30,000 pound torque castellated nuts. The package also removes valves, tests seals integrity, and replaces a 3,000 pound, seven cubic foot control pod. Inspection and non-destructive testing are often applied.

Dr. Leifer's work with the Veterans Administration Rehabilitation Center has been concerned with applying telerobotics to increase the productivity and self-sufficiency of disabled persons. He indicated that for younger handicapped persons, investment in a $100,000 tool to get back into the work force is obviously attractive. Also, he emphasized that physical therapy and other patient care is growing rapidly in the face of demands to push health care costs down. Acute (versus chronic) care is only 30 percent of all medical care. The therapy and chronic care market can be addressed by telemanipulation on the low-cost, low-intelligence end. A robot using force feedback has been developed by Athtec Corp. for human performance evaluation and will be marketed for health care. Otherwise, the panel was not aware of any company marketing a telerobotic product to the $85 billion plus medical industry.

As enlightening as the described potential advantages for telerobots were, IBM offered just as informative reasons for not using them. Dr. Hollis pointed out that the electronics/computer/semiconductor industry requires high speeds (approaching 100 [!] motions per second), low tolerance precision and accuracy (microns or sub-microns), typical assembly intergrating 35-45 parts with different tools, and operation in a fairly large workplace. An automated, sensory-feedback error recovery capacity is necessary to deal with assembly process exceptions without informing an operator any more than every thirty minutes. One operator should be able to tend several robots. Within the industry, robots assemble while humans take things apart for maintenance and repair. While the automation trend has been somewhat slower than expected, IBM exemplifies the industry picture with its automation commitment. This includes design and execution in 2- and 3-D assembly, geometric modeling, queing and scheduling systems, 2- and 3-D machine vision support, new actuation developments, and evolving computer architecture for real-time control.

Of necessity, a number of possible application areas for telerobotics were not represented by the panel. For example, not included were firefighting and agriculture. A survey before the workshop turned up intelligent-machines-in-agriculture efforts going on at the Universities of Georgia, Florida, Purdue, California-Davis, and at Weyerhauser and Batelle-Northwest. Studies covered soil analysis, planting seeds, harvesting, nursery maintenance and pruning.

SPECIFIC NEEDS

Beyond the general interests and more specific activities described under Motives, the panel indicated some items that might head a NASA telerobotics technology transfer shopping list. Not surprisingly, new and improved software received considerable attention-- but not always specifically related to telerobotics. Some examples of these requirements include:

1. Expert systems to diagnose equipment condition for monitoring and predicting failures, combined with machine maintenance during off-shifts.

2. A CAD/CAM type simulation system for mining applications.

3. Methods to reduce the coding required for handling exceptions in electronic assembly.

4. Intelligence built into teleoperated machines so that they know when they need help and can decide on proper response.

5. More expert systems work on the design and planning end rather than on the analytic or diagnostics end. For example, a system searching through a large set of design alternatives, for associating them and arriving at a device synthesis.

6. Scheduler/planners, an area where NASA excels.

7. Data capture codes, to prevent production process decision information from being lost in host computers rather than being readily accessible to a localized plant requirement.

8. Emulators, which incorporate human factors simulation, as with cockpit or flight control simulation. The implications for telerobotics training and simulation are obvious. Recent examples are a Honeywell system for Army tank operators and a Mercedes automotive trainer.

Joe Naser and Ray Gilbert discussed a software technology transfer project on which EPRI and NASA are collaborating. The transfer derives from an expert system developed for the Space Shuttle at the NASA Kennedy Space Center. The original system was for liquid oxygen handling, which then evolved to a knowledge-based automatic test equipment system (KATE) for system monitoring, signal validation, fault location and diagnosis, automatic control and reconfiguration. EPRI is taking advantage of this "off-the-shelf" product to further develop it and apply it in simulators and eventually in a nuclear power plant production system. Joe Byrd added that such software is needed by their industry but must be begun in simple applications and gradually assimilated into the system. He also commented that it is often easier to incorporate new hardware than software for nuclear power.

On the low end of teleoperator hardware, Dr. Leifer reiterated that the medical field will first best incorporate "dumb" systems which serve to extend a patient's control a little beyond his reach. However, voice control is a more sophisticated requirement often added to such systems.

Mining, nuclear and undersea experts stated that sensors and additional software adding some autonomy to telerobots are important. A basic example is to allow a mobile system to take itself from point A to B without operator input and then to request instructions. USBM needs guidance systems. Lasers are a possibility but will not work in undulating seams. Inertial systems, even using ring-laser gyros, probably lack sufficient accuracy over long cuts. USBM has been assisted by the NASA Lewis Research Center with IR and ultrasonic systems and are now examining vibrations in the mining machines and mine strata to determine whether the machine is in the coal seam. A sensor for tracking a coal seam would be an invaluable tool. Control research requirements begin with closed-loop control, then task planning for simple or reflexive machine control in a well-defined open area, and finally developemnt of strategies for mining more than one area.

An important point that was emphasized for nuclear plant automation is that there will be only retrofit systems for some time to come. There are no current new orders for nuclear power plants, although EPRI would like to design automation into planned Advanced Lightwater Reactor systems. For this reason the Savannah River plant is emphasizing testing of such equipment as the Odetics walking machine which may be able to wend its way through a nuclear power plant maze.

One interesting need is for disposable robots (expendable, low-cost) or disposable modules. Both the Army and Savannah River laboratories would like these futher developed. The Army is also interested in NASA developments of 1) computer-aided driving of remote vehicles, applying 3-D displays and point-to-point navigation updated every 30 seconds, 2) 2- and 3-D vision, 3) manipulators, and 4) low data rate communications.

Few high tech improvements will be made soon to most earth handling equipment, said Mr. Leach, although they have simple remote operation packages and have announced a new unmanned electric forklift truck for warehousing. There are no plans to move toward camera-based type teleoperator systems. He also noted that with large equipment it is important to provide feedback other than such force feedback as that felt through the back pressure of hydraulic wheels. An example is audio feedback, which equipment operators have lost because of the evolution of, first, mufflers and then turbochargers and electronically controlled transmissions which eliminate the ability to calibrate the shifter with engine noise.

Ken Sebok asserted, on the other hand, that undersea operations are very similar to those in space and their problems may have mutual solutions. Vehicles operate in an essentially weightless environment without a source of ambient air and with sealing against pressure differentials required. Ocean activities must also consider problems of corrosion, high external pressures, ocean currents, low visibility and light attenuation. NASA could possibly assist with 1) sensors for acquiring a landing site, 2) sensors for alignment (e.g. valve replacement), 3) sensor monitors, 4) manipulators and end effectors, 5) methods for achieving soft failures, 6) non-destructive inspection and test methods, 7) remote welding, 8) vision systems to overcome lack of depth perception (stereo cameras and headsets are hard on operators), 9) inertial navigation, 10) stabilized platforms for better microwave uplinks from ocean surface to satellites, and 11) miniaturization of electrical and mechanical devices and systems.

Dr. Brian Wilcox, head of a telerobotic research group at the Jet Propulsion Laboratory, provided the panel with some additional "supply side" information for NASA technology transfer. Further discussion by the panel was prompted by Dr. Wilcox' descriptions of 1) light-weight manipulator arms for space applications whose construction and control paradigms could be of commercial interest, 2) the NASA commitment to force-synchronized dual arm control, to include handling of extended rigid objects, and 3) planning to include critical autonomous functions (such as on-board force sensing instead of force reflection) on space-based teleoperators controlled from the ground. This is essential to work around the 1 to 2 second time delay through the TDRSS satellite up- and down-links.

CONCLUSION

The discussions of motives and requirements for telerobotics application demonstrated that, in many cases, lack of progress was a result not of limited opportunities but of inadequate mechanisms and resources for promoting opportunities. Support for this conclusion came from Telerobotics, Inc., one of the few companies devoted primarily to telerobot systems. They have produced units for such diverse applications as nuclear fusion research, particle accelerators, cryogenics, firefighting, marine biology/undersea systems and nuclear mobile robotics. Mr. Flatau offered evidence that telerobotics research is only rarely supported by the private sector and that it often presents a difficult market.

Questions on the mechanisms contained within the NASA technology transfer process for promoting commercial opportunities were fielded by Ray Gilbert and Tom Walters. A few points deserve emphasis:

1, NASA/industry technology transfer occurs in both directions and NASA recognizes the opportunity to learn a great deal from industry in the fields of automation and robotics

2, Promotion of technology transfer projects takes a demand side approach, with requests to industry for specific problem identification. NASA then proposes possible solutions.

Over

123

3. o Commitment of motivated and technically qualified people on each end of a technology transfer is essential.

4. o NASA assures protection of proprietary interests and provides incentives such as exclusive licensing of NASA patents as part of the technology transfer process.

5. o NASA often enlists the assistance of other agencies (e.g., Dr. Hodge mentioned the DoD Joint Technical Panel on Robotics), associations, or technical societies to recommend or participate in transfers with industry.

The Office of Commercial Programs and its agents for technology transfer solicit the interest of industry and seek their specification of problems or requirements with which NASA might assist. Prompt attention to any inquiries may be obtained by contacting the Technology Applications Team, Research Triangle Park, NC (919) 541-6156. OCP gratefully acknowledges the opportunity for the Commercial Users Panel afforded by the Jet Propulsion Laboratory and greatly appreciates the contributions of the panel experts.

SYSTEM ARCHITECTURE

PAGE 124 INTENTIONALLY BLANK

188015 S13-63
188045
48

Task Allocation Among Multiple Intelligent Robots

L. Gasser and G. Bekey
University of Southern California
Los Angeles, CA 90089-0782

U 620312S

Abstract Research

~~We~~ describe the design of a decentralized mechanism for allocating assembly tasks in a multiple robot assembly workstation. Currently, ~~our~~ the approach focuses on distributed allocation to explore its feasibility and its potential for adaptability to changing circumstances, rather than for optimizing throughput. Individual "greedy" robots make their own local allocation decisions using both dynamic allocation policies which propagate through a network of allocation goals, and local static and dynamic constraints describing which robots are eligible for which assembly tasks. Global coherence is achieved by proper weighting of "allocation pressures" propagating through the assembly plan. Deadlock avoidance and synchronization is achieved using periodic reassessments of local allocation decisions, ageing of allocation goals, and short-term allocation locks on goals.

1 Introduction

The coordination of several robots in a flexible assembly workstation is a problem of growing importance. Three levels of coordination are necessary:

1. *Planning*: Assembly tasks must be decomposed and represented as cooperative arrangements among several assembly robots [4].
2. *Resource Allocation*: Particular robots must be assigned individual tasks in a detailed assembly plan, in ways that assure all tasks are carried out with optimal throughput [5].
3. *Coordinated Motion Planning*: Robot effectors must be controlled dynamically as they move close together in concert, while avoiding collisions.

In this paper we are concerned only with the second coordination level - allocating tasks to robots. In a workstation of limited size, for tasks of limited complexity, we may be able to allocate robots to tasks using a centralized global allocation mechanism, for example, one based on heuristic search of the space of possible allocations. As static task or workstation complexity increases, however, a global solution becomes more costly. Moreover, if we assume that orders to the flexible assembly station arrive randomly, necessitating reconfiguration, it will be very difficult to recalculate the global allocation plan each time to "fold in" new orders with common subassemblies.

For these reasons, we have decided to explore a decentralized approach to task allocation. In this scheme, a collection of robots is *greedy* for work; they are eager to take on whatever work they can do, and as much of it as possible. Each robot makes its own decision about what task to take on, based on its own local decision criteria. The robots' greed is mediated by a set of dynamically changing *allocation policies*, which assures that global throughput requirements are met, and which encourages individual robots to assign themselves the most appropriate tasks given the global circumstances, such as order due dates, parts availability, etc.

Our focus here (and our motivation for investigating concurrency) is to explore *the feasibility of a distributed solution* and *adaptability to changing circumstances*, rather than optimal throughput, or maximum production or reallocation speed. We have not yet addressed the temporal scheduling of task assignments to achieve desired throughput results, though we shall discuss some ideas for handling temporal constraints on processing.

2 The Nature of Assembly

Assembly is the process of composing higher-level structures from parts (primitive structures) and lower-level subassemblies using *assembly operations* such as welding, fastening, screwing, riveting, inserting, etc. The assembly task allocation process must account for producing some number of copies of the high-level assemblies it generates - there is some *lot* of assemblies to make. Each lot is assembled on the basis of an *assembly plan* for an individual object in the lot, and on the basis of some *due date* for the entire lot. The due date places a time constraint on the assembly process.

Based on a preliminary analysis of several common manufactured objects, we have derived a canonical form for the high level description of coordinated assembly tasks in an assembly plan. Figure 1 shows our scheme. An assembly plan is shown as a tree-structured composition of subassembly plans. Nodes in this tree are primitive assembly operations. Assembly operations may require either single robots or groups of coordinated robots. We have only illustrated coordinated assembly tasks which require pairs of robots, but the scheme remains analogous for either single-robot tasks or those requiring more than two robots.

Any high-level assembly A is composed of several subassemblies, in this case S_1 and S_2. The assembly operation O_3 which assembles S_1 and S_2 to produce A comprises two subtasks, L_{3a} and L_{3b}, (which might be a HOLD and an INSERT operation, respectively). Each subtask has an associated *resource constraint* $R_{L_{3a}}$ and $R_{L_{3b}}$, respectively. Resource constraints have static and dynamic components. The static resource constraints describe the machines which must be used to perform the subtask, and may indicate the type of tooling required, the lifting capacity or application force required, etc. Dynamic resource constraints describe constraints which vary with the allocation process (such as the real physical proximity of the required partner robot), once some robot has has taken on one task of the pair. The same description notation is used at each level of the assembly plan. Lowest-level subassemblies are made from *parts* (P).

2.1 Allocation Pressure

The existence of any unfulfilled task in the assembly plan creates two subgoals: a global *goal* of allocating some robot to the task and a local *performance goal* goal of performing the task once it has been allocated. However, we use a decentralized approach to allocation, meaning that the individual decisions about which robots respond to which goals are taken by the robots individually, not by a global allocator. Still, to maintain global coherence, the global task-allocation goals must be ordered to reflect global priorities. We do this by establishing dynamic local *policies* to guide individual allocation decisions. These policies are not explicit decision rules; instead they are weighting factors attached to each allocation goal indicating the global importance of the goal. The weights are called *allocation pressures* for the goals. A higher allocation pressure makes a given goal more attractive to any robot. The combination of allocation pressures, resource constraints, and the opportunistic decisions of individual robots controls the allocation process over time.

Allocation pressures attached to goals are dynamic and local. They change over time as some tasks are completed and others become more pressing. They are created by propagating allocation pressures through the assembly plan. Allocation pressures come from three sources, and are termed *production pressure*, *coordination pressure*, and *consumption pressure*. First, lower-level subassemblies must be created before higher-level ones can be assembled; this places a precedence on the order of assembly, and thus establishes some precedence for task allocation (e.g., when there are fewer robots than assembly tasks, or some robots are faster, or better-suited for certain tasks than others). As each assembly order arrives, it carries a *due date*. The due date for the completed lot provides the top-level *production pressure* which propagates downward (i.e. toward finer-grained subassemblies) with increasing force. This serves to encourage robots to choose lower in the assembly plan at first, since it would make no sense to take on higher-level assembly tasks if there were nothing to assemble. As the due date approaches, the production pressure *increases*, ageing the allocation goals over time.

Second, if one robot (X) decides to assume some subtask L_{ia}, which is part of a coordinated task O_i, it must be sure that it can induce some other robot (say, Y) to take on the corresponding subtask L_{ib}, in order to proceed. We call this the *principle of complementarity in allocation*. As one robot makes a decision to assume a task, it creates 1) a new set of dynamic resource constraints on the corresponding subtask (because the allocation of one robot determines the physical location where the task will occur), and 2) creates *coordination pressure* to encourage some available and appropriate robot to assume the complementary subtask (since without a partner, the operation is not possible).

Third, the production of lower-level assemblies creates an upward-propagating *consumption pressure*. This encourages some robots to take on the higher-level assembly tasks in order that work-in-process inventories (e.g. parts storage bins) and work flows remain balanced. Consumption pressure increases as more lower-level subassemblies are produced. It is initiated at the parts level, by arrival or availability of parts at the workstation.

Allocation pressures are propagated using weighted propagation functions on the arcs of the assembly plan. The precise nature of the propagation functions is to be determined by experimentation and theoretical analysis, which await further research.

2.2 Local Allocator Decisionmaking Criteria

Individual robots make local allocation decisions by examining the available unsatisfied allocation goals and selecting those with compatible resource constraints. From this set, the most highly-rated allocation goal is selected, and the robot allocates itself (see implementation section below). This allocation decisionmaking takes place with some fixed periodicity, as well as happening any time a robot becomes available for work. It is important to have repeated, periodic checks, for adaptive allocation and deadlock avoidance.

If a robot has no current task (i.e. it is free to take on any compatible task) then it may decide purely on the basis of compatibility, However, if the robot is engaged, yet is doing a periodic re-allocation check, it must also consider the *changeover cost* in deciding whether to take on a new task.

Changeover costs include the costs of retooling, opportunity costs for not getting the new work done, and the prevailing performance goals for the task it is already performing.

These decisionmaking criteria are purely local, both with respect to the individual tasks, and to the temporal unfolding of the global assembly work. It is possible that, for maximum throughput, an individual robot should avoid taking on a task which is immediately available, and should wait for an upcoming but currently lower-rated task to which it is better suited. If the upcoming task is one in the current set of globally-known allocation goals, this requires each robot to incorporate in its decisionmaking criteria either 1) meta-level control policies which can be constructed by some planner with a more global view or by integrating information about what other robots are doing by communicating with them [1], or 2) some predictive knowledge about the expected trajectory of the allocation pressures through the system. If the upcoming allocation goal will be generated by an assembly order which itself has not been generated, the individual robot will need even higher-level information about the likely arrival of new assembly orders.

With concurrent access to all allocation goals in an assembly plan, there is a potential for deadlock. If two robots with conflicting reach constraints simultaneously choose complementary subtasks, (e.g. L_{ia} and L_{ib} of task O_i) the operation cannot proceed, but there will be no way to deallocate any robot. Moreover, there is no criteria for deciding which robot should be deallocated. This is only a problem for subtasks of the same operation, because they must have *simultaneous allocation* of appropriate resources. It is not a problem for subtasks of different operations, which can be allocated concurrently. Inappropriate allocation of robots across different operations will lead to inefficiency, but the adaptation mechanisms provided by the propagating allocation pressures, ageing of goals with respect to due dates, and the re-assessments of allocation decisions will force the system to correct itself.

We use two mechanisms to prevent deadlock among subtask allocations. First, allocation access to all subtasks of a single assembly operation must be locked so that only one allocator at a time can make an allocation decision. After this decision is made, the new dynamic constraints posted on the complementary allocation goals assure that only appropriate partners choose the complementary subtasks. This locking, while it reduces concurrency, does not create undue overhead, because the allocation decisions are made infrequently, and are short by comparison to the amount of time it takes for actual assembly operations.

Second, if no partner chooses the complementary subtask within the time constraint provided by the allocation reassessment period, the allocation goal's value may change, and the robot is free to deallocate itself and take on a more highly-rated goal (given the changeover cost - see below).

3 Implementation Approach

The allocation system described here has not yet been implemented, but here we present our design for implementation. The implementation is planned for MACE [2,3], our concurrent Distributed AI testbed. The basic structure of the system is designed to be a global but distributed blackboard system [3]. The blackboard itself contains the overall assembly plan and propagation links, and a collection of concurrently-executing allocation decisionmakers, one associated with each robot in the workstation.

3.1 Representation of the Assembly Plan

The assembly plan is represented as a collection of goals on a globally accessible but distributed blackboard. The blackboard may be pre-segmented according to machine types and allocation constraints, to provide some efficiency in allocation and communication, and to allow for increased concurrency as individual allocators access different parts of the blackboard [3]. Each allocation goal in the assembly plan is represented as a collection of constraints on the type of robot which can assume the task. Each goal has two constraint sections: a dynamic and a static part. Constraints are descriptions of robot characteristics expressed in a flexible pattern-language [2], to allow for partial matches, restricted matches, and variable bindings for the allocators. Static constraints are fixed parts of the assembly plan, whereas dynamic constraints are updated as allocation decisions are made. The updating is done by a MACE computational agent which manages blackboard access. Allocation locks are implemented using the mailbox synchronization provided by MACE, and are controlled by the blackboard level manager.

Allocation goals are linked to one another with any of four types of uni-directional links. Each link comprises a *type* and a *propagation function* for propagating allocation pressures or constraints. The four types of links are *production-pressure links*, *consumption-pressure links*, *coordination-pressure links*, and *constraint-propagation links*. Constraint-propagation links connect subtasks of a single assembly operation, and describe how to update dynamic allocation constraints.

3.2 Local Allocation Decision-Making Agents

Local decisionmaking agents are individual MCAE agents which access the global blackboard using messages and demons. MACE provides a facility for remote demons, so that when a goal for which a particular robot is particularly well-suited is posted or its evaluation changes, the robot can be notified opportunistically. Direct access to allocation goals is achieved with messages. Local decisionmaking criteria are built into rules within each allocator agent.

4 Conclusions

We have presented the design of a promising scheme for decentralized allocation of tasks in a multiple robot assembly workstation. Further research remains on the theoretical analysis of weights and feedback through the system. This analysis should include a control-theoretic analysis of the system, to show the weighting and constraint conditions under which it is possible to assure allocations will occur with maximum throughput, and to indicate the problematic condtitions under which the system can avoid thrashing, or at least damp it. After implementation, we also expect experimental analysis to derive appropriate weights, reallocation periodicity, and adaptive performance behavior.

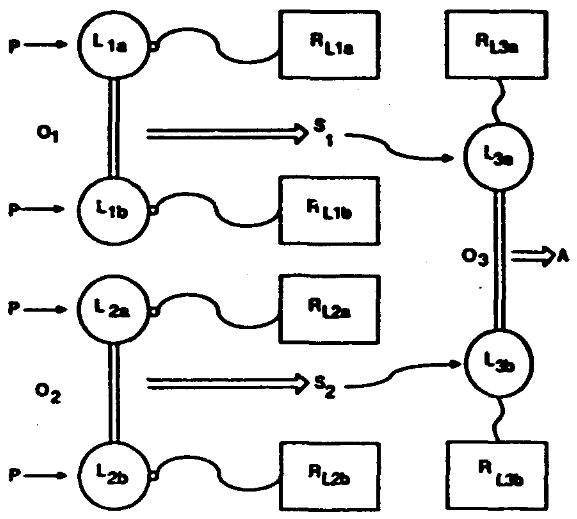


Figure 1.

References

[1] Durfee, E., Lesser, V. and Corkill, D. "Coherent Cooperation Among Communicating Problem-Solvers." *IEEE Transactions on Computers* (to appear) 1987.

[2] Gasser, L. Braganza, C., and Herman, N. "MACE: A Flexible Testbed for Distributed AI Research," to appear in M. Huhns, Ed., *Distributed Artificial Intelligence*, Pitman Publishers, 1987.

[3] Gasser, L. Braganza, C., and Herman, N. "Implementing Distributed AI Systems Using MACE" in *Proceedings of the 3rd IEEE Conference on Artificial Intelligence Applications*, Orlando, Fla., February, 1987 (to appear).

[4] K. Konolige and N. J. Nilsson. "Multi-Agent Planning." in *Proc. National Conference on AI*, 1980.

[5] Sadeh, N. and Gasser, L. "Hierarchical Scheduling and Resource Allocation," Paper submitted to the *International Joint Conference on Artificial Intelligence*, Aug. 1987.

18-346 : 514-63
8P

Execution Environment for Intelligent Real-Time Control Systems

J. Sztipanovits
Vanderbilt University
Nashville, TN 37235

VE 909080

1. ABSTRACT

Modern telerobot control technology requires the integration of symbolic and non-symbolic programming techniques, different models of parallel computations, and various programming paradigms. ~~This paper describes~~ the Multigraph Architecture, which has been developed for the implementation of intelligent real-time control systems. The layered architecture includes specific computational models, integrated execution environment and various high-level tools. A special feature of the architecture is the tight coupling between the symbolic and non-symbolic computations. It supports not only a data interface, but also the integration of the control structures in a parallel computing environment.

is described.

2. INTRODUCTION

There is an ever-increasing demand for improving the information processing capabilities of robot controllers, measurement systems, or process control systems. The ultimate goal is to enhance system autonomy, adaptivity and functional performance. Since some of these features were previously provided by human operators, systems exhibiting these properties are often qualified to be "intelligent."

Essential extension of capabilities is always based on the application of new techniques. Current trends in intelligent systems include: (1) the use of parallel/distributed computing architectures, (2) the use of parallel/distributed programming models, and (3) the application of various artificial intelligence (AI) programming techniques. Since the new techniques typically are not substitutes but extensions of the conventional techniques, integration has become a key factor in building intelligent systems. Symbolic and numerical computations, various programming paradigms and different parallel computing models have to be merged in the frame of a possibly unified architecture.

The critical system component, where most of the implementational problems of system integration have to be solved, is the execution environment. The execution environment provides run-time support for the architecture, and couples various programming models to each other and to the underlying hardware system.

This paper describes an experimental architecture - the *Multigraph Architecture* - and the corresponding execution environment developed for building integrated systems. After the summary of the background of this research, the design considerations are outlined. Then, the main components of the Multigraph Architecture are discussed, which is followed by the summary of various applications and future plans.

3. BACKGROUND

A. Intelligent Systems

There is a rapidly growing research interest in the application of AI techniques in robot controllers, measurement systems and process control systems. General, architectural issues of intelligent systems are analyzed in [1]. The generic architecture of intelligent systems is characterized by the introduction of the "knowledge-level," which includes "knowledge-intensive" system components providing high-level perception, modelling and planning functionalities.

The structure and operation of the knowledge-level system components are typically model-driven. New possibilities offered by knowledge-based, model-driven automation in telerobotics are described in [2]. Architectural issues of model-driven instrumentation are discussed in [3] and the application of new techniques in the Knowledge-based Experiment Builder of a magnetic resonance imaging system is discussed in [4]. The prospective of model-driven, knowledge-based systems in controllers is outlined in [5].

A common view regarding the structure of intelligent systems operating in real-time environment is that they must have layered architecture where "high-level," knowledge-based system components synthesize, monitor and control the operation of the "low-level" sensory and processing activities.

B. Graph models of computations

An important class of parallel computational models are the graph models. The computational graphs (or control graphs) are directed graphs, where nodes represent units of computations and arcs represent dependency relationships. The general properties of graph models are analyzed in [6], and their classification is given in [7]. In dataflow models, arcs arise from data dependences, and data are passed along the arcs in execution time. In control-flow models, not the data, but pointers to the data are carried; therefore, this model requires the availability of shared memory. The computational units can be scheduled data-driven or demand-driven. Data-driven scheduling means that a unit is executable if the necessary input data are available. In demand-driven scheduling, only those nodes which are necessary to provide the requested data are activated.

Graph models have essential advantages in the context of intelligent real-time systems.

- Graph models can uniformly describe parallel computations for different multiprocessor architectures, such as distributed and shared memory systems.

- The granularity of the model can be "tuned" by selecting the size of the computational units.

- The "imperative" parts of the computation (i.e. the code of the computational units) are naturally separated from its logic structure represented by the control graph. This separation makes it possible to dynamically modify the computational structure.

- The control graph can be easily represented in declarative form. The declarative representation is the key for using symbolic processing techniques to synthesize various computational structures, such as real-time signal processing systems [4].

4. DESIGN CONSIDERATIONS

The Multigraph Architecture (MA) provides software framework for building intelligent systems in real-time, parallel computing environment. The main layers of the architecture are: the (1) Physical layer, (2) System layer, (3) Module layer and (4) Knowledge base layer. The basic properties of the individual layers are summarized below.

A. Physical layer

Computational heterogeneity, various physical constraints (such as distance between computing nodes), and the typically high computation load require the support of different multiple-processor configurations: tightly-coupled architectures with shared memory, loosely-coupled computer networks, and their combination. Special hardware components such as array processors or i/o devices might also belong to the hardware configurations.

B. System layer

The primary function of the system layer is to provide access mechanisms to the hardware resources. In the current implementations of MA, the system layers are off-the-shelf operating systems, which facilitate services such as standard i/o, task management, intertask (interprocessor) communication and synchronization and real-time clock. An important requirement for the higher-levels is flexibility to ensure the portability to different operating systems.

C. Module layer

One of the most critical requirements for MA is to support the synthesis and dynamic modification of various low-level computational structures (signal processing systems, control systems, etc.) in parallel computing environment. The key idea in the solution is the introduction of the module layer, which serves as an interface between the knowledge base layer and the system layer. The module layer has a special graph-oriented computational model, the Multigraph Computational Model (MCM), which provides the following possibilities:

- high-level (possibly very high-level) declarative languages can be defined on the knowledge-based layer to represent various computational structures such as procedural networks, constraint networks, reasoning networks etc.;

- these declarative forms can be interpreted and mapped into a computation graph on the module layer;

- the run-time support of MCM can schedule the elementary computations and "pass" them to the system layer for execution, taking advantage of the available parallelism of the computational structures;

- appropriate interpretation techniques can ensure the dynamic modification of the computation graph.

The name "Module layer" suggests the view that this layer is a "module library" consisting of typically small program modules written in C, Fortran, LISP etc. These modules are structured to form a complete program by the definition of the computation graph.

D. Knowledge base layer

High-level, symbolic computations are implemented on the knowledge base layer. Although the actual structure of the knowledge-based system components are strongly application dependent, the parallel computing environment and the features of the underlying module layer make the elaboration of a generic programming model desirable. The main purposes of the high-level programming model are:

- to support the structurization of the knowledge-based operations into concurrent activities,
- to facilitate a standardized, high-level communication system among the activities, and
- to provide interface to the module layer and MCM.

A strict requirement is that these services have to be implemented as extensions to one of the standard LISP systems in order to preserve the compatibility with different AI toolsets.

5. OVERVIEW OF THE MULTIGRAPH ARCHITECTURE

The basic computing models used on the different layers of MA and their relationships are represented in Figure 1.

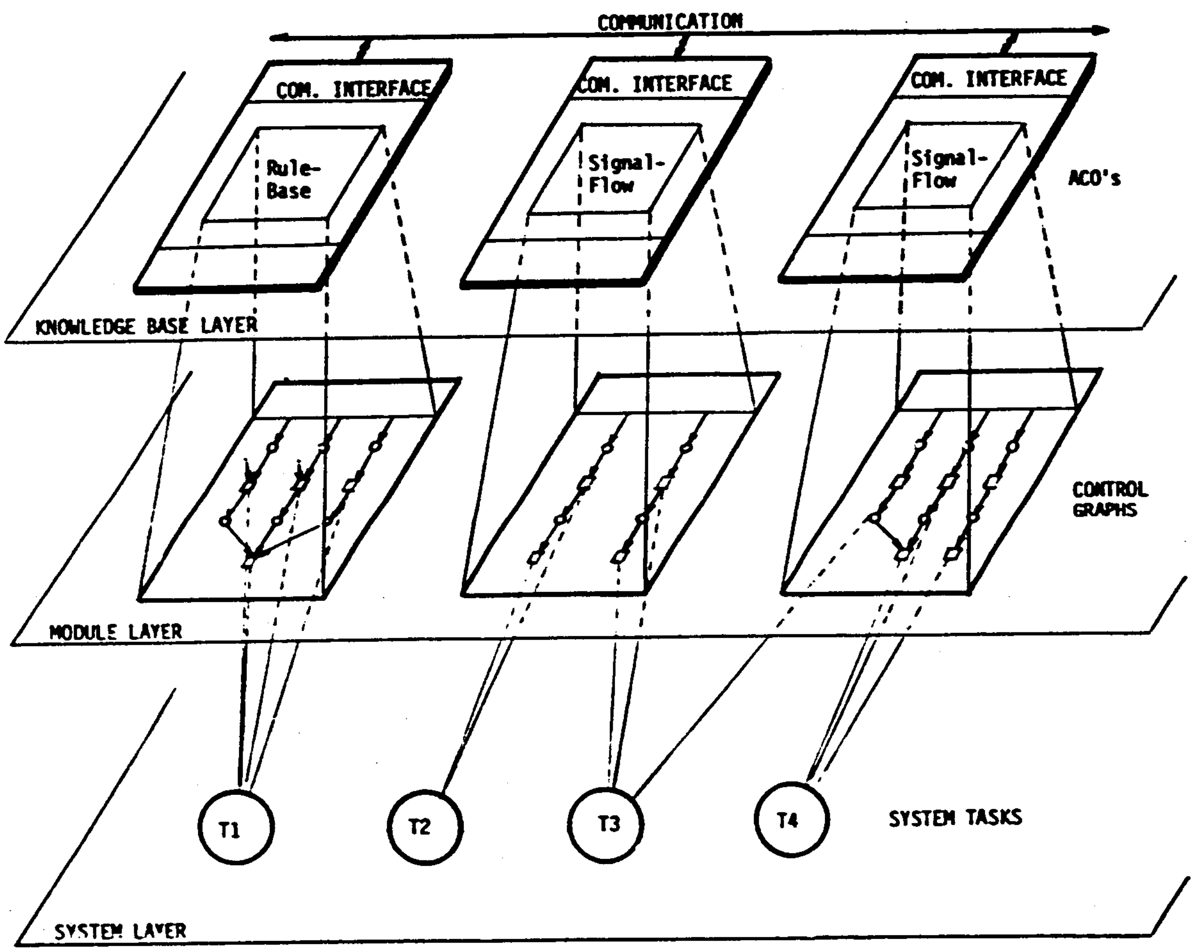


FIGURE 1. Layers of the Multigraph Architecture

A. Autonomous Communicating Objects (ACO)

The basic system structurization principle on the knowledge base layer is provided by the concept of **Autonomous Communicating Objects.** ACO is a straightforward extension of the "object" concept of object-oriented languages such as Flavor [8] in the following sense:

- ACO's are fully autonomous systems that can run virtually or physically parallel,
- ACO's can be dynamically allocated and can compete for the same resources,
- they communicate with each other by means of a fully asynchronous communication protocol.

The main purpose of ACO's is to provide a standardized "object shell" around a variety of heterogeneous knowledge-based system components. The communication "methods" are standard elements of the object shell, and hide the details of an actual implementation from the application programmers.

Various object types have been developed for supporting specific applications. These objects typically include a "knowledge base," which is represented by a special representation language. Some of these object types, such as **Procedural Network Object** (PNO) and **Rule Network Object** (RNO) are described in [9].

B. Multigraph Computational Model (MCM)

The different object-types are facilitated with an appropriate interpreter or incremental compiler, which maps the actual knowledge base into a computation graph on the module layer [9]. While ACO's serve as a symbolic representation and interface to possibly complex functional components of the system (e.g., a signal processing system, rule-based system, associative database system, etc.), the computation graph on the module layer constitutes their actual execution environment. This relationship between the ACO's and their execution environment has the following advantages:

- Execution of the operations represented by ACO's occurs in a parallel execution environment offered by MCM.
- The interpreter (or incremental compiler) "methods" of ACO's, which build the computation graph, can dynamically modify the graph, as a response to an external message, or to a feedback from the execution environment (a mechanism for implementing "self- modifying" signal processing systems is described in [4]).
- The system fully integrates two different parallel computing models. ACO's form the "macro-structure" of the system, and they communicate by using the services of loosely coupled distributed systems (typically message passing). The computation graphs provide the "micro-structure" of the system components. MCM efficiently supports medium-level (subroutine size) computational granularity, and can take advantage of tightly-coupled multiprocessor architectures with shared memory.

C. System tasks

The computational model on the system layer is provided by the actual operating system. The key concept is the "system task," which represents a "slice" from the processing capacity, and can access to various resources. The elementary computation units that are scheduled by the run-time support of MCM are executed by the system tasks.

6. MULTIGRAPH COMPUTATIONAL MODEL

MCM can be characterized as a **control-flow model**. The control structures of computations are represented by bipartite graphs that are built of **actornodes, datanodes** and **connection** specifications (see Figure 2).

The actornodes are associated with the elementary computational units, called **scripts**, which can be written either in LISP or in any other language, such as C, Fortran, Pascal, etc. The scripts do not know about their position in the control graph: they communicate with other graph components through the input/output ports of the actornodes. The actornodes are associated with a local datastructure, called **context**, which can be accessed by the script. If the code of the script is reentrant, it can be attached to several actornodes. In different computation problems the scripts may be quite different: a script may be an interrupt-driven i/o handler, a transformation of the input data arriving to the actornode, or an interpreter module, which interprets the symbolic form stored in the context of the actornode.

Datanodes store and pass the data generated by actornodes. They can be either **streams** with multiple output ports, or **scalars**. The streams maintain the partial sequence order of the data generated during the computations, which preserves the overall consistency.

The control graph can be operated in **data-driven** or in **demand-driven** mode, or in a combination of the two modes. In data-driven mode, the data sent to a datanode propagate a "control token" to the connected actornodes. The actornodes will fire according to the specified control discipline: in **ifall** mode, at least one

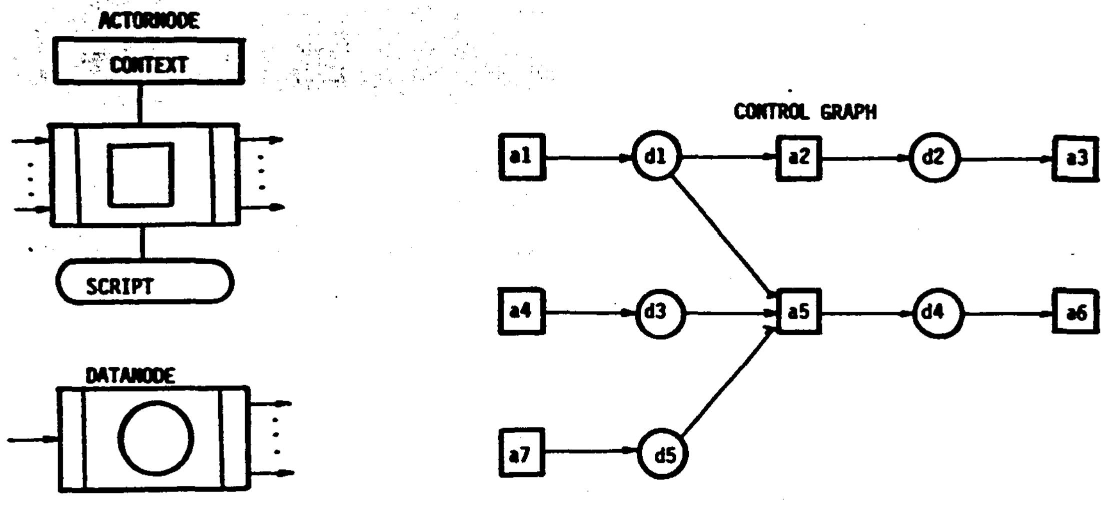


FIGURE 2. Components of the Multigraph Computation Model

control token must be sent to all of the inputs of the actornode: in ifany mode, every received control token will cause firing. In demand-driven mode, the request for data sent to a datanode will generate a control token if the datanode is "empty." This control token will fire all of the actornodes that are potentially able to provide the requested data. The demand propagates backward along the control graph until data is generated. From this point a forward propagation starts, which finally provides the requested data. A more detailed description of the computational model can be found in [10, 11].

The run-time support for the MCM is provided by the Multigraph Kernel (MK). The structure of the MK is shown in Figure 3. The control graph is represented in the descriptors, which are manipulated by various kernel functions. The Control Interface functions are used for dynamically building and modifying the control graph. These functions are imbedded in a LISP system, where the various graph-builder interpreters and incremental compilers are implemented. The Module Interface includes the data/demand propagation kernel calls for the scripts. An important feature of the system is that actornodes with scripts written in different languages can be mixed in the same control graph. (The necessary transfer routines are invisible to the user.) This feature is used for creating tight coupling between symbolic and non-symbolic computations. Tight coupling means that not only data structures can be passed between the two kinds of computations, but there may be a fully integrated control structure.

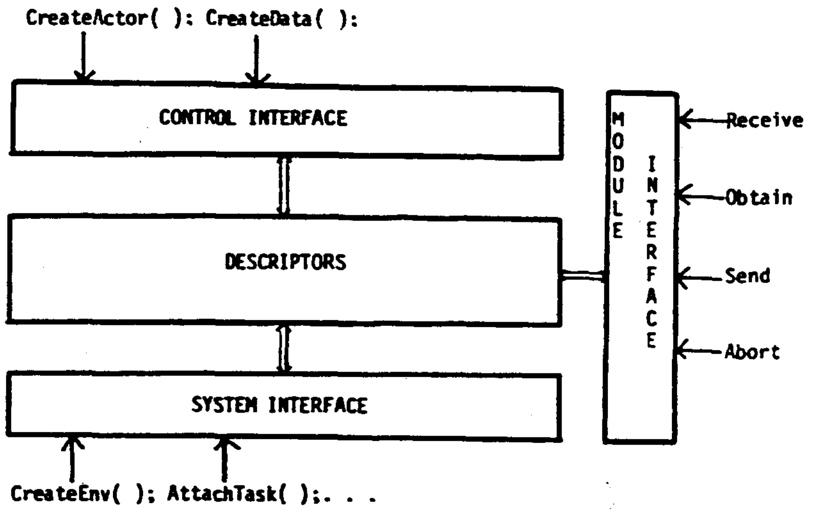


FIGURE 3. Structure of the Multigraph Kernel

The most complex part of MK is the System Interface. The System Interface schedules the elementary computations that are defined during the data/demand propagation. The computational units (the scripts of the fired actornodes) are passed to the available system tasks for execution. The environment mechanism of the System Interface ensures that subsets of the control graph can be dynamically associated with one or more system tasks that include the necessary resources for executing the scripts. This mechanism provides a very straightforward way for dynamic resource management in multiprocessor configurations.

Two important implementation issues are the granularity and the memory model. The lower limit for the reasonable computational granularity is basically determined by the overhead of MK. Since MK currently is implemented in software, the overhead is introduced by the control token propagation functions. On the 68000 processor-based IBM 9000 system (clock frequency is 8Mhz), the overhead is about 800 microseconds: on the VAX 785 implementation is less then 200 microseconds. Due to the construction of the MK, the overhead is basically independent from the size of the control graph.

Since MCM is a control-graph model where the pointers to data structures rather than the data are passed along the graph, the model requires the presence of shared memory for those tasks (and processors) that are assigned to the same subgraph as execution resource (see Figure 4). In order to provide flexibility toward architectures which do not support shared memory access for the processors (hypercube architectures or distributed computer configurations), a simple mechanism is implemented to link control graphs that are allocated in the local memory of the separate nodes. The scripts of receiver and transmitter actornodes provide a logical link between the separated subgraphs and implement the data transfer by using the actual services of the underlying system layer (e.g., the message passing services of DECNET in the VMS/DECNET implementation). At these links, the control-graph model is "transformed" to dataflow model, since the actual datastructures - and not just pointers - are passed to the "remote" nodes.

This method makes it possible to generate large processing networks from their symbolic representation in distributed computing environment [12].

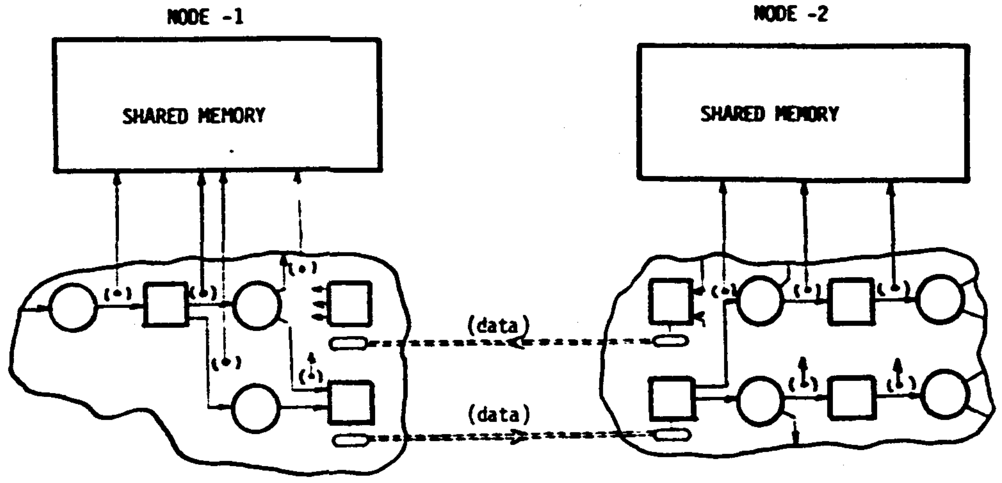


FIGURE 4. Memory Model of MCM

7. STRUCTURE OF THE EXECUTION ENVIRONMENT

The simplified structure of the execution environment supporting MA can be seen in Figure 5. MK is implemented as an additional layer to a standard operating system. Depending on the computer architecture and on the features of the particular operating system, MK may exist in one or more copies. The System Interface of MK is a well structured, modular program which makes porting the kernel relatively easy, even to devastatingly different operating systems and real-time supervisors.

The Module Interface functions of MK can be invoked from LISP as well as from other languages. This ensures that scripts can be written in different languages, and existing module libraries can be easily interfaced to MK. The Control Interface of MK is imbedded in LISP since currently we use LISP as implementation language of the knowledge base layer.

Various high-level software components such as the generic object shell for ACO's and the standard methods of different ACO types are implemented in LISP.

For distributed computer configurations, the LISP system (in the first implementation FRANZ LISP, recently changed to Common Lisp) has been expanded with the Communicating LISP System facility, which provides task management and asynchronous message passing primitives [13].

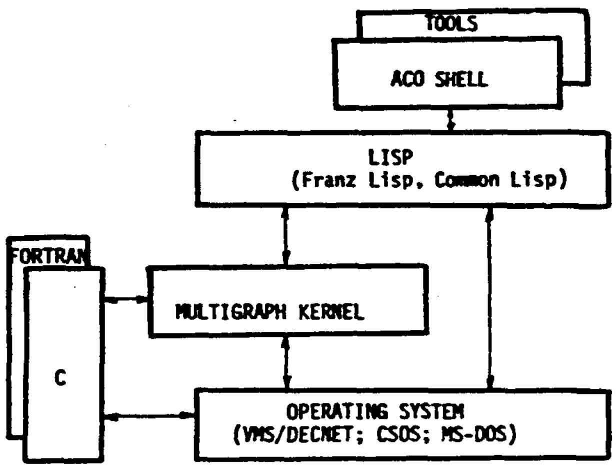


FIGURE 5. Structure of the Execution Environment

8. EXPERIENCES

MA has been implemented on very different computer configurations and has been used for various applications.

An Intelligent Test Integration System (ITIS) has been implemented in the Space Station Laboratory of the Boeing Aerospace Company in Huntsville, AL [12]. The purpose of ITIS is to support the automatic generation of test systems in real-time, distributed computing environment. ITIS is implemented as a knowledge base layer above the conventional test system components, and can build complex test configurations from the symbolic specification of test scenarios. The computing environment is a VAX network with the VMS/DECNET operating system expanded with special hardware units.

A different application of the layered MA architecture is the Knowledge-based Experiment Builder (KEB), which was developed for the M.I.T/IBM experimental MRI (Magnetic Resonance Imaging) system [4]. The core of the KEB is a high-level representation language for signal processing schemes and a smart interpreter, which can generate the appropriate version of the real-time signal processing system, and which is able to reconfigurate it for specific events. The computing environment of this system is the IBM 9000 computer with the CSOS real-time operating system.

Various computational structures have been developed and are being investigated for the MCM, such as a hierarchical planner [14], knowledge-based simulation builder [15], pattern-driven inference system [16], etc.

9. CONCLUSIONS AND FUTURE PLANS

The integration of symbolic and conventional programming techniques, parallel computing models of different granularity, and various programming paradigms are essential conditions for the successful implementation of intelligent real-time systems. The Multigraph Architecture has proven to be a good approach to solve the problems of integration. It provides a generic framework, programming models for structurizing software components and various tools for the actual implementation.

We have practical experiences with implementing systems in single-processor (IBM-AT/MS-DOS), single-processor multitasking (VAX/VMS and IBM 9000/CSOS) and distributed (VAX network VMS/DECNET) computing environments. As a next step, we intend to implement the execution environment for tightly-coupled multiprocessor configuration.

The system offers a convenient method to describe and implement "self-modifying" signal processing systems. Further research is needed to utilize this capability in the design of structurally adaptive measurement control systems.

10. ACKNOWLEDGEMENTS

The research described in this paper was supported in part by Boeing Aerospace Company, IBM Corporation and Vanderbilt University. The author especially wishes to acknowledge the contributions of Csaba Biegl and Gabor Karsai to the design and implementation of the execution environment, and Byron R. Purves of Boeing Aerospace Company and Colin G. Harrison of IBM to the design of the application systems.

11. REFERENCES

[1] A. Newell, "The Knowledge Level," Artificial Intelligence, Vol. 18, 1982, pp. 87-127.

[2] S. Lee, G. Bekey, A.K. Bejczy, "Computer Control of Space-Borne Teleoperators with Sensory Feedback," Proc. of the IEEE International Conference on Robotics and Automation, St. Louis, MO, 1985, pp. 205-214.

[3] J. Sztipanovits, "Knowledge-Based Approach in Measurement and Instrumentation," Proc. 3rd International Conference on Measurement in Clinical Medicine, Edinburgh, Scotland 1986, pp. 29-33.

[4] J. Sztipanovits, C. Biegl, G. Karsai, J. Bourne, C. Harrison, R. Mushlin, "Knowledge-Based Experiment Builder for Magnetic Resonance Imaging Systems," Proc. of the 3rd IEEE Conference on Artificial Intelligence Applications, Orlando FL., 1987 (in press).

[5] K.J. Astrom, "Auto-Tuning Adaptation and Expert Control," Proc. American Control Conference, Boston MA, 1985, pp. 1514-1519.

[6] J.C. Browne, "Formulation and Programming of Parallel Computations: A Unified Approach," COMPCON, Spring 1985, pp. 624-631.

[7] D.D. Galski and Jih-Kwon Peir, "Essential Issues in Multiprocessor Systems," IEEE Computer, June 1985, pp. 9-27.

[8] D. Moon, R. Stallman, D. Weinreb, "LISP Machine Manual," The MIT AI Lab., Cambridge MA, 1984.

[9] J. Sztipanovits, R. Purves, G. Karsai, C. Biegl, S. Padalkar, R. Williams, T. Christiansen, "Programming Model for Coupled Intelligent Systems in Distributed Execution Environment," Proc. of the SPIE's Cambridge Symposium on Advances in Intelligent Robotics Systems, Cambridge, MA, 1986 (in press).

[10] J. Sztipanovits, "MULTIGRAPH: Parallel Architecture for Intelligent Systems," Dept. of Electrical Engineering, Vanderbilt University, Techn. Report #86-01, 1986.

[11] C. Biegl, "Multigraph Kernel User's Manual," Dept. of Electrical Engineering, Vanderbilt University, 1985.

[12] J. Sztipanovits, B. Purves, S. Padalkar, J. Rodriguez, K. Kawamura, R. Williams, H. Biglari, "Intelligent Test Integration System," Proc. of the Conference on Artificial Intelligence for Space Applications, Huntsville, AL, 1986, pp. 177-185.

[13] S. Padalkar, "Communicating LISP System Facility," M. Sc. Theses, Dept. of Electrical Engineering, Vanderbilt University, 1987.

[14] G. Karsai, "Hierarchical Planning with Objects," Proc. of the 19th Southeastern Symposium on Systems Theory, Clemson, SC, 1987 (in press).

[15] C. Biegl, "Knowledge-based Generation of Simulation Models," Proc. of the 19th Southeastern Symposium on Systems Theory, Clemson, SC, 1987 (in press).

[16] C. Biegl, "Florence (Dataflow Oriented Inference Engine) User's Manual," Dept. of Electrical Engineering, Vanderbilt University, 1986.

188097 515-63

108.

Distributed Intelligence for Supervisory Control

W.J. Wolfe and S.D. Raney
Martin Marietta Denver Aerospace
Denver, CO 80201

MI 411300

1. ABSTRACT

Supervisory control systems must deal with various types of "intelligence" distributed throughout the layers of control. Typical layers are real-time servo control, off-line planning and reasoning subsystems and finally, the human operator. Design methodologies must account for the fact that the majority of the intelligence will reside with the human operator. ~~This paper focuses on~~ hierarchical decompositions and feedback loops as conceptual building blocks that provide a common ground for man-machine interaction. Examples of types of parallelism and parallel implementation on several classes of computer architecture are also discussed. Are discussed.

~~2. INTRODUCTION~~

Designing systems that involve a lot of interaction with humans is a difficult task. Although humans are intelligent they are also unpredictable. It is easier to conceive of completely "autonomous" systems, requiring little man-machine interaction. But with such systems human operators run the risk of losing control and at some point the operator may want to redirect, assist, troubleshoot or enhance the system. Thus, it is desirable to be operating somewhere in the middle ground, where systems are designed to be autonomous only in the sense that they are part of a larger system that provides a means of interaction with humans. The concept of "supervisory control" provides such a framework.

2.1 General System Design Goal

Put in simple terms, it is desirable to have systems that are reliable, flexible and expandable (Figure 1). Reliability implies that the system rarely fails and that there is a quick fix for failures such as plugging in a new readily available module. Along with prior testing and evaluation is the development of run-time aids such as fault detection, isolation and diagnosis systems, all of which improve reliability. A flexible system is one that can perform many different functions or tasks and can operate in diverse environments. Such a system must consider many alternatives and be able to "understand" its environment. After a system is fielded it is invariably discovered that new capabilities are desired or required. An expandable system can easily incorporate new functions and more knowledge.

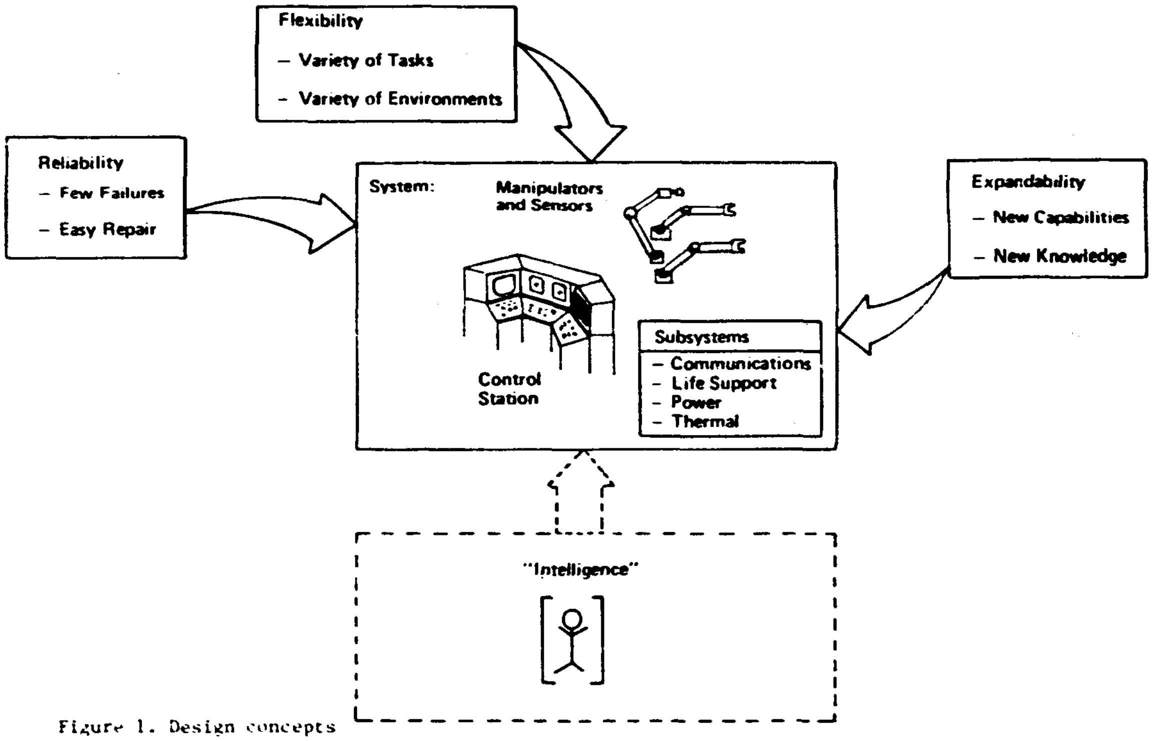


Figure 1. Design concepts

Many of these design goals boil down to providing the system with "intelligence". Although there has been much progress in the development of systems that display limited forms of intelligence,it is often the case that when intelligence is required human operators must be incorporated. Therefore, the development of complex systems now and in the near future will rely heavily on human operators (Figure 2).

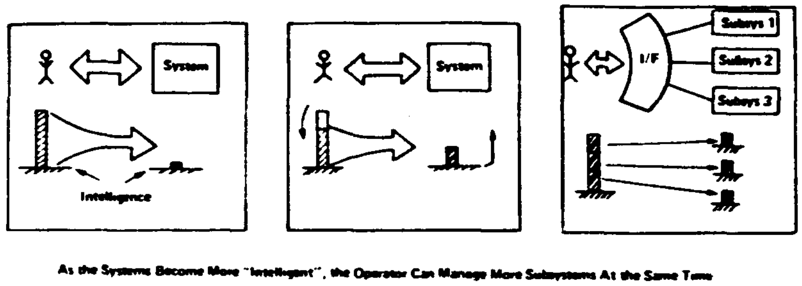

System
Intelligence
System
I/F
Subsys 1
Subsys 2
Subsys 3
As the Systems Become More "Intelligent", the Operator Can Manage More Subsystems At the Same Time

Figure 2. A definition of "supervisory control"

2.2 Supervisory Control

To effectively operate a system, a human must focus attention and can only expend a certain amount of energy. The more autonomous the system, the less attention required from the human operator. If the goal is to have less and less for the human operator to do the human would ultimately become a "watchman" asleep until an alarm sounds. Unfortunately, human expertise should be used more efficiently. Thus, the proper direction to move in is that as a particular system becomes "smarter", the operator devotes less attention to it and devotes increasingly more attention to a second system, and so on. From this perspective, the operator has various, possibly autonomous, subsystems under his control as he orchestrates the performance of the overall system.

2.3 Amplifiers of Human Capabilities

Machines, in general, enhance human capabilities. Machines give strength, reach and dexterity not normally possessed by humans; they can project presence into areas that would not normally be entered; they allow detection of things that are beyond the capability of human senses and they can enhance our analytical ability by rapidly analyzing data and summarizing the results (Figure 3). In this sense machines act as "amplifiers" of our capabilities, including the amplification of our intelligence. From this point of view, the design of a complex system is seen in terms of a centrally placed human operator - the source of intelligence - supported by advanced man-machine interfaces that provide an array of "tools". The tools are not intended to replace man, but to make it easier for him to perform difficult tasks.

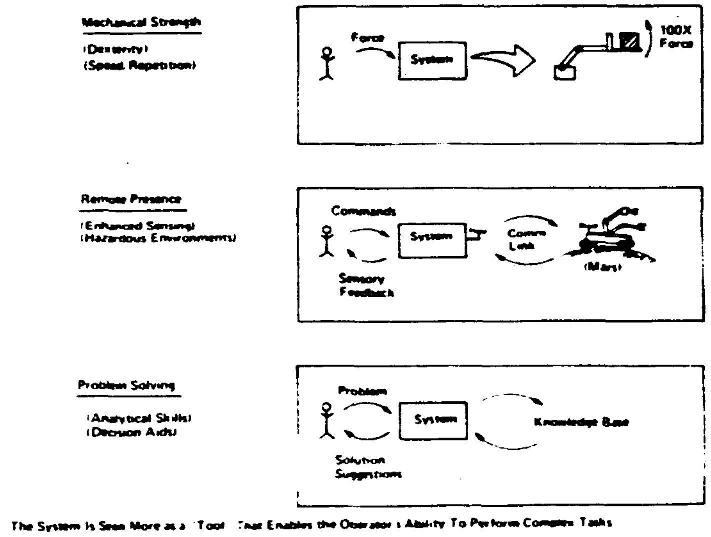


Figure 3. The system acts as an "amplifier" of human capabilities

3. LEVELS AND LOOPS

In seeking a common ground for both man and machine, two design techniques stand out: hierarchical decompositions and feedback loops (Figure 4). Although these two concepts are not mutually exclusive, they have contrasting properties, emphasizing different aspects of the system under scrutiny. Just about every system has elements of both. In fact, the same system might be described in terms of a hierarchy or a particular feedback loop, depending on the emphasis desired.

First we will discuss hierarchies.

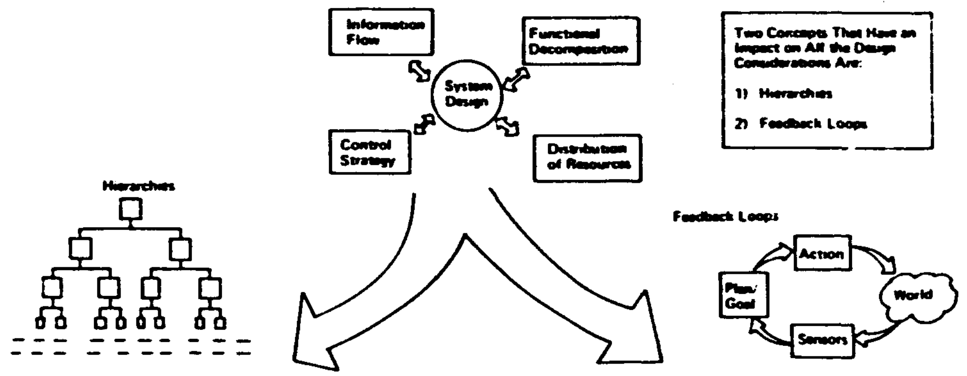


Figure 4. System design methodologies-levels and loops

3.1 Hierarchies

Hierarchies are pervasive, appearing in societal structures such as the military, corporations, churches, and in more abstract areas such as general problem solving (Figure 5). Some of the simplest hierarchies appear in knowledge representation, resulting from natural taxonomies such as "part-of", "is-a" or "kind-of". Similarly, spatial decompositions such as topographical maps or geometric descriptions of a robot environment treated at various levels of resolution provide examples of hierarchical decomposition. Approaches to complex problems such as speech and image understanding invariably involve levels, with the lowest ones associated with "primitives", the middle ones with various patterns and the highest ones with symbolic interpretations. In [1], the organization of functions in a robotic environment such as an automated factory is described as a hierarchy, with high levels related to planning and reasoning about the task and low levels identified with servo control.

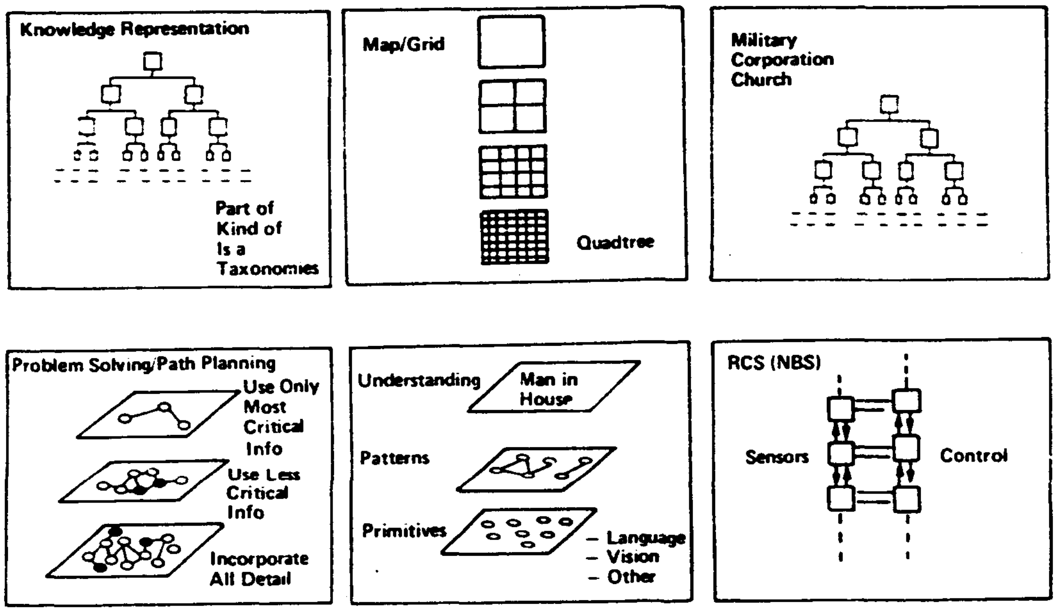


Figure 5. Examples of hierarchies

After looking at several hierarchies some general properties emerge. In more complicated scenarios, such as the decompostion of tasks for a robot, the highest levels are associated with reasoning and planning typically using very general global or abstract knowledge. As a result the highest levels tend to rely on thoughtful, slow, symbolic processing (Figure 6). The overall goals of the system are set at the highest levels, typically considering many alternatives, representing the "goal-driven" aspects of the system. In contrast, the lowest levels are associated with "action", using specific local knowledge. At the lowest levels are numeric, algorithmic processing that results in fast reflex action representing the "data-driven" aspects of the system. The overall hierarchy usually represents a decomposition of functions that is equivalent to the successive reduction of a problem into smaller and easier subproblems.

The dynamics in a hierarchy become apparent when it is realized that information can flow in either direction, creating loops that can straddle several levels (Figure 7). Furthermore, each level must be able to communicate with levels above and below, possibly requiring separate, local, languages, and also possibly swamping the middle levels with too many messages.

A hierarchical approach is not always easy to apply. For complex systems, defining the appropriate levels is quite

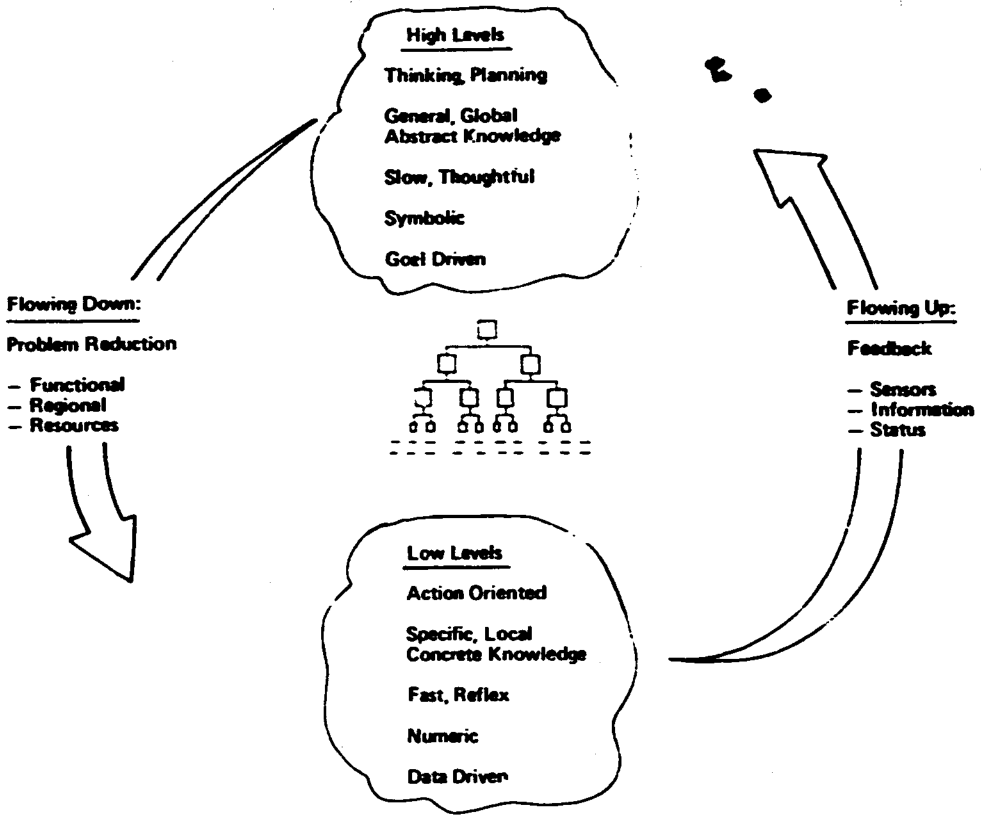


Figure 6. General aspects of hierarchies

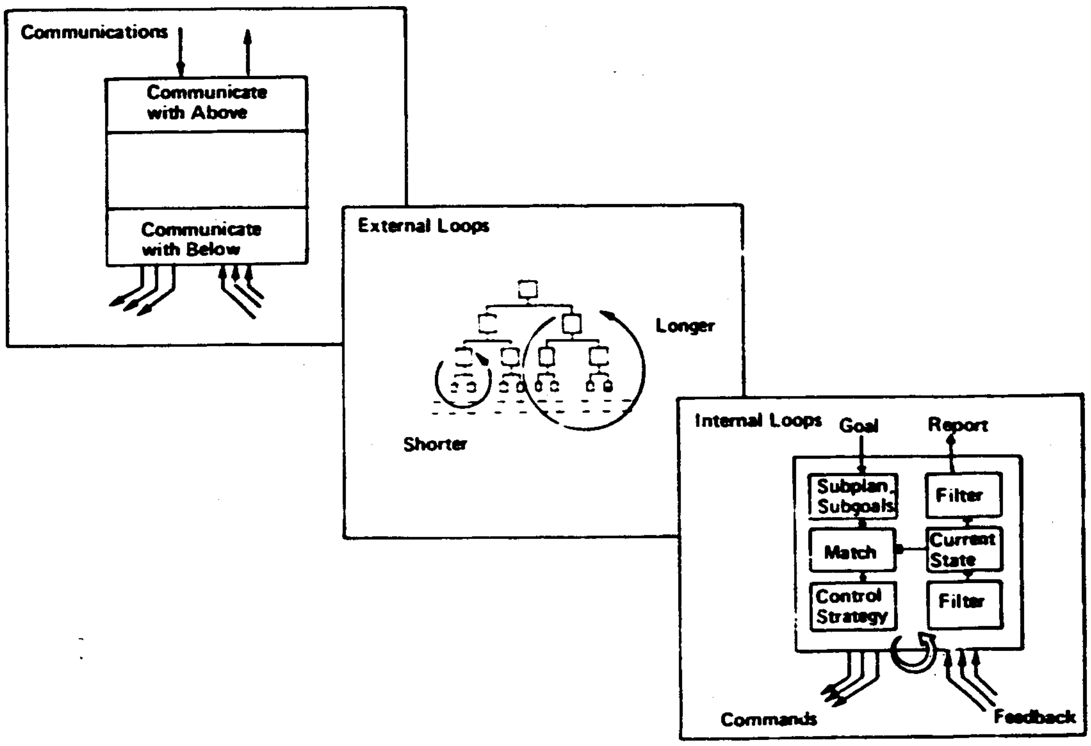
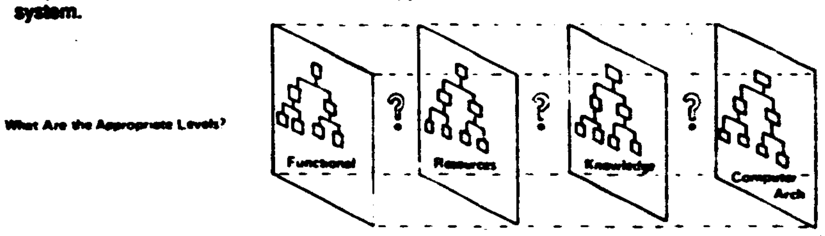


Figure 7. Interlevel communication and loops within hierarchies

complicated, running up against the same difficulties associated with general problem solving and knowledge representation. The problem reduction inherent in the hierarchical decomposition can fall short because of complex interrelations among subproblems. Nevertheless, a hierarchical approach is extremely powerful in its ability to guide the design of a complex system.



- The Levels Are Relatively Easy To Define for Taxonomies, Variable Resolution Maps, Independent Functions (Work Cells), etc
- For Complex Situations, We Are Up Against
  - General Problem Solving Problem Reduction
  - Knowledge Representation

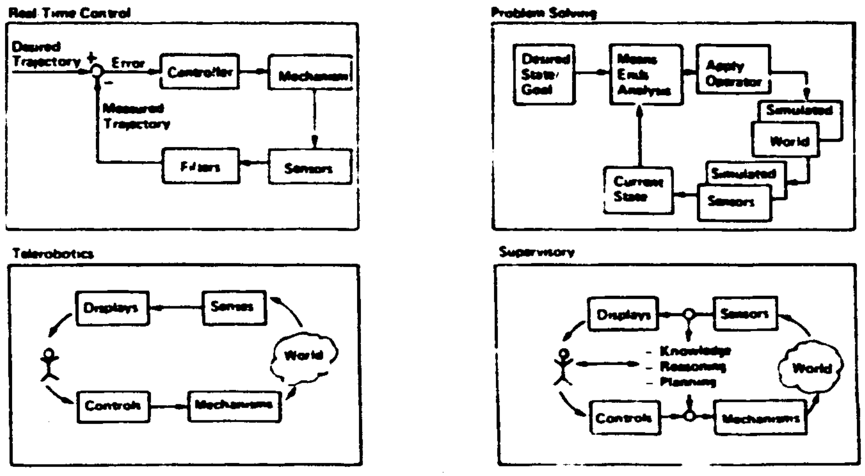
If Levels Are Not Defined Properly, The System Could Easily Get Swamped with Message-Passing Bottlenecks

Figure 8. The challenge of hierarchies

3.2 Feedback Loops

While the use of hierarchies reflects the need to decompose large systems into smaller tractable subsystems, feedback reflects the need to account for uncertainty. If a human operator or an autonomous system had perfect knowledge there would be no need for feedback since things would always go according to plan. Unfortunately, knowledge is often incomplete and erroneous and as a result actions must be followed by some form of checking.

The basic process underlying a feedback loop consists of two steps, repeated until successful : 1) using feedback, either from sensors or other sources of information, compare current state with desired state; 2) based on the results of the first step, decide on an action that will move the system closer to the desired state. Figure 9 depicts several types of feedback loops.



- Feedback Helps Account for Uncertainty
- The Emphasis Is on Incremental Progress Toward a Goal

Figure 9. Feedback loops

The first step assumes that sufficient information can be gathered from the particular domain of interest (ie. "world") and represented in a form that can be compared with some representation of the goal. For example, a simple servo control loop matches a desired with a measured trajectory to create an error signal for the "controller" which attempts to drive the error to zero by the proper actuation. In this way a relatively simple model of the "world" can be very effective. By focusing on the critical information as represented by the error signal, a rapid response can be easily formulated; but robust behavior (such as adaptation to load changes, collision avoidance and compliant control) is difficult to achieve without detailed dynamic models. That is, in most cases the decision on what to do at each iteration is not obvious. Although the iterative nature of the feedback loop implies incremental or local progress toward the goal, global knowledge must be used as well to avoid dead ends and local extrema.

Another basic example is the "means-ends" analysis used as a problem solving paradigm. Here, the representations of states, goals and available operators can be quite abstract, but the approach can be described in terms of a loop. A comparison of the current with the desired state detects a difference that is used to select operators that can reduce the difference. Typically, this produces a search process as various operators are tried out on a simulated "world". Once again, though, one of the major difficulties in applying such a procedure is the need to recognize global contraints at the incremental level.

Although there are various problems in applying feedback loops, it is clear that they are very powerful as a means of isolating and representing the critical aspects of a system, especially when uncertainty plays a major role.

4. Example : A SUPERVISED ROBOT

Our example is a "go-fetch" robot. The point of this example is to demonstrate that simple sounding tasks are tremendously complicated when considered for automation. Although various hierarchies and feedback loops can be readily identified, the interrelationships among them creates complexity that often requires human level intelligence.

Consider the task as beginning with a command to go get a particular object and deliver it to some specified location. The highest levels of a functional hierarchy might break the task into three subtasks : search for the object; navigate in the environment; manipulate the object (Figure 10). One of the basic problems already is that each of these will require an ability to reason about the task and the environment that is difficult to specify, and furthermore, there will be a need for the robot to communicate with the supervisor and possibly other robots.

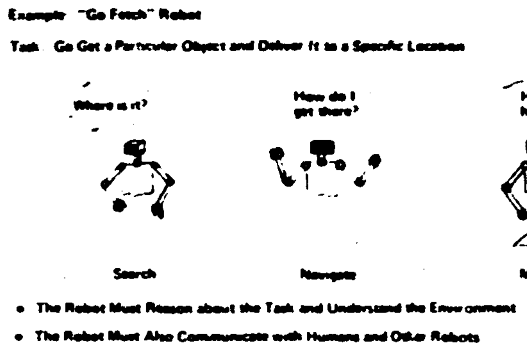

Example "Go Fetch" Robot
Task: Go Get a Particular Object and Deliver It to a Specific Location
Where is it?
How do I get there?
How do I handle this thing?
Search
Navigate
Manipulate
The Robot Must Reason about the Task and Understand the Environment
The Robot Must Also Communicate with Humans and Other Robots

Figure 10. Typical task

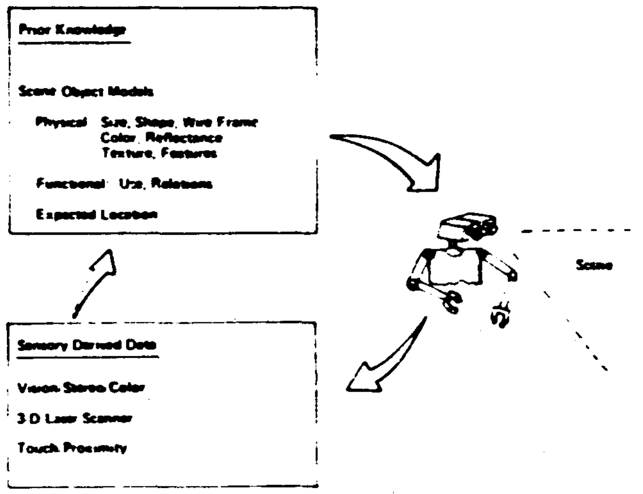


Figure 11. Search

4.1 Search

Unless the environment is extremely constrained, searching for an object will rely primarily on vision (and possibly range images). This sensory data must be processed and compared with internal models of the object's physical properties, functional aspects and expected location. In general, this is known to be a very difficult task, involving hierarchical structures that attempt to organize knowledge driven processes arising from expectations, and the data driven processes arising from the reduction of sensory data into image primitives such as edges, lines and surfaces. Searching for objects will require a significant amount of help from the human supervisor.

4.2 Navigate

Navigation entails the interplay of an interesting combination of information : 1) prior knowledge of the environment, typically described in global terms since the details would be hard to keep track of (for example, a description of the structural components such as the main passageways and known beacons could be available, but small obstacles would have to be discovered); 2) The robot's sensors such as vision and proximity would have to provide the local information about the environment, and use it to avoid obstacles and recognize landmarks; 3) internal motion sensing could be used to give the robot an estimate of its position independent of the external sensors; 4) finally, planning paths that negotiate local difficulties while getting to the goal requires geometric reasoning capability.

In order to successfully navigate, the robot must use all these sources of information, recognizing landmarks, obstacles, tight spots and dead ends while reasoning about progress toward the goal. Once again, a navigation task would be easy in a very structured environment, but in real applications the problem is quite difficult, requiring the human operator's assistance.

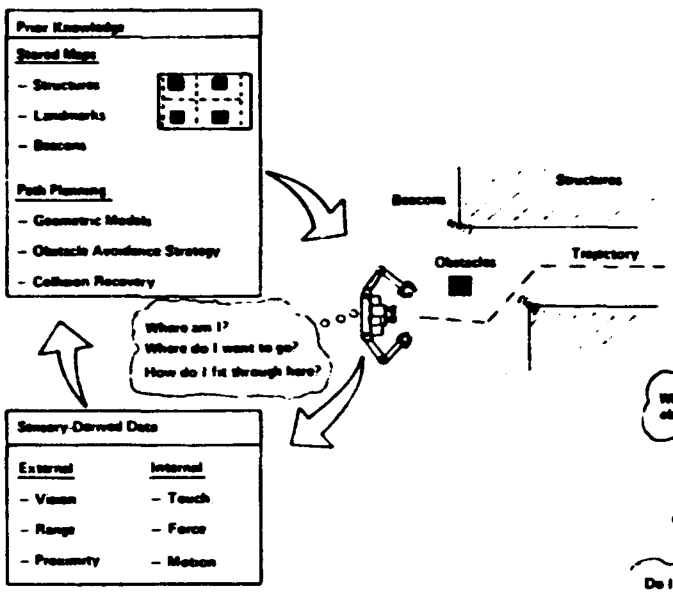


Figure 12. Navigate

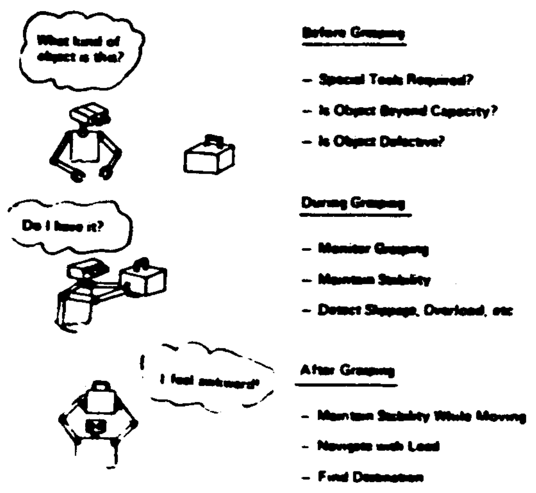


Figure 13. Manipulate

4.3 Manipulate

If the robot is successful in getting to the object (Figure 13), it must, among other things: determine whether neighboring objects must be moved to get at it, inspect the object to see if it is defective, detect the need for special grappling tools, find grasp points, and estimate the weight and strength of the object. After grasping the object, sensors must monitor overall stability and detect slippage and overload. Once the object is securely in the robot's grasp, the robot must navigate to the goal as a different dynamic system which may alter the chosen route. The majority of these subtasks will require the intervention of a human operator.

4.4 Complexity

Even a cursory view of the go-fetch example indicates that there is a tremendous complexity involved. Upon analysis, the various subtasks are interrelated, and the identification of various structures such as hierarchies and feedback loops can only have limited effect, since even they are tangled among each other. Defining "primitive" tasks, such as peg-in-hole and turn-crank, is an excellent way to control the complexity. Although this approach is somewhat limited by the discrete primitives and combinatorial explosion, it represents a solid hierarchical approach that can build up complex tasks from easily understood primitives (see [3] for an example of this approach).

It is not uncommon for a robotics engineer to gradually realize that what at first appeared to be easy to automate is extremely complex, defying all attempts at automation. The history of vision and natural language research provide classic examples, but this phenomenon runs throughout the robotics world.

Humans have an uncanny ability to simplify things. The environment presents an extremely complex array of information. Humans process it, extract the important details, attach symbols, and reason about the symbols. This innate ability to simplify and deal powerfully with abstractions is an advantage, but it also has a tendency to create the illusion that things really are simple. The robotics engineer often finds himself wrestling with exactly this problem. The natural language used in describing everyday tasks does not - and should not - emphasize the complexity involved in automating them. It is up to the robotics engineer to tear down these "simplifying" abstractions to ascertain the primitives and underlying structure that is amenable to automation.

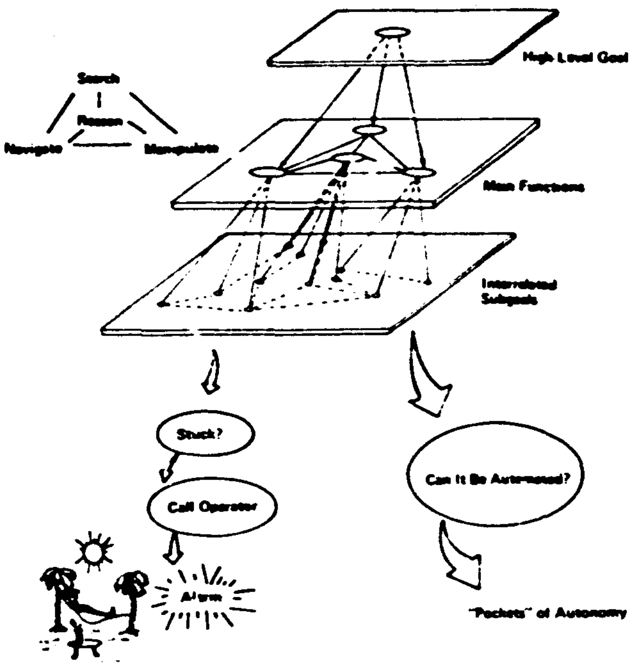


Figure 14. Hierarchy

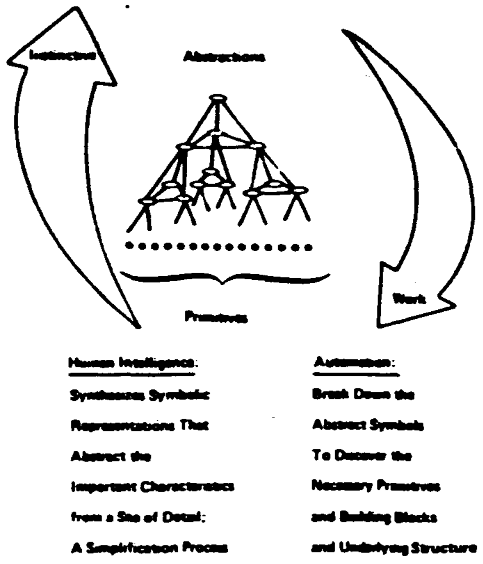


Figure 15. Complexity

4. PARALLELISM IN SUPERVISORY CONTROL SYSTEMS

Real-time execution of distributed supervisory control systems will require implementation on parallel and advanced computer architectures. This requirement is driven by the types of processing and processing requirements in supervisory control systems. As is the case with understanding and describing a problem, it is useful here to think in terms of levels or hierarchies. Since the types of processing in a supervisory control system are so diverse, ultimate implementation will require a network of special purpose, parallel, and serial architectures. In the remainder of this section we briefly consider the types of processing in a supervisory control system and the suitability of various architectures to these levels. Furthermore, we suggest that the biggest problem facing parallel implementation of supervisory control systems is lack of mature software tools and techniques for implementation of these systems on distributed and parallel architectures.

5.1 Processing Requirements

The processing requirements of a supervisory control system can be roughly divided into three levels which suggest the types of architectures which are beneficial. These levels are not necessarily mutually exclusive as tasks may occur which straddle several levels.

5.1.1 Low Level Processing

This level is is connected with interaction with the environment through sensors and commands to effectors and actuators. Typical tasks performed at this level are: sensor data processing/filtering such as image feature extraction (convolutions), associative retrieval, and simple trajectory generation. This level is generally characterized by the following requirements: 1) High bandwidth I/O rates; 2) Numeric processing; 3) Algorithmic processing rather than search among alternates; 4) Identical, repeated operations on bounded regular data structures.

Special purpose hardware is often applied to much of the processing at this level. The special purpose hardware can be serial or parallel, but typically is designed for very specific functions such as convolutions and does not support general purpose processing.

SIMD processors [4] are particularly well suited to this level of processing. Examples of this class of parallel processor are the Illiac IV and MPP [5]. These machines are particularly well suited for problems involving identical operations on bounded regular data structures such as matrix multiplication, parallel sorting, and fast fourier transforms.

MIMD processors [4] such as the BBN Butterfly and Cm* have also been applied to these types of problems, however, they have not achieved the same level of performance as SIMD machines [5,6].

5.1.2 Mid Level Processing

This level concerns more general purpose processing than either the lower or higher levels. This level generally performs operations on filtered, preprocessed sensory data from the lower level. Typical functions performed here are connectivity analysis (object detection), simple vehicle route planning, and optimization problems using techniques such as dynamic programming. This level is characterized by the following requirements: 1) Medium bandwidth I/O rates; 2) Varied operations on regular or irregular data structures containing similar data types; 3) Optimized choice among a bounded set of alternatives.

SIMD machines can be effectively applied to this level of processing if the data to be operated on can be allocated uniformly across the processing elements and the processing performed on the data elements is similar. In the absence of these, many of the elements in the processing array remain idle too much of the time resulting in inefficient use of the system. In such cases, MIMD processors can be applied.

The MIMD, or Multiprocessor architectures, are the most general parallel processors and are capable of executing multiple control threads in parallel. Tightly coupled (shared memory) MIMD computers are applicable when the problem requires a high degree of interprocess communication and a high degree of sharing of objects/resources in the system. These machines can support a finer degree of process granularity due to the high interprocess communication bandwidth (on the order of memory bandwidth). If the problem calls for minimal data/resource sharing, then loosely coupled multiprocessors are applicable and will not be subject to performance degradation due to resource contention as is often the case in a shared memory system.

5.1.3 High Level Processing

Typical functions performed at this level are route and path planning by heuristic search such as A*, mission planning, logical inferencing, and object recognition and understanding. This level operates on data that has been preprocessed by each of the lower levels and typically controls the lower levels. It is characterized by: 1) Low bandwidth I/O rates; 2) Varied operations on different data types and irregular data structures; 3) Symbolic processing; 4) Heuristic search of a large number of alternatives.

MIMD machines are the most widely applied architectures to this level of processing. This is due to the flexibility in these systems since different processors can be applied to different aspects of the problem. Examples are parallel branch-and-bound search and parallel execution of logic programs [7].

5.4 Parallel Programming

The biggest challenge facing the use of parallel processors for supervisory control systems is the complexity of programming many parallel machines. This difficulty arises because programmers must often familiarize themselves with low level details about the parallel architecture as well as parallel programming techniques before they can become effective users of a parallel machine. In order to take advantage of the parallel hardware, conventional programs must often be rewritten and embedded with system calls for memory and process allocation. In addition, when software is ported from one parallel machine to another, it must again be rewritten to account for a different architecture and set of parallel language features. Parallel programming technology is maturing, and advances in the following areas will result in more efficient programming:

1) Optimizing compilers for serial languages. Such compilers will allow for portability of existing codes to parallel machines by making the underlying parallel architecture transparent to the user. Furthermore, this will allow for portability of software across different parallel architectures. Progress in this area has been made for both conventional languages such as C and Fortran [8] as well as AI languages such as functional languages and Prolog [9]. We are currently working on a parallel interpreter for Prolog execution on the Butterfly which automatically optimizes these programs for parallel execution.

2) A common set of parallel programming and language abstractions which can be applied to the various languages and ported across different parallel machines. This coupled with optimizing compilers allows for the use of parallel architectures at multiple levels of expertise. Novices can write programs which are automatically parallelized and optimized for the particular architecture. As they become more experienced and knowledgeable about the particular architecture being used, they can optimize programs even further by embedding these abstractions - or compiler directives - within their programs.

3) A common set of resource sharing and process intercommunication protocols which will allow for programs or processes operating on heterogeneous processors in a network to communicate.

6. CONCLUSION

There are many open issues concerning the development of supervisory systems for Space Station but the key factor is the man-machine interplay. System design methodologies must allow the humans who possess the majority of the intelligence to use the subsystems as tools. The subsystems should amplify human capabilities by allowing the operators to direct, troubleshoot and enhance system performance. From a conceptual standpoint, both hierarchical decompositions and feedback loops are common to man and machine thereby providing a simple framework for interaction. Unfortunately, typical tasks are quite complex, requiring that humans solve them. As experience is gained, the human operators will be in the position to encode new procedures, provided that they have the right development tools.

Parallel implementation of these supervisory control systems will provide the performance necessary to sustain real-time execution of such systems. Ideally, distributed computer systems consisting of serial, special purpose, and parallel processors will allow for optimal performance of the different processing requirements of such systems. The major difficulty, however, in using parallel architectures effectively is the complexity of programming them. Recent and forthcoming advances in parallel software technology will reduce the impact of this problem and allow for parallel implementation of these systems.

7. ACKNOWLEDGMENTS

We would like to express our thanks to other members of the Advanced Automation Technology group for their help in writing this paper: Wendell Chun, John Barnes, Peter VanAlta, John Tietz, Kayland Bradford and Roger Schappell.

8. REFERENCES

[1] Barbera, Fitzgerald and Albus, "Concepts for a Real-Time Sensory Interactive Control System Architecture," Proceedings of The Fourteenth Southeastern Symposium on System Theory, April, 1982.

[2] W. Erickson and P. Cheeseman, "Issues in the Design of an Executive Controller Shell for Space Station Automation," Optical Engineering, November, 1986, Vol. 25, No. 11.

[3] J. Barnes, "A Task-Based Metric for Telerobotic Performance Assessment", Workshop on Space Telerobotics, Jet Propulsion Laboratory, Pasadena, California, January 20-22, 1987.

[4] M. Flynn, "Some Computer Organizations and Their Effectiveness", IEEE Transaction on Computers, C-21, no. 9, Sept. 1972, pp. 948-960.

[5] K. Hwang and F. Briggs. "Computer Architecture and Parallel Processing," McGraw-Hill, 1984.

[6] R. Rettberg and R. Thomas, "Contention Is No Obstacle to Shared Memory Multiprocessing," Com. of the ACM, Dec. 1986, Vol. 29, no. 12, pp. 1202-1213.

[7] B. Wah, G. Li, and C. Yu, "Multiprocessing of Combinatorial Search Problems," IEEE Computer, June 1985, pp. 93-108.

[8] D. Padua and M. Wolfe, "Advanced Compiler Optimizations for Supercomputers," Com. of the ACM, Dec. 1986, Vol. 29, no. 12, pp. 1184-1202.

[9] J. Conery and D. Kibler, "Parallel Interpretation of Logic Programs", ACM Symposium on Functional Programming Languages and Computer Architecture, Oct. 1981.

188578 S6-63 68

An Architecture for Heuristic Control of Real-Time Processes

P. Raulefs and P.W. Thorndyke
FMC Corporation
Santa Clara, CA 95052

FY 563409

Abstract *Process management* combines complementary approaches of heuristic reasoning and analytical process control. Management of a continuous process requires monitoring the environment and the controlled system, assessing the ongoing situation, developing and revising planned actions, and controlling the execution of the actions. For knowledge-intensive domains, process management entails the potentially time-stressed cooperation among a variety of expert systems. By redesigning a blackboard control architecture in an object-oriented framework, we obtain an approach to process management that considerably extends blackboard control mechanisms and overcomes limitations of blackboard systems.

1. Introduction

Many future military and space applications will require control of autonomous or semiautonomous systems operating in dynamic environments. *Real-time heuristic control* is the task of applying knowledge-intensive reasoning methods to supervise and manage such dynamic systems. Effective control requires monitoring and assessing rapidly changing situational data from the environment in which the system operates, developing and evaluating planned actions, and executing those actions to achieve desired goals. This requires the control system to maintain a model of the operating environment and to interpret sensor data and planned actions in light of this model.

For operating environments too complex to model using traditional process control models, knowledge-based qualitative models can provide an effective means of approaching the tasks of dynamic situation assessment and planning. However, existing AI-based techniques for data interpretation, situation assessment, and planning cannot accommodate requirements for operating in time-stressed situations where critical conditions and assumptions may be varying dynamically. Further, existing methods cannot easily accommodate the synchronization of interacting processes engaged in situation assessment, planning and execution control.

Blackboard control systems [4, 2] provide a first step towards solving these problems. Our analysis (Section 3.2) shows, however, that they lack the facilities needed to meet requirements of real-time responsiveness and hybrid, layered system architectures.

After analyzing the generic real-time heuristic control problem more closely in Section 2, we present in Section 3 the Heuristic Control Virtual Machine, or HCVM, as an approach that considerably extends blackboard control architectures to solve problems associated with their shortcomings. The underlying idea of the HCVM is to cast a blackboard control architecture into an object-oriented framework [3] and then exploit the additional features provided by object-oriented programming to design solutions to the above problems.

We show how flexible, even dynamically variable interaction modes between objects can be used to achieve real-time responsiveness. Using objects with standard interfaces as 'wrappers' around both heuristic and analytical procedures allows building hybrid systems that integrate knowledge-based and conventional process control components. Control reasoning using explicit notions of time and temporal relations about activities in *both* the controlled environment *and* the control system provides an approach to meet timeliness requirements.

Finally, current experience suggests a repertoire of methods applicable to a broad domain of applications. Section 4 discusses how the HCVM could provide a computational model for a more comprehensive software engineering environment for building process management applications.

2. The Process Management Problem

Process control is concerned with monitoring and assessing changes in dynamic systems, followed by planning and executing actions to control dynamic systems. Analytical techniques employed by conventional process control utilize mathematical models of controlled systems, expressed in terms of differential equations of time-varying functions relating sensor data and controlled variables. Conventional process control is limited to applications where such models and sufficient sensor data are available. Dynamic systems with this property are often successfully managed by human operators using heuristic expertise instead of analytical reasoning. In fact, a great number of process control systems operate in environments where they are complemented by human operators. This observation suggests to view analytical and heuristic process control as complementary. We refer to *process management* as the combination of both.

Heuristic techniques play a dual role in process management: *Supervisory control* applies heuristic judgment to make decisions about results elaborated by analytical *or* heuristic methods, and *direct control* interprets sensor data to make and execute control decisions.

Applications impose two types of requirements on process management systems. *Functional requirements* consist of functions and tasks to be performed. *Performance requirements* constrain time and other resources available to carry out tasks. *Real-time performance* is a particular challenge difficult to overcome by knowledge systems.

Several characteristics of intended application domains influence functional and performance requirements. We consider domains where up to tens of thousands of sensors generate data about many fewer subsystems or components. Data sources are distributed over space and time, such as satellites producing bursts of data every few hours. Both continuous and discrete data occur with often dynamically varying data rates and urgency. Data are unreliable because of potential sensor and communication failures and noise.

Functional requirements concern monitoring the environment, situation assessment, planning and re-planning, plan execution control, and coordinating and managing the system's own activities.

1. **Monitoring.** The monitoring function includes:
   * ***Monitoring the environment:*** Receive incoming data; filter, fuse, interpret and abstract incoming data for data reduction, critical event and alert generation to interrupt, override or pre-empt current activity.
   * ***Monitoring system performance:*** Inspect current process management activities, possibly in conjunction with results of monitoring incoming data, to detect critical control events and raise control alerts.

2. **Situation Assessment.** The process management system must diagnose the current situation of the environment; determine how it evolved from previous history, and predict future developments; determine goals to be achieved by process management activity. Situation assessment is done dynamically, i.e. it extends, modifies and revises previous assessment, and it will have to incorporate new information when it becomes available.

3. **Action Planning.** From results of Monitoring and Situation Assessment, produce a plan of process management actions to reach the goals. Plans will include *conditional actions* with alternatives if conditions do not hold. *Re-planning* modifies plans to adapt to dynamically changed situations in the environment (known when new data have been received) *or* the process management system (when new assessments, critical events have been produced).

4. **Plan execution and Monitoring.** Plans produced by Action Planning will be high-level descriptions yet to be compiled into executable instructions to effectors or the user interface. Monitoring plan execution determines whether actions are carried out as specified and have anticipated effects.

5. **Managing Own Activities.** The system must allocate resources to the above tasks, start their activities, and monitor their progress. To do so in a timely fashion, it must assign priorities, schedule activities, and manage interrupts.

The performance requirements specific to real-time systems concern their responsiveness to changes in their environment:

1. **Timeliness.** All scheduled activity must be completed sufficiently early to have its desired effect on the environment. Any task that completes successfully, but too late to have its intended impact violates the timeliness requirement.

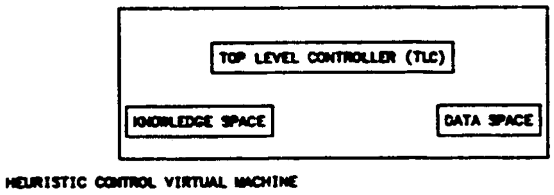
2. **Interruptibility.** Interrupts suspend or terminate activities in favor of activating others. Interrupts require a model separating atomic, non-interruptible from composite, interruptible tasks that is compatible with the timeliness requirement. Composite tasks can be interrupted to give control to tasks of higher priority. Interrupts are accompanied by strategies for orderly resumption of tasks that preserve the integrity of the overall task.

Real-time performance cannot be achieved by merely optimizing hardware and software to operate fast. For timely responses, a process management system has to plan and monitor the temporal performance of its own activities together with those of its environment. Interruptibility requires the system architecture to provide facilities for setting up tasks of variable and even dynamically varying size, and for managing suspension and resumption of tasks to transfer control. Interruptibility contributes to achieve timeliness by providing a mechanism to postpone tasks and move up others to complete more time-stressed tasks earlier.

3. HCVM: An Architecture for Real-Time Process Management

3.1 HCVM Overview

The Heuristic Control Virtual Machine (HCVM) is a knowledge-processing operating system designed as an environment for engineering real-time, knowledge-intensive process management applications. The HCVM (see Fig. 3-1) consists of three collections of objects: The top level controller (TLC), the knowledge space and the data space. The knowledge space consists of knowledge handlers that embody heuristic expertise and analytical procedures. The elements of the data space are data handlers carrying information obtained from outside communication or produced by knowledge handlers. All activity of knowledge and data handlers is managed by the TLC.



```
<TOP LEVEL CONTROLLER>  :?:= <knowledge mgr> <data mgr> <control mgr>
                                   <control plan>

<KNOWLEDGE SPACE>   ::= "a collection of knowledge handlers"
<KNOWLEDGE HANDLER> ::= <trigger condition> <precondition> <body>
<TASK HANDLER>      ::=                     <precondition> <body>

<DATA SPACE>        ::= "a collection of data handlers"
<DATA HANDLER>      ::= <information> <local data manipulation methods>
```

Figure 3-1: HCVM Overview

A Knowledge handler is activated by the TLC when (1) it has been triggered, (2) it has been scheduled for execution by the TLC, and (3) is in a state where its <precondition> is true. Trigger conditions are propositions about information stored in data handlers. Executing a knowledge handler consists of carrying out its <body> that may result in changes in the data space and communication with the environment external to the HCVM. Information stored in data handlers changes when knowledge handlers produce such changes (e.g. after inferring a new diagnosis), or when new information from the external environment is communicated to the HCVM and affects a particular data handler (e.g. after having received new sensor data). When a data handler has received some update information, it may execute some internal code to, for example, check consistency, perform some smoothing and averaging operation on new and previous values, or determine qualitative abstractions from numerical values.

The TLC repeatedly executes a cycle of invoking the knowledge, data and control manager modules. The knowledge manager arranges executing knowledge handlers that have been marked executable by the control manager. Then the data manager carries out resulting changes in the data space. The control manager maintains a control plan that specifies which knowledge handler(s) are to be executed next and in which order. To arrive at a control plan, the control manager may perform extensive control planning about tasks to be done in the external environment and the HCVM. To do this, the control manager may employ knowledge and data handlers, thus restricting a series of control cycles to control reasoning.

3.2 Shortcomings of Blackboard Architectures

The HCVM as described above is essentially a *blackboard control architecture* [4] cast into an object-oriented programming framework. Knowledge handlers correspond to knowledge sources, the data space to a combined domain and control blackboard, and the TLC includes the facilities for blackboard control reasoning. However, this does not suffice to meet the requirements described in the previous section. We now explain the mechanisms by which the HCVM considerably extends blackboard control architectures in connection with the problems they are meant to solve.

Considering the requirements posed in section 2, blackboard architectures have particular difficulties in handling the following problems:
Interruptibility of tasks in any computational model is determined by the grain size of non-interruptible, atomic units. Knowledge sources are such units in blackboard systems: Once a knowledge source is activated, it is executed until it terminates. A knowledge sources provides a facility to collect closely related expertise and procedures so that interactions between knowledge sources are restricted to less closely related capabilities. This principle of organizing knowledge can be in conflict with timeliness and interruptibility requirements to structure activity into tasks of sufficiently small grain size.

Timeliness is difficult to achieve because blackboard control is excessively expensive when scheduling tasks does not require extensive control reasoning, such as for tasks that are known to be executed in a sequence, one after another.

Responsiveness to an incoming glut of data with dynamically varying information density and urgency is difficult to achieve with blackboard systems. Incoming data need to be analyzed first before control planning can schedule activities to respond, implying at least two execution cycles of the blackboard system that even may be interrupted by high-priority alerts.

Hybrid and layered system architectures where activities such as blackboard control, or those of individual knowledge sources are handled by conventional "control boxes" or blackboard systems in themselves are difficult to build as blackboard Systems. Interactions, including property inheritance, switching activations from lower to higher level tasks and vice versa, and managing interrupts between different levels have to go through blackboard control cycles although this is often not an adequate mechanism to use.

To help solve these problems, the HCVM introduces several additional mechanisms and features.

3.3 Synchronous and Asynchronous Interaction Modes between Handlers

Besides the *blackboard mode* of triggering, scheduling and then invoking knowledge handlers, the HCVM provides for two additional modes of interaction between modules: First, *direct message passing* activates a module by sending it a message, as usual for object-oriented systems. The second is *remote routine call* by which an object has another object execute a routine and then return a result to the caller. For the first two modes, activities of objects causing those of others are not directly linked to the activated objects. In the blackboard mode, both activities are separated by arbitrary other activities. For the blackboard and the message passing mode, invoking and invoked object may logically and physically proceed concurrently, and their activities are not synchronized. For remote routine call, the caller and callee are synchronized by the caller being suspended until the the callee returns control to resume the caller.

The HCVM adds an additional feature to these standard mechanisms of interaction between modules: interaction modes may vary dynamically. For example, a knowledge handler may be invoked by blackboard control triggering and activating it, by some other handler sending it a message (knowledge handlers are *named* objects), or by some other handler invoking a routine (e.g., to execute a set of rules) that the knowledge handler provides.

Handlers in the knowledge space that are not supposed to be invoked by blackboard control are realized as task handlers in the HCVM: A task handler is like a knowledge handler, but it lacks a <trigger condition> (see Fig. 3-1).

Knowledge and task handlers may have bodies built up from task handlers, with control passing from one task handler to another in a way that may vary dynamically (e.g., by evaluating conditions to decide which one is the next to execute).

Although these facilities appear to be straightforward extensions of blackboard systems by incorporating additional standard module interaction modes, they open a significant problem of designing coherent interrupt management ~~schemes~~. For example, a knowledge handler may be interrupted after having finished some task handler. Resuming execution of this knowledge handler may occur after significant changes, suggesting to not simply continue with the next previously scheduled task handler. A thorough discussion of techniques for such capabilities goes beyond the scope of this paper. We note, however, that they provide flexible interrupt schemes to attend rapidly changing needs in an optimal way.

3.4 Multiple HCVMs and Heterogeneous Handlers

Large-scale process management applications may lead to knowledge handlers of considerable size, blackboard control that is extremely complicated, and I/O-communication that is difficult to handle within a single HCVM-object because of immense data rates. Each of these activities is best performed by a system providing all the facilities offered by an HCVM.

The other problem of process management applications is that knowledge-based reasoning closely interacts with traditional process control equipment, such as PID-controllers and other types of "control boxes".

The object-oriented nature of the HCVM allows quite easily to treat these issues by two approaches:

* As an HCVM is just an object, knowledge and task handlers, as well as a TLC or its components may in themselves be full-fledged HCVMs. As mentioned above, a problem beyond the scope of this paper is how to manage interactions (such as property inheritance and interrupts) between "ordinary" and HCVM-knowledge/task handlers.

* A knowledge or task handler may encapsulate a physical process control unit, where HCVM-interactions with such a handler may consist in communicating sensor data and set points.

3.5 Communication with External Environment

Unforeseen dynamic changes in an external environment with the properties described in section 2 lead to two problems of dynamically changing needs for communication between environment and process management system.

* Varying information density of incoming data require process management to allocate varying resources for buffering and preprocessing. Much of preprocessing concerns the monitor functions of screening and compacting input data, where screening discards less relevant data as early as possible.

* Varying urgency of incoming data needs to be determined and anticipated by process management for planning and re-planning its tasks and resource allocation. In particular, communication capacity has to be dynamically adjusted to varying urgency so that high-bandwidth communication is only devoted to messages of highest importance.

The HCVM provides four mechanisms of increasing complexity to support solving these problems:

1. I/O-communication is handled by a "Communication Manager" executed as part of the TLC-loop. For each TLC-cycle, the Communication Manager empties all I/O-buffers by distributing incoming data into the HCVM, and sending data in output buffers away. This method is adequate when information densities and urgencies of communicated data vary approximately in the same way as the amount of time that is spent on processing the other TLC-tasks.

2. I/O-communication is done by a knowledge handler triggered when data with specified properties (e.g., sufficient volume and urgency) appear in I/O-buffers. This is adequate as long as there is no need to differentiate between processing rates in different I/O-buffers.

3. I/O-communication is done by several knowledge handlers specialized to process data for particular I/O-buffers.

4. I/O-communication is done by knowledge handlers activated both synchronously and asynchronously.

3.6 Control Reasoning Support

Process management is concerned with three types of actions. *Domain actions* are carried out in the environment and their effects are observed in the process management system. *Control actions* are executed by process management to influence the course of domain actions. *Self control actions* are those guiding the process management system to perform its own tasks. The task of 'managing its own activities' (see section 2) recursively repeats monitoring, situation assessment, planning and plan execution.

The HCVM is well-suited to support control reasoning. Knowledge handlers embodying explicit control knowledge are triggered when data handlers record relevant conditions and events for environment and process management. A collection of such knowledge and data handler classes to be instantiated with creating an HCVM constitutes a *control framework* that drives the control manager in the TLC to select and schedule process management activities. Knowledge handlers could either themselves be of different grain-size, or be composites of knowledge/task handlers focusing on particular dynamic situations. The control manager of the HCVM dynamically manages priorities and temporal dependencies among knowledge handlers scheduled for execution, providing a mechanism to respond to dynamically changing needs.

4. Mapping Architecture to Application Structures

4.1 Multiple Agents Reasoning About Hierarchical Component Structures

We have applied the HCVM to the development of a continuous process management problem of mineral refinement. This domain has the following characteristics:

* The environment consists of hundreds of components with thousands of sensors gathering data about the states of components.

* Components are organized in a functional hierarchy where interactions among functions coincide with control and data flow.

* Each major component requires independent process management within given requirements dynamically influenced by other major components.

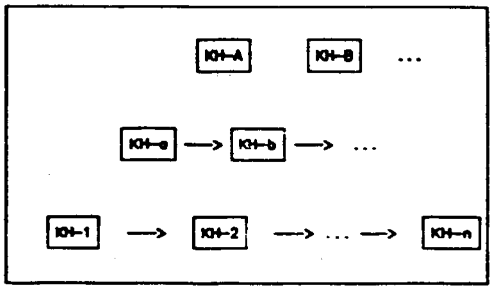


Figure 4-1: Hierarchy of Process Management Agents

Fig. 4-1 shows a process management system structure where a hierarchy of knowledge handlers reflects the component hierarchy. Each knowledge handler is an HCVM-object performing process management for a particular component. Higher-level knowledge handlers combine sensor data with monitoring and situation assessment results from lower-level agents to form more abstract, strategic assessments and plans.

Lower-level knowledge handlers (e.g., KH-1, KH-2, ... in Fig. 4-1) are executed in a fixed order which is implemented as sequential execution where trigger conditions are ignored. Interrupts, however, may lead to terminate execution of any such task, and resume execution of the *sequence* at any given point. Resumption is implemented as triggering knowledge handlers, scheduling the first in the sequence, and discarding the rest from the control plan (as subsequent elements of the sequence, they are executed, anyway).

4.2 Towards Process Management Engineering Environments

The HCVM provides a *computational model* in a more comprehensive architecture for real-time knowledge systems, as shown in Fig. 4-2. In this role, the HCVM is a software environment to implement the problem-solving frameworks that support building capabilities to perform the generic functions of process management described in Section 2.

Example
To describe how the HCVM supports such a system organization, we review how a real-time process management system implemented on an HCVM performs its tasks [6].

> Sensor data are received as separate streams of data packages in several input ports. A Communication Manager (see approach 1 in Section 2.5) turns data packages over to package-specific knowledge handlers for screening. Unless critical values are detected, packet handlers send relevant and possibly pre-processed data into the data space.
>
> Data handlers provide capabilities of a process data base management system: Data are stored in a way that supports retrieving historical information, such as "get the temperature distribution between the last sharp drop and subsequent sharp increase of pressure". Data handlers also perform data abstraction from numerical to qualitative values (quantization), smoothing and averaging operations, and locally decidable reliability checks.
>
> Knowledge/task handlers are activated upon the occurrence of changes in the data space, or upon the persistence of properties about data over a specified period of time. For situation assessment, knowledge/task handlers determine situations and events constituting higher-level descriptions about conditions and changes in the environment. Conditions and changes occur in the contexts of temporally extended patterns of situations and events, or *processes*. Situation assessment results in recognizing which processes are active, how they interact, and to which states they have progressed.
>
> Recognition of processes activates knowledge/task handlers that plan adequate responses. Ongoing planning may be interrupted and revised following alerts generated when critical values are observed when screening incoming data, or critical situations or events are inferred by situation assessment. Interrupts are frequently used to interleave situation assessment and planning. For example, planning may request more detailed information on the occurrence of an anticipated critical event before committing to on of several possible courses of action.

The example indicates how process management functions are accomplished by using generic capabilities, such as data abstraction, event and process recognition, response planning, and control management. A software engineering environment for process management would provide the capabilities comprising the problem-solving frameworks of Fig. 4-2.

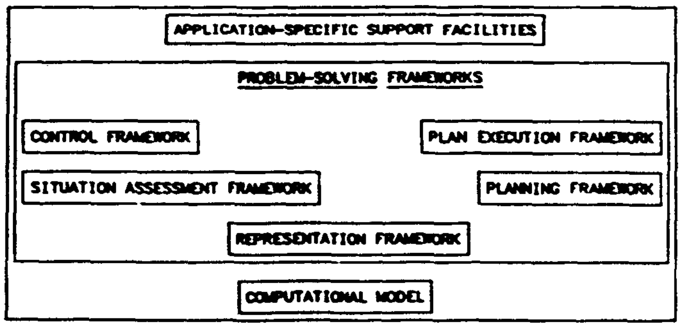


Figure 4-2: Organization of Real-time Knowledge Systems

5. Conclusions and Future Work

We have shown how the HCVM extends blackboard control architectures to overcome many of their shortcomings. Our work has so far indicated that the HCVM provides an adequate computational model to support the variety of tasks in process management, thus giving some evidence that it could be an appropriate basis for future real-time process management engineering environments. To become credible, this claim must be substantiated by actually building, applying and evaluating such systems.

As with blackboard architectures, the HCVM carries a significant computational overhead for simply managing its own structure. Unlike blackboard architectures, however, much of this overhead can be contained and even avoided by synchronous interaction modes between handlers. Empirical evidence needs to be gathered about the extent of such overhead, the improvements achievable by less expensive interaction modes, and the trade-offs with regard to lesser degrees of real-time responsiveness.

Another unexplored direction is distributing knowledge [5, 1] and data spaces across processors in distributed computing environments. A particularly interesting feature of the HCVM is that it involves highly synchronized communication via shared memory as well as loose message-passing interaction. Hardware architectures supporting both types of interaction provide appropriate support for these features, but there is a wide spectrum between tightly synchronous and loosely asynchronous interactions yet to be investigated.

6. Acknowledgements

The HCVM has been primarily developed by Teknowledge, Inc. in a cooperative project with FMC Corporation. The Teknowledge group was lead by Michael R. Fehling until October, 1986. The other members are Stephanie Forrest, Michael Wilber, and Wayne Caplinger. A previous version of the HCVM was developed by Michael Fehling and colleagues as the SCHEMER system. Michael Wilber (Teknowledge) implemented most of the HCVM, and Stephanie Forrest (Teknowledge) designed and implemented an application framework. Bruce D'Ambrosio, Anita Wong, and Khanh Ho from FMC contributed to the process management application system in our laboratory.

References

[1] Durfee, E. H., Lesser, V. R., Corkill, D. D.
Increasing Coherence in a Distributed Problem Solving Network.
In *Proc. of the 9th IJCAI-85*, pages 1025-1030. August, 1985.

[2] Erman, L. D., Hayes-Roth, F., Lesser, V. R., Reddy, D. R.
The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty.
ACM Computing Surveys 12:213-253, 1980.

[3] Goldberg, A. and D. Robson.
Smalltalk-80: The language and its implementation.
Addison-Wesley, Reading, Mass., 1983.

[4] Hayes-Roth, B.
A blackboard architecture for control.
Artificial Intelligence 26:251-321, 1985.

[5] Lesser, V. R. and Corkill, D.
Functionally accurate cooperative distributed systems.
IEEE Trans. Systems, Man and Cybernetics 1:81-96, 1981.

[6] Raulefs, P., D'Ambrosio, B., Fehling, M. R.,Forrest, S., Wilber, M.
Real-time process management of materials composition processes.
In *Proc. 3rd IEEE Conference on Applications of Artificial Intelligence.* IEEE, Orlando, Fla., Feb. 1987 (in press).

[illegible] S17-63
118

Hierarchical Control of Intelligent Machines Applied to Space Station Telerobots

NG 315720

J.S. Albus, R. Lumia, and H. McCain
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

A hierarchical architecture is described which supports space station telerobots in a variety of modes. The system is divided into three hierarchies: task decomposition, world model, and sensory processing. Goals at each level of the task decomposition hierarchy are divided both spatially and temporally into simpler commands for the next lower level. This decomposition is repeated until, at the lowest level, the drive signals to the robot actuators are generated. To accomplish its goals, task decomposition modules must often use information stored in the world model. The purpose of the sensory system is to update the world model as rapidly as possible to keep the model in registration with the physical world. ~~This paper describes~~ the architecture of the entire control system hierarchy and how it can be applied to space telerobot applications. An [illegible]

1. INTRODUCTION

One of the major directions on which the robot research community has concentrated its efforts is concerned with planning and controlling motion. Given a specific task, a motion plan must be calculated which meets the task requirements. Then, the plan must be executed; there must be sufficient control for the robot to adequately effect the desired motion.

Trajectories are often planned as straight lines in Cartesian space (1). Whitney (2,3) developed the resolved motion rate control method for Cartesian straight line motions. Paul (4,5,6) used homogeneous coordinate transformations to describe a trajectory as a function of time, and Taylor (7) used coordinated joint control over small segments to keep the trajectory within a specified deviation of the desired straight line trajectory.

While the research described above employs a "kinematic" approach to robot control, another direction of research takes the manipulator "dynamics" into account in the description of robot motion. The dynamic equations of motion are described either by the Lagrangian formulation (8) or by the Newton-Euler equations (9). Algorithms and computer architectures have been suggested which promise real-time dynamic robot control (10,11).

Another aspect of motion control is concerned with the variables being controlled. The research described to this point was concerned primarily with position control. The robot moved from an initial position to a goal position. While this is perhaps the most common mode, there are many applications for robots which suggest that other variables should be controlled. For example, force control would be desired for assembly operations. Raibert and Craig (12) suggest a method for hybrid position/force control of manipulators.

These examples point to the more general problem of sensory processing. For a great deal of robot motion research, sensory processing has been limited to joint positions, velocities, and accelerations. However, other sensors are often required to accomplish tasks. The control community has concentrated on the control aspects of the robot and as a result, little emphasis has been placed on sophisticated sensory processing.

Machine vision, an offshoot of image processing research, has recently been associated with advanced robot applications. One of the most interesting directions in this research area is concerned with sensor controlled robots. Operating with the constraints imposed by real-time robot control, early methods used structured light and binary images (13,14,15,16). These approaches, though developed at different institutions, shared many concepts. One of the important subsequent research efforts went toward the development of model-based image processing. Bolles and Cain (17) used models of objects to guide the algorithms in a hypothesis/verification scheme known as the local feature focus method. The concept has recently been extended from two dimensional (i.e. nearly flat) objects to three dimensional objects (18). Although the approaches described here have led to a better understanding of real-time vision processing, the systems lacked a sophisticated interconnection with the robot control system.

The Automated Manufacturing Research Facility (AMRF), developed at the National Bureau of Standards, is a hierarchically organized small-batch metal machining shop (19). It separates sensory processing and robot control by a sophisticated world model. The world model has three complementary data representations. Lumia (20) describes the CAD-like section of the model. Shneier, Kent, and Mansbach (21) describe the octree and table representations supported by the model. The model generates hypotheses for the features which are either verified or refuted by empirical evidence. The sensory system's task is to update the appropriate parts of the world model with new or revised data as rapidly as possible. The control system accesses the world model as desired to obtain the current best guess concerning any aspect of the world. Shneier, Lumia, and Kent (22) describe the sensory system and its operation in greater detail. The AMRF was the first deliberate attempt to tie together sensory processing, world modeling, and robot control in a generic fashion. The system developed for the AMRF is applicable to more than manufacturing. This paper describes its use in space telerobotics.

2. A FUNCTIONAL SYSTEM ARCHITECTURE

The fundamental paradigm is shown in Figure 1. The control system architecture is a three legged hierarchy of computing modules, serviced by a communications system and a common memory. The task decomposition modules perform real-time planning and task monitoring functions, and decompose task goals both spatially and temporally. The sensory processing modules filter, correlate, detect, and integrate sensory information over both space and time in order to recognize and measure patterns, features, objects, events, and relationships in the external world. The world modeling modules answer queries, make predictions, and compute evaluation functions on the state space defined by the information stored in common memory. Common memory is a global database which contains the system's best estimate of the state of the external world. The world modeling modules keep the common memory database current and consistent.

2.1. Task Decomposition - H modules
(Plan, Execute)

The first leg of the hierarchy consists of task decomposition H modules which plan and execute the decomposition of high level goals into low level actions. Task decomposition involves both a temporal decomposition (into sequential actions along the time line) and a spatial decomposition (into concurrent actions by different subsystems). Each H module at each level consists of a job assignment manager JA, a set of planners PL(i), and a set of executors EX(i). These decompose the input task into both spatially and temporally distinct subtasks as shown in Figure 2. This will be described in greater detail in section 4.

2.2. World Modeling - M modules
(Remember, Estimate, Predict, Evaluate)

The second leg of the hierarchy consists of world modeling M modules which model (i.e. remember, estimate, predict) and evaluate the state of the world. The "world model" is the system's best estimate and evaluation of the history, current state, and possible future states of the world, including the states of the system being controlled. The "world model" includes both the M modules and a knowledge base stored in a common memory database where state variables, maps, lists of objects and events, and attributes of objects and events are maintained. By this definition, the world model corresponds to what is widely known throughout the artificial intelligence community as a "blackboard" (23). The world model performs the following functions:

1. Maintain the common memory knowledge base by accepting information from the sensory system.

2. Provide predictions of expected sensory input to the corresponding G modules, based on the state of the task and estimates of the external world.

3. Answer "What is?" questions asked by the executors in the corresponding level H modules. The task executor can request the values of any system variable.

4. Answer "What if?" questions asked by the planners in the corresponding level H modules. The M modules predict the results of hypothesized actions.

2.3. Sensory Processing - G modules
(Filter, Integrate, Detect, Measure)

The third leg of the hierarchy consists of sensory processing G modules. These recognize patterns, detect events, and filter and integrate sensory information over space and time. The G modules at each level compare world model predictions with sensory observations and compute correlation and difference functions. These are integrated over time and space so as to fuse sensory information from multiple sources over extended time intervals. Newly detected or recognized events, objects, and relationships are entered by the M modules into the world model common memory database, and objects or relationships perceived to no longer exist are removed. The G modules also contain functions which can compute confidence factors and probabilities of recognized events, and statistical estimates of stochastic state variable values.

2.4. Operator Interfaces (Control, Observe, Define Goals, Indicate Objects)

The control architecture defined here has an operator interface at each level in the hierarchy. The operator interface provides a means by which human operators, either in the space station or on the ground, can observe and supervise the telerobot. Each level of the task decomposition hierarchy provides an interface where the human operator can assume control. The task commands into any level can be derived either from the higher level H module, or from the operator interface. Using a variety of input devices such as a joystick, mouse, trackball, light pen, keyboard, voice input, etc., a human operator can enter the control hierarchy at any level, at any time of his choosing, to monitor a process, to insert information, to interrupt automatic operation and take control of the task being performed, or to apply human intelligence to sensory processing or world modeling functions.

The sharing of command input between human and autonomous control need not be all or none. It is possible in many cases for the human and the automatic controllers to simultaneously share control of a telerobot system. For example a human might control the orientation of a camera while the robot automatically translates the same camera through space.

2.4.1 Operator Control interface levels

The operator can enter the hierarchy at any level. The operator control interface interprets teleoperation in the fullest sense: a teleoperator is any device which is controlled by a human from a remote location. While the master-slave paradigm is certainly a type of teleoperation, it does not constitute the only form of man-machine interaction. At different levels of the hierarchy, the interface device for the human may change but the fundamental concept of teleoperation is still preserved. Table 1 illustrates the interaction an operator may have at each level.

The operator control interface thus provides mechanisms for entering new instructions or programs into the various control modules. This can be used on-line for real-time supervisory control, or in a background mode for altering autonomous telerobot plans before autonomous execution reaches that part of the plan.

2.4.2 Operator monitoring interfaces

The operator interfaces allow the human the option of simply monitoring any level. Windows into the common memory knowledge base permit viewing of maps of service bay layout, geometric descriptions and mechanical and electrical configurations of satellites, lists of recognized objects and events, object parameters, and state variables such as positions, velocities, forces, confidence levels, tolerances, traces of past history, plans for future actions, and current priorities and utility function values. These may be displayed in graphical form, for example using dials or bar graphs for scalar variables, shaded graphics for object geometry, and a variety of map displays for spatial occupancy.

2.4.3 Sensory processing/world modeling interfaces

The operator interface may also permit interaction with the sensory processing and/or world modeling modules. For example, an operator using a video monitor with a graphics overlay and a light pen or joystick might provide human interpretative assistance to the vision/world modeling system. The operator might interactively assist the model matching algorithms by indicating with a light pen which features in the image (e.g. edges, corners) correspond to those in a stored model. Alternatively, an operator could use a joystick to line up a wireframe model with a TV image, either in 2-D or 3-D. The operator might either move the wireframe model so as to line up with the image, or move the camera position so as to line up the image with the model. Once the alignment was nearly correct, the operator could allow automatic matching algorithms to complete the match, and track future movements of the image.

2.5. Common Memory

2.5.1. Communications

One of the primary functions of common memory is to facilitate communications between modules. Communications within the control hierarchy is supported by a common memory in which state variables are globally defined.

Each module in the sensory processing, world modeling, and task decomposition hierarchies reads inputs from, and writes outputs to, the common memory. Thus each module needs only to know where in common memory its input variables are stored, and where in common memory it should write its output variables. The data structures in the common memory then define the interfaces between the G, M, and H modules.

The operator interfaces also interact with the system through common memory. The operator displays simply read the variables they need from the locations in common memory. If the operator wishes to take control of the system, he simply writes command variables to the appropriate locations in common memory. The control modules that read from those locations need not know whether their input commands derived from a human operator, or from the next higher level in the autonomous control hierarchy.

2.5.2 State Variables

The state variables in common memory are the system's best estimate of the state of the world, including both the external environment and the internal state of the H, M, and G modules. Data in common memory are available to all modules at all levels of the control system.

The knowledge base in the common memory consists of three elements: maps which describe the spatial occupancy of the world, object-attribute linked lists, and state variables.

3. LEVELS IN THE CONTROL HIERARCHY

The control system architecture described here for the Flight Telerobot System is a six level hierarchy as shown in Figure 3. At each level in this hierarchy a fundamental transformation is performed on the task.

Level 1 transforms coordinates from a convenient coordinate frame into joint coordinates. This level also servos joint positions, velocities, and forces.

Level 2 computes inertial dynamics, and generates smooth trajectories in a convenient coordinate frame.

Level 3 decomposes elementary move commands (E-moves) into strings of intermediate poses. E-moves are typically defined in terms of motion of the subsystem being controlled (i.e., transporter, manipulator, camera platform, etc.) through a space defined by a convenient coordinate system. E-move commands may consist of symbolic names of elementary movements, or may be expressed as keyframe descriptions of desired relationships to be achieved between system state variables. E-moves are decomposed into strings of intermediate poses which define motion pathways that have been checked for clearance with potential obstacles, and which avoid kinematic singularities.

Level 4 decomposes object task commands specified in terms of actions performed on objects into sequences of E-moves defined in terms of manipulator motions. Object tasks typically define actions to be performed by a single multiarmed telerobot system on one object at a time. Tasks defined in terms of actions on objects are decomposed into sequences of E-moves defined in terms of manipulator or vehicle subsystem motions. This decomposition checks to assure that there exist motion freeways clear of obstacles between keyframe poses, and schedules coordinated activity of telerobot subsystems, such as the transporter, dual arm manipulators, multifingered grippers, and camera arms.

Level 5 decomposes actions to be performed on batches of parts into tasks performed on individual objects. It schedules the actions of one or more telerobot systems to coordinate with other machines and systems operating in the immediate vicinity. For example, Level 5 decomposes service bay action schedules into sequences of object task commands to various telerobot servicers, astronauts, and automatic berthing mechanisms. Service bay actions are typically specified in terms of servicing operations to be performed by all the systems (mechanical and human) in a service bay on a whole satellite. This decomposition typically assigns servicing tasks to various telerobot systems, and schedules servicing tasks so as to maximize the effectiveness of the service bay resources.

Level 6 decomposes the satellite servicing mission plan into service bay action commands. Mission plans are typically specified in terms of satellite servicing priorities, requirements, constraints, and mission time line. The level 6 decomposition typically assigns satellites to service bays, sets priorities for service bay activities, generates requirements for spare parts and tool kits, and schedules the activities of the service bays so as to maximize the effectiveness of the satellite servicing mission. To a large extent the level 6 mission plans will be generated off line on the ground, either by human mission planners, or by automatic or semiautomatic mission planning methods.

4. DETAILED STRUCTURE OF THE H MODULES

The H module at each level consists of three parts as shown in Figure 4: a job assignment manager JA, one or more planners PL(s), and one or more executors EX(s).

The job assignment manager JA is responsible for partitioning the task command TC into s spatially or logically distinct jobs to be performed by s physically distinct planner/executor mechanisms. At the upper levels the job assignment module may also assign physical resources against task elements. The output of the job assignment manager is a set of job commands JC(s), s=1, 2, ..., N where N is the number of spatially, or logically, distinct jobs.

For each of these job commands JC(s), there exists a planner PL(s) and a executor EX(s). Each planner PL(s) is responsible for decomposing its job command JC(s) into a temporal sequence of planned subtasks PST(s,tt). Planning typically requires evaluation of alternative hypothetical sequences of planned subtasks. The planner hypothesizes some action or series of actions, the world model predicts the results of the action(s) and computes some evaluation function EF(s,tt) on the predicted resulting state of the world. The hypothetical sequence of actions producing the best evaluation function EF(s,tt)max is then selected as the plan PST(s,tt) to be executed by the executor EX(s).

$$PST(s,tt) = PL(s)\quad JC(s),EF(s,tt)max$$

where tt is the time sequence index for steps in the plan. tt may also be defined as a running temporal index in planning space, tt = 1, 2, ..., th where th is the value of the tt index at the planning horizon. The planning horizon is defined as the period into the future over which a plan is prepared. Each level of the hierarchy has a planning horizon of one or two expected input task time durations.

Each executor EX(s) is responsible for successfully executing the plan PST(s,tt) prepared by its respective planner PL(s). If all the subtasks in the plan PST(s,tt) are successfully executed, then the goal of the original task will be achieved. The executor operates by selecting a subtask from the current queue of planned subtasks and outputting a subcommand STX(s,t) to the appropriate subordinate H module at time t The EX(s) module monitors its feedback FB(s,t) input in order to servo its output STX(s,t) to the desired subtask activity.

$$STX(s,t+n) = EX(s)\quad PST(s,t),FB(s,t)$$

where n = the number of state clock periods required to compute the function EX(s). n typically equals 1. The feedback FB(s,t) also carries timing and subgoal event information for coordination of output between executors at the same level. When the executor detects a subgoal event, it selects the next planned subtask from the queue.

Executor output STX(s,t) also contains requests for information from the world model M module, and status reports to the next higher (i+1) level in the H module hierarchy. The feedback FB(s,t) contains status reports from the H module at the i-1 th level indicating progress on its current task.

5. CONCLUSION

This paper has described a hierarchically organized control system and has shown how this generic system can be applied to telerobotic applications in space by considering the requirements of a flight telerobotic servicer for the space station.

REFERENCES

(1) M. Brady, et.al., ed. Robot Motion: Planning and Control, (Cambridge, MIT Press, 1982).

(2) D.E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses," IEEE Trans. Man-Machine Systems MMS-10, 1969, p. 47.

(3) D.E. Whitney, "The Mathematics of Coordinated Control of Prostheses and Manipulators," Journal of Dynamic Systems, Measurement, Control, Dec. 1972, p. 303.

(4) R.P. Paul, "Manipulator Path Control," IEEE Int. Conf. on Cybernetics and Society, New York, p. 147.

(5) R.P. Paul, "Manipulator Cartesian Path Control," IEEE Trans. Systems, Man, Cybernetics SMC-9, 1979, p. 702.

(6) R.P. Paul, Robot Manipulators: Mathematics, Programming, and Control, (Cambridge, MIT Press, 1981.)

(7) R. H. Taylor, "Planning and Execution of Straight-line Manipulator Trajectories," IBM J. Research and Development 23 1979, p. 424.

(8) J.M. Hollerbach, "A Recursive Formulation of Lagrangian Manipulator Dynamics," IEEE Trans. Systems, Man, Cybernetics SMC-10, 11, 1980, p. 730.

(9) J.Y.S. Luh, M.W. Walker, and R.P.C. Paul, "On-line Computational Scheme for Mechanical Manipulators," J. Dynamic Systems, Measurement, Control, 102, 1980, p. 69.

(10) C.S.G. Lee, P.R. Chang, "Efficient Parallel Algorithm for Robot Inverse Dynamics Computation," IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-16, No. 4, July/August 1986, p. 532.

(11) E.E. Binder, J.H. Herzog, "Distributed Computer Architecture and Fast Parallel Algorithm in Real-Time Robot Control," IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-16, No. 4, July/August 1986, p. 543.

(12) M.H. Raibert and J.J. Craig, "Hybrid position/force control of manipulators," J. Dynamic Systems, Measurement, Control, June, 1981, p. 126.

(13) W.A. Perkins, "A Model Based Vision System for Industrial Parts," IEEE Trans. on Computers, Vol. C-27, 1978, p. 126.

(14) G.L. Gleason, G.J. Agin, "A Modular Vision System for Sensor-controlled Manipulation and Inspection," Proc. 9th Int. Symposium on Industrial Robots, 1979, p. 57.

(15) M.R. Ward, et.al., "CONSIGHT An Adaptive Robot with Vision," Robotics Today, 1979, p. 26.

(16) J. Albus, E. Kent, M. Nashman, P. Mansbach, L. Palombo, M.O. Shneier, "Six Dimensional Vision System," SPIE, Vol. 336, Robot Vision, 1982, p. 142.

(17) R.C. Bolles, R.A. Cain, "Recognizing and Locating Partially Visible Objects: The Local Feature-Focus Method," Int. Journal of Robotics Research, Vol. 1, 1982, p. 57.

(18) R.C. Bolles, P. Horaud, M.J. Hannah, "3DPO: Three Dimensional Parts Orientation System," Proc. of The Int. Joint Conf. on Artificial Intelligence, August 1983, p. 1116.

(19) J.A. Simpson, R.J. Hocken, J.S. Albus, "The Automated Manufacturing Research Facility of the National Bureau of Standards," Journal of Manufacturing System, Vol. 1, No. 1, 1983, p. 17.

(20) R. Lumia, "Representing Solids for a Real-Time Robot Sensory System," Proc. Prolamat 1985, Paris, June 1985.

(21) M.O. Shneier, E.W. Kent, P. Mansbach, "Representing Workspace and Model Knowledge for a Robot with Mobile Sensors," Proc. 7th Int. Conf. Pattern Recognition, 1984, p. 199.

(22) M.O. Shneier, R. Lumia, E.W. Kent, "Model Based Strategies for High-Level Robot Vision," CVGIP, Vol. 33, 1986, p. 293.

(23) A. Barr, E. Feigenbaum, The Handbook of Artificial Intelligence, (Los Altos, William Kaufman, 1981).

TABLE 1 -- OPERATOR INTERACTION AT EACH LEVEL

LEVEL	TYPE OF INTERACTION
At the servo	replica master, individual joint position, rate, or force controllers.
above level 1	joy stick to perform resolved motion force/rate control
above level 2	indicate safe motion pathways. Robot computes dynamically efficient movements
above level 3	graphically or symbolically define key poses. menus to choose elemental moves.
above level 4	specify tasks to be performed on objects.
above level 5	reassign telerobots to different service bays. insert, modify, and monitor plans describing servicing task sequences.
above level 6	reconfigure servicing mission priorities.

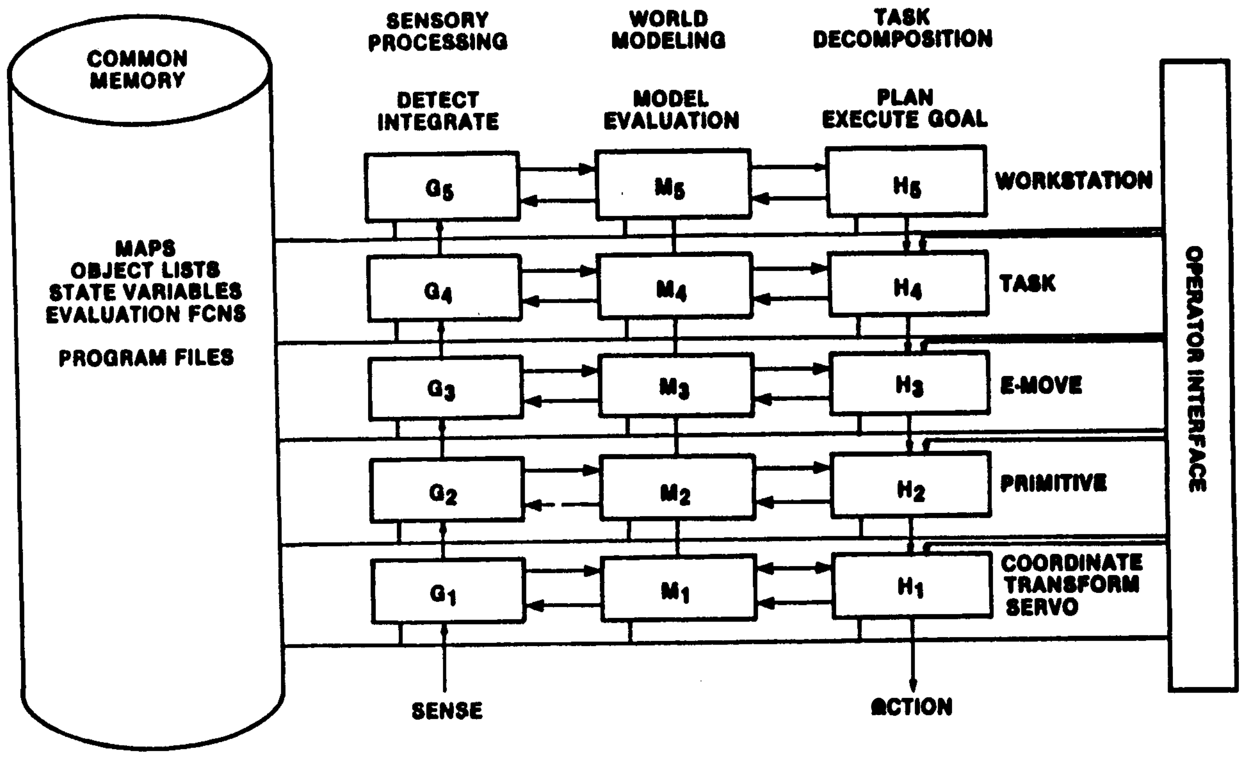


FIGURE 1 : A Hierarchial Control System Architecture for Intelligent Vehicles

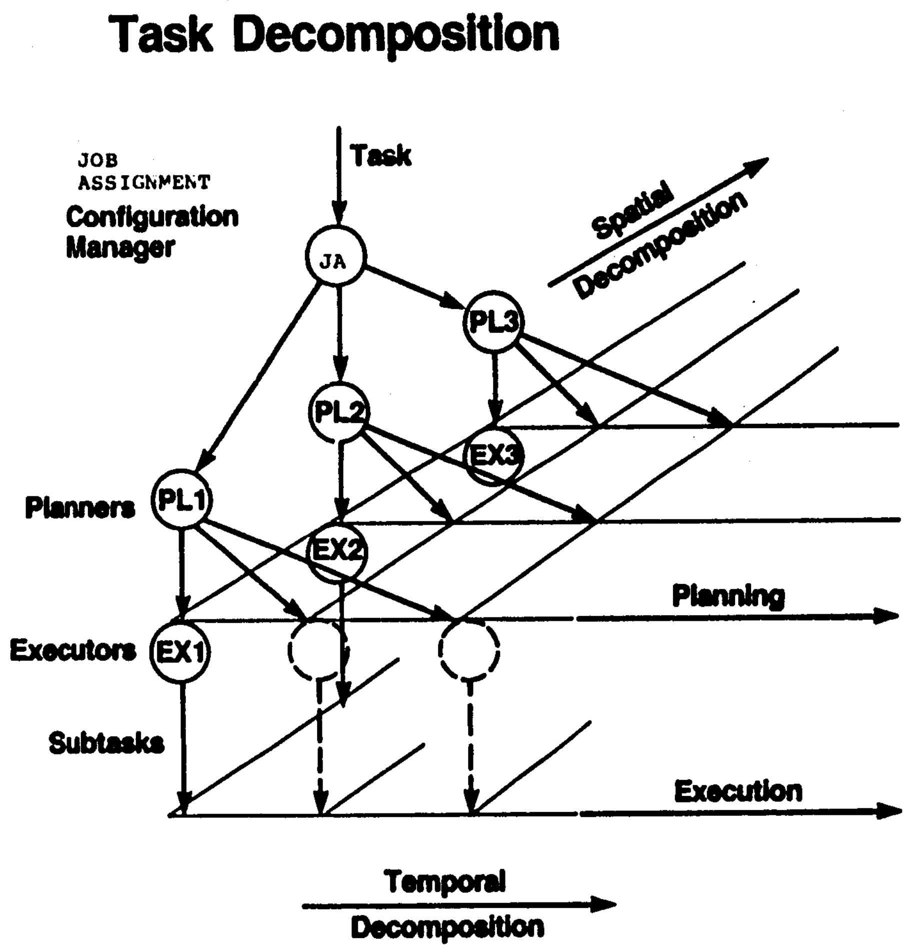


FIGURE 2: The job assignment JA performs a spatial decomposition of the task. The planners PL (j) and executors EX (j) perform a temporal decomposition

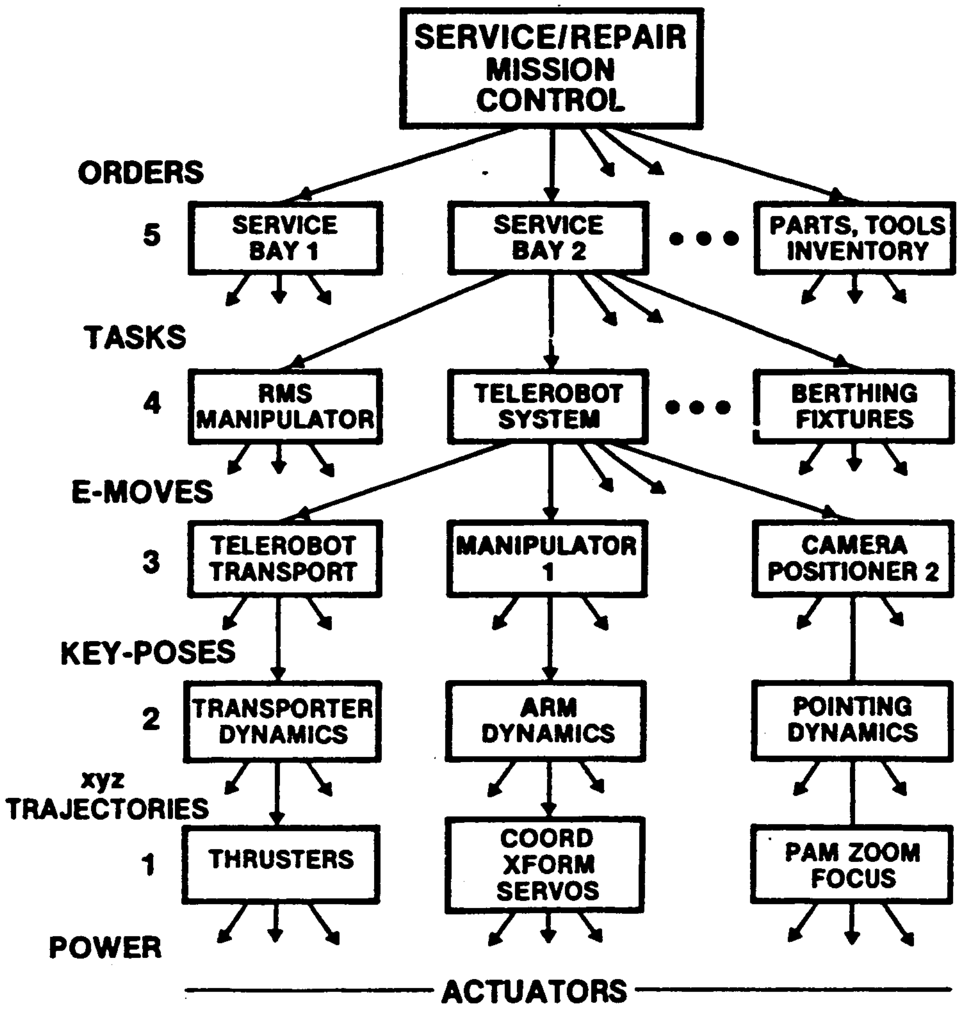


FIGURE 3: A six level Hierarchial Control System Proposed for Multiple Autonomous Vehicles

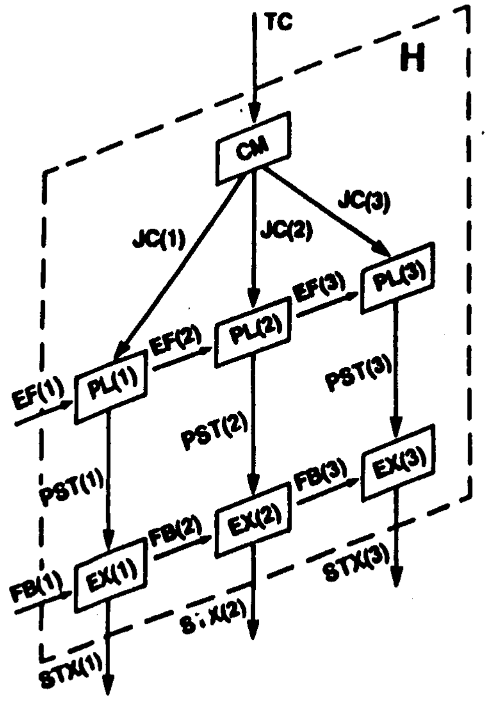


FIGURE 4: The H Module at each level has three parts: A job assignment module JA, Planners PL and a set of exocutors EX.

S18-61
188120
68.

Programming Methodology for a General Purpose Automation Controller

M.C. Sturzenbecker, J.U. Korein, and R.H. Taylor
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598

IA 657203

Abstract

The General Purpose Automation Controller is a multi-processor architecture for automation programming. A methodology has been developed whose aim is to simplify the task of programming distributed real-time systems for users in research or manufacturing. Programs are built by configuring *function blocks* (low-level computations) into processes using data flow principles. These processes are activated through the *verb* mechanism. Verbs are divided into two classes: those which support devices, such as robot joint servos, and those which perform actions on devices, such as motion control. This programming methodology was developed in order to achieve the following goals: 1) Specifications for real-time programs which are to a high degree independent of hardware considerations such as processor, bus, and interconnect technology. 2) A "component" approach to software, so that software required to support new devices and technologies can be integrated by reconfiguring existing building blocks. 3) Resistance to error and ease of debugging. 4) A powerful command language interface.

Introduction

Recent system designs aimed at solving problems in automation control have made significant use of *multi-processing* [1-7]. Typically these systems incorporate a variable number of processors performing computations in parallel and exchanging data by means of some *interconnect technology*. These technologies are usually compatible either with a *shared memory model* [6] of data exchange or with a *message-passing model* [5,7]; occasionally, systems may exhibit features of both models. If all processors execute the same program, the system is said to be SIMD (*Single Instruction, Multiple Data*), however the greatest flexibility is achieved with a MIMD (*Multiple Instruction, Multiple Data*) system.

Multi-processor systems not only offer the prospect of increased computing power to meet the ever-increasing requirements of real-time control, they carry the potential for a high degree of configurability. One of the obstacles to rapid progress in robotics research and the deployment of programmable automation in scientific and manufacturing applications is the lack of configurability inherent in most currently available systems. The need for configurability arises from the need to integrate new devices, employ new strategies, or add processing power incrementally without making major changes to the system; for software this implies a need for "fast prototyping", the ability to construct software that can rapidly adjust to changing requirements [8,9]. However, multi-processor systems cannot be used to tackle this problem unless another problem is simultaneously addressed: the lack of tools and methodologies to simplify the task of programming such systems.

What is a Programming Methodology?

A true programming methodology is not simply a collection of tools or techniques, rather it provides a set of concepts, usually formally or semi-formally defined, which serve as a basis for problem decomposition and program design. [10] This is a fairly definitive requirement which is not covered by vague terms indicative of a general approach, such as "top-down design" or "object-oriented programming".

A methodology has a number of advantages over a "toolbox", of which the following are representative:

- The evolution of tools and techniques does not always promote ease of use, and of themselves tools do not help to guarantee good program design. [11]
- A methodology provides a powerful language for describing the function of a program or specifying its design.
- A methodology can provide a framework for reasoning about the correctness of a program.
- Programs developed according to a methodology are often easier to maintain.

This paper describes such a programming methodology developed in conjunction with a General-Purpose Automation Controller (GPAC) project at IBM [8,9]. The ultimate aim of this methodology is to simplify the task of programming real-time, distributed systems for manufacturing engineers, control specialists, and other system users whose primary area of expertise is not computer science.

Currently, most distributed programs are written either in a concurrent language for which a distributed environment has been built (e. g., Ada [12,13]), a conventional sequential language with enhancements for distributed programming [4,7], or a special-purpose language [14]. None of these approaches is particularly adept at dealing with highly reconfigurable systems. To change the behavior of a system one must make changes to the program's source language, re-compile, re-link, re-download, etc. The usual approach is to provide primitives [15,16] (either as part of the language syntax or as system calls) for communication, synchronization, mutual exclusion, and even control over the granularity of concurrency. Application programming in such a system involves not only getting the sequential algorithms right but also managing the interaction between concurrent modules and using the primitives correctly. Many of the programs developed using these approaches are dependent for their performance, if not for their correctness, on a particular model of data exchange. Finally, there is no true methodology associated with these approaches, although useful tools such as debuggers, simulators, and syntax-directed editors are often provided.

Basic Concepts

The GPAC programming methodology depends upon a set of basic concepts relating to both hardware and software.

Hardware

The fundamental hardware concept in GPAC is that of a *real-time processor*. Each real-time processor is a distinct entity which can perform one or more computational tasks. Each real-time processor has its own instruction stream, hence the GPAC model is MIMD. A set of real-time processors sharing a common communications bus or network is a *real-time system* (RTS). Programs are developed on a *programming system* (PS) and are then downloaded into the real-time processors. Associated with each real-time processor are one or more *interrupt sources*. An interrupt source, in the abstract, is simply a request to a real-time processor to perform some work. Interrupt sources have *priorities*; work performed in connection with one interrupt source can be preempted by an interrupt source with a higher priority. Interrupt sources can be generated by other real-time processors or by devices connected to the real-time system. It is important to remember that the notions of interrupt source and priority are fundamentally abstract, and may be realized in the actual system in a number of ways. One last hardware concept is the *physical device*. A physical device is a gateway by which data can be passed to and from external hardware and is accessible by a particular real-time processor. Sensors, actuators, pendant interfaces - to name but a few - are examples of physical devices.

In GPAC, as in other systems [16], implementation of the PS is decoupled from that of the RTS, so that the architecture of the PS can be quite different from that of the RTS, as it should be since the requirements are different.

Function Blocks

The fundamental software concept in GPAC is that of a *function block* [9]. A function block is a basic computational unit assigned to one or more real-time processors; it communicates through *input ports* and *output ports*. In addition, a function block may communicate with *physical devices*, and may report *conditions*, or events which require exceptional action by the system. Finally, a function block may have some *formal parameters*, which are for its internal use only and normally not visible from outside. These five components comprise the *interface* to the function block.

A function block can be viewed externally as a "black box" which takes inputs, performs some computation, and produces outputs (and in some cases, reports conditions). We denote the inputs, outputs, devices, conditions, and parameters of a function block F by I_F, O_F, D_F, C_F, and P_F respectively.

The most basic form of function block is called an *application subroutine*. In the current system, this is coded in the C language [17] and corresponds to a C function. A set of macros is used to specify the interface; this hides any implementation details from the programmer.

An application subroutine is written in sequential code, and of itself contains no notion of concurrency. If certain coding conventions are followed, the application subroutine is also *re-entrant*, and multiple *instances* of it may be active either on the same real-time processor or on different real-time processors. A *function block instance* is obtained by *binding* the ports to specific data objects, supplying actual references for the physical devices, and values for the conditions and formal parameters. Other preconditions for the creation of a function block instance are its assignment to a particular real-time processor and interrupt source, and determination of the means by which it is scheduled.

Data Flow Graphs

More complex function blocks can be built up from simpler ones such as application subroutines. These complex function blocks are called *data flow graphs*.

A data flow graph G consists of:

- A set f_G of function blocks.
- A set I_G of input ports, a set O_G of output ports, and a set L_G of *local cells*.
- A mapping $\rho : I_G \cup O_G \cup L_G \rightarrow 2^{\mathbf{data}_G}$,[1] where $\mathbf{data}_G$ are the *data nodes* of G:

$$\mathbf{data}_G = \bigcup_{F \epsilon f_G} I_F \cup O_F$$

All the data nodes must be mapped to by ρ, i. e., for any $d \epsilon \mathbf{data}_G$ there exists p such that $d \epsilon \rho(p)$.

A data flow graph is composed of communicating function blocks, but externally it is indistinguishable from an application subroutine. Data flow graphs are useful for expressing *distributed execution* of an algorithm. For example, a servo algorithm usually involves reading some value from a sensor, performing some computation, and writing another value to an actuator. In a distributed system, however, there is no guarantee that the sensor and actuator will both be accessible from the same real-time processor. Thus, in general, this algorithm cannot be executed by a single application subroutine, an instance of which is constrained to run on a single processor.

Data flow graphs are also useful for pasting together application subroutines of general utility. Perhaps we wish to add some digital filtering to the input in our servo example. This can be done most conveniently by building a data flow graph, provided some digital filtering module has already been installed as part of the software component data base.

Ports of Function blocks

Each input and output port of a function block has a *type* and a *mode*. The type is simply a C type declaration. Ports can be bound to data objects only if those objects have a compatibile type. The mode of a function block port describes the relationship between the modification, or *updating* of the object to which the port is bound, and the frequency with which the function block is executed.

There are three port modes:

- *synchronous* - The object bound to the port is updated on every invocation.
- *ratio* - The object bound to the port is updated at a specified sub-frequency of the frequency of invocation.
- *asynchronous* - There is no fixed relationship between the updating of the object to which the port is bound and the frequency of invocation.

Thus, if a function block has a synchronous input port, an instance of it can be scheduled for execution only when a new value of the object bound to the port is made available by another function block instance which writes the object through an output port.

If a function block has no synchronous inputs, then it can be scheduled by assignment of an *execution interval* or a *trigger*. The execution interval specifies a frequency at which the function block must be executed; a trigger associates execution of a function block directly to occurrence of an interrupt source.

Another rule enforced by the GPAC system is that values produced by a function block are not made available to other function blocks until the former has completed its current invocation. This gives an atomic flavor to function blocks and helps to insure that the world will always be seen in a consistent state.[2]

Advantages of the Function Block Concept

Specification of real-time computations using the function block concept has the following advantages:

- Real-time requirements are separated from the executable code. This means that instances of the same code can be in-

[1] 2^S denotes the set of all subsets of S

[2] Errors occurring in the transmission of values over a network may, however, yield such inconsistencies; the methodology assumes that handling of these errors is transparent to the application.

voked with differing real-time requirements, that requirements can be changed dynamically.

- Function blocks have no knowledge, and consequently no dependence, on the particular data exchange model. Although the current GPAC architecture is based on shared memory, the system could be implemented using message passing without affecting the design of applications written for it.
- By standardizing the interface to function blocks and hiding the implementation details, we hope to encourage programmers to design routines that will be easily reusable. One problem which has emerged historically in the "component" approach to software design is that, in the absence of a methodology, the success of the approach is dependent upon the good judgment and foresignt of programmers.
- *Consistency in* data typing is automatically checked. Primitives for synchronization and mutual exclusion are unnecessary, hence their misuse is impossible. The result is a system that is more robust and easier to debug and maintain.

Processes in GPAC

Work done by a real-time system is normally divided into *processes*. In GPAC, these processes are merely sets of communicating function block instances. Function block instances are *installed* before they can be executed; this consists of setting up the port bindings and attaching the function block instance to the proper processor and interrupt source. Consistent use of data objects bound to ports and of function block scheduling is checked during installation.

A typical GPAC process will consist of three phases:

- Installation of function block instances to be used.
- Execution (either once, or repetitively) of one or more function block instances.
- Waiting for termination of all the function block instances, or for a function block instance to report a condition that results in process termination.

As an example, consider a process to implement coordinated motion of several robot joints. This process requires two function blocks: one which computes the coefficients of a *trajectory* (i. e., desired joint position as a function of time) based on the current positions, target positions, speed limits, etc., and another which generates intermediate points along the trajectory and outputs these commanded positions to the joint servo modules. (The servo modules themselves are considered part of a different process, as we shall see shortly.) The first function block, called the *trajectory planner*, is executed once, the second, called the *setpoint generator*, is executed repetitively at some time interval (e. g., 20 msec). The whole process terminates when the current (sensed) position of the joints becomes close enough to the target position.

Command Lists

We now describe the above process behavior using GPAC terminology:

1. Install the trajectory planner and the setpoint generator.
2. Execute the trajectory planner once.
3. Activate the setpoint generator for repetitive execution.
4. Wait until the setpoint generator reports that the motion has completed (i. e., the sensed position is close enough), or that some other condition (unexpected force sensed, time limit exceeded, etc.) has occurred.

This description of a process is called a *command list* in GPAC. The elements of a command list are called *function block commands* and take function block instances as arguments. A process consists of a command list and a set of *termination conditions*.

Command List Transitions

In the case of robot motion, however, termination conditions are often simply signals that the system should proceed with the motion using a somewhat different plan. This is particularly true in *compliant motion* and specialized combinations of motions such as *centering grasp* [8]. In these cases the termination of a process results in a *transition* to another process.

Verbs

We now have developed the necessary concepts to define a *verb* in GPAC. A verb V consists of:

- A set f_V of function blocks.
- A set p_V of parameters.
- A set o_V of output values.
- A subset $\mathbf{inst}_V$ of p_V and a mapping $\chi : \mathbf{inst}_V \rightarrow 2^{\mathbf{data}_V}$ where $\mathbf{data}_V$ is the set of all *data nodes* in V, i. e., the union of all inputs, outputs, and physical devices in the function blocks of V:

$$\mathbf{data}_V = \bigcup_{F \in f_V} I_F \cup O_F \cup D_F$$

- A mapping $\psi : p_V - \mathbf{inst}_V \rightarrow 2^{\mathbf{vals}_V}$ where $\mathbf{vals}_V$ is the set of all *values* accessible within V, which includes all the ports as well as the conditions and parameters:

$$\mathbf{vals}_V = \bigcup_{F \in f_V} I_F \cup O_F \cup C_F \cup P_F$$

- A mapping $\omega : o_V \rightarrow \mathbf{vals}_V$.
- A set c_V of command lists, of which one is distinguished as *initial*.
- A set t_V of termination conditions.
- A mapping $\tau : \mathbf{trans}_V \rightarrow c_V$ where $\mathbf{trans}_V$ is the *transitions* of V, a subset of $c_V \times t_V$.
- A mapping $\upsilon : \mathbf{term}_V \rightarrow 2^{o_V}$, where $\mathbf{term}_V$ is the set of *terminations* of V :

$$\mathbf{term}_V = \{ t : \exists c \in c_V \ni (c,t) \notin \mathbf{trans}_V \}$$

$\mathbf{inst}_V$ are the *instantiation parameters* of the verb, and χ is the *instantiation mapping*. These elements describe the communication of data to and from the instantiated function blocks of the verb. The verb parameters not in $\mathbf{inst}_V$ are called *input parameters* and the mapping ψ is the *input mapping*. These elements describe how initial values are set up to be accessed by the instantiated function blocks of the verb. The mapping ω is the *output value mapping* which defines a set of values that may be returned from the verb.

Instantiation parameters are either data objects which can be bound to function block ports or references to physical devices. Input parameters are simply values which can be stored in the objects bound to ports or in conditions or parameters of a function block. As in the case with the mapping ρ for data flow graphs, every member of $\mathbf{data}$, must be a member of some set in the range of χ; that is, every data node is mapped to by some instantiation parameter.

If $\tau(c_1,t) = c_2$ then if condition t occurs while the process described by c_1 is executing, that process is aborted and the process described by c_2 commences. If no transition is specified for a given termination condition and process, then receipt of that condition within the process causes termination of the entire verb.

The mapping υ describes the *termination actions* of a verb. Each termination of a verb can return a subset of the output values defined for the verb.

Verb Instances

The instantiation parameters of a verb provide bindings for the ports of the constituent function blocks of the verb. Thus, *application* of parameters to a verb yields function block instances for each of the constituent function blocks, and the collection of these defines a *verb instance*. A verb instance in GPAC closely resembles the concept of a *task* in more traditional systems. Verb instances can be started, halted, suspended, and resumed.

The distinction between verb and verb instance can easily be appreciated by analogy with an operating system utility residing as a binary image on secondary storage. When the utility is invoked by a user, its inputs and outputs are bound to actual files and devices and a task, or main memory image, is created. Furthermore, multiple instances of the same utility can sometimes be active in the system simultaneously. The same is true for multiple instances of a verb in GPAC.

Verb instance commands are passed from the Programming System to the Real-Time System, where they are executed by the *supervisory software*. In every RTS, there is one real-time processor which is distinguished as the *supervisor*. The supervisory software is downloaded (possibly along with applications code) into this processor, and it maintains communication with the PS while the RTS is running its applications. While the PS sends verb instance commands to the RTS, the RTS sends notification of *verb instance termination* to the PS.

All real-time processors, whether or not they are supervisors, contain a *real-time kernel* which is primarily responsible for handling interrupts, context switching, and dispatching of application subroutines. The supervisor handles all activity related to:

- Installation, activation, and deactivation of function block instances.
- Transitions between command lists.
- Termination of verb instances.

State Vector Variables and Logical Devices

We have previously referred to the representation of data within GPAC without providing any details. Data which must be communicated between function blocks is represented by the concept of a *state vector variable* (SVV). It is perhaps easiest to think of an SVV as shared memory, but it need not be implemented that way. A state vector variable contains a buffer for the current value, and optionally a buffer for a set of *previous values*, in case a history needs to be maintained. State vector variables are bound to function block ports in order to effect communication between funcion block instances. Associated with each SVV is a *type*, and it can only be bound to ports of the same type.

Although a state vector variable is essentially an independent concept, the primary method of creating and using the SVV in GPAC is through the more powerful concepts of a *logical device* and a *logical device type*. A logical device type consists of:

- An optional verb.
- A specification for a set of SVVs.
- A specification for a set of parameters.

A logical device can be considered an *instance* of a logical device type and consists of:

- An optional verb instance.
- A set of SVVs.
- A set of parameters.

A robot joint is an excellent example of a logical device type. With each joint in the system we normally want to maintain the following data:

- The current *sensed* position of the joint.
- The current *commanded* position of the joint.
- The *goal*, or target position.

This data will be stored in SVVs, since it will be updated by function block instances in real time. We may also associate certain parameters with a robot joint:

- The maximum speed at which the joint can be moved.
- The maximum force or acceleration to which it may be subjected, etc.

These values do not normally vary in real time and hence do not need to be stored in state vector variables.

Logical Devices as Verb Parameters

A verb may be *applied* to one or more logical devices; alternatively, we might say that logical devices can appear as the *objects* of verbs. In this case, the state vector variables and parameters which comprise the logical device will be entered into the parameter list from which the verb instance will be constructed. Thus they can be bound to the ports and parameters of function blocks which will perform the computations associated with the verb.

A robot joint is *controlled* by some real-time process (servo). This is implemented as a verb in GPAC. The definition of a logical device type for the robot joint contains specifications for the servoing verb, the SVVs, and the parameters. Then, when an instance of a joint is created, the actual verb instance, SVVs, and parameters are created for that device.

Before a device may be used as the object of some action (e. g., before a *joint* may be *moved*) it must be *enabled*. Enabling a device is equivalent to starting the verb instance which controls the device.

Verb Composition

Verbs may be *composed*, or combined to form new verbs. A composition of verbs can be formally defined by taking unions of all the verb components; the verbs may then be sequenced by supplying an augmented transition mapping. In this way verbs with a highly specialized semantics can be assembled out of simpler, more general ones. (A verb with only one command list is called *simple*.) An example of this is "centering grasp" found in [8] and [9].

Levels of Control

The GPAC methodology supports programming for various *levels of control*: [9]

- *Closed-loop control* is achieved through low-level real-time computations such as application subroutines and data flow graphs.
- *Concurrency control* is concerned with specification of computations executing in parallel and is achieved through configuration of command lists and simple verbs.
- *Sequential control* deals with plans and strategies for executing complex motions or tasks and is achieved primarily by verb composition.

Programming for different levels of control involves different issues and areas of expertise; this is reflected in the methodology.

User Interface

AML/X

The GPAC user interface is a program running on the Programming System. Commands entered by a user are converted from a high-level form into actual messages to be sent to the Real-Time System. Also, configuration of system hardware and software is done via the user interface.

Because of the many abstractions and generic entities in the GPAC system, the high-level language AML/X was used to implement the user interface. AML/X is a language designed at IBM for automation programming and other applications [18]. It has a number of interesting features, among which the three most relevant to GPAC are:

- Data abstraction capability. An abstract data type is called a *class*; instances of classes consist of *instance variables* which can be manipulated using *methods*, a specialized type of subroutine call. The methods define the interface to the data type, except that some instance variables can be declared *exposed*, which makes them visible externally and hence part of the interface.
- Operator overloading. Operators can be defined on class instances, using a syntax similar to the method syntax. In particular, the act of applying parameters to an object is regarded as an operator in AML/X; this allows GPAC verb invocation to have the same syntactic form as subroutine invocation.
- Exception handling. AML/X has a rich set of constructs for raising and handling exceptional conditions [18].

AML/X can be easily interfaced to lower-level languages like C and FORTRAN. In fact, the communication between the PS and RTS in GPAC is handled by a C subroutine package which is called from AML/X.

Configuration and Execution Phases

The GPAC user interface consists of two parts: a *configuration phase* and an *execution phase*.

In the configuration phase the hardware layout is described, in terms of real-time processors available, interrupt sources, and physical devices attached to the processors. Next, previously compiled and linked object code modules are downloaded into the real-time processors. (For the most part, these modules are simply libraries of application subroutines linked with a real-time kernel.) Then generic objects such as function blocks, verbs, and logical device types are defined. Finally, some instances of logical devices may be created and enabled.

The execution phase consists primarily of invocation of verbs. Verbs are applied to devices to create verb instances, and commands concerning these verb instances are then passed to the real-time system.

Configuration and execution phases can be interleaved to a certain extent. New function blocks, verbs, and logical devices may be defined at any time, and existing verbs may be redefined. This can be done *while* real-time applications are running (as long as no instances of the affected verbs are still executing). The goal is to provide a highly interactive programming environment in which real-time system behavior can be modified in a very flexible and dynamic way. This meets the needs of both robotics researchers, who wish to experiment with alternative control strategies and new sensor and actuator technologies, and manufacturing engineers, who are responsible for reconfiguring workcells to meet changing work requirements.

Modes of Verb Invocation

The full life cycle of a verb instance can be described as follows:

- A verb is invoked by applying it to a parameter list. These parameters are processed by the PS and the information needed to create an instance of the verb is sent to the RTS.
- The RTS allocates and fills in the necessary data structures, and reports completion of this procedure to the PS.
- The PS sends a command to start the verb instance.
- The initial command list of the verb is executed on the RTS. The verb instance continues to execute until some condition forces it to terminate or until a *suspend* or *halt* message is received from the PS.
- When the verb instance terminates, the RTS sends notification to the PS. Depending on the reason for the termination, an exception may be raised on the PS.
- Finally, the PS sends a command to delete the verb instance. The RTS complies, cleaning up and deallocating all data structures.

In the simplest mode of invocation, this entire scenario is carried out synchronously. The user simply applies a list of arguments to a object of the verb class, e. g.:

```
move(joint,goal,speed);
```

The above operation will not complete until the corresponding verb instance terminates in the RTS and is deleted.

A more efficient, although more verbose, mode of invocation is provided by introducing an asynchronous phase:

```
vi: BIND move.asynch(joint,goal,speed);
/* other work can be done here */
vi.wait();
```

asynch is a verb method that results in creating and starting a verb instance, and returning a verb instance object without waiting for termination. At some later point in time, the verb instance can be waited for, and it is deleted after its termination.

Finally, a verb instance can be created and then *multiply invoked* with new input parameters each time:

```
vi: BIND move.new_instance(joint);
vi.start(goal,speed);
/* do something else */
vi.wait();
vi.start(another_goal,another_speed);
/* do something else */
vi.wait();
vi.delete();
```

Current Status and Future Work

The GPAC methodology has been implemented in conjunction with several different hardware architectures. The relative ease with which applications can be ported back and forth between these architectures is one encouraging result of our work. We have also observed that real-time computations are easy to program, require a minimal amount of debugging, and have predictable behavior once debugged. Finally, the system can be easily reconfigured: new devices and processors can be added without laborious changes and recompilations.

We have used a certain amount of formalism to describe the concepts underlying the methodology rather than relying on configuration examples in AML/X. We chose to do this because it is the formalism (not the current configuration syntax) which is really critical to understanding the methodology, because we did not wish to require detailed knowledge of AML/X of the reader, and because the syntax itself is subject to considerable change and enhancement as we gain experience with the system.

In the future, more sophisticated applications will be attempted using this methodology. We will then be able to judge the efficacy of the methodology for practical problems in automation. Graphical tools will be added to simplify verb and data flow graph configuration. Knowledge-based enhancements and natural-language-like interfaces are also contemplated.

References

1. F. Ozguner and M. L. Kao, "A Reconfigurable Multiprocessor Architecture for Reliable Control of Robotic Systems", *IEEE International Conference on Robotics and Automation*, St. Louis, March 1985.

2. S. Ahmad, "Real-Time Multi-Processor Based Robot Control", *IEEE Int. Conf. on Robotics and Automation*, San Francisco, April 1986.

3. V. Dupourque, H. Guiot, O. Ishacian, "Towards Multi-Processor and Multi-Robot Controllers", *IEEE Int. Conf. on Robotics and Automation*, San Francisco, April 1986.

4. I. Lee and S. Goldwasser, "A Distributed Testbed for Active Sensory Processing", *IEEE Int. Conf. on Robotics and Automation*, St. Louis, March 1985.

5. D. Siegel, et. al., "Computational Architecture for the Utah/MIT Hand", *IEEE Int. Conf. on Robotics and Automation*, St. Louis, March 1985.

6. L. S. Haynes and A. J. Wavering, "Real Time Control System Software: Some Problems and an Approach", *IEEE Int. Conf. on Robotics and Automation*, San Francisco, April 1986.

7. R. D. Gaglianello, "A Distributed Computing Environment for Robotics", *IEEE Int. Conf. on Robotics and Automation*, San Francisco, April 1986.

8. J. U. Korein, G. E. Maier, R. H. Taylor, and L. F. Durfee, "A Configurable System for Automation Programming and Control", *IEEE International Conference on Robotics and Automation*, San Francisco, April 1986.

9. G. E. Maier, R. H. Taylor, J. U. Korein, "A Dynamically Configurable General Purpose Automation Controller", *Fourth IFAC/IFIP Symp. on Software for Computer Control*, Graz, Austria, May 1986.

10. S. N. Griffiths, "Design Methodologies - A Comparison", in *Tutorial: Software Design Strategies*, G. Bergland and R. Gordon, eds. 1979. pp. 189-213.

11. G. D. Bergland, "Structured Design Methodologies", in *Tutorial: Software Design Strategies*. pp. 162-181.

12. U. S. Dept. of Defense, *Reference Manual for the Ada Programming Language*.

13. V. Dupourque, "Using Abstraction Mechanisms to Solve Complex Task Programming in Robotics", *IEEE Int. Conf. on Robotics and Automation*, San Francisco, April 1986.

14. M. D. Donner, "The Design of OWL: A Language for Walking", *Proceedings of the SIGPLAN '83 Symp. on Prog. Lang. Issues in Software Systems*. pp. 158-165

15. K. G. Shin and M. E. Epstein, "Communication Primitives for a Distributed Multi-Robot System", *IEEE Int. Conf. on Robotics and Automation*, St. Louis, 1985.

16. K. Schwan, T. Bihari, B. W. Weide, and G. Taulbee, "GEM: Operating System Primitives for Robots and Real-Time Control Systems", *IEEE Int. Conf. on Robotics and Automation*, St. Louis, 1985.

17. B. Kernighan and D. Ritchie, *The C Programming Language*, Prentice-Hall, 1978. July 1982.

18. *IBM Manufacturing System: A Manufacturing Language Reference Manual*, No. 8509015, IBM Corporation, 1983.

19. L. R. Nackman and R. H. Taylor, "A Hierarchical Exception Handler Binding Mechanism", *Software - Practice and Experience*, 14, 10, Oct. 1984. pp. 999-1007.

S19-63
88
188101

Real-Time Hierarchically Distributed Processing Network Interaction Simulation

JJ 574450

W.F. Zimmerman and C. Wu
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

1. Abstract

The Telerobot Testbed is a hierarchically distributed processing system which is linked together through a standard, commercial Ethernet. Standard Ethernet systems are primarily designed to manage non-real-time information transfer. Therefore, collisions on the net (i.e., two or more sources attempting to send data at the same time) are managed by randomly rescheduling one of the sources to retransmit at a later time interval. Although acceptable for transmitting noncritical data such as mail, this particular feature is unacceptable for real-time hierarchical command and control systems such as the Telerobot. Data transfer and scheduling simulations, such as token ring, offer solutions to collision management, but do not appropriately characterize real-time data transfer/interactions for robotic systems. Therefore, models like these do not provide a viable simulation environment for understanding real-time network loading. A real-time network loading model is being developed which allows processor-to-processor interactions to be simulated, collisions (and respective probabilities) to be logged, collision-prone areas to be identified, and network control variable adjustments to be reentered as a means of examining and reducing collision-prone regimes that occur in the process of simulating a complete task sequence. The phase-one development results are presented, ~~in this paper~~. Results include 1) the theoretical foundation for the network flow model, 2) an overview of the simulation design and constraints, and 3) the software design. Ultimately, the simulation will be used to examine potential loading problems as out-year demo performance improvements cause increased data traffic. The simulation will also provide a systematic means of managing resulting loading problems.

2. Introduction

Distributed processing systems are becoming extremely common for passing mail between processors that are collocated in the same facility or separated by large geographic distances. Therefore, viable commercial systems have been developed that place processors on communication networks. For purposes of passing mail between processors, studies on Ethernet local network efficiencies have shown mean response frequencies on the order of 39.5 ms, with 100% of all traffic arriving by 200 ms (Ref. 1). Average utilization (on the order of 122 bytes per packet) with 10 hosts (processors) on the net indicates mean arrival times on the order of 10 ms (Ref. 1). Under normal operating conditions, the above response times are excellent. If variable hosts require an intermediate protocol package to ensure message consistency, then an additional mean overhead of 7 to 10 ms is not unusual. Again, if mail passing is the primary occupation of the network, then 20-50 ms is perfectly acceptable.

The Telerobot Testbed is presently employing an Ethernet (with DECnet protocol) system to facilitate interprocessor communication within the overall system computer hierarchy. Where normal mail passing finds response times on the order of 20-50 ms perfectly acceptable, robotic systems find delays in excess of 10 ms undesirable. The key reason why it is important to minimize signal delays is potential control instabilities at the lowest level of the control hierarchy (i.e., the manipulator end-effector servo control level). In hierarchical command and control systems, commands must be passed from the system exec level through several intermediate control levels before they reach the servo control level. In the Telerobot, the operator acts as the system exec by confirming the automated task sequence with the AI planner (the next level). The planner forwards high-level task commands to the run-time controller (the third level in the hierarchy) where each task is broken down into a string of primitives containing important end-point state variables, trajectory via points, and force/torque information. The run-time controller then forwards requests and commands to the manipulator control and sensing/perception processors for control execution (the fourth level of the hierarchy). Additionally, because of mismatches between processor protocols, a network interface package is necessary to maintain protocol consistency over the net. Even though there is a discrete hierarchy for command passing, each processor will be simultaneously managing outgoing commands and incoming requests (i.e., requests for world-state updates from higher level processors). In a static environment in which all fixtures are stationary, the effect of collisions on the net (two or more hosts trying to send data at the same time) will be to merely degrade the rate at which the manipulators and end-effectors respond to incoming commands. However, in a dynamic environment in which objects or obstacles are moving (such as a rotating satellite), the degradation in speed could cause significant control problems and potential damage to the manipulators and end-effectors.

Techniques have been developed which ensure that network collisions are minimized. One of the most popular methods is token ring. The token ring technique basically assumes that message packets arrive according to a random process (Ref. 2). A single control token circulates around the ring from one host processor to the next. When a host observes that the token has been received, a data packet is queued for transmission. The token basically completes an array of "and" gate inputs and allows the packet to be transmitted. Upon completion, the token is passed to the next host, and so on, until it returns to enable another packet to be sent. Although this approach minimizes collisions on the net, it does not enable asynchronous network traffic (such as would be experienced by the telerobot) to propagate back and forth. For example, if an emergency stop signal was required in response to a calculated error, then control problems could arise if the processor that needed to send the signal had to wait a full cycle to receive the token.

Therefore, in developing a simulation for the Telerobot command and control hierarchy, attention had to be paid to developing a more stochastic network event process. Stochastic petri-nets (Ref. 3) and stochastic activity networks (Ref. 4) provided the most fruitful basis for modeling and simulating message traffic on the telerobot distributed processing network. These models were useful because they basically model 1) the arrival of packets, 2) the queuing of packets, 3) the propagation of data, and 4) the detection of collisions. The Telerobot distributed processing hierarchy can be characterized as shown in Figure 1.

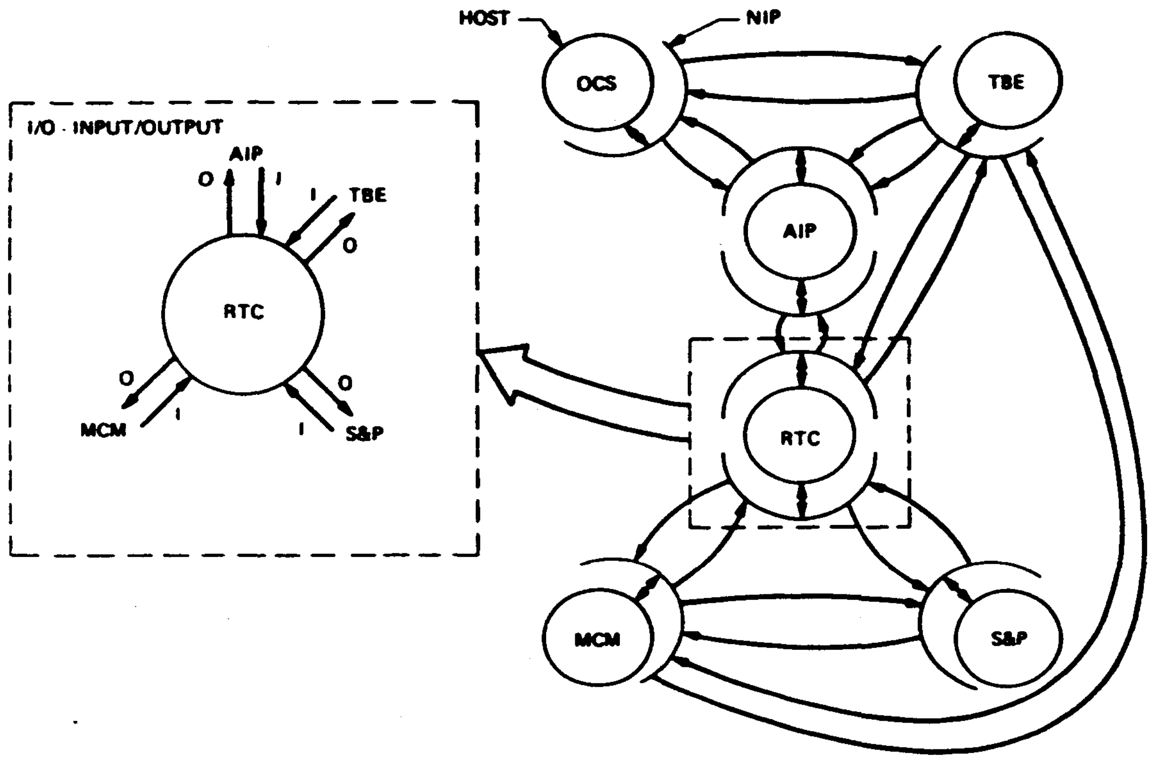


Figure 1. Telerobot Distributed Processing Hierarchy (Automated Control)

Figure 1 shows the operator control station (OCS) interfacing with the testbed exec (TBE) and the AI planner (AIP). During teleoperation the intermediate levels of the hierarchy are bypassed and the OCS/TELEOP interfaces directly with the manipulator control mechanization (MCM) and sensing/perception (S&P) subsystems. The TBE also interfaces with all other subsystems. However, for the FY 1987 and 1988 demos any robust control capability for the TBE will be suppressed; only system initialization and configuration will exist, which primarily requires a one-on-one interface with each subsystem in the hierarchy during start-up. At the AIP level of the hierarchy the planner interfaces with the OCS, TBE, and the run-time controller (RTC). Again, because of the hierarchical design, the TBE and OCS interfaces occur respectively during the system start-up and initialization phase, followed by the task sequence (menu) confirmation phase. The RTC interface occurs alternately as each task element in the sequence is presented to the operator for review/confirmation after the RTC has retrieved and forwarded the various world-state parameters to fill the menu. The RTC interface is also initiated after the menu has been appropriately completed and the AIP has forwarded the first set of execution commands. The RTC then receives those commands and proceeds to pass specific control primitives down to the MCM and S&P subsystems. It should be noted that during task execution at the MCM and S&P level of the hierarchy (the servo control level), any of the upper levels (i.e., the AIP or RTC) can request status reports to monitor task progression and update their respective world-state data bases. The reader should note the indicated interaction between the MCM and S&P subsystems. This interaction exists during the execution of task macros (e.g., grappling a slowly rotating satellite). During the execution of a task macro, communication with the upper levels of the hierarchy can be temporarily delayed to insure that communication delays are minimized in order to prevent potential control instability associated with the dynamic environment. For the FY 1987/1988 demos, the MCM and S&P subsystems will interface with each other through either a parallel interface (in which case the net will be bypassed) or the planned DECnet network interface package (NIP). Figure 1 clearly shows that the area of concern, in terms of asynchronous requests, potential collisions, and subsequent delays in execution, is at the third level of the hierarchy (RTC). Table 1 summarizes the various operational modes, processor involvement, data transmission frequencies, and data rates as expected for the Telerobot Testbed.

Table 1. Expected Testbed Processor Performance

Operational Mode	Processors Involved	Freq. of Data Trans.	Order of Mag. Data Rate (Bits/Sec)
Start-up/Shut-down	OCS - TBE TBE - AIP TBE - RTC TBE - MCM/S&P	$<10^{-3}$ Hz	$<10^{2}$ bps
Status/Initial-ization	TBE - OCS TBE - AIP TBE - RTC TBE - MCM/S&P AIP - RTC RTC - MCM/S&P	10^{-1}-10^{-2} Hz	10^{3} bps
Planning	OCS - AIP AIP - RTC RTC - MCM RTC - S&P	.5 - 1 Hz	10^{3} bps
Execution - Autonomous	AIP - RTC RTC - MCM/S&P TBE - AIP/RTC/MCM/S&P	RTC - 36 Hz	10^{3}-10^{4} bps
		MCM - 36 Hz	10^{3}-10^{4} bps
		S&P - 5 Hz/50 Hz during vision servoing	10^{4}-10^{5} bps
- Teleop	OCS - Teleop Teleop - MCM OCS - MCM OCS - S&P	$\leq$ 5 Hz	10^{3} bps

Considering the above expected loading, the network activity is very low for the start-up, status/initialization, and planning operational modes. This is because at any given time only two processors are actually talking to each other. This is one of the advantages of the hierarchical design. Furthermore, the communication is at a fairly low rate and on the order of "question-answer" type interactions. Since the bandwidth of the Ethernet is on the order of 10 Mbps, the network will only be utilized at a max of .001-.01% capacity. The planning mode similarly falls into the low utilization category. As illustrated by Figure 1 and confirmed by the above table, the area of concern revolves around the autonomous control execution mode. Both the transmission frequency and data rates increase substantially. Even though the utilization only increases to .1-1% capacity, the concern with this particular portion of the system in terms of modeling arises from projected substantial increases in the out-year utilization. For modeling this critical area in the distributed processor configuration it appears that at some time in the task execution mode, three to five processors might be competing for access to the net. Also, the longer the net is occupied by one processor, the greater the odds become that more than one processor will be competing for access to the net. The major overhead variables that surface from the above design and discussion (and confirmed by the literature (Refs. 3 and 4)) are as follows:

1. The number of processors (hosts) attempting to communicate with each other at one time.
2. The frequency at which the processors communicate.
3. The size of the data blocks being communicated.
4. The internal subsystem processing time per standard data block.
5. The internal NIP protocol delay.
6. The queue time for backlogged data packets before they get transmitted (from the NIP to the host).
7. The retransmission delay resulting from a collision.
8. The Ethernet transmission interval per data block.

For this phase-one development activity, the simulation will only model network interactions involving the AIP, RTC, MCM, and S&P. The frequency at which the processors communicate or transmit data packets will be synchronized for task execution commands (i.e., a data packet sent by processor n to processor n+1 must be acknowledged on receipt before processor n+1 can send a packet to processor n+2) and randomly selected for data update requests. The size of the data blocks being transmitted will be randomly selected within the respective bps ranges given in Table 1. The NIP internal overhead will be a constant (i.e., a mean value of 7 ms). The queue time delay will be 0 if the queue is empty, and increased accordingly as the queue increases based on actual experience with the NIP. The retransmission interval will be based on the Ethernet hardware specs and each retransmission time will be selected randomly within that interval. The internal processing time will be established by multiplying the specified average hardware internal overhead per data block times the randomly selected data block size. Similarly, the Ethernet overhead is determined by multiplying the inverse of the 10 Mbps times the randomly selected data block size. Since all processors are collocated in one facility, distance will not be a factor in the Ethernet overhead.

3. Model Description

The basic problem is to establish the time interval required to move an event (e) successfully from processor (n) to processor (n+1), which in turn must move the event to processors (n+2) and (n+3). If, at any time, the state processing and sojourn times for an event generated by a given processor are equal to (or overlap) the state processing and sojourn times for an event generated by an interacting processor, then a collision occurs and the respective events receive an additional retransmission time delay, are placed in a queue, and a new state is calculated/tested to determine if a collision occurs. The event must pass successfully through the four processors before it is discarded and a new event randomly selected. Before the event is discarded, the vital statistics of 1) total sojourn time, and 2) number of collisions are collected and stored for calculating the overall system delay and collision probabilities. The events and interactions are structured as an input/output flow problem with each processor being linked by tagged events (e.g., an event that successfully leaves the AIP (processor n=1) as an output becomes an input to the RTC (processor n+1=2), and so on through the hierarchy). Mathematically, the overhead state time interval (S) for processor n and event 1 (e=1) can be stated as the sum of all internal processor state overheads (s_o):

$$S_{e_o} = \sum_{e=1}^{1} \sum_{o=1}^{p} s_{e_o} \Big|_{n=1} \qquad (1)$$

where, the state overheads are the delays indicated in Section 2 above.

By not suppressing the upper bound on the event summation, the total overhead time for processor (n) can be calculated using:

$$S_{e_o} = \sum_{e=1}^{u} \sum_{o=1}^{p} s_{e_o} \Big|_{n=1} \qquad (2)$$

where (u) represents the total events or commands and requests associated with a given task sequence. It then follows that the total delay (S tot) associated with events passing through the hierarchy (in this case only the AIP (n=1), RTC (n=2), MCM (n=3), and S&P (n=4) can be given as:

$$S_{tot} = \sum_{n=1}^{4} S_{e_o} \qquad (3)$$

Conditionally, if an event arrival at a processor conflicts with another event input/output arrival at the same processor, then a collision (and additional delay) is imposed. The following collision conditions hold for a given event arrival:

either,

$$\sum_{o=1}^{p} s_{e_o}\Big|_{n=1} = \sum_{o=1}^{p} s_{e_o}\Big|_{n=n+1} \qquad (4)$$

or, given the Ethernet overhead (E)

$$\sum_{o=1}^{p} (s_{e_o}) - E_{e_o}\Big|_{n=n+1} < \sum_{o=1}^{p} s_{e_o}\Big|_{n=1} < \sum_{o=1}^{p} s_{e_o}\Big|_{n=n+1} \qquad (5)$$

The above later condition simply means that the net is occupied and processor n+1 cannot accept the command from processor n. The command from processor n would then be placed in processor n+1's queue and assigned a delay interval commensurate with its place in the queue.

Calculation of the collision probability for each processor, and the hierarchy as a whole, is done by merely employing standard statistical expressions using the logged collisions and total events processed.

The last major constraint employed for the model was a definition of "real-time." This was necessary since collisions are really only important if they indeed cause a subsequent degradation in system performance (i.e., speed). Therefore, after each event ripples through the hierarchy and the collision/time delay data are logged, the system delay is compared against the actual lag time of the system to determine if the lag time was exceeded. It is presently planned to use the manipulator control lag as the real-time baseline. For example, this means that if control lag for manipulator movement (under the planned operating speed) is 20 ms, then a collision aggregate delay of 15 ms would not affect operating speed at all. However, as response and operating speeds increase with improvements in manipulator control design, acceptable aggregate delays might be driven down into the 2-3 ms range.

The above model is still being examined from the standpoint of completeness and available data. As the software development proceeds and data bases are assembled, additional changes may be incorporated to further define the processor interaction environment. The next section briefly summarizes the software program being developed to implement the model.

4. Description of Simulation Program

In this section we will illustrate the construction of an event scheduling simulator for the network model discussed in the previous sections. In this simulator each event, created and scheduled with a time tag, represents the main activity of the network model. This time tag decides the ordering of events in the event pool of the system. This ordering also determines the execution of the events and therefore describes the operation of the network model. During the operation, there is always one available event.

As shown in Figure 2, the simulation program consists of six major components: initialization, event access, housekeeping, network activity, event handling, and report generation. The initialization module prompts the required system parameters such as number of desired RTC commands, initializes global variables, and sets the initial conditions of the system. The event access module uses a time tag and event number for each event to select a most recent event for execution. The execution of an event means activation of the event, transmission of the communication packet, and processing of the command actions if any. The housekeeping module advances the system clock based on the time tag of the current event, generates data requests periodically, and collects the required statistics data from the network activity module. The network activity module evaluates the interactions (state overhead) of the available events in the Ethernet network. The event handling module performs the processing of the command actions to generate further activity in the system. It contains several handlers to deal with different types of events. The report generation module assembles the required system delay and collision probabilities once the simulation terminates. The system will terminate when the event access module cannot get an available event, which means the event pool is empty.

Among these six modules, only the network activity module and event-handling module directly relate themselves to behavior of the network model. Therefore, they will be discussed in more detail.

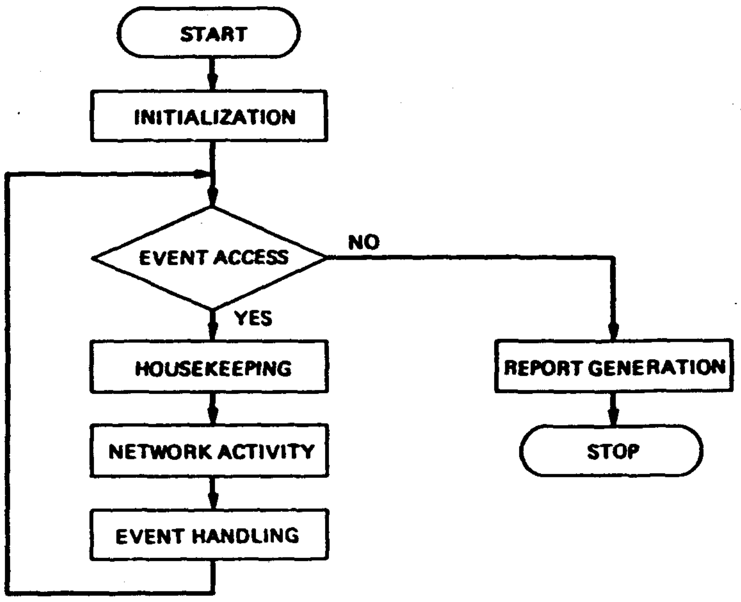


Figure 2. The Main Flow Diagram of the Simulation Program

The flowchart for the network activity module is shown in Figure 3. The main function of this module is to detect and record the collision of events. Collision occurs when two or more events are trying to occupy the network at the same time. Therefore, in order to detect the collision, the simulator computes the period of the current event residing on the network and with a given processor, and then searches through the event pool to check if any other events in the event pool will be arriving during this same period. If a collision occurs, all the involved events will be removed from the event pool and be appended to the network queues associated with the processor. Note that a random time delay will be added to each collided event for retransmission. If no collision occurs, the event becomes active and is ready to be processed (passed to the next processor) by the event handling module. The passed event is an output of the latter processor and an input to the next processor.

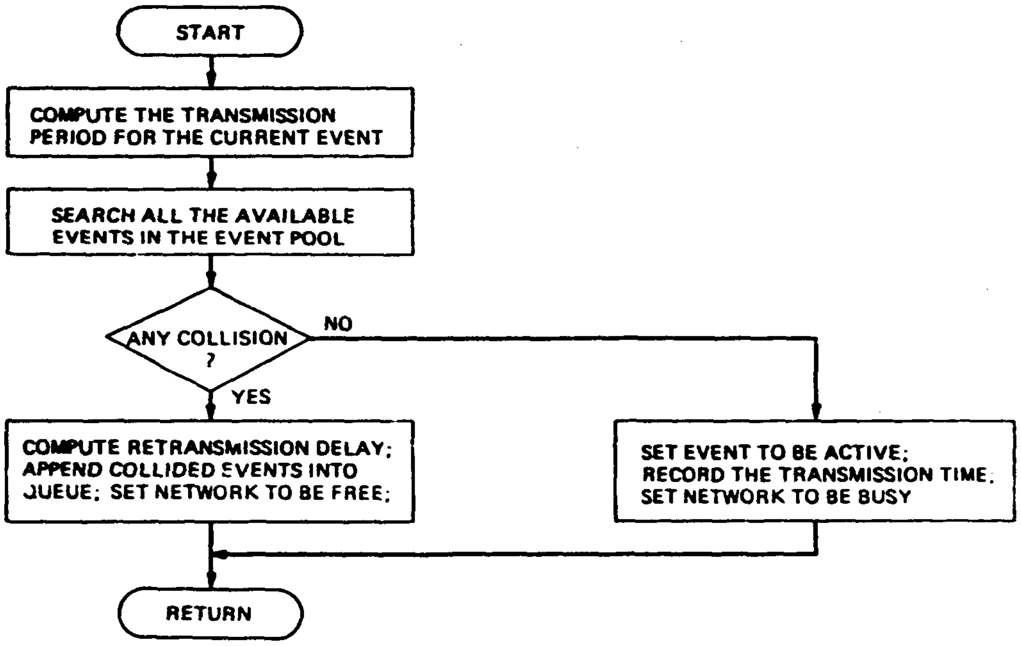

START
COMPUTE THE TRANSMISSION PERIOD FOR THE CURRENT EVENT
SEARCH ALL THE AVAILABLE EVENTS IN THE EVENT POOL
ANY COLLISION ?
NO
YES
COMPUTE RETRANSMISSION DELAY; APPEND COLLIDED EVENTS INTO QUEUE; SET NETWORK TO BE FREE;
SET EVENT TO BE ACTIVE; RECORD THE TRANSMISSION TIME; SET NETWORK TO BE BUSY
RETURN

Figure 3. The Flow Diagram of the Network Activity Module

The flow diagram of the event-handling module is shown in Figure 4. There are four types of events in this simulator: command, acknowledge, request, and response. They will be handled differently. Once a command event is received, the subsystem will generate an acknowledge event back to the originating subsystem. Then the requested command is passed to the next processor. Concurrently, the AIP and the RTC processors use request events to acquire data from lower level of the hierarchy to update their own data bases periodically. The response event is the answer for the command and request events. Since the command event is executed sequentially, the completion of the current event also activates the next command event until all events are executed in the dummy task sequence.

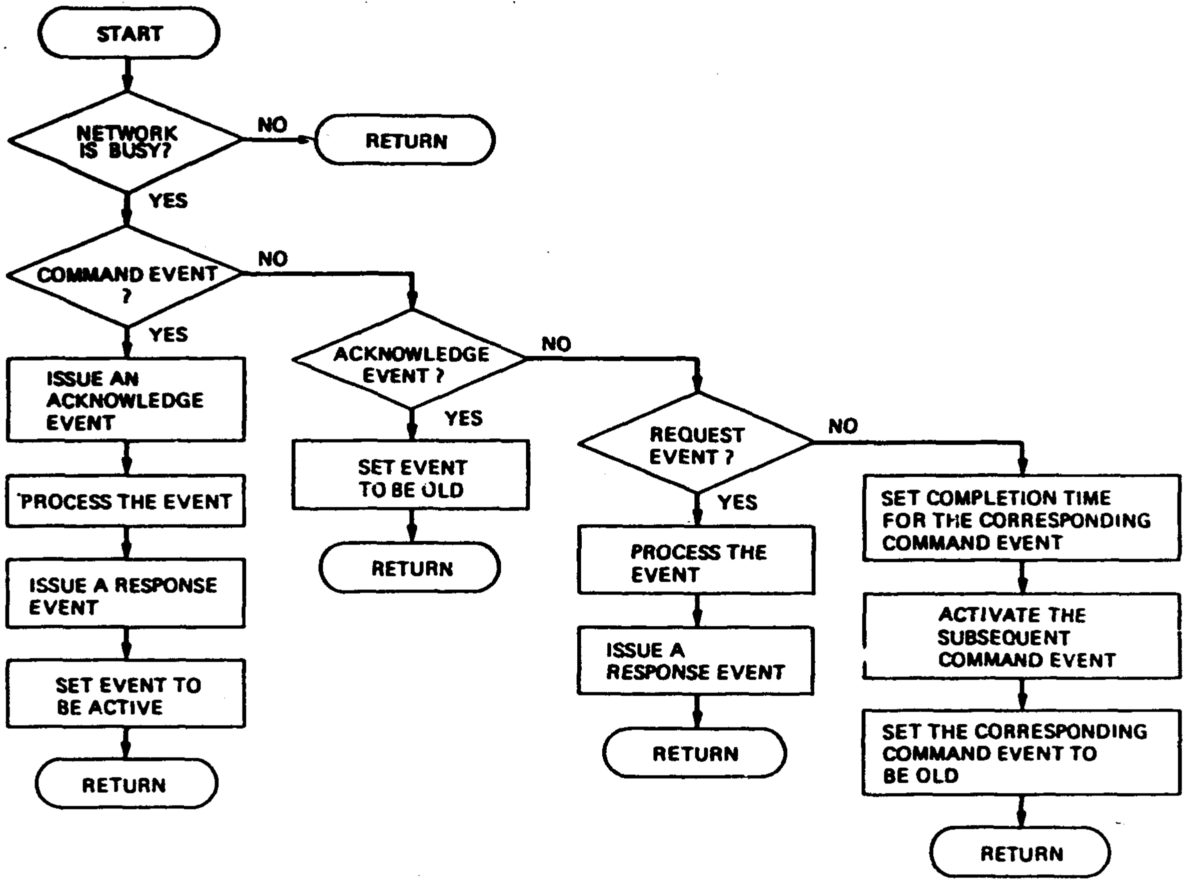


Figure 4. The Flow Diagram of the Event-Handling Module

5. Conclusions

In this paper we discuss the simulation model for a real-time hierarchically distributed system using an Ethernet local area network. The probability and the impact of message collision on the network is the main interest. The conceptual design of the simulation program was completed. The implementation is under way.

Although the simulation is not operable at this time, a considerable amount of information concerning network design and operation has been obtained. This information has been used to design the Telerobot processor protocol and communication architectures in such a manner so as to minimize network interference. These design features include 1) a high bandwidth operating environment (i.e., 10 Mbps) to minimize the overhead on the Ethernet, 2) the overall hierarchical design which prioritizes and reduces the functions that lower levels in the hierarchy must perform, 3) inclusion of a special parallel interface at the servo control level of the hierarchy to facilitate off-net high data rate transfer during critical dynamic coordination tasks (e.g., satellite grappling), 4) adjustment of the Ethernet retransmission interval to correspond with processor priority relative to control execution (e.g., during error management, the RTC and MCM processors require greater access to the net than the AIP), and 5) the inclusion of a queue I/O in the network interface protocol package to accept incoming commands/requests even though the net is occupied.

Using this simulator we can evaluate the system performance in terms of collisions during message transfer on the network, network utility, the data packet retransmission delays, and data request rate. For example, we can determine the response of the network to a utility level (i.e., transmission frequency), establish the lower and upper bound of the data packet size, obtain an optimal data request rate, calculate new data flow control parameters to minimize collisions, and consequently resolve any system bottlenecks.

6. Acknowledgments

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the Office of Aeronautics and Space Technology (OAST), National Aeronautics and Space Administration (NASA).

7. References

[1] J.F. Shoch and J.A. Hupp, "Measured Performance of an Ethernet Local Network," Communication of the ACM, Vol. 23, No. 12, December 1980.
[2] P. Hass and G. Shedler, "Regenerative Simulation Methods for Local Area Computer Networks," IBM J. Res. Develop., Vol. 29, No. 2, March 1985.
[3] M.A. Marsan et al., "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," ACM Transactions on Computer Systems, Vol. 2, No. 2, May 1984, pp. 93-122.
[4] M.H. Woodbury and W.H. Sanders, "An Ethernet Model Simulation Using Stochastic Activity Networks," draft report, Oct. 1986.
[5] B.W. Stuck and E. Arthurs, "A Computer and Communications Network Performance Analysis Primer," Prentice Hall, 1985.
[6] H. Kobayashi, "Modeling and Analysis: An Introduction to System Performance Evaluation Methodology," Addison-Wesley, 1981.

520-61
58
188102

Metalevel Programming in Robotics: Some Issues

A. Kumar N. and N. Parameswaran
Indian Institute of Technology
Madras, India

ID 660960

ABSTRACT. Computing in robotics has two important requirements: efficiency and flexibility. Algorithms for robot actions are implemented usually in procedural languages such as VAL and AL. But, since their excessive bindings create inflexible structures of computation, ~~we propose in this paper~~ Logic Programming ~~as~~ is a more suitable language for robot programming due to its non-determinism, declarative nature, and provision for metalevel programming. Logic Programming, however, results in inefficient computations. As a solution to this problem, ~~we~~ discuss a framework in which controls can be described to improve efficiency. ~~We~~ have divided controls into: (i) in-code and (ii) metalevel and discussed them with reference to selection of rules and dataflow. ~~We have~~ illustrated the merit of Logic Programming by modelling the motion of a robot from one point to another avoiding obstacles.

1. Introduction

Computing in robotics requires both efficiency and flexibility. Large scale real time computation, both symbolic and numeric, is necessary for even apparently simple robot movements. Robot's actions must be as time-efficient as possible at the end-effector and actuator levels[1]. At the same time, planning, backtracking, consulting knowledge about the world etc., requires flexibility.

The world of the robot is dynamically changing, as in the case of telerobots, and the change in the world must be continually noted. Accordingly, the planning of the robot at the object and the objective levels has to be altered / readjusted. This is a strong case in favour of flexible computation for robotics at the object and the objective levels.

In robotic computation, a third requirement is effective man machine communication [2]. This requires that the programming environment be as conducive to natural input and the language be as readable as possible.

Traditionally, languages for robots are procedural, such as VAL[3], AL [4] and RAPT [5]. They provide for efficient computation, but are not flexible or easily modifiable.

We propose that Logic Programming is more suitable for robot programming due to its non-determinism, declarative nature, and provision for metalevel programming. Description of the world is most conveniently done in the declarative semantics of Logic Programming. Dynamic change in the world can be easily incorporated by the addition / deletion of logical assertions in the program. The fact that visual world can be best described declaratively has been exploited in [6] for declarative graphics.

However, a major problem in using Logic for computation in robotics is its inefficiency. Typically, a Logic Programming language such as Prolog follows the depth-first strategy with chronological backtracking and a few control features such as cut to improve efficiency [7]. However, recently, many features have been introduced in Prolog, to improve efficiency, such as intelligent backtracking[8,9], annotated variables[10], and metalevel programming[11,12].

Of greatest relevance to robotic computation is the last feature viz., metalevel programming. Here, the specification of control strategy for the object level program (OLP) is expressed seperately, at a different level called metalevel. It permits one to intervene in the interpretation of the object level programs to define new strategies of control. It also allows one to specify one's own interpreter.

Since metalevel description is kept entirely seperated from logic level, the basic procedures and world description at the object level are left entirely untouched by efficiency considerations. Object level programs are still as declarative and flexible as ever. An additional element of flexibility is that the control strategy for any computation can be changed to suit the needs of the current goal and world.

2. In-code and Metalevel Controls

For robot programming, we present control at two levels: (a) in-code, (b) metalevel; and discuss them with respect to two types of controls : (i) selection control (ii) data flow control.

In-code control refers to control expressed textually along with the OLP clauses and the syntax and the semantics are defined as part of the OLP language. At this level we provide:

* efficient constructs which are procedural in nature.
* liberal bindings of as much dataflow as possible, using the idea of locations.

However, at the global level, we propose largely a declarative language.

Metalevel control refers to control over selection of rules and is textually separated from OLP. At this level, control is expressed in the form of rules (different representations are possible) so that it is easy to modify when required.

This strategy of splitting the controls has the following consequences on computation :

1. Locally, the computation is expressed as efficiently as possible at the cost of flexibility. But, since it is only local, the resulting inflexibility may be acceptable as, in the worst case, it can always be replaced when modifications are required.

2. Globally, the structure of computation is kept as flexible as possible, at the cost of efficiency i.e.,the programs need not be radically re-altered when the world or the input changes for a robot. Efficiency is improved by carefully choosing metalevel rules. Hence, readability and modifiability of programs are preserved, which is very important for robotic computation.

The following are the control features we have introduced to improve the efficiency:

CONTROL FLOW:

Incode: The scope of in-code controls is restricted to individual clauses and they are directed to specific predicates.

(i) Forward Jump F(X): Let F(X) be a predicate in the body of a clause C. The execution of this predicate results in the transfer of control to the predicate in the clause C, to which X is instantiated, skipping the predicates between F(X) and X in C. F(X) fails when X is uninstantiated or when backtracking.

(ii) Backward Cut !(X): Let !(X) be a predicate in the body of a clause C. !(X) succeds in the forward direction trivially. While backtracking, if !(X) is encountered, control is transferred to the predicate X occuring within the clause C.

Metalevel: Several control features defined in Metalog[11] actually fall under this: CHOOSECLAUSE, INHIBCLAUSE, INHIBACK, FACTOR and BACKFAIL. In addition, we propose OR-parallelism as a control structure on the set of modules. This makes it possible to control parallelism by controlling the number of processes. An important restriction we have placed here is that these controls are expressed with respect to clauses and not individual predicates. Thus the control here will tell which clause to use but not which predicate in a clause. Controls of this type affect the top-to-bottom selection of clauses for interpretation within a program. The scope of all the metalevel declarations is the entire program.

(i) OR Parallelism

(PARCHOOSECLAUSE literal i[1],i[2],...,i[n] clausebody) :- condition.

The execution of this predicate results in the simultaneous selection of n clauses for interpretation in OR parallellism where the clause heads are unifiable with literal i[j] denotes the j-th clause among the m clauses whose heads unify with literal. When n = 1, and i[1] = 1, PARCHOOSECLAUSE reduces to CHOOSECLAUSE of Metalog[11].

DATAFLOW:

Incode: In Prolog, data flow within a clause occurs through side-effects in the sense of [13]. We introduce new predicates create and assign to let data flow both in the forward and backward (on backtracking) directions. This unlimited data flow within a clause enhances efficiency considerably.

(i) create(L1,L2,...,Ln): Let C be the clause in which the predicate create occurs.

Execution of this predicate in the forward direction results in the creation of locations Li, i = 1,...,n. These locations can be used to store values using the assign predicate described below. Create fails during backtrack.

(ii) assign(L,expression): Execution of this predicate results in storing the value of the expression in L, where L is a location. It fails when L is not created, and during backtrack.

Metalevel: At this level the rules define global data locations for the program. Data flow can occur across the clauses through parameters or global locations when it is convenient. Distributed computing for coordinated activities in a colony of robots can be achieved by appropriate metalevel descriptions of control and data flow.

Global locations

GLOBAL L1,L2,...,Ln.

This declaration at metalevel defines n locations L1,...,Ln that can be used as locations to store data and use them later in the program; they are visible over the entire program.

3. ILLUSTRATION

In this section, we address the problem of a robot moving in a world, avoiding obstacles. For simplicity, the world is divided into a number of squares. The obstacles are distributed in the world as shown in Figure 1. The start position of the robot is assumed to be the square (1,1), denoted by p(1,1). The target is the square p(6,4). We restrict the motions of the robot to horizontal and vertical directions only.

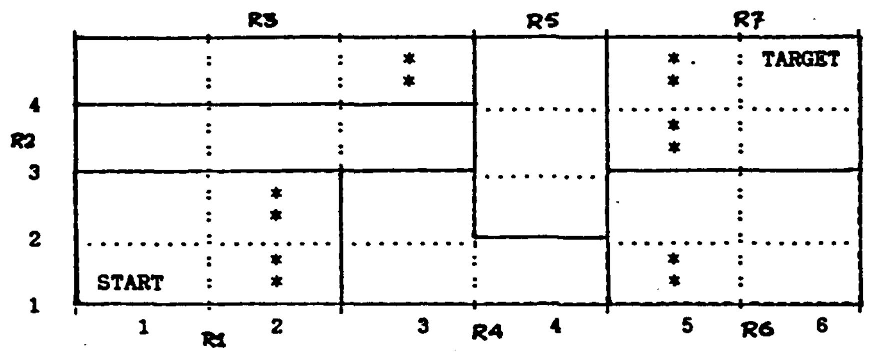


Figure 1: The robot world.

The solution is provided at two levels: object level, and metalevel. At the object level, the mechanical actions and the simple heuristics the robot employs on encountering obstacles are described. At the metalevel, knowledge about the distribution of the obstacles in the world and the mechanisms for supplying information about local regions in the world are described in order to guide the robot towards its target. The knowledge encoded in the object level description is complete in the sense that the knowledge is sufficient, in principle, for the robot to reach any given target, by exhaustive search. Metalevel knowledge is incomplete in this sense, but it can greatly aid in reducing the exhaustive search, and thus improving the efficiency.

At the object level, we have the following heuristic to guide the robot: Let the current position of the robot be (Cx,Cy) and the target location (Tx,Ty), where Tx >= Cx and Ty >= Cy. If Tx = Cx and Ty = Cy, then the robot has reached its target, and therefore it halts; else, the robot moves to the square (Cx + 1, Cy) if Tx > Cx and (Cx + 1, Cy) is obstacle-free, or to the square (Cx, Cy + 1) if Ty > Cy and (Cx, Cy + 1) is obstacle-free. Otherwise, it consults the expert in the current region.

At the metalevel, the squares are grouped into regions. For example, p(1,1), p(1,2), p(2,1) and p(2,2) form R1.(see Figure 1.) Each region has an expert who has the knowledge of the obstacles present in that domain. The expert has the ability to suggest a way out to each of its neighbouring regions, when consulted. For example, when consulted, R1 shows the way out to R2 via the square p(1,3). At the metalevel, we also have the following control heuristic to avoid infinite paths: In any region Ri,the robot ignores the advice of the expert of this region, if the advice leads the robot back to any of the previously travelled region Rj. In case of failure of a solution suggested by an expert, the robot backtracks to any alternative solutions that the expert may have suggested. If all the suggestions of the current expert fail to provide an exit, the robot will take up any untried alternative suggestions of the earlier expert; and so on.

The robot initially traces the following squares: p(1,1), p(1,2), p(1,3), p(2,3), p(3,3), p(4,3), p(4,4). At this point, the robot is not able to move further on its own, and so it consults R5. R5 suggests the way out to R2, R4 and R6. The way out to R2 is ignored. If the robot accepts the way out to R4, via p(4,1) or p(3,2), the robot finds itself once again at a dead end. However, on backtracking, it tries the third solution suggested by R5, thus entering R6, and the target eventually.

```
/* Robot: Object level program */
/* Initialise stack for local expert and start */

toptrav( p(*X,*Y), p(*X1,*Y1) ) :-  create ( *list ),
                                    assign (*list, [] ),
                                    trav ( p(*X,*Y), p(*X1,*Y1) ).

trav( p(*X,*Y), p(*X,*Y) ) :- !
trav( p(*X,*Y), p(*X1,*Y1) ) :-
                         advance(p(*X,*Y),p(*X1,*Y1),p(*X2,*Y2)),
                         trav( p(*X2,*Y2), p(*X1,*Y1) ).

advance( p(*X,*Y), p(*X1,*Y1), p(*X2,*Y2) ) :-  *X1 > *X,
          *X2 is *X + 1,
          checkobst( *X2,*Y ), /* A built-in procedure that checks
                                            whether the square (X2,Y) has an obstacle */
          in_region( p(*X2,*Y), *R ), /* Finds out the region to which (X2,Y) belongs */
          test_append( *R, *list ), /* Checks whether *R has already been traversed */
          move( p(*X2,*Y) ), /* A built-in procedure that enables the robot move by a square */
          F(  !  ),  move( p(*X,*Y) ),  !( move ).  /* The robot retraces its path
                                                       in  case  of failure */

advance( p(*X,*Y), p(*X1,*Y1), p(*X2,*Y2) ) :-  *Y1 > *Y,
                          *Y2 is *Y + 1,
                          checkobst( *X,*Y2 ),
                          in_region( p(*X,*Y2), *R ),
                          test_append( *R, *list ),
                          move( p(*X,*Y2) ),
                          F( ! ), move( p(*X,*Y) ), !( move ).

advance( p(*X,*Y), p(*X1,*Y1), p(*X2,*Y2) ) :-
                          in_region( p(*X,*Y), *R1),
                          consult(  *R1,  p( *X,*Y ),  p( *X2,*Y2 )).  /* If met with a dead end,
                                                                         consult local expert */

/* To check whether R has been already consulted */

test_append( *R, *list ) :- member( *R, *list ), !.
test_append( *R, *list ) :- append( *R, *list, *list1 ),
                            assign( *list, *list1 ).
```

The metalevel description defines the local experts, and the position of obstacles. Associated with each expert is its knowledge regarding the position of squares which the robot should try to reach in order to avoid infinite paths.

```
/* Local Experts: Metalevel program */

consult( *R1, p(*X,*Y), p(*X2,*Y2) ) :- out( *R1, *Z ),
                        member( p(*X2,*Y2), *Z ),
                        in_region( p( *X2, *Y2 ), *R2 ),
                        not( member( *R2, *list ) ),
                        advance(p(*X,*Y),p(*X2,*Y2),p(*X2,*Y2)).

region( R1, [ p(1,1), p(1,2), p(2,1), p(2,2) ] ). /* Definition of region R1 */
region( R2, [ p(1,3), p(2,3), p(3,3) ] ).
region( R4, [ p(3,1), p(3,2), p(4,1) ] ).
                ..
                ..
out( R5, R2, [ p(3,3) ] ).
out( R5, R4, [ p(3,2), p(4,1) ] ).
out( R5, R6, [ p(5,2) ] ).
                ..
                ..
```

```
/* Description of location of obstacles */
obst( 2,1 ).            obst( 2,2 ).        obst( 3,4 ).
obst( 5,1 ).            obst( 5,3 ).        obst( 5,4 ).
```

4. Conclusion

In this paper, we have discussed the merits of Logic programming for robotic computations. Towards improving the efficiency of computation, while retaining the declarative nature of programming, we have split the control into two levels: in-code and metalevel. We have proposed several constructs at both these levels, that improve efficiency. We have illustrated the elegance of metalevel logic programming and the usage of the predicates we have introduced, on a simple robot problem.

REFERENCES:

1. Kogan Page,Logic and Programming., Robot Technology series Vol 5., London 1983.

2. Ambler A.P., 'Languages for programming robots', in Robotics and Artificial Intelligence, NATO ASI series 1984.

3. 'Users guide to VAL', Unimation Inc.

4. Goldman R., Mujtaba S.M.,'The Al users manual', Stan-CS-79-718, Stanford Univ.,Jan 1979.

5. Popplestone R.J.,Ambler A.P.,Bellos T.M., 'An interpreter for a Language for describing assemblies', AI, vol.14, 1980, pp79-107.

6. Richard Helm and Kim Marriott, 'Declarative Graphics', Proc. of Third Intl. Conf. on Logic Programming, July 1986 (Lecture Notes in Comp. Science 225).

7. W. F. Clocksin and C. S. Mellish, 'Programming in Prolog' Springer Verlag, 2nd. edition 1984.

8. M. Bruynooghe and L. M. Pereira, 'Deduction revision by intelligent backtracking', in 'Implementations of Prolog' ed. J. A. Campbell, Ellis Horwood Ltd., 1984.

9. P. T. Cox, 'Finding backtrack points for intelligent backtracking' in 'Implementations of Prolog' ed. J. A. Campbell, Ellis Horwood Ltd., 1984.

10. Clark K. L., and McGabe F. G., 'IC-PROLOG : Language features' Proc. Logic Programming Workshop, Debrecen, July 1980.

11. Mehmet Dincbas et al., 'Metacontrol of Logic programs in Metalog' in Proc. of the Intl. Conf. on Fifth Generation Computer Systems, ICOT, 1984.

12. Colmerauer A., 'PROLOG-II:Manuel de reference et modele theorique', GIA, Faculte des Sciences de Luminy, Mars 1982 .

13. Terrence W. Pratt, 'Programming Languages: Design and Implementation', Prentice Hall Inc., 2nd edition 1984.

S21-61

188103

An Approach to Distributed Execution of ADA Programs

R.A. Volz, P. Krishnan, and R. Theriault
University of Michigan
Ann Arbor, MI 48109

MX 270710

Abstract

Intelligent control of the Space Station will require the coordinated execution of computer programs across a substantial number of computing elements. It will be important to develop large subsets of these programs in the form of a single program which executes in a distributed fashion across a number of processors. ~~In this paper we describe~~ a translation strategy for distributed execution of Ada programs in which library packages and subprograms may be distributed. A preliminary version of the translator is operational. Simple data objects (no records or arrays as yet), subprograms, and static tasks may be referenced remotely.

is described.

1. INTRODUCTION

Intelligent control of the Space Station will require the coordinated execution of computer programs across a substantial number of computing elements. It will be important to develop large subsets of these programs in the form of a single program which executes in a distributed fashion across a number of processors. The single program approach to programming closely coordinated actions of multiple computers allows the advantages of language level software engineering developments, e.g., abstract data types, separate compilation of specifications and implementations, and extensive compile time error checking to be fully realized across machine boundaries. In this paper we describe one approach to a translation system for distributed execution of Ada programs. We consider loosely coupled homogeneous systems in which the program/processor binding is specified within the distributed program (static binding).

There have been a number of proposals for distributed programming languages, [1], [2], [3], [4], [5], [6], [7],& [8] to name but a few. Most of these language proposals have emphasized models for communication and synchronization and/or a unified treatment of data abstractions and multi-processing that are amenable to correctness proving. With few exceptions, e.g. [7], however, they have considered neither the real-time aspects of the languages nor the full problem space (see [9]) involved in distributed execution. Only a few of these languages likely to see widespread adoption and use, though the important principals they lay down are likely be adopted in future language designs.

Ada, on the other hand, will see widespread use, and explicitly admits distributed execution. As yet, there have been only a few attempts at actually distributing its execution. Tedd, et. al. [10], describe an approach that is based upon clustering resources into tightly coupled nodes (shared bus) having digital communications among the nodes. They then limit the language definition for inter-node operations (e.g., no shared variables on cross node references); they are currently in the process of implementing their approach. Cornhill has introduced the notion of a separate partitioning language [11] that can be used to describe how a program is to be partitioned after the program is written. [12] describes this language in greater detail. Again, neither of these approaches recognizes the full problem space involved in the distributed execution of programs. [9] describes a number of difficulties which both approaches must face if they are to remain within the current Ada definition.

[9] introduces three major dimensions to the problem of distributed program execution: the memory access architecture, the binding time, and the degree of homogeneity. The range of distributed execution systems that can be represented by these dimensions is larger than any of the distributed language efforts to date, including Ada, can address. For example, one of the major design decisions that must be made in a distributed language is the units of the language that may be distributed. It is likely that one will want to make the distributable units a function of these dimensions. For example, one may want to allow

[1] Ada is a registered trademark of the U. S. Government (Ada Joint Program Office)

[2] This work has been sponsored by General Dynamics, General Motors, and NASA

shared variables for some architectures and disallow them for others. [10] does this by disallowing shared variables across node boundaries. Yet the Ada Reference Manual is not clear on this point. There are also questions of to what extent the specification of distribution should be part of the language (as opposed to being stated in a separate configuration phase); [9] discusses these and several other issues. It is clear that the Ada language definition is really not complete with respect to distributed execution.

The work described in this paper is, thus, principally an experimental device to help identify the basic issues and point toward their solution. We restrict consideration to homogeneous loosely coupled computers with static binding specified within the program (one point in the problem space indentified in [9]), and follow the suggestion in [9] and allow only library packages and subprograms to be distributed. We endeavor to avoid placing any further restrictions on the language, and study the implications of effecting distributed execution within these constraints. This implies the need for remote access to data objects, subprograms and tasks.

Rather than write a complete Ada translator, we adopt a pretranslator approach in which we translate a distributed Ada program (- in our system a distributed Ada program is a normal Ada program with SITE pragmas indicating where units are to be placed) into a set of normal Ada programs. We then use existing Ada compilers to translate each of the programs in the set. This approach has the dual goals of being a simpler experimental mechanism and utilizing existing work where possible. It also has a few limitations, which will be pointed out in the remainder of the paper.

The next section presents an overview of the approach. Section 3 then discusses the critical problems and the details of the approach taken. Section 4 analyzes the project performance of the method and section 5 summarizes the current status of the work and discusses directions that must be explored in the future.

2. OVERVIEW OF THE PROBLEM AND APPROACH

The Problem

We presume that the computers upon which a program is to be distributed are interconnected by a communication network, as shown in figure 1. Since we are allowing distribution of library packages and subprograms, our translation system must provide a means of accomplishing the following remote operations:

- Access to procedures and functions declared in remote library units,
- Reading and writing of data objects declared in remote library packages (and hence stored remotely), i.e., remotely shared data is allowed in our model,
- Making [timed/conditional] entry calls on tasks declared in remote library packages,
- Declaring/allocating (local) variables whose types are declared in remote library packages,
- Elaborating tasks whose types are declared in remote library packages,
- Managing task termination for tasks elaborated across machine boundaries.

The approach

The first issue that must be considered is the representation of the distribution. In our system, we write a single program and place a pragma called SITE before each library unit to specify the location on which that library unit is to reside. For example, consider a mobile space robot system consisting of several mobile vehicles (each with a robot mounted on it) and an overall system controller. If it were desired to have one vehicle controlled by computer number 2 and the overall control using it (as well as several other similar systems) placed on computer 1, a sample of the relevant code would look as follows:

```
pragma SITE (2);
package VEHICLE is
    procedure MOVE(..);
          :
end VEHICLE;
    ______________________________

pragma SITE(1);
with VEHICLE;
procedure CONTROL is
   :
begin
   :
   VEHICLE.MOVE(..);
   :
end CONTROL;
```

Figure 2 illustrates the overall operation of our system. A pre-translator translates our single Ada program with SITE pragmas into a set of independent Ada programs which include library modules of our design to effect communication among processes. The translation system would replace all calls (within CONTROL) to the procedure VEHICLE.MOVE(..) with the appropriate remote procedure call. Similarly any references in CONTROL to data objects or task entries defined in package VEHICLE would be translated into appropriate remote references. Each of the individual programs can thus be compiled by an existing Ada compiler. This approach simplifies the translation process considerably since our pre-translator is much less complex than a full Ada compiler.

The remote operations and interprocess communication are managed by a set of *agents* created for units that can be remotely referenced and an underlying process to process mailbox system: During the pass through the pre-translator, all references to remote objects are replaced with appropriate references to *agents*. Each program unit (for example, suppose it is unit A) which might be referenced from another, remote, unit (call it B) has three categories of agent associated with it. There is a *local agent* on the site holding A, a *remote agent* on the site holding B (and every other site referencing A), and a *pointer agent* placed on the same site as the local agent. A is essentially unchanged by the pre-translator, and both the local and remote agents can be generated from only the specification of A. All references within B to code or data objects in A are translated into appropriate calls to the remote agent of A (which resides on the same site as B), which in turn uses the message system to pass the service request to the local agent of A. The local agent performs the necessary functions, returning any objects requested. The pointer agent serves to propagate object accesses involving access variables pointing to other sites, and is only required if such access variables are used.

The relevant entities and flow of data for the example above are shown in Figure 3. CONTROL_T is the translation of CONTROL with the references to objects in VEHICLE replaced with calls into VEHICLE_REM_AGENT.

3. TRANSLATION STRATEGY

In order to solve the problems raised in the previous section the following issues must be resolved:

- development of a general remote object accessing methodology,
- translation of source code references to remote objects,
- management of other remote service functions, e.g., creating objects, and
- generation of the agents.

The solution to these problems, while leading to reasonably efficient code, involves a rather complex set of multiple pass operations and the generation and use of a number of auxiliary files of intermediate information. Thus, a set of utilities also are needed to allow the user to perform these operations in a straightforward manner.

By far the most complex of these issues is the development of a general remote object accessing method. This is complicated by the need to address arbitrarily nested record and array components and the fact that component pointers may point to logically nested records or arrays on other processors. We thus concentrate our discussion on matters relating to object access. The solution to most of the other issues follows the resolution of this problem in a reasonably straightforward manner.

The structure of the agents is critical to solving this problem, and is generally in three parts: 1) elements to access code objects, 2) elements to manage the address chain leading through qualified names of records and arrays, and 3) elements to manage other services. The interprocessor mail system and message structure is closely integrated with the agent structure.

We begin this section with a discussion of the overall agent structure and its use for accessing code objects, and then discuss related important issues of access via fully qualified names, the postal message structure, and the translation process.

Agent Structure

As mentioned in the previous section, three kinds of agents are generated whenever a library unit specification is encountered by the pretranslator: a local agent, a remote agent, and a pointer agent. Each agent is generated as a separate package, and assigned a unique name that is derived from the source package name. The agents can be generated simply from the package or subprogram specifications.

The terms *local* and *remote* agent are used with respect to the processor holding the library unit which they represent. That is, the local agent resides on the same node as the unit it represents, while the remote agent resides at each other node referencing the unit. Thus, a remote action of some kind begins with the referencing unit making a call (after it is processed by the pre-translator) to the remote agent of the unit being accessed. For example, if the cell controller CONTROL makes a call to VEHICLE.MOVE(..), the translated procedure CONTROL_T makes a call to the remote agent VEHICLE_REM_AGENT. We thus consider remote agents first.

Remote Agents

Remote agents are merely collections of procedures and functions that effect remote calls. In the case of subprogram and task entry calls, they present an interface to the calling package identical to that of the original source package. In our previous example, the package VEHICLE_REM_AGENT will contain a procedure MOVE(..) that will receive the call intended for package VEHICLE. These procedures and functions format an appropriate message record (described below), and dispatch it to the appropriate site via the postal service. When a return value is received from the local agent (on the other processor) via the postal service, this value will be returned to the calling unit. From the perspective of the calling unit, the facts that the action is remote and that there are (at least) two agents in between it and the called unit are transparent, except for the longer time required. Thus, no translation of subprogram or (simple) task calls is required in the calling unit, unless they use arguments residing remotely.

In the case of remote data object references, a transparent interface is not possible. Instead, a set of procedures to get and put the values of remote objects of various types is generated. Again, these procedures are generated from the specification of the library package being referenced. In this case, the referencing unit (CONTROL in our previous example) must be translated to replace the object reference with a call to the appropriate get or put routine. Since the get and put routines can be overloaded (with respect to the specific argument types used) the translation is straightforward. The specific arguments used and the detailed actions of the get and put routines are closely intertwined with the management of fully qualified names, and will be discussed later.

Two other points are worth mentioning. Since the structure of the remote agent was chosen to minimize the impact on the referencing unit, the translation required by the pretranslator is a minimum. Also, since sending and receiving messages from another processor is time consuming (relative to normal instruction processing times), the point after the transmission of a message is treated as a sychronization point so that other tasks may obtain the services of the cpu while the reply to the message is in progress.

Local Agents

The local agents are the most complicated of the three agent types. Their task is to service requests from remote sites needing to access data objects, subprograms, or task entries. A local agent consists of N+2 tasks where N is the total number of functions, procedures, and task entries, contained in the source specification of the unit the agent is helping to represent. One of these tasks is associated with each of the aforementioned subprograms and task entries.

One of the remaining two tasks is designated as the local agent *main* task. This task consists of a single loop that requests message records from the postal service (via a task entry call), interprets the request, and dispatches the request to the appropriate handler (task or procedure) within the local agent. Messages requesting access to data objects, are serviced immediately within the main task by calling a GETPUT procedure (described below) and an immediate reply is sent.

```
with PACKAGE_NAME;              package being represented
task body AGENT_MAIN is
   M: MESSAGE_TYPE;
begin
   loop
      POSTAL.MAIL_BOX.GET(M); - - get message
      case M.OBJ_ENUM is          - - Branch according to object name.
         - - Object references
         when NAME_1 => GETPUT(M, PACKAGE_NAME.NAME_1);
                  ⋮
         when NAME_K => GETPUT(M, PACKAGE_NAME.NAME_K);
         - - Subprogram calls and task entries
         when NAME_K1 => MANAGER.DEPOSIT_NAME_K1(M);
                  ⋮
         when NAME_N => MANAGER.DEPOSIT_NAME_N(M);
      end case;
      SEND_RETURN(M);
   end loop;
end AGENT_MAIN;
```

The above abstraction is only for a single distributed package. Actually, the message type would be embedded in a yet more general record having a varient part for each distributed package, and the actual code would be slightly more involved.

It is imperative that the main task not be blocked for it provides concurrent access to all objects and types in the specification of the unit it represents, and if blocking occurred here, other, parallel, requests could be delayed. In particular, the agent must not be blocked by a unit it calls on behalf of a remote client, as could occur if the agent directly called the unit (the subprogram called might, for instance, become blocked on an I/O wait). That is why a *task* is associated with each subprogram and task entry. The main task places the message received in a buffer, by calling an entry in a buffer manager task (the last of the tasks in the local agent). A flag counter corresponding to the requested call [illegible] also incremented at this time.

The buffer manager task has N additional entries, whose acceptances are conditioned on a positive value of each of the corresponding N counters (indicating that there is a message to be retrieved).. There are N call manager tasks (the N tasks corresponding to subprograms and task entries), whose sole purpose is to retrieve a message record from the corresponding conditional task entry in the buffer manager task, execute the appropriate call to the source package body, and then return a reply to the remote site.

The following code abstraction illustrates the manager task and the tasks corresponding to the subprograms and task entries that may be called.

```
task MANAGER is
    entry DEPOSIT_E1(MESG : in MESSAGE);
    entry DEPOSIT_S1(MESG : in MESSAGE);
          ⋮
    entry EXTRACT_E1(MESG : out MESSAGE);
    entry EXTRACT_S1(MESG : out MESSAGE);
          ⋮
end;

task body MANAGER is
    E_FLAG: array(1..MAX_ENTRIES) of INTEGER;
begin
    loop
        select
            accept DEPOSIT_E1(MESG : in MESSAGE) do
                - - deposit the message for e1
                E_FLAG(1) := E_FLAG(1) + 1;
            end;
                      ⋮
        or
            when E_FLAG(1) > 0 =>
                accept EXTRACT_E1(MESG : out MESSAGE) do
                    - - extract a message from the buffer and return it
                    E_FLAG(1) := E_FLAG(1) - 1
                end;
                      ⋮
        end select;
    end loop;
end MANAGER;
```

The suffix EI indicates the Ith entry point, and the suffix SI indicates the Ith subprogram. The structure of the entry task for entry E1 is as follows:

```
task DO_E1
begin
    loop
        MANAGER.EXTRACT_E1( );
            E1( );
            - - send back message;
    end loop;
end DO_E1;
```

The messages are provided by a mailbox system that delivers messages to the correct local agent. The message interpretation and task calls by the agent essentially achieves a routine to routine communication between routines in the remote and local agents in a way that prevents delays in the response to one request from locking out other parallel requests.

Pointer agents

Our allowed model of distribution permits access variables to point to objects on machines other than the one holding the access variable. Since access variables, as defined within a local Ada program, clearly cannot contain both the machine identity and an address, whenever an access type definition is encountered in the source package, it is replaced by a record structure containing two fields: a site number, and the original access type. This new record type is then used in place of the access type. The site number is always checked against the current site number, to determine whether the object being pointed to is on the local site, or on a remote site.

Because access variables can be passed from one machine to another, it is possible for a processor to hold an access variable pointing to an object on a site which it does not directly reference and for which it does not therefore have an agent. We therefore include pointer agents to allow access to objects on remote sites. The structure of pointer agents is similar to that of local agents, except that provision for subprogram calls need not be made.

Remote Data Object Access

Three characteristics of Ada data objects cause difficulty in developing a general mechanism for handling references to remote objects: 1) the objects may be composite objects, 2) they may have concatenated names, and 3) parts of a fully concatenated name may be access variables pointing to objects on other machines.

The first issue manifests itself when one must copy a composite object (as opposed to a component of the object) from one site to another. For example, suppose that site 2 uses a record A on the right hand of an assignment statement and that A is located on site 1. Eventually, the agent and message system must convert A to a bit string for transmission. It would usually be desireable that the part of the system that performs the conversion not be aware of the structure of the object (from object oriented design principles). However, if the object contains a memory address as part of its structure, the result received could be meaningless. For example, suppose the record A contains a variable length array, as shown below.

```
subtype S is INTEGER range 1..MAX;
type IA is array (INTEGER range <>) of INTEGER;
type R(L: S := 1) is
    record
        B: IA(1..L);
        C: INTEGER := 0;
    end record;
A: R;
```

One decision for the memory allocation for the record might be to allocate the storage for the array from a heap and place only a pointer to the array (or possibly its dope vector) in the record. The need to perform whole object (record) assignments in Ada might discourage such a memory allocation scheme, but nevertheless, it is certainly a possibility. A bit by bit copy of the block of data corresponding to the record A, would then copy this address, which would have no usefulness when received by the requesting unit; in particular, the bit by bit copy of the record block would not result in the array values being transmitted. To avoid this problem, the routine that does the final message transmission must, indeed, contrary to the above assumption, have knowledge of the record structure so that the array values themselves may be transmitted, and not just the address of the array. Since we are describing a pre-translator approach that uses existing Ada compilers, this knowledge is dependent upon the implementation of the underlying compilers used.

To see the second issue, suppose that site 2 contains a statement like $X := A.C$. How does one construct an address for A.C? Or describe, in a general way, to the agents what element is to be returned? The syntax "A.C" exists only on site 2, and the only information available there from the specification of the package containing A is the logical record structure of A, not its physical structure. Again, implementation dependent knowledge of the rules used for construction of the physical structure of records is necessary.

If one were to now add a fourth component, D, to the type R above, that is an access type, and if the value of A.D were to point to another record stored on site 3, the third issue arises. The method used to calculate the address of the item to be retrieved must not only contain implementation dependent knowledge, but it must be distributed as well.

Strategies for Remote Object Access

We are studying two methods of obtaining composite (as well as scalar) objects: 1) using knowledge of the rules for storage allocation and physcial record and array construction, develop the distributed algorithms for calculating the address of the target object and then implement these, possibly in assembly or some other lower level language, and 2) use minimal implementation dependent knowledge and the logical structure of records and arrays to utilize standard Ada mechanisms to perform the object transfers. We expect the former to lead to more compact (in terms of code size) solutions, but to require a more detailed knowledge of the internal workings of the underlying compilers, while the latter will require less knowledge of the internal mechanisms used by the compilers at the expense of a larger amount of code (automatically generated, however) in the agents. Since the latter is also more in keeping with the philosophy of using existing compilers where possible with minimal knowledge of their internals, and since developing this approach will aid in developing the algorithms for the first approach, we have followed this one first, and it is this one that will be described below. In subsequent work, we will explore the direct calculation of object addresses.

Access to remote objects is based upon the following things:

- An enumerated type, T_ENUM, whose values are the names of every type and field declared in the package for which an agent is being generated, and those in packages included via a **with**.
- An enumerated type, N_ENUM, whose values are the names of every data object declared in the package for which an agent is being generated, and those in packages included via a **with**.
- A collection of GETPUT procedures, one for each record or array type defined, whose functions are to either handle the request for an object reference if the request is for an object of the type the GETPUT handles, or to call another GETPUT if the object requested is, or is derived from, one of the fields of the type.
- A variant message structure containing appropriate fields indicating the type of data required, the fields within records to be used, and an actual data object of the type being referenced.

From the perspective of the local agent, a remote direct (not via access variables) data object access begins with the local agent main task receving a message from the postal system. One of the fields in this record contains a value of type N_ENUM that indicates the outermost name in the fully qualified name of the object being referenced. The local agent main task then performs a case statement on this value. There is thus a case for each object name. Each case calls a GETPUT procedure and passes it the message, the object named, and a count of the number of name components to the fully contatenated name sought (including array arguments).

If the object passed is a scalar object, the count will be zero and the request can be satisfied directly by the GETPUT procedure by simply copying a value between the appropriate field in the message record and the object passed to it. Another field in the message record contains the type of the object to be returned.

If the COUNT is not zero, then either an array element is being sought, or a fully concatenated name has not yet been fully expanded. In the former case, the indices for the array element (or slice) are contained in other fields of the message record and the GETPUT can select the appropriate element(s) of the array. These either directly satisfy the request or are used to recurse as described next.

If the GETPUT is handling a record type, there will be another field in the message record corresponding to this type of record which will contain a value of type T_ENUM (containing the field name to be selected). The GETPUT contains a case statement conditioned on this field indicator. There is thus a case corresponding to each field possible in the record. The action of each branch of the case is similar. Another GETPUT is called, passing to it the message record and object pointed to by a concatenation of the object name passed in and the corresponding field name.

Below is an abstraction of a typical GETPUT routine for a record type. The forms for other types are similar, but tend to be even a bit simpler.

```
procedure GETPUT(M: in out MESSAGE; OBJ: in out T; COUNT: NATURAL) is
begin
   COUNT := COUNT - 1;
   if COUNT = 0 then            -- the name is fully expanded
      if <a get request> then
         -- copy value from OBJ to appropriate field in message record;
      else
         -- copy value from appropriate field in message record to OBJ;
      end if;
      return;
   end if;
   case <field name from message record> is
      when F1 => GETPUT(M, OBJ.F1,COUNT);
                 ⋮
      when FN => GETPUT(M, OBJ.FN,COUNT);
   end case;
end;
```

Here T is a record type of an object being passed in, and F1..FN are the fields in the record type. If one of the fields, FI, say, were an access variable, that access variable would have been replaced by a record (as described in the pointer agent section above) and the action for the corresponding case would first check to see if the requested object were on the current site or elsewhere. If local, then the call to GETPUT would be made as shown above. If elsewhere, then an appropriate message would be propagated to the pointer agent on the indicated site.

Message Record Structure

The interprocessor message structure is key to the operation of the above object referencing scheme. For each source package, a different message record type is defined. These records consist of a fixed part and a variant part. There is one case of the variant part for each type of data object defined in the source package. In the case of a subprogram or task entry call, the variant part of the record contains fields for all of the arguments, and if applicable, a function result. The fixed part of the record contains field selectors which are used for accessing fields of records, as described above. A simple example of a message record type is given below. It should be self-explainatory from the previous discussion.

```
type MESS_T( DATA_TYPE: T_ENUM ) is
   record
      OBJ_ENUM : N_ENUM ;          -- indicates outermost object
      TYPE1_FIELD : T_ENUM;        -- record type TYPE1
      TYPE2_X1 : TYPE2_X1_T;       -- 2-dim array type TYPE2
      TYPE2_X2 : TYPE2_X2_T;
         :
      case DATA_TYPE is            -- reflects data to be exchanged
         when TYPE1_D =>
            TYPE1_VAL : TYPE1;
         when TYPE2_D =>
            TYPE2_VAL : TYPE2;
         when CALL1_D =>              -- function CALL1
            CALL1_ARG1 : FLOAT;
            CALL1_RESULT : FLOAT;
         when CALL2_D =>              -- subprogram CALL2
            CALL2_ARG1 : INTEGER;
            CALL2_ARG2 : INTEGER;
         when FLOAT_D =>
            FLOAT_VAL : FLOAT;
         when INTEGER_D =>
            INTEGER_VAL : INTEGER;
      end case;
   end record;
end;
```

Since the postal service deals with all types of messages, a global message record type is defined. The global message record also consists of a fixed part, and a variant part. The various cases of the variant part are, as one might guess, merely the different message records for each source package. The fixed part contains the destination package number, and the return address, which consists of the source site number, and a logical channel number.

Translation Procedure

The translations required for the methods outlined above involve numerous steps and are quite involved. In this section we describe briefly the procedures to be used and a utility that has been prepared to simplify use of the pre-translator.

The first step in the translation procedure is to insure that the program to be distributed is correct. This is accomplished by compiling it for a single system. The programmer must do this before invoking the pre-translator.

When a correct program is available, the translation and compilation procedure consists of the following steps: 1) determination of the order of pretranslation of source files, 2) pretranslation of source files, 3) pre-link operations, 4) determination of the order of compilation of original sources (including agents) for target sites, 5) compiling and linking of individual site programs. Two utilities have been written to facilitate some of these steps.

The pre-compilation utility (ADAUTIL) will translate the network of package dependencies implicit in a set of source files to a set of file dependencies in Unix "makefile" format. The list of relevant source files must be specified, and one or more targets (main programs) must be specified. Since the order of pretranslation is identical to the order of Ada compilation, ADAUTIL takes an option specifying whether a makefile to run the pretranslator, or a makefile to run the Ada compiler is desired.

The second utility, called MESSUTIL, performs step three above. The operations done during setp 3 are: 1) constructing the global message record from all relevant package message records, 2) constructing a package of package site constants, 3) constructing main procedures for each site, and 4) constructing a meta makefile capable of performing steps 4 and 5 above.

Two scripts were written to simplify the pretranslation process. One script performs steps 1 to 3 above, and the other invokes the meta makefile to perform steps 4 and 5. If any non-Ada object modules need to be linked into any site, the meta makefile may be edited in between the running of the two scripts.

4. DISCUSSION OF THE APPROACH

One of our principal concerns with the system developed is the run-time overhead associated with the mechanisms we used. We can model this performance in terms of the run-time overhead associated with various kinds of remote references. From the tests performed in [13] we know that task rendezvous times exceed procedure call times by one and a half to two orders of magnitude, and that task elaboration times are several times larger than rendezvous times. We can also reasonably expect the network communications times to be sizable. For example message end-to-end times for MAP are on the order of 100ms, more or less independent of message size [14], for the Intel hypercube, a few milliseconds, and for the NCUBE hypercube, several hundred microseconds to a millisecond, where the latter two depend somewhat upon message size, the variable component of message size being 1-10 microseconds/byte [15]. We thus neglect all local procedure and function call times, and model our overhead in terms of the number of messages and rendezvous required.

Thus, let t_m and t_r be the times to complete a message transfer and local rendezvous, respectively and let n_m^o and n_r^o be the number of messages and local rendezvous required for a remote operation of type o. Then, the time to complete a remote operation is

$$n_m^o \cdot T_m + n_r^o \cdot t_r$$

In these cases, we represent the overhead by the pair (n_m^o, n_r^o).

Whenever there are task elaborations involved, we represent the number by E. It is listed separately since it is generally not necessary to do the task elaboration with each access, but only when tasks or procedures are first elaborated. Nevertheless, even though many of these need be done only once immediately after system load, the number of tasks in the system could have an impact on the scheduling algorithms to be used and the efficiency of any runtime system, and the number E is thus important

The following sections present briefly the costs associated with each of the remote operations.

Data Objects — (2,4), E = 0

Access/Updates to data objects require two messages and four rendezvous. One message is to send the request and the second to receive an acknowledgement. The rendezvous are for the mail system. This presumes that the requested object is on the first remote site accessed. If there is a continuation to other sites through pointers, the above numbers must be mulitplied by the number of remote accesses required to satisfy the request.

Task Objects — (2, 6), E = # of entries

Task objects are accessed through entry calls. This requires two messages as for data objects and six rendezvous for synchronization (4 for the mail system and 2 for the handler).

The number of task elaborations that need to be done initially is equal to the number of entries to the task. Entry calls to task objects created from task types require no special handling by themselves. However, each task object created from a remote type requires two messages for creation and four rendezvous for synchronization. All further access are as in the case of task objects.

Procedures and Functions —(2, 6); E = 1

Since the local agent treats procedure and function calls in the same way as task entry calls, the analysis is analogous.

Pointers

There are two factors to consider here, the overhead when the object pointed to is remote, and the overhead when the object is local. Remember that all pointers are replaced with records having a site number and a pointer. This requires that all accesses via pointers begin with a check of whether or not the object is local or remote. If remote, the time of the check will be insignificant in comparison to the time required for the remote access and may be neglected. In this case the overhead depends upon the type of objected being referenced, and will follow the results obtained above.

However, if the access is local, the overhead is more significant. The exact amount of degradation will depend upon how an individual compiler implements pointer accesses and if then else constructs. In a simple test in which we wrote as efficient assembly language code as we could for local pointer accesses with and without the pointer record construct used here, the difference was a factor of four. In interpreting this, however, one must take into account the magnitude of time involved (only a few microseconds are the most) and the frequency of occurence. With these considerations taken into account, we do not feel that much overall time will be added to local accesses.

Summary Analysis Comments

To place the above analyses in perspective, one must compare typical times for message transfers and rendezvous. Some typical network times were mentioned above. Rendezvous times on the order of 500-600 microseconds have been reported for an 8 mHz IBM PC/AT, and on the order of 300-400 microseconds for Motorola 68000 processors. It is also the case that these times have been dropping significantly with each new release of Ada compilers intended for real-time applications, and are predicted by Ada vendors to become yet considerably smaller over the next year or two. Thus, except for the fastest networks, the message times will either be close to the rendezvous times or dominate them, and the approach taken will be primarily influenced by the network message times.

There is further issue that may be of concern, the number of tasks and GETPUT routines needed in the local agents. These have a linear dependence upon the number of entries (and subprograms) and types present in a remotely accessed package. While this may seem rather large, one is not likely to access a large number of things remotely, and those that are accessed remotely can be packaged separately from those that are not, thus keeping the number of extra tasks and routines to a minimum.

5. STATUS AND CONCLUSIONS

At the present time, the distributed translation system is operational for distributed packages with simple objects in their visible parts, i.e., no record or array definitions. Scalar data objects, subprograms and declared tasks may be directly referenced (no timed or contional calls). Tests have been successfully completed with up to three VAX processors cooperating on the execution of a single program. The implementation of the strategy described for referencing arrays and records (with fully concatenated names) is nearly complete, and expected to be in operation within a few weeks.

Nevertheless, there is still considerable work to be accomplished before the distribution of library packages and subprograms is complete. Although the strategy has been determined (see [16]), work has not yet been begun on handling timed/conditional task entry calls. Similarly, the dynamic creation of tasks is not complete. Two strategies will be implemented in this case. In the first, the created objects will be placed on the site elaborating the definition of the task type. In the second, the task object will be placed on the site creating the task through a declaration or **new** operator. The first is simpler to implement, but may make the task objects remote from the unit executing the code calling for their creation, while the second implementation is considerably more complex, and as noted in [9], may contain hidden remote object references. Finally, task termination must be properly handled.

More importantly, there are many issues of language definition that must be addressed. Our work has only addressed one point in the problem space to date, homogeneous, loosely coupled systems with static distribution. Additional representation mechanisms are needed to describe limitations dependent upon architectural considerations, to describe binding mechanisms, and to describe processor types (so that implicit data conversions can be accomplished). Moreover, it is probably necessary to require greater use of representational specifications on data objects to which remote access is allowed. Finally, there should be a more explicit definition of the allowed units of distribution.

References

[1] Hoare, C.A.R., "Communicating sequential processes", *Communications of the ACM*, vol. 21, no. 9, Aug. 1978.

[2] Strom, R.E. and Yemini, S., "NIL: an integrated language and system for distributed programming", *Sigplan '83 Symposium on Programming Language Issues in Software Systems*, vol. 18, no. 6, June, 1983, pp. 73-82.

[3] Liu, M.T. and Chung-Ming Li, "Communication distributed processes: a language concept for distributed programming in local area networks", *Local Networks for Computer Communications, IFIP Working Group 6.4, International Workshop on Local Networks*, Aug., 1980, pp. 375-406.

[4] Van DenBos, J. and Plasmeijer, R. and Stroet, J., "Process comm. based on input specifications", *ACM Trans. of Programming Languages & Systems*, vol. 3, pp. 224-250, July, 1981.

[5] Andrews, G.R., "Synchronizing resources", *ACM Trans. of Programming Languages & Systems*, vol. 3, no. 4, pp. 405-430, Oct., 1981.

[6] Mao, T.W. and Yeh, R.T., "Communication Port: A Language Concept for Current Programming", *IEEE Trans. Software Eng.*, vol. SE-6, no. 2, pp. 194-204, March, 1980.

[7] Holt, R.C., "A short introduction to concurrent euclid", *Sigplan Not.*, vol. 17, no. 5, pp. 60-79, May, 1982.

[8] Hansen, P.B., "Edison-A multiprocessor Language", *Software-Prac. and Exper.*, vol. 11, no. 4, pp. 325-361, April, 1981.

[9] Volz, R.A. and Mudge, T.N. and Buzzard, G.D. and Krishnan, P., "Translation and Execution of Distibuted Ada Programs: Is It Still Ada? ", *IEEE Transactions on Software*, Spring 1987.

[10] M. Tedd, S. Crespi-Reghizzi, and A. Natali, *Ada for multi-microprocessors*, Cambridge University Press, Cambridge, 1984.

[11] D. Cornhill, "Partitioning Ada programs for execution on distributed systems", *1984 Computer Data Engrg. Conf.*.

[12] Honeywell Systems Research Center, "The Ada Program Partitioning Language", the Distributed Ada Project, Sept., 1985.

[13] R.M. Clapp, L.J. Duchesneau, R.A. Volz, T.N. Mudge, and T. Schultze, "Toward real-time performance benchmarks for Ada," *Communications ACM*, vol. 29, no. 8, pp. 760-778, Aug. 1986.

[14] Volz, R.A. and Naylor, A.W., *Final Report of the NSF Workshop on Manufacturing Systems Integration*, held November 1985 in St. Clair, Michigan and organized by the Robotic Systems Division, Center for Research on Integrated Manufacturing, College of Engineering, The University of Michigan, Ann Arbor, MI. 48109, 1985

[15] Mudge, T. N., G. D. Buzzard, & T. S. Abdel-Rahman, "A high Performance Operating System for the NCUBE," *Proceedings of the 1986 Conference on Hypercube Mulitprocessors*, Knoxville, Tennessee, Oct. 1986.

[16] Volz, R. A. and T. N. Mudge, "Timing Issues in the Distributed Execution of Ada Programs," to appear in special issue on Parallel and Distributed Processing, *IEEE Transactions on Computers*, 1987.

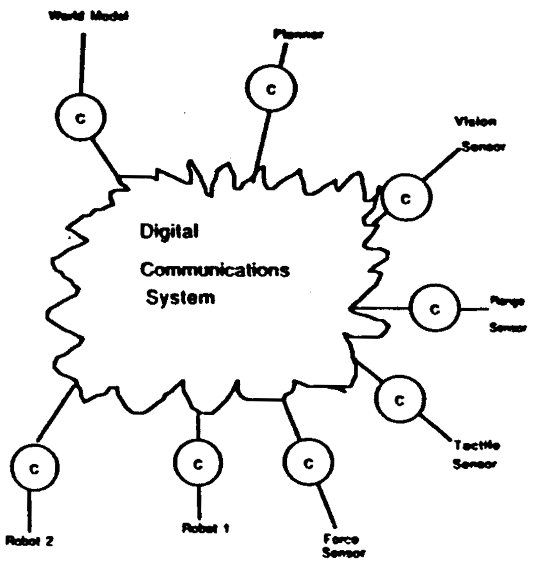


Figure 1: Loosely coupled system upon which we seek distributed program execution

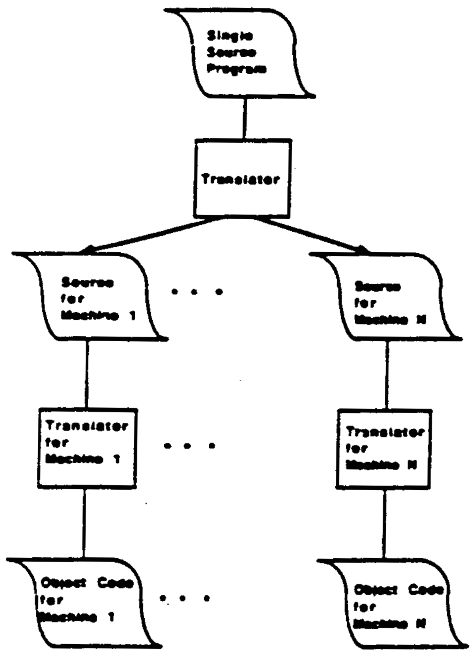


Figure 2: Overall operation of translation system

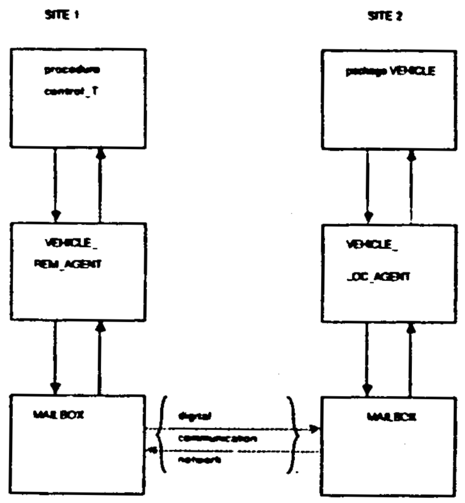


Figure 3: Structure of translated example program

S22-61
128
188104

Computational Structures for Robotic Computations

C.S.G. Lee and P.R. Chang
Purdue University
West Lafayette, IN 47907

P939 1052

Abstract

~~This paper addresses~~ The computational problem of inverse kinematics and inverse dynamics of robot manipulators by taking advantage of parallelism and pipelining architectures is discussed. For the computation of inverse kinematic position solution, a maximum pipelined CORDIC architecture has been designed based on a functional decomposition of the closed-form joint equations. For the inverse dynamics computation, an efficient p-fold parallel algorithm to overcome the recurrence problem of the Newton-Euler equations of motion to achieve the time lower bound of $O(\lceil \log_2 n \rceil)$ has also been developed.

1. Introduction

Robot manipulators are highly nonlinear systems, and their motion control is usually specified in terms of the path traveled by the manipulator hand in Cartesian coordinates. To perform a simple kinematic path control, the controller is required to compute accurately the joint angles of the manipulator along the desired Cartesian path at an adequate and acceptable rate. To perform a dynamic path-tracking control, one must repeatedly compute the required generalized forces, from an appropriate manipulator dynamics model, using the measured data of displacements and velocities of all the joints, and the accelerations computed from some justifiable formulae or approximations, to drive all the joint motors. In order to achieve fast convergence of the control algorithm, a sampling rate of no less than 60 *Hz* is preferable because the mechanical resonant frequency of most industrial manipulators is around 5–10 *Hz*. The above kinematic and dynamic path control reveals a basic characteristic and common problem in robotic manipulator control — intensive computations with a high level of data dependency. Despite their impressive speed, conventional general-purpose uniprocessor computers can not efficiently handle the kinematic and dynamic path control computations at the required computation rate because their architectures limit them to a mostly serial approach to computation, and therefore limit their usefulness for robotic computational problems. This paper addresses these intensive robotic computational problems by taking advantage of parallelism and pipelining architectures.

Considering that most industrial robots have simple geometry, the kinematic path control requires the computation of the solution of joint angles which can be obtained by various techniques. The inverse transform technique [1] yields a set of explicit, closed-form, non-iterative joint angle equations which involve multiplications, additions, square root, and transcendental function operations. Based on an actual implementation on a multiprocessor system† [2,3] having a circuit to synchronize the CPUs and software scheduling for computing the joint solution, the best reported computation time was 3.6 *ms* for a six-link manipulator versus 20 *ms* running on a uniprocessor system. If we use a CORDIC (COordinate Rotation DIgital Computer) architecture [4], the computation time reduces to 40 μs, a speed-up factor of 500††!

For the dynamic path-tracking control, there are a number of ways to compute the generalized forces/torques applied to the joint motors [5], among which the computation of joint torques from the Newton-Euler (NE) equations of motion is the most efficient and has been shown to possess the time lower bound of $O(n)$ running in uniprocessor computers [6,7], where n is the number of degrees of freedom (DOF) of the manipulator. Based on the study of Luh, Walker, and Paul [7], it requires $(150n - 48)$ multiplications and $(131n - 48)$ additions per trajectory set point for a manipulator with rotary joints. It is unlikely that further substantial improvements in computational efficiency can be achieved, since the recursive NE equations are efficiently computing the minimum information needed to compute the generalized forces/torques: angular velocity, liner and angular acceleration, and joint forces and torques. For a Stanford robot arm (a total of 308 multiplications and 254 additions is required to compute the joint torques [8]), this amounts to 25 *ms* processing time on a uniprocessor system and 5.69 *ms* running on an experimental multiprocessor system with 7 processors [9]. If we use the parallel algorithm with 6 processors as proposed in this paper, this reduces the computation from 852 multiplications and 738 additions running on a uniprocessor to 197 multiplications and 183 additions for a PUMA

This work was supported in part by the National Science Foundation Engineering Research Center Grant CDR8500022. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the author and do not necessarily reflect the views of the funding agency.

† The multiprocessor system consists of a MC6809 CPU and seven Z80 CPUs. Each Z80 is accompanied by two 9511 APUs, local memory, and I/O interfaces.

†† A speed-up factor is defined as the ratio of the computational time of a task running on a uniprocessor system to the computational time of the same task running on the proposed architecture (i.e., 20 $ms/40\mu s = 500$).

robot (due to different kinematic structure of PUMA and Stanford robots, direct comparison on processing time is invalid) [10].

This paper discusses the development of a maximum pipelined CORDIC architecture for the computation of inverse kinematic position solution to achieve the pipelined time of 40 μs and an efficient p-fold parallel algorithm to achieve the time lower bound of computing the joint torques. The CORDIC architecture was designed based on a functional decomposition of the closed-form joint equations. Delay buffers are necessary to balance the pipelined CORDIC architecture to achieve maximum pipelining. The buffer assignment problem is solved by the integer linear programming technique. The efficient p-fold parallel algorithm can be best described as consisting of p-parallel blocks with pipelined elements within each parallel block to achieve the time lower bound of $O(\lceil \log_2 n \rceil)$ of computing the joint torques based on the Newton-Euler equations of motion, where n is the number of degrees of freedom of the manipulator. The algorithm can be implemented with a group of microprocessors without complex intercommunication among processors and bussing of data. A modified inverse shuffle scheme is suggested for connecting the processors together with efficient intercommunications.

2. Inverse Kinematic Position Computation

The general kinematic problem of a 6-DOF robot arm concerns the problem of finding the generalized coordinates $\mathbf{q} = [q_1, q_2, \cdots, q_6]^T$, together with the vector of their generalized velocities and the vector of their generalized accelerations in the n-dimensional space such that the characteristics of the motion of the free end, the hand, coincide with the pre-specified Cartesian trajectory. This inverse problem has earned considerable attention because of its importance in relating the Cartesian trajectory of the hand to the corresponding joint-variable trajectory of the manipulator. This paper focuses only on the inverse kinematic position solution.

In solving the inverse kinematic position problem, we are always interested in obtaining a closed-form solution (i.e. an algebraic equation relating the given manipulator hand position and orientation to one of the unknown joint displacements), which yields all the possible solutions in a fixed computation time. Fortunately, most industrial robots have simple geometry and exhibit closed-form joint solution. Utilizing the inverse transform technique [1], the joint angle equations of a six-link manipulator with simple geometry reveal the computation of a large set of elementary operations: real number multiplications, additions, divisions, square roots, trigonometric functions and their inverse. However, these elementary operations, in general, cannot be efficiently computed in general-purpose uniprocessor computers. In order to obtain a fixed computation time for the joint angle solution, time-consuming transcendental functions (sine, cosine, and arc tangent) are implemented as table look-up at the expense of the solution accuracy. The CORDIC algorithms [11-14] are the natural candidates for efficiently computing these elementary operations. They represent an efficient way to compute a variety of functions related to coordinate transformations with iterative procedures involving only shift-and-add operations at each step. Thus, cordic processing elements are extremely simple and quite compact to realize [14] and the interconnection of CORDIC processors to exploit the great potential of pipelining and multiprocessing provides a cost-effective solution for computing the inverse kinematic position solution.

2.1. CORDIC Algorithms and Processors

In conventional uniprocessor computers, computation of elementary functions such as square roots, sine, cosine, hyperbolic sine and cosine and their inverse consumes a considerable amount of effort than multiplication operation. These elementary functions can be efficiently computed by the cordic algorithms which can be described by a single set of iterative equations parametrized by a quantity m (= −1,0,1) which determines the type of rotations. To establish connections between CORDIC and rotation-based algorithms, let the angle of rotation θ be decomposed into a sum of n sub-angles $\{d_i;\ i = 0, n-1\}$

$$\theta = \sum_{i=0}^{n-1} u_i d_i \tag{1}$$

where the sign u_i (± 1) is chosen based on the direction of rotation. Similarly, the plane rotation matrix $\mathbf{R}(\theta)$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{2.a}$$

or hyperbolic rotation matrix $\mathbf{R}(\theta)$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cosh\theta & \sinh\theta \\ -\sinh\theta & \cosh\theta \end{bmatrix} \tag{2.b}$$

can also be decomposed into a product of sub-angle rotation matrices

$$\mathbf{R}(\theta) = \prod_{i=0}^{n-1} \mathbf{R}(d_i) \tag{3}$$

Thus, a single rotation of θ angle can be replaced by n smaller rotations with d_i angle each. In the cordic algorithms, d_i is chosen such that

$$d_i = \begin{cases} \tan^{-1}(2^{-s(i)}) & , m = 1 \text{ (circular)} \\ 2^{-s(i)} & , m = 0 \text{ (linear)} \\ \tanh^{-1}(2^{-s(i)}) & , m = -1 \text{ (hyperbolic)} \end{cases} \tag{4}$$

where m determines the type of rotations and $\{s(i); i = 0, n-1\}$ is a non-decreasing integer sequence. Using d_i from (4), $\mathbf{R}(d_i)$ can be written as

$$\mathbf{R}(d_i) = p_i \begin{bmatrix} 1 & -mu_i 2^{-s(i)} \\ u_i 2^{-s(i)} & 1 \end{bmatrix} \tag{5}$$

where p_i is a scaling factor and equals to $(1 + m\, 2^{-2s(i)})^{-1/2}$. Let $\mathbf{R}^N(\theta)$ and $\mathbf{R}^N(d_i)$ be the normalised form of $R(\theta)$ and $\mathbf{R}(d_i)$, respectively, then from (3), we have

$$\mathbf{R}(\theta) = \prod_{i=0}^{n-1} p_i \prod_{i=0}^{n-1} \mathbf{R}^N(d_i) = k_m \prod_{i=0}^{n-1} \mathbf{R}^N(d_i) = k_m\, \mathbf{R}^N(\theta) \tag{6.a}$$

where

$$k_m = \prod_{i=0}^{n-1} p_i = \prod_{i=0}^{n-1} (1 + m\, 2^{-2s(i)})^{-1/2} \quad ; \quad \mathbf{R}^N(\theta) = \prod_{i=0}^{n-1} \begin{bmatrix} 1 & -mu_i 2^{-s(i)} \\ u_i\, 2^{-s(i)} & 1 \end{bmatrix} \tag{6.b}$$

Usually, k_m is a machine constant and $k_m \simeq 0.6072$ (for $m = 1$) or 1.00 (for $m = 0$) or 1.205 (for $m = -1$), when $n \geq 10$ [12, 15]. The normalised rotation matrix of (6.b) indicates that each small rotation can be realised with one simple shift-and-add operation. Hence, the computation of a trigonometric function can be accomplished with n shift-and-add operations, which is comparable to conventional multiplications. This makes a CORDIC ALU a very appealing alternative to the traditional ALU for implementing the elementary functions. In general, the normalised CORDIC algorithm can be written as follows:

```
FOR $i = 0, 1, \cdots, n-1$, DO
```

$$\begin{bmatrix} x_{i+1}^N \\ y_{i+1}^N \end{bmatrix} = \begin{bmatrix} 1 & -m\; u_i\; 2^{-s(i)} \\ u_i\; 2^{-s(i)} & 1 \end{bmatrix} \begin{bmatrix} x_i^N \\ y_i^N \end{bmatrix} \tag{7.a}$$

$$z_{i+1}^N = z_i^N + u_i\; d_i \tag{7.b}$$

where $x_0^N = x_0$, $y_0^N = y_0$, m determines the type of rotation, d_i is chosen as in (4), and the auxiliary variable z_i^N is introduced to accumulate the rotation after each iteration. And the corresponding "unnormalised" CORDIC algorithm is described as:

```
FOR $i = 0, 1, \cdots, n-1$, DO
```

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = p_i \begin{bmatrix} 1 & -m\; u_i\; 2^{-s(i)} \\ u_i\; 2^{-s(i)} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{8.a}$$

$$z_{i+1} = z_i + u_i d_i \tag{8.b}$$

where $x_0 = x_0$ and $y_0 = y_0$. It can be shown that z_i and z_i^N will accumulate the angle of the total rotation and have the same value after n iterations. However, the end results of (x_n, y_n) from the iterations of (8.a) and the end results of (x_n^N, y_n^N) from the iterations of (7.a) are related according to

$$x_n = k_m\; x_n^N \quad ; \quad y_n = k_m\; y_n^N \tag{9}$$

Consequently, one may evaluate x_n^N and y_n^N by using only the shift-and-add operations in (8.a), then realise x_n and y_n by other simple methods such as ROM look-up tables and regular combinatorial logic, etc. Fortunately, it is possible to find a simple way to normalise the scale factor k_m using the same shift-and-add hardware [14, 15]. The supplementary operations that are used to force the scale factor k_m to converge toward unity can be either performed after all the operations of (7.a) are terminated, that is,

$$x_{i+1}^c = (1 + \gamma_i 2^{-i})\, x_i^c \quad ; \quad y_{i+1}^c = (1 + \gamma_i\; 2^{-i}) y_i^c \tag{10}$$

where $x_0^c = x_n^N$, $y_0^c = y_n^N$, and $0 \leq i \leq n-1$, or interleaved with the operations of (7.a), that is,

$$x_i^c = (1 + \gamma_i\; 2^{-i})\, x_i^N \quad ; \quad y_i^c = (1 + \gamma_i\; 2^{-i})\, y_i^N \tag{11}$$

where $0 \leq i \leq n-1$. The parameter γ_i in (10) or (11) may be -1 or 0 or 1 depending on the value of i and the type of rotations (i.e. m) [14, 15].

Haviland et al. [14] realised the CORDIC algorithm on a CMOS chip and showed that the processing time of the CORDIC chip is 40 μs. They also suggested $n = 13$ as the minimum cycle time of a two-byte (24-bit) fixed point operation. However, in practice, they used $n = 24$. For a conventional CORDIC module, it reqpires 5 shift-and-add modules to compute one CORDIC iteration and one normalisation iteration in parallel (that is, 3 shift-and-add modules for (7.a) and (7.b), and 2 shift-and-add modules for (11)). The desired output can be obtained in 24 iterations ($n = 24$). Thus, 24 iterations of 5 shift-and-add modules computing in parallel will be enough to realise CORDIC algorithms. This indicates that the CORDIC processing time is no slower than the time for a serial multiplier computing two 24-bit operands.

Figure 1 summarises the elementary functions that can be obtained from the CORDIC processor when m is set to -1, 0, or 1. In this figure, a CORDIC processor is depicted as a box with three inputs x_0 , y_0 , z_0, which are the initial values of x_i , y_i, and z_i in (8), as well as three outputs that correspond to the final values of x_n , y_n , and z_n in (8). Thus, the outputs x_n , y_n , z_n are the desired elementary functions, when m is appropriately set to -1, 0, or 1. These CORDIC processors will be configured and connected, based on a functional decomposition of the joint angle equations, to arrive at an efficient architecture for the computation of inverse kinematic position solution.

2.2. CORDIC Architecture for Inverse Kinematic Position Computation

The design philosophy is to examine the inverse kinematic position solution for its computational flow and data dependencies in order to functionally decompose the computations into a set of CORDIC computational modules (CCMs) with an objective that each CCM will be realizable by a CORDIC processor.* This functional decomposition is not unique and can be best represented by a directed task graph with a finite set of nodes denoting CCMs and a corresponding finite set of communication edges denoting operands movements. An examination of the inverse kinematic position equations in Appendix A shows a limited amount of parallelism with a large amount of sequentialism in the flow of computations and data dependencies. This serial nature of the computational flow lends itself to a pipelined CORDIC processors implementation [16]. The decomposition of the inverse kinematic position equations is done by looking at the equations which can be computed as elementary functions by CORDIC processors as listed in Figure 1.

The task of computing the inverse kinematic position of a PUMA robot arm, based on the equations in Appendix A, can be decomposed into 25 subtasks in which each subtask corresponds to a CCM and can be realized by a CORDIC processor. For example, θ_1 can be computed by the following 3 subtasks:

Subtask 1: $x1_n = r = (p_x^2 + p_y^2)^{1/2}$

$$z1_n = \tan^{-1}(\frac{p_y}{p_x}) = -\tan^{-1}(\frac{-p_y}{p_x})$$

$$\text{CORDIC Processor:} \quad \text{CIRC2} = \begin{cases} x_0 = p_x \\ y_0 = -p_y \\ z_0 = 0 \end{cases}$$

Subtask 2: $x2_n = \sqrt{r^2 - d_3^2} = \sqrt{x1_n^2 - d_3^2}$

$$\text{CORDIC Processor:} \quad \text{HYPE2} = \begin{cases} x_0 = x1_n \\ y_0 = d_3 \\ z_0 = 0 \end{cases}$$

Subtask 3: $z3_n = \theta_1 = z1_n - \tan^{-1}(\frac{d_3}{x2_n})$

$$\text{CORDIC Processor:} \quad \text{CIRC2} = \begin{cases} x_0 = x2_n \\ y_0 = d_3 \\ z_0 = z1_n \end{cases}$$

The computational flow of these 25 tasks together with the input data can be represented by the directed acyclic data dependency graph (ADDG) with switching nodes and parallel edges as shown in Figure 2 and the details about the decomposition of the inverse kinematic position solution into CCMs can be found in [4]. In Figure 2, each computational node, indicated by a circle, represents a CORDIC computational module, and each switching node, indicated by a dot, performs no computations but just switches data to various CCMs. The operands or data move along the edges. A major bottleneck in achieving maximum throughput or maximum pipelining in Figure 2 is the different arrival time of the input data at the multi-input CCMs (e.g. nodes T18 and T22 in Figure 2). The computations of multi-input CCMs can not be initiated until all the input data have arrived. This different arrival time of input data lengthens the pipelined time. Thus, the ADDG is said to be *unbalanced* and fails to achieve maximum pipelining. Several techniques [17]-[19] have been suggested to remedy this data arrival problem by inserting appropriate number of buffers (or delays) in some of the paths from the input node s to the multi-input CCMs to "balance" the ADDG and achieve maximum pipelining. This buffer assignment problem for balancing the ADDG can be reduced to an integer linear optimization problem. Detailed formulation of the optimal buffer assignment problem as an integer linear optimization problem can be found in [4]. After solving the buffer assignment problem, realisation of the balanced ADDG results in a maximum pipelined CORDIC architecture. For a PUMA robot arm, the architecture consists of 25 CORDIC processors and 141 buffer stages with 4 tapped-delay-line-buffers [4]. The initial time delay of the pipeline is equal to 18 stage latency (or 720 μs), where the stage latency of a CORDIC processor is assumed to be 40 μs [14]. The pipelined time of the CORDIC architecture equals to one stage latency or 40 μs. The realisation of the maximum pipelined CORDIC architecture is shown in Figure 3.

3. Inverse Dynamics Computation

The general inverse dynamic problem for an n-link manipulator can be stated as follow: given the joint positions and velocities $\{q_j(t)\,,\,\dot{q}_j(t)\,\}_{j=1}^{n}$ which describe the state of the manipulator at time t, together with the joint accelerations $\{\ddot{q}_j(t)\}_{j=1}^{n}$ which are desired at the point, solve the dynamic equations of motion for the joint torques $\{\tau_j(t)\}_{j=1}^{n}$ as follows:

$$\tau(t) = f\,(q(t), \dot{q}(t), \ddot{q}(t)) \tag{12}$$

where $\tau(t) = [\tau_1, \tau_2, \ldots, \tau_n]^T$, $q(t) = [q_1, q_2, \ldots, q_n]^T$, $\dot{q}(t) = [\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_n]^T$, $\ddot{q}(t) = [\ddot{q}_1, \ddot{q}_2, \ldots, \ddot{q}_n]^T$, $f(\cdot)$ is an nx1 nonlinear vector function and superscript T denotes transpose operation on matrices and vectors.

At present, much attention has been focused on the computational issues of the inverse dynamics based on the Newton-Euler (NE) formulation, resulting in various multiprocessor-based control systems [8,21-24]. The recursive structure of the NE equations of motion is obviously well suited to standard single-instruction-stream and single-data-stream (SISD) computers. It is, however, not an efficient parallel processing for new single-instruction-stream and multiple-data-stream (SIMD) computers that are capable of performing many simultaneous operations. Our approach in designing efficient algorithms for computing the robot inverse dynamics is to look at the computational complexity of the problem first. In particular, we need to know what is the limitation of speeding up the computation of the inverse dynamics while running on p processors, where $1 \leq p \leq n$. That is, we would like to establish a time lower bound for the inverse dynamics computation problem so that several efficient computational schemes can be compared and contrasted. Then efficient algorithms achieving the time lower bound can be designed for the computation of the inverse dynamics. The following notations and lemma will be used to derive the time lower bound of the inverse dynamic problem.

Notations:

(1) *Linear arithmetic expression* is any well-formed string composed of four arithmetic operators $(+,-,\times,/)$ or, for convenience, two operators $+$(or $-$), $\times$ (or $/$), left and right parentheses, and atoms, which are constants or variables. We denote a linear arithmetic expression E of m distinct atoms by $E{<}m{>}$, e.g. $E{<}4{>} : a + b - c \;/\; d$.

(2) $T_p\,[f_1(\cdot)\,,\,f_2(\cdot)\,,\,\ldots\,,\,f_n(\cdot)]$ = Minimum computing time needed to evaluate $[\,f_1(\cdot)\,,\,f_2(\cdot)\,,\,\ldots\,,\,f_n(\cdot)\,]$ using p processors.

Lemma 1: The time lower bound of $T_p\,[E{<}m{>}]$ [25]. The shortest parallel time to evaluate a linear arithmetic expression $E{<}m{>}$ using p processors is bounded below by and equal to $O(\lceil m/p \rceil + \lceil \log_2 p \rceil)$, that is,

$$T_p\,[E{<}m{>}] \geq O(\lceil m/p \rceil + \lceil \log_2 p \rceil)$$

Theorem 1: The shortest parallel time to evaluate the joint torques $\{\tau_j(t)\}_{j=1}^{n}$ in equation (12) using p processors is bounded below by $O(k_1\,\lceil n/p \rceil + k_2\,\lceil \log_2 p \rceil)$, where k_1 and k_2 are specified constants, that is,

$$T_p\,[\tau_1, \tau_2, \ldots, \tau_n] \geq O(k_1\,\lceil n/p \rceil + k_2 \lceil \log_2 p \rceil) \tag{13}$$

The proof of Theorem 1 can be found in [10]. Two extreme cases follow from Theorem 1:

(a) If $p = 1$, then the shortest computing time $T_p\,[\tau_1, \tau_2, \ldots, \tau_n]$ is not lower than $O(n)$. Thus, the NE formulation is the most efficient algorithm of evaluating the inverse dynamics running in uniprocessor computers.

(b) If $p = n$, then the shortest parallel computing time $T_p\,[\tau_1, \tau_2, \ldots, \tau_n]$ is not lower than $O(\lceil \log_2 n \rceil)$.

Theorem 1 indicates that an efficient algorithm running on p processors may not achieve the same time order as $O(k_1 \lceil n/p \rceil + k_2 \lceil \log_2 p \rceil)$. However, if a parallel algorithm possesses the time lower bound, then it must be the most efficient algorithm of evaluating the inverse dynamics. Theorem 1 also indicates that, although NE formulation is very efficient for computing the inverse dynamics, a better solution is to find an efficient parallel algorithm, running on p processors, that possesses a time order of $O(k_1 \lceil n/p \rceil + k_2 \lceil \log_2 p \rceil)$. A parallel algorithm running on an SIMD machine and achieving the time lower bound is discussed next.

The recursive NE equations of motion are very efficient in evaluating the inverse dynamics whether they are formulated in the base coordinate frame [6] or the link coordinate frames [7]. The clear advantage of referencing both the dynamics and kinematics to the link coordinates is to obviate a great deal of coordinate transformations and to allow the link inertia tensor to be fixed in each link coordinate frame, which results in a much faster computation in a uniprocessor computer. However, the recursive structure of this formulation is in an inhomogeneous linear recursive form, e.g. $\omega_i = a_i \omega_{i-1} + b_i$, where $a_i = {}^{i}\mathrm{R}_{i-1}$ (a 3×3 rotation matrix) and $b_i = {}^{i}\mathrm{R}_{i-1}\, z_0 \dot{q}_i$ †, which requires more calculations and arrangements for parallel processing than the homogeneous linear recursive form. On the other hand, the NE formulation in the base coordinates can be re-arranged and transformed into a homogeneous linear recurrence form, e.g. $\omega_i = \omega_{i-1} + \dot{q}_i\, z_{i-1}$, which is more suitable for parallel processing on an SIMD computer, yielding a much shorter computing time.

Once the NE equations of motion are formulated in the base coordinates in a homogeneous linear recurrence form, then a parallel algorithm, called recursive doubling [16,17,26-27], can be utilised to compute the kinematics in the forward equations and the dynamics (or torques) in the backward equations [5]. The homogeneous linear recurrence problem of sise $(n+1)$ can be described as follows: given $x(0) = a(0) \neq$ identity and $a(i)$, $1 \leq i \leq n$, find $x(1)$, $x(2)$, ... ,$x(n)$ by an algorithm running on an SIMD computer of n processors, where

† a_i and b_i are used here as variables.

$$x(i) = x(i-1) * a(i) \tag{14}$$

and "$*$" denotes an associative operator, which can be a matrix product or addition, a vector dot product or addition, etc. If $a(0)$ is equal to an identity, then the linear recurrence of size $(n+1)$ can be reduced to the case of size n by taking a shift operation as follows: $a(0) \leftarrow a(1)$, $a(1) \leftarrow a(2)$, ..., $a(n-1) \leftarrow a(n)$, where "$\leftarrow$" denotes a replacing operation. The recursive doubling algorithm basically solves the homogeneous linear recurrence form in (14) by splitting the computations of a serial associative operation. For an arbitrary $x(n)$, the generalization of the idea allows $(n+1)/2$ parallel operations at the first splitting, $(n+1)/4$ at the second, ..., and $(n+1)/2^k$ at the k th until $\lceil \log_2(n+1) \rceil$ splittings, then $x(n)$ is computed with one final operation. Similarly, it can be shown that $x(i)$ can be computed in $\lceil \log_2(i+1) \rceil$ splittings, where $1 \leq i \leq n$. In other words, with the recursive doubling algorithm, $x(1)$, $x(2)$, ... , $x(n)$ can be computed concurrently no later than the time step $\lceil \log_2(n+1) \rceil$.

The procedure in computing the inverse dynamics from the NE equations of motion formulated in the base coordinates is to re-arrange the kinematic equations and the dynamic equations in a homogeneous linear recursive form, and the following input parameters relating the formulation must be given or evaluated in advance:

(a) The 3×3 rotation matrices ${}^{i-1}\mathbf{R}_i$, $i = 1,2,...,n$, which indicates the orientation of link i coordinates referenced to link $(i-1)$ coordinates, need to be evaluated in advance.

(b) ${}^i\mathbf{p}_i^*$ denotes the origin of link i coordinate frame from the origin of link $(i-1)$ coordinate frame expressed with respect to link i coordinates; ${}^i\mathbf{s}_i$ denotes the location of the center of mass of link i from the origin of link i coordinate frame expressed with respect to link i coordinates; and ${}^i\mathbf{J}_i$ denotes the inertia matrix of link i about its center of mass expressed with respect to link i coordinates, must be given in advance. Note that ${}^i\mathbf{p}_i^*$, ${}^i\mathbf{s}_i$, and ${}^i\mathbf{J}_i$ are constants when referred to their own link coordinates.

(c) λ_i is a joint indicator which specifies link i is rotational or translational as follow:

$$\lambda_i = \begin{cases} 0 \ , & \text{if link } i \text{ is rotational} \\ 1 \ , & \text{if link } i \text{ is translational} \end{cases}$$

(d) Let $\omega_0 = \dot{\omega}_0 = 0$, $\ddot{\mathbf{p}}_0 = [g_x, g_y, g_z]^T$ and $|\mathbf{g}| = 9.80621 m/s^2$. If external force $\mathbf{f}_e$ and external moment $\mathbf{n}_e$ are exerting on link n, then $\mathbf{f}_{n+1} = \mathbf{f}_e$, $\mathbf{n}_{n+1} = \mathbf{n}_e$; otherwise, $\mathbf{f}_{n+1} = \mathbf{n}_{n+1} = \mathbf{0}$.

The procedure of evaluating the NE equations of motion as a linear recurrence problem is then given below (note b_i are used here as variables):

STEP 1. Compute the rotation matrix ${}^0\mathbf{R}_i$ with respect to the base coordinates for $i = 1,...,n$

$${}^0\mathbf{R}_i = {}^0\mathbf{R}_{i-1} * {}^{i-1}\mathbf{R}_i \tag{15}$$

STEP 2. Compute $\mathbf{p}_i^*$, $\mathbf{z}_i$, and $\mathbf{s}_i$ for $i = 1,2,...,n$

$$\mathbf{z}_i = {}^0\mathbf{R}_i * \mathbf{z}_0 \ , \ \ \mathbf{z}_0 = [0,0,1]^T \ ; \ \ \mathbf{p}_i^* = {}^0\mathbf{R}_i \ {}^i\mathbf{p}_i^* \ ; \ \ \mathbf{s}_i = {}^0\mathbf{R}_i \ {}^i\mathbf{s}_i \tag{16}$$

The evaluation of $\mathbf{z}_i$ only involves taking the third column of ${}^0\mathbf{R}_i$.

STEP 3. Compute

$$b_i = \mathbf{z}_{i-1}\dot{q}_i \ (1 - \lambda_i) \tag{17}$$

and

$$\omega_i = \omega_{i-1} + b_i \tag{18}$$

STEP 4. Compute

$$b_i = (\mathbf{z}_{i-1} \ddot{q}_i + \omega_{i-1} \times \mathbf{z}_{i-1} \dot{q}_i)(1 - \lambda_i) \tag{19}$$

and

$$\dot{\omega}_i = \dot{\omega}_{i-1} + b_i \tag{20}$$

STEP 5. Compute

$$b_i = \dot{\omega}_i \times \mathbf{p}_i^* + \omega_i \times (\omega_i \times \mathbf{p}_i^*) + (\mathbf{z}_{i-1} \ddot{q}_i + 2 \ \omega_i \times (\mathbf{z}_{i-1} \dot{q}_i)) \ \lambda_i \tag{21}$$

and

$$\ddot{\mathbf{p}}_i = \ddot{\mathbf{p}}_{i-1} + b_i \tag{22}$$

STEP 6. Compute

$$\ddot{\mathbf{r}}_i = \dot{\omega}_i \times \mathbf{s}_i + \omega_i \times (\omega_i \times \mathbf{s}_i) + \ddot{\mathbf{p}}_i \tag{23}$$

STEP 7. Compute

$$\mathbf{F}_i = m_i \ddot{\mathbf{r}}_i \tag{24}$$

STEP 8. Compute

$$\mathbf{N}_i = \mathbf{J}_i \ \dot{\omega}_i + \omega_i \times (\mathbf{J}_i \ \omega_i) \tag{25}$$

For the sake of saving the calculations of evaluating $\mathbf{J}_i = {}^0\mathbf{R}_i\, {}^i\mathbf{J}_i\, {}^i\mathbf{R}_0$, which Luh et al. [7] showed that the computation was quite complicated, (25) is modified to

$$ {}^i\omega_i = {}^i\mathbf{R}_0\, \omega_i = ({}^0\mathbf{R}_i)^T \omega_i \quad ; \quad {}^i\dot{\omega}_i = {}^i\mathbf{R}_0\, \dot{\omega}_i = ({}^0\mathbf{R}_i)^T \dot{\omega}_i \tag{26}$$

$$ {}^i\mathbf{N}_i = {}^i\mathbf{J}_i\, {}^i\dot{\omega}_i + {}^i\omega_i \times ({}^i\mathbf{J}_i\, {}^i\omega_i) \quad ; \quad \mathbf{N}_i = {}^0\mathbf{R}_i\, {}^i\mathbf{N}_i \tag{27}$$

STEP 9. Compute .

$$ \mathbf{f}_i = \mathbf{f}_{i+1} + \mathbf{F}_i \tag{28}$$

STEP 10. Compute

$$ \mathbf{b}_i = \mathbf{N}_i + (\mathbf{p}_i^* + \mathbf{s}_i) \times \mathbf{F}_i + \mathbf{p}_i^* \times \mathbf{f}_{i+1} \tag{29}$$

and

$$ \mathbf{n}_i = \mathbf{n}_{i+1} + \mathbf{b}_i \tag{30}$$

STEP 11. Compute

$$ \tau_i = \begin{cases} (\mathbf{n}_i)^T \mathbf{z}_{i-1} \ , & \text{if } \lambda_i = 0 \\ (\mathbf{f}_i)^T \mathbf{z}_{i-1} \ , & \text{if } \lambda_i = 1 \end{cases} \tag{31}$$

Previously undefined terms, expressed in the base coordinates, are given as follows: m_i is the mass of link i, ω_i is the angular velocity of link i, $\dot{\omega}_i$ is the angular acceleration of link i, $\ddot{\mathbf{p}}_i$ is the linear acceleration of link i, $\ddot{\bar{\mathbf{r}}}_i$ is the linear acceleration of the center of mass of link i, $\mathbf{F}_i$ is the total force exerted on link i at the center of mass, $\mathbf{N}_i$ is the total moment exerted on link i at the center of mass, $\mathbf{f}_i$ is the force exerted on link i by link $i-1$, $\mathbf{n}_i$ is the moment exerted on link i by link $i-1$, τ_i is the torque exerted by the actuator at joint i if rotational, force if translational, q_i is the joint variable of joint i (θ_i if rotational and d_i if translational).

Equation (15) shows that the evaluation of ${}^0\mathbf{R}_i$ is a simple recursive matrix product form. Equations (18), (20), (22), (28), and (30) only involve simple recursive vector addition form. The other equations in the NE equations can be computed parallelly. Thus, the evaluation of the total computational complexity of the parallel algorithm for a PUMA robot arm can be derived as follows:

(a) The parallel evaluation of (15) using recursive doubling indicates $(27\lceil\log_2 n\rceil - 19)$ scalar multiplications and $(18\lceil\log_2 n\rceil - 14)$ scalar additions.

(b) Equations (18), (20), (22), (28), and (30) all have the same recursive vector addition form, the total parallel evaluation of these equations requires $(6\lceil\log_2 n\rceil + 9\lceil\log_2(n+1)\rceil)$ scalar additions.

(c) The parallel evaluation of the other equations in the NE formulation, e.g. $\mathbf{p}_i^* = {}^0\mathbf{R}_i\, {}^i\mathbf{p}_i^*$, $\mathbf{s}_i = {}^0\mathbf{R}_i\, {}^i\mathbf{s}_i$, $\mathbf{F}_i$, $\mathbf{N}_i$, τ_i, and all the $\mathbf{b}_i$ of (17), (19), (21), and (29) can be calculated by simple parallel computations, yielding a constant computation of 135 scalar multiplications and 98 scalar additions.

Combining the results of (a), (b), and (c), the total computational complexity of the parallel algorithm applied to a PUMA robot arm is $(27\lceil\log_2 n\rceil + 116)$ scalar multiplications and $(24\lceil\log_2 n\rceil + 9\lceil\log_2(n+1)\rceil + 84)$ scalar additions. Note that it is of time order $O(\lceil\log_2 n\rceil)$ because we are using $p = n$ processors. If further reduction on the coefficients of $(\lceil\log_2 n\rceil)$ is desirable, this can be accomplished by using matrix multiplier chips. This would reduce the coefficients 27 and 18 in evaluating (15) as discussed in (a). If $n = 6$, then the complexity of the parallel NE algorithm is 197 multiplies (mults) and 183 additions (adds) as compared with the complexity of the NE algorithm running on a uniprocessor [7]: 852 mults and 738 adds. Moreover, even if n becomes large, say $n = 12$ (for redundant robots), then the number of multiplications and additions increases only by 27 and 33, respectively. Thus, we have shown that considerable savings in computation time can be achieved from embedding the inverse dynamic computation in a parallel algorithm, which has a time complexity of logarithmic in the number of joints, $O(\lceil\log_2 n\rceil)$.

3.1. An Efficient Parallel Algorithm With p-Fold Parallelism

Last section showed that the bottleneck of parallel computation of the inverse dynamics depends on solving the homogeneous linear recurrence of the N-E formulation. If the restriction that one microprocessor "handles" one joint is relaxed, it is desirable to obtain an efficient parallel algorithm which can greatly improve the evaluation of the linear recurrence using p processors. A parallel algorithm of evaluating the inverse dynamics with a restricted number of p processors has been developed to achieve the time lower bound of $O(k_1\lceil n/p\rceil + k_2\lceil\log_2 p\rceil)$. The proposed p-fold parallel algorithm can be best described as consisting of p-parallel blocks with pipelined elements within each parallel block. The results from the computations in the p blocks form a new homogeneous linear recurrence of size p, which again can be computed using the recursive doubling algorithm. The parallel algorithm with p-fold parallelism (PFP) is summarized and presented as follows:

Algorithm PFP (p-fold Parallelism). This algorithm divides the computations into p-parallel blocks of computations. The jth processor computes the elements in the jth block serially. The results from the the p-parallel blocks form a new homogeneous linear recurrence of size p, which can be computed by the recursive doubling algorithm.

P1. [Initialization.] Given $a(0) \neq$ identity, let $s = \lceil (n+1)/p \rceil$, $m = ps$, and set

$$M(i) = \begin{cases} a(i-1) & , 1 \leq i \leq n+1 \\ 0 & , n+1 \leq i \leq m \end{cases} \tag{32}$$

where s indicates the block size (number of elements in a block).

P2. [Divide into p blocks.] Divide $M(i)$, $1 \leq i \leq m$, into p blocks as follows:
jth block $= \{M((j-1)s+1), M((j-1)s+2), \ldots, M(js)\}$ and let $N(1,j) = M((j-1)s+1)$, $1 \leq j \leq p$
where $N(i,j)$ indicates the ith element in the jth block.

P3. [Compute $N(i,j)$ in a DO loop.] The jth processor serially computes $N(i,j)$ in the jth block as:

```
For $i = 2$ step 1 until $s$ Do
    $N(i,j) = N(i-1,j) * M((j-1)s+i)$ , $1 \leq j \leq p$        (33)
END
```

It is seen that $x(1), x(2), \ldots, x(s-1)$ has been evaluated in the DO loop as well as $N(i,1)$, $2 \leq i \leq s$. That is, $x(1) = N(2,1)$, $x(2) = N(3,1)$, ..., $x(s-1) = N(s,1)$.

P4. [Form a new homogeneous linear recurrence of size p.] Let $\gamma(j) = N(s,j)$ and $y(j) = x(js-1)$, for $1 \leq j \leq p$ and referring to (14), (32), and (33), we have

$$\begin{aligned} y(j) = x(js-1) &= x((j-1)s-1) * a((j-1)s) * a((j-1)s+1) * \ldots * a(js-1) \\ &= y(j-1) * M((j-1)s+1) * M((j-1)s+2) * \ldots * M(js) = y(j-1) * N(s,j) \\ &= y(j-1) * \gamma(j) \end{aligned} \tag{34}$$

Equation (34) is a new homogeneous linear recurrence of size p which can be parallelly evaluated by the algorithm FOHRA, running in time proportional to $O(\lceil \log_2 p \rceil)$ and yielding the results $x(s-1) = y(1)$, $x(2s-1) = y(2)$, ..., $x(ps-1) = y(p)$. (Note that if $(n+1)$ is divisible by p, then $x(n) = y(p)$; otherwise, $x(ps-1) = y(p) = 0$).

P5. [Compute intermediate $x(i)$ in equation (35).] Without loss of generality, assuming that $(n+1)$ is not divisible by p, then there are $n-p-s+3$ intermediate terms of $x(i)$ that need to be determined. They are:

$$x(js+i) \quad , 0 \leq i \leq s-2, 1 \leq j \leq p-2 \quad \text{and} \quad x((p-1)s+i) \quad , 0 \leq i \leq n-(p-1)s \tag{35}$$

Referring to (7), (33), (34), and (35), thereby giving

$$\begin{aligned} x(js+i) = x(js-1) * a(js) * a(js+1) * \ldots * a(js+i) &= y(j) * M(js+1) * M(js+2) * \ldots * M(js+i+1) \\ &= y(j) * N(i+1,j) \quad , 0 \leq i \leq s-2, i \leq j \leq p-2 \end{aligned} \tag{36}$$

and $x((p-1)s+i) = y(p-1) * N(i+1,p-1)$, $0 \leq i \leq n-(p-1)s$

where $N(i+1,j)$, $y(j)$ of (36) have been evaluated in steps P3 and P4, respectively. Equation (36) shows that if $(n-p-s+3)$ tasks are of the same evaluation, then the calculations of these equal tasks are suited to an SIMD computer of p processors. It is shown that parallel evaluation of (36) requires $\lceil (n-p-s+3)/p \rceil$ time steps (note that if $(n+1)$ is divisible by p, then $x(n)$ can be evaluated in step P4, yielding $(n-p-s+2)$ equal tasks in (36), thereby requiring $\lceil (n-p-s+2)/p \rceil$ time steps).

END PFP

It is seen that the total parallel computing time T_p of the homogeneous linear recurrence of size $(n+1)$ using p processors is:

$$T_p = \begin{cases} \lceil (n+1)/p \rceil + \lceil (n-p-s+3)/p \rceil + \lceil \log_2 p \rceil - 1, & \text{if } (n+1) \text{ is not divisible by } p. \\ \lceil (n+1)/p \rceil + \lceil (n-p-s+2)/p \rceil + \lceil \log_2 p \rceil - 1, & \text{if } (n+1) \text{ is divisible by } p. \end{cases}$$

Applying the above p-fold parallel algorithm to the N-E formulation for an n-link rotary manipulator, it is able to achieve the time lower bound of $O(k_1 \lceil n/p \rceil + k_2 \lceil \log_2 p \rceil)$.

The above n-fold parallel algorithm is suited to be run on an SIMD computer. A cascade structure can be used for connecting the PEs. An alternate structure is to position a network between the processors and memories. The interconnection pattern, called the "perfect shuffle [26, 27]," has the number of links between processors proportional to n. An attractive interconnection pattern, called "inverse perfect shuffle [26,27]," is suitable for the implementation of solving the homogeneous linear recurrence and can be obtained by reversing the arrows of the perfect shuffle. Details about this network connection for solving the homogeneous linear recurrence for computing the joint torques can be found in [10].

4. Conclusion

A maximum pipelined CORDIC architecture for computing the inverse kinematic position solution and an efficient parallel algorithm for computing the joint torques have been discussed. To achieve maximum throughput, delay buffers are required to balance the pipeline. For a PUMA robot arm, the CORDIC architecture consists of 25 CORDIC processors and 141 buffer stages with 4 tapped-delay-line buffers. The initial time delay of the pipeline is equal to 720 μs and the pipelined time of the CORDIC architecture equals to one stage latency or 40 μs. The parallel algorithm for computing the joint torques has a time complexity of logarithmic in the number of joints. The interconnection networks for the processors have also been investigated to improve the utilization of communication and internal buffering between processors in an SIMD computer. Using the concepts of the proposed parallel algorithm, it would be possible to devise a VLSI chip capable of implementing the inverse dynamics computation at speed primarily bounded by the proposed parallel algorithm.

5. Appendix A: Inverse Kinematic Position Solution

The inverse kinematic position problem can be stated as: Given the position/orientation of the manipulator hand and the link/joint parameters, determine the joint angles so that the manipulator can be positioned as desired. That is, given

$$\mathbf{T} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the joint angle equations are:

$$r = (p_x^2 + p_y^2)^{1/2} \tag{A-1}$$

$$\theta_1 = \tan^{-1}\left[\frac{p_y}{p_x}\right] - \tan^{-1}\left[\frac{d_3}{\pm\sqrt{r^2 - d_3^2}}\right] \tag{A-2}$$

$$f_{11p} = p_x C_1 + p_y S_1 \;\; ; \;\; f_{11o} = o_x C_1 + o_y S_1 \;\; ; \;\; f_{13p} = -p_x S_1 + p_y C_1 \;\; ; \;\; f_{13o} = -o_x S_1 + o_y C_1 \tag{A-3}$$

$$f_{11a} = a_x C_1 + a_y S_1 \;\; ; \;\; f_{13a} = -a_x S_1 + a_y C_1 \tag{A-4}$$

$$d = f_{11p}^2 + f_{12p}^2 - d_4^2 - a_3^2 - a_2^2 \;\; ; \;\; e = 4a_2^2 a_3^2 + 4a_2^2 a_4^2 \tag{A-5}$$

$$\theta_3 = \tan^{-1}\left[\frac{a_3}{-d_4}\right] - \tan^{-1}\left[\frac{d}{\pm\sqrt{e - d^2}}\right] \tag{A-6}$$

$$\theta_{23} = \tan^{-1}\left[\frac{a_2(f_{11p}S_3 - p_z C_3) + d_4 f_{11p} - a_3 p_z}{a_2(f_{11p}C_3 + p_z S_3) + a_3 f_{11p} + d_4 p_z}\right] = -\tan^{-1}\left[\frac{(p_z C_3 - f_{11p}S_3) + (-\frac{d_4}{a_2})f_{11p} + (\frac{a_3}{a_2})p_z}{(p_z C_3 + f_{11p}S_3) + (\frac{a_3}{a_2})f_{11p} + (\frac{d_4}{a_2})p_z}\right] \tag{A-7}$$

$$\theta_2 = \theta_{23} - \theta_3 \tag{A-8}$$

$$\theta_4 = \tan^{-1}\left[\frac{-S_1 a_x + C_1 a_y}{C_{23}(C_1 a_x + S_1 a_y) - S_{23} a_z}\right] = \tan^{-1}\left[\frac{f_{13a}}{C_{23} f_{11a} - S_{23} a_z}\right] \tag{A-9}$$

$$\theta_5 = \tan^{-1}\left[\frac{C_4[C_{23}(C_1 a_x + S_1 a_y) - S_{23} a_z] + S_4[-S_1 a_x + C_1 a_y]}{S_{23}(C_1 a_x + S_1 a_y) + C_{23} a_z}\right] = \tan^{-1}\left[\frac{C_4(C_{23} f_{11a} - S_{23} a_z) + S_4 f_{13a}}{S_{23} f_{11a} + C_{23} a_z}\right] \tag{A-10}$$

$$\theta_6 = \tan^{-1}\left[\frac{-C_5[C_4(C_{23} f_{11o} - S_{23} o_z) + S_4 f_{13o}] + S_5(S_{23} f_{11o} + C_{23} o_z)}{-S_4(C_{23} f_{11o} - S_{23} o_z) + C_4 f_{13o}}\right] \tag{A-11}$$

where $(-\frac{d_4}{a_2})$, $(\frac{d_4}{a_2})$, $(\frac{a_3}{a_2})$ are constants, $C_i \equiv \cos\theta_i$, $S_i \equiv \sin\theta_i$, $C_{ij} \equiv \cos(\theta_i + \theta_j)$, and $S_{ij} \equiv \sin(\theta_i + \theta_j)$.

6. References

1. Paul, R.P., Shimano, B.E., and Mayer, G., "Kinematic Control Equations for Simple Manipulators," *IEEE Trans. Syst. Man, Cybern.*, Vol. SMC-11, No. 6, pp. 456-460, 1981.
2. Kametani, M. and Watanabe, T., "Hardware and Software of a Multiprocessor System Applied for Robot Control," *1984 Proc. of Industrial Electronic Conf.*, pp. 749-758.

3. Watanabe, T. et al., "Improvement of the Computing Time of Robot Manipulators Using a Multiprocessor," *Proc. of ASME Winter Annual Meeting*, Miami, Florida, 1985, pp. 13-22.

4. Lee, C.S.G. and Chang, P.R., "A Maximum Pipelined CORDIC Architecture for Inverse Kinematics Computation," Technical Report TR-EE-86-5, School of Electrical Engineering, Purdue University, January 1986.

5. Fu, K. S., Gonzales, R. C., and Lee, C.S.G., *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, September, 1986.

6. Orin, D.E., R.B. McGhee, M. Vukobratovic, and G. Hartoch, "Kinematic and Kinetic Analysis of Open-chain Linkages utilizing Newton-Euler Methods," *Math. Biosc.*, Vol. 43, 1979, pp. 107-130.

7. Luh, J.Y.S., M.W. Walker, and R.P.C. Paul, "On-line Computational Scheme for Mechanical Manipulator," *Trans. of ASME, J. of Dynam. Syst., Meas. and Contrl.*, Vol. 102, pp. 69-76, June 1980.

8. Luh, J.Y.S. and C.S. Lin, "Scheduling of Parallel Computation for a Computer-controlled Mechanical Manipulator," *IEEE Trans. Syst. Man, and Cyber.*, Vol. SMC-12, No. 2, pp. 214-234, March/April 1982.

9. Kasahara, H. and Narita, S., "Parallel Processing of Robot Arm Control Computation on a Multiprocessor System," *IEEE J. of Robotics and Automation*, Vol. RA-1, No. 2, June 1985, pp 104-113.

10. Lee, C.S.G. and Chang, P.R., "Efficient Parallel Algorithm for Inverse Dynamics Computation," *EEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-16, no. 4, July, 1986, pp. 532-542.*

11. Volder, J.E., "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electronic Computers*, Vol. EC-8, No. 3, Sept. 1959, pp. 330-334.

12. Walther, J.S., "A Unified Algorithm for Elementary Functions," *AFIPS Conf. Proc.*, Vol. 38, 1971, pp. 379-385.

13. Ahmed, H. M., J. M. Delosme and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *IEEE Computer*, Vol. 15, No. 1, pp. 65-82, Jan. 1982.

14. Haviland, G. L. and A. A. Tussynski, "A CORDIC Arithmetic Processor Chip," *IEEE Trans. Comput.*, Vol. C-29, No. 2, pp. 68-78, Feb. 1980.

15. Dewide, P. et al., "Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms," in *VLSI and Modern Signal Processing*, S. Y. Kung, H. J. Whitehouse, T. Kailath, (eds.), Prentice-Hall, Inc., Englewood Cliffs, NJ, pp. 257-276.

16. Kogge, P.M., "Parallel Solution of Recurrence Problems," *IBM J. Res. Develop.*, Vol. 18, pp. 138-148, Mar. 1974.

17. Kogge, P.M. and Stone, H.S., "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. on Comput.*, Vol. C-22, pp. 789-793, Aug. 1973.

18. Kung, H.T. and Lam, M., "Wafer-Scale Integration and Two-level Pipelined Implementation of Systolic Arrays," *J. of Parallel and Distributed Computing*, Vol. 1, No. 1, Sept. 1984, pp. 32-63.

19. Dennis, J. B. and R. G. Gao, "Maximum Pipelining of Array Operations on Static Data Flow Machine," *Proc. of 1983 Int'l. Conf. on Parallel Processing*, pp. 331-334, Aug. 1983.

20. Leiserson, C.E. and Saxe, J. B., "Optimizing Synchronous Systems," *J. VLSI and Computer Systems*, Vol. 1, 1983, pp. 41-68.

21. Lee, C.S.G., Mudge, T.N., and Turney, J.L., "Hierarchical Control Structure using Special Purpose Processors for the Control of Robot Arms," *Proc. 1982 Pattern Recognition and Image Processing Conf.*, Las Vegas, Nevada, June 14-17, 1982, pp 634-640.

22. Lathrop, L.H., "Parallelism in Manipulator Dynamics," M.I.T. Artificial Intelligence Tech. Rep. No. 754, Dec. 1983.

23. Nigam, R. and C. S. G. Lee, "A Multiprocessor-Based Controller for the Control of Mechanical Manipulators," *IEEE J. of Robotics and Automation*, Vol. 1, No. 4, Dec. 1985, pp. 173-182.

24. Orin, D.E., "Pipelined Approach to Inverse Plant Plus Jacobian Control of Robot manipulators," *Proc. 1984 IEEE Int'l Conf. on Robotics and Automation*, Atlanta, GA, pp. 169-175, March 1984.

25. Horowitz, E. and Sahni, S. *Fundamentals of Computer Algorithms*, Computer Science Press Inc., 1978, pp. 488-494.

26. Stone, H.S., *Introduction to Computer Architecture*, Science Research Associate Inc., 1975, pp. 319-373.

27. Stone, H.S., "Parallel Processing with Perfect Shuffle," *IEEE Trans. on Comput.*, Vol. C-20, pp. 153-161, Feb. 1971.

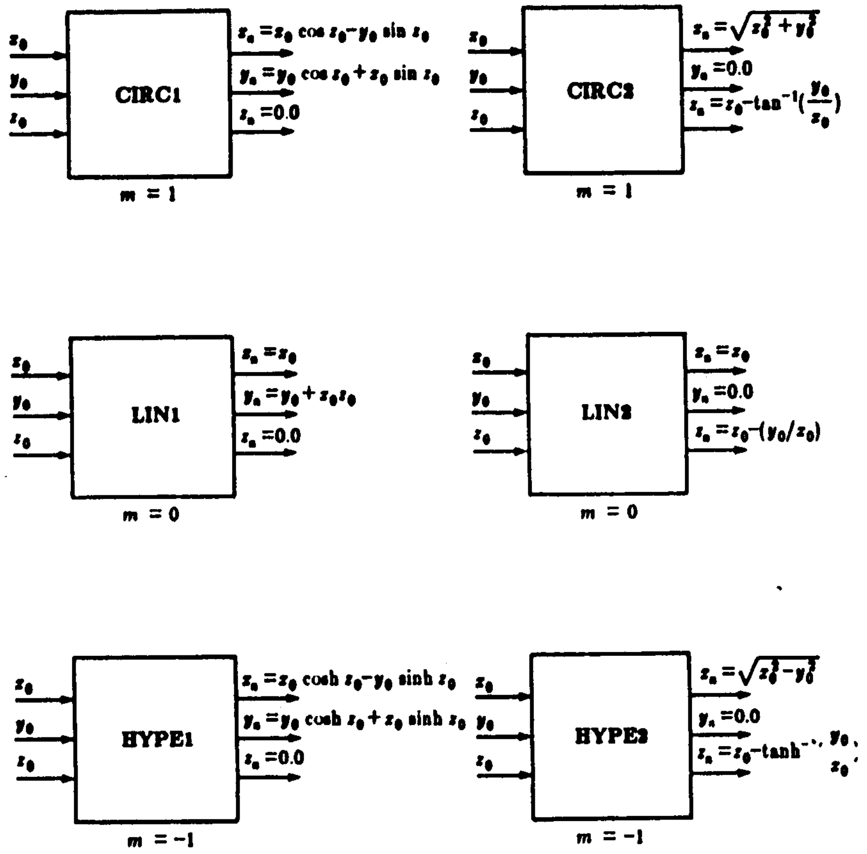

CIRC1
x_0
y_0
z_0
$x_n = x_0 \cos z_0 - y_0 \sin z_0$
$y_n = y_0 \cos z_0 + x_0 \sin z_0$
$z_n = 0.0$
$m = 1$
CIRC2
$x_n = \sqrt{x_0^2 + y_0^2}$
$y_n = 0.0$
$z_n = z_0 - \tan^{-1}(\frac{y_0}{x_0})$
$m = 1$
LIN1
$x_n = x_0$
$y_n = y_0 + x_0 z_0$
$z_n = 0.0$
$m = 0$
LIN2
$x_n = x_0$
$y_n = 0.0$
$z_n = z_0 - (y_0/x_0)$
$m = 0$
HYPE1
$x_n = x_0 \cosh z_0 - y_0 \sinh z_0$
$y_n = y_0 \cosh z_0 + x_0 \sinh z_0$
$z_n = 0.0$
$m = -1$
HYPE2
$x_n = \sqrt{x_0^2 - y_0^2}$
$y_n = 0.0$
$z_n = z_0 - \tanh^{-1} \frac{y_0}{x_0}$
$m = -1$

Figure 1 Elementary Functions Computed by CORDIC Processors

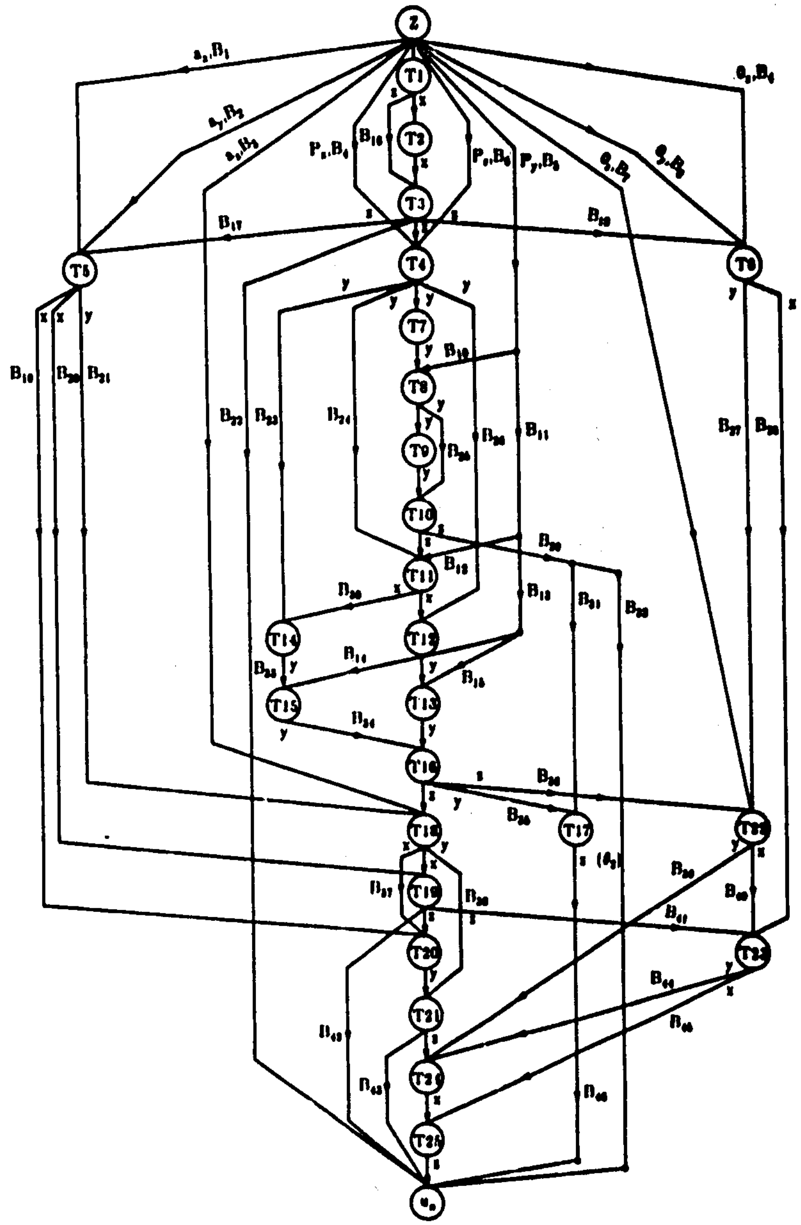
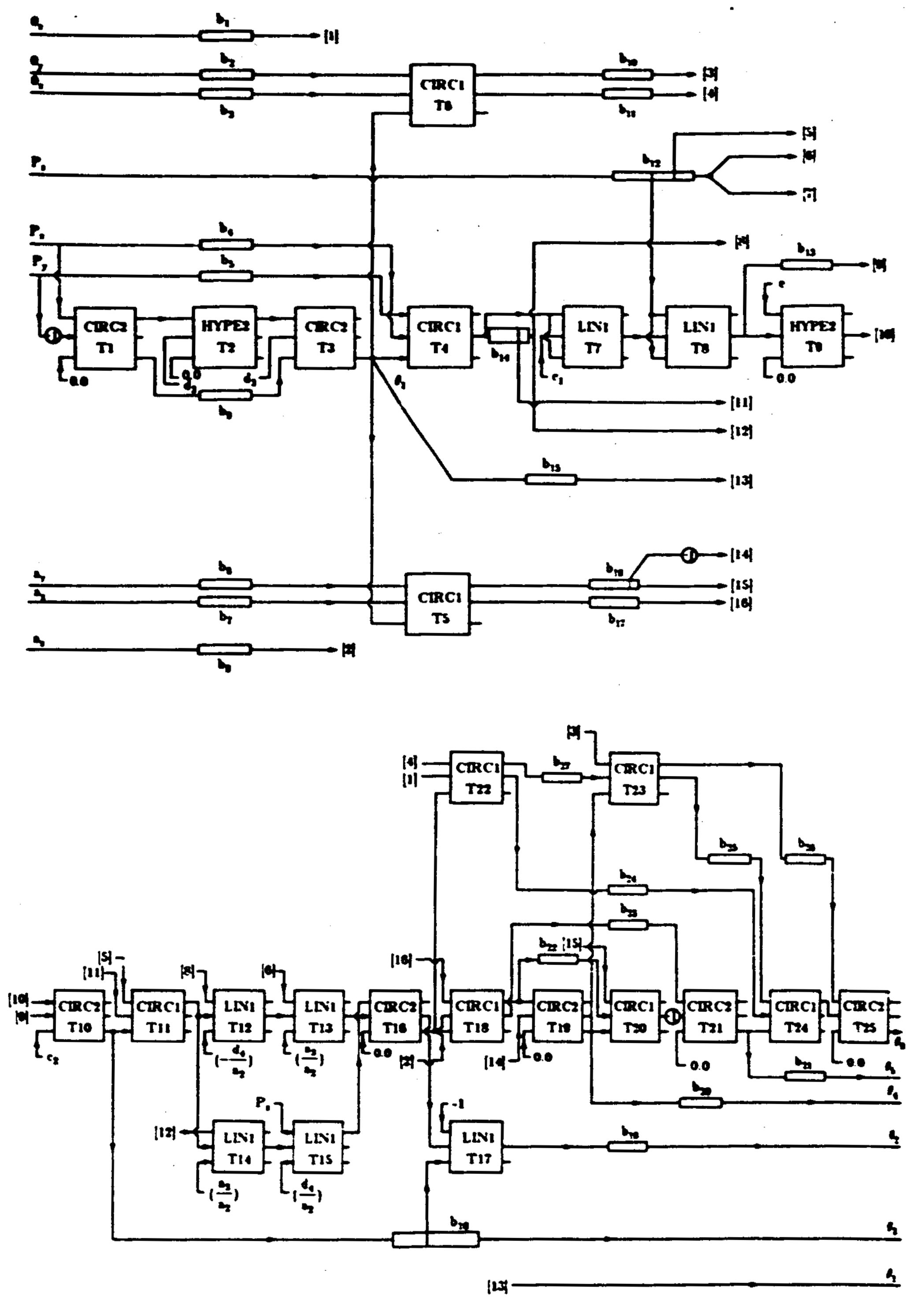


Figure 2 Task Graph for A PUMA Inverse Kinematic Position Solution



Buffers:

$|b_1| = 12$, $|b_2| = |b_3| = |b_4| = |b_5| = |b_6| = |b_7| = 3$, $|b_8| = 12$, $|b_9| = 12$, $|b_{10}| = 10$, $|b_{11}| = 8$, $|b_{12}|$: $TDLB-(10,5,3,2,2)$, $|b_{13}| = 1$, $|b_{14}|$: $TDLB-(5,4,1,1)$, $|b_{15}| = 15$, $|b_{16}|$: $TDLB-(10,9,1)$, $|b_{17}| = 8$, $|b_{18}|$: $TDLB-(10,4,6)$, $|b_{19}| = 5$, $|b_{20}| = 4$, $|b_{21}| = 2$, $|b_{22}| = 1$, $|b_{23}| = 2$, $|b_{24}| = 3$, $|b_{25}| = 1$, $|b_{26}| = 2$, $|b_{27}| = 1$

Constants:

$c_1 = -(d_4^2 + a_3^2 + a_2^2)$; $c_2 = \tan^{-1}(\frac{a_3}{-d_4})$

Figure 3. Realisation of Figure 2 with CORDIC Processors

S23-60
188105
11P.

A Run-Time Control Architecture for the JPL Telerobot

J. Balaram, A. Lokshin, K. Kreutz, and J. Beahan
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

JJ 574458

1. Abstract

~~This paper describes~~ an architecture for implementing the process-level decision making for a hierarchically structured telerobot currently being implemented at the Jet Propulsion Laboratory (JPL). Constraints on the architecture design, architecture partitioning concepts, and a detailed description of the existing and proposed implementations are provided.

[illegible]

2. Introduction

The architecture of a telerobot is required to support autonomous and teleoperated activities in a manner designed to obtain the maximum synergy of function between robotics and teleoperation. The JPL telerobot implements such a system for applications involving various servicing and repair operations in space.

The architecture proposed for the JPL telerobot decomposes the functionality of system operations into three hierarchical levels. For the autonomous component of the architecture, these levels consist of the Task Planning Level at the top of the hierarchy, followed by the Process-Level at the intermediate level, and the Actuation and Sensor Level at the bottom of the hierarchy. Each of these levels is designed to operate robustly by using local feedback to detect and recover from local errors.

The Task Planning level is concerned with the overall context of the task, including global planning, overall execution monitoring and task replanning. The basis for the decision making is the Task Sequence Logic with relatively little incorporation of Task Domain Physics. As a consequence, activity timing at this level is driven by task scheduling requirements, and the decision making methods used are predominantly symbolic and not numeric. In the JPL telerobot the functions at this level are mechanized by an Artificial Intelligence Planner subsystem (AIP) with the logical planning of a Module swap-out sequence being a representative example of internal subsystem activity.

The Process Level is concerned with the planning, execution and monitoring of subtasks such as grasping objects, object assembly etc. Activities at this level require only a local subtask context within which various sensor and actuation subsystems are commanded and coordinated to accomplish a given subtask. Decision making at this level has to be cognizant of the physics and geometry of the task domain and the occurrence of various run-time physical events in the telerobot and its environment. Decision making is characterized by a hybrid of symbolic and numeric processing with activity timing determined by the event line of physical sensed events. Mechanization at this level is provided by a Run-Time Control subsystem (RTC) which performs the various trajectory determinations, reflex specifications etc. for subtasks such as grasp centering and activation, parts mating etc.

The Actuation and Sensor Level serves to provide the basic manipulation and sensor capabilities of the robot. The activity at this level is intimately tied to the physics of the domain but no task context is required. Activity occurs in continuous time (practically realized by high rate servo sampling) and is predominantly numeric in nature. Functions at this level are mechanized by the Manipulation and Control Mechanization subsystem (MCM) and the Sensing and Perception subsystem (S&P) with compliant motion execution or object tracking constituting representative examples of subsystem activity.

Similar levels may be identified with the Human Operator, with High Level cognitive and planning functions at the top of the hierarchy, motor skill coordination at the intermediate level and muscular and sensory systems at the lowest level.

Coordination between the autonomous levels of the hierarchy and the Human Operator is provided by three methods. At the lowest level coordination is enabled by providing the Operator with the ability to directly interface with the robot arm and sensors via force reflecting joysticks and stereo camera systems. At the top of the hierarchy the coordination is provided via interactive task planning activity between the Task Planner and the Operator. At the intermediate level the coordination is achieved via a mechanism known as Traded/Shared control described further below.

Traded control refers to the situation where the Operator relinquishes control over process level operations for certain segments of the task sequence execution while retaining it for others. The transition between the operator controlled segments and autonomous program controlled segments is achieved via a 'hand-off' protocol that ensures that information about the world state is correctly communicated between the operator and the machine.

Shared control refers to the situation where the Operator and the autonomous programs jointly participate in decision making at the process level. The contribution of the autonomous programs could be passive in nature as in the form of monitoring operator actions, e.g., ensuring that joystick actions will not lead to possible collision with the environment. The participation may also be much more active with the autonomous programs actually contributing to the manipulation of objects in the environment, e.g., letting the operator control the wrist of the robot arm while the autonomous programs control its global positioning.

The overall structure of such a system architecture is depicted in Figure 1.

3. Run-Time Control Architecture Requirements and Constraints

We now focus on the architecture for the Run-Time controller. The functions of this architecture are to:

1) Mechanize Process Level Operations
2) Permit Man-Machine Coordination

The specific design of this architecture is influenced by the context within which the telerobot operates, the details of which are described below:

1) Man-in-Loop

This feature has the most significant impact on the design of an RTC architecture. The existence of the capability of teleoperation implies that the autonomous capabilities of the autonomous RTC architecture need not be complete. It is sufficient to have an architecture that is capable of performing autonomously for a reasonably high percentage of the times the task is required. Since most computational algorithms to plan and mechanize simple physical subtasks such as grasps etc. are exponentially hard to perform, this feature of the architecture enables algorithms to be selected based on their likelihood of successful performance.

The architecture is also required to functionally interface to both the Operator and the AIP planner. This implies that an Interpreter style command language is required to enable the human to effectively communicate with the system. The Operator must also be able to interact with the system in a manner that does not involve a knowledge of the detailed implementation and computational aspects of the system. This requirement is usually taken to mean the implementation of a Task Oriented Language where actions are implicitly specified in terms of the required task. This concept is implemented in the RTC via a Task Command Language augmented by a constraint specification scheme. The constraint specification language not only allows the operator (or the AIP) to designate the required physical task, but also allows the specification of a set of constraints that are to be maintained by the autonomous system during the execution of such a task. The autonomous system then determines the appropriate actions to execute the task and also satisfy the constraints. A sample entry of the RTC command dictionary is shown in Figure 2. The operator is also required to interact with the system in a shared mode of operation as described earlier. The constraints specification command language allows a natural and easy extension to permit the specification of shared control activity.

2) Specialized Environment

The telerobot is required to operate in a space environment performing tasks such as satellite repair, structure assembly etc. The specialized nature of the tasks implies that standard factory automation paradigms are inapplicable. Further, unlike a highly changing manufacturing environment, the space application environment does have a substantial body of off-line knowledge available in the form of actual astronaut experience, earth simulators etc. The existence of this knowledge indicates that script based techniques are feasible for decision making, because they permit the transfer of the available knowledge to the system with the least effort. Of course, as the telerobot takes over more autonomous functions, the scripts could be replaced with modules capable of reasoning and generating their own activity.

The other constraint of the specialized environment is the perspective shaped by the fact that the Run-Time Controller will eventually form part of an Embedded system implemented in space qualifiable hardware/software. Also, since the controller is part of an end-to-end telerobot system, requirements of overall system design affect the design of the local subsystem architecture. While these constraints are not rigid since the first implementations of the JPL telerobot are necessarily ground based research prototypes, the requirements for an evolutionary growth in the system require that these factors be considered while designing and building the baseline architecture.

3) Parallel Activity Tracks

A requirement of the telerobot architecture is the management of parallel task execution and the simultaneous use of many physical resources available to the robot. Physical resources consist of various manipulation resources such as robot arms, end effectors, mobile platforms etc., or various sensor resources. Each of these resource could be active simultaneously as in the example of dual arm independent or coordinated actions. The architecture must be embedded with appropriate resource logic to ensure that resource conflicts do not occur when simultaneous activity tracks are in progress. Further, the error detection and recovery for each of these independently controlled resources must be coordinated.

4) Computational Bottlenecks

Computational resources must also be managed. Most robotics algorithms are computationally very intensive, far outstripping the capabilities of uniprocessor machines. Concurrent processing in a distributed multiprocessor environment is necessitated, and the system should therefore be capable of performing the dynamic computation task allocation necessary to optimally use the available computational resources available to the system.

In the RTC, the computational paradigm that has been adopted is the message passing multiprocessing system.

5) Robust Operations

Robust operations at the any level of the hierarchy require the implementation of feedback, where the actual progress of a subtask execution in the world is monitored to guide further actions. At the sensor and servo level, this feedback is implemented via various control system schemes designed to operate with a continuous physical world. The RTC, on the other hand, implements a feedback scheme that operates in event space. A more detailed description of an event space formulation of the RTC decision making is given in the next section.

4. Event Control

The notion of event space and event control is intimately connected with the time horizon of the subsystem of interest. It can be argued that reaction time for a subsystem at any level of the hierarchy should be sufficient to react to a possible world change that is relevant to the goal commanded to that level. Therefore for any subsystem, its time horizon should be comparable with the time constant of the control processes at that level. While the precise determination of appropriate time horizons must necessarily be ad-hoc, it nevertheless allows the definition of a hierarchical control scheme, where a higher level relies on the lower subsystem for the execution of commands associated with small time constants.

A hierarchy in the chain of command is naturally coupled with a hierarchy in the processing of sensory information as well [1]. This leads to an important simplification in the overall task execution monitoring activity in the telerobot. That is, the subsystem at the commanding level is relieved from the mundane supervision of command execution at the lower levels, and is instead free to focus on monitoring and anticipating a smaller (and finite) number of crucial events.

Taken together, a "trusted soldier principle" [2] may be formulated. According to this principle the overall process-level decision space can be factored into two distinct components. One component is acted upon by the RTC, while the other is the sole responsibility of the sensor and servo subsystems.

On any level of hierarchy, Event Control requires the mapping from the space of sensor event sequences:

$$\{S_j^k\},\ i = 1..M_k\ K=1..S_n$$

to the space of actuation event sequences available for this level:

$$\{A_i^k\},\ i = 1..N_k\ K=1..S_n$$

where 'i' is the running index for the sequences and S_n represents the number of lower level actuation/sensor subsystems.

While this formalism can be used to describe a traditional servo system with digital elements in it, its main goal is to create a finite parameterization of actuation and sensor events therefore making error recovery through search possible.

The "trusted soldier" principle leads to a unified command format. Commands to a lower level specify goals, constraints on the allowable ways to reach this goal, specifications on when activity should be stopped, and report formats as well.

In our case the communication between the levels of the event control space (RTC) and the continuum space (MCM and S&P) is performed by a Primitive which is modelled as follows:

Primitive T = (P,M,F,R)

where:

P - desired trajectory (position x force space)
M - desired class of control law
F - desired reflex predicate (stop function)
R - desired event reporting function

The components P and M specify the physical motion to be executed by the robot. The component F determines the termination of the specified physical motion via a sensor predicate which when true triggers the appropriate reflex action (typically a stop). The component R determines the sensor predicates that generate the event reports to the RTC.

The specification of P and M determines the desired map from the actuation event space to the continuum sensor/actuation space. The component R specifies the map from the sensor space to the sensor event space. The component F specifies the precomputed reflex actions that must be taken by the MCM and represent the algebraic (memoryless) and local components of the decision map that are to be executed within the fast decision time constants of the MCM subsystem. We want to emphasize that the stop function F is to be defined before the start of the actual command execution, therefore allowing preplanning for all possible outcomes of T.

Event Control in the RTC

The interface between the RTC and the lower level subsystem adopts the "trusted soldier" principle of operation. That is, the lower level subsystem has the full responsibility of performing the desired command T. If T is unsuccessful then the role of the RTC is solely that of coordinating and interpreting the various reports R (from all of the subsystems) to determine the next command T to be sent to one of the lower level subsystems.

Given this mode of operation, consider the actions to be taken by the RTC on reception of a command from the AIP or the operator. The command may be eventually translated according to one of the four following cases:

1. COMMAND --> (P,M,F,R)

2. COMMAND --> (P_1,M_1,F_1,R_1)

.
.

--> (P_n,M_n,F_n,R_n)

3. COMMAND --> $[(P_1,M_1,F_1,R_1)\ (P_2,M_2,F_2,R_2)\dots(P_m,M_m,F_m,R_m)]$

4. COMMAND -->

$[(P_{11},M_{11},F_{11},R_{11})\ (P_{12},M_{12},F_{12},R_{12})\dots(P_{1m},M_{1m},F_{1m},R_{1m})]$

.
.

$[(P_{n1},M_{n1},F_{n1},R_{n1})\ (P_{n2},M_{n2},F_{n2},R_{n2})\dots(P_{nm},M_{nm},F_{nm},R_{nm})]$

Case 1 corresponds to a single possible primitive T (albeit parameterized) that can be sent to the lower level subsystem. An example can be a command to move an arm to a given joint position using given joint interpolation.

Case 2 corresponds to the case where one of many primitives T may be sent to the lower level subsystem. A command that specifies end position, but not the specific trajectory to be selected by the RTC, can be used as an example here.

Case 3 corresponds to the case where a sequence of primitives T is necessary for process-level command execution, and case 4 deals with the situation where more than one sequence is available.

Since only one command can be send to a lower level at a time, the ambiguity that exists in the cases 2 and 4 must be removed. There are several possible ways to do it. A "hard rule" approach assumes an existence of an implicit agreement between the commanding subsystem and the RTC on how a single fourplet should be chosen. For example it may be assumed that a straight line interpolation should be always used. In this case there are no real differences between cases 1 and 2. On the other hand a "soft rule" would allow the RTC to rank all candidates and start execution from the "best" one. If it failed, and the action were reversible, then another candidate could be tried. The RTC implements a mixed approach.

It ranks possibilities internally based on an RTC internal criteria; but after a choice is made and execution started, the rule becomes "hard," and an error in execution would require the RTC to report it back to the commanding level without exploiting the rest of the options.

Commands to the RTC which result in Cases 1 and 2 are called Actions. Commands resulting in Cases 3 and 4 are called Skills. Skills require a sequence of event control decision making prior to the issuance of each lower level subsystem primitive T. In Case 3 the sequence of primitives is constrained by the one possible mapping indicated earlier, but the numerical parameterization of the individual T is determined at run-time by the RTC. On the other hand, Case 4 allows the possibility that a different sequence of primitives together with their individual parameterization may be selected during the middle of the execution of a specific sequence. Since the creation and execution of this sequence is dynamically determined by the RTC, the ability to change the parameterization of a primitive T, or the actual change of a sequence, gives the RTC the ability to perform "trimming" (i.e., error recovery of subtask command execution). Note that by virtue of this definition of trimming, Actions cannot be trimmed.

Using an analogy from control theory, the modules that implement the 'forward loop' operations of Action selection or a nominal Skill sequence determination are incorporated into a Process Level Planner, and the 'feedback loop' that analyzes events and determines the need for trimming is incorporated in Monitor, Predictor, Evaluator and Trimmer modules. The feedback paradigm corresponding to this event control system is shown in Figure 3.
It should be noted that two key assumptions govern the functionality and design of these modules. It is assumed that a nominal task sequence is known from off line analysis, and that in a case of ultimate failure, control can always be surrendered to the Operator.

For the MCM a particular implementation of command components P,M and R is given by:

P - a set of via points in joint/task space, or spline functions.
M - position mode or compliant mode control.
R - standard report information returned to the RTC after stop (or reflex actions): joint positions, force/torque sensors readings, and the reason for end of execution.

The choice of the stop functions set F is more complicated. It is clear that F must be complete in the sense that some condition must be eventually triggered. The triggering conditions must also be unambiguous, namely only one stop function should be triggered at a time. Such a choice of the stop functions makes the number of possible outcomes for each MCM command finite.

The process level planning is based on scripts. This allows the embedding of careful off-line analysis of the possible outcomes. In the future an expert system can be implemented to make a choice of an appropriate trimming action. Another advantage of scripts lies in their ability to incorporate ad-hoc domain specific knowledge for planning of a nominal, feed-forward part of a skill. For example an approach position for a grasp can always be determined as a set distance away from the grasping point. Another specific feature of the current implementation is the existence of a command-object cross table which provides for every action, a list of parameters needed for nominal planning of this Action for each known object. It can be filled off-line or, if appropriate modules are available, on-line upon demand. The modular nature of this architecture allows system modification and extension in an orderly fashion.

Skill implementation includes two different phases: planning and execution. First an appropriate script employs a generate, test and modify paradigm to make a nominal sequence of actions. At the same time the script provides a list of possible trimming actions that can be employed to recover from the errors on the intermediate steps. Then the execution is performed according to the following loop:

```
WHILE Action_Stack Not_Empty
loop:
     Bind Action Parameters;
     Set_up Evaluation_context;
     Send Action to the MCM;
     Wait for Report;
     Evaluate Report;
     if SUCCESS then
          null;
     else
          Determine trimming
            if ANY AVAILABLE then
              put trim_action on Action_Stack;
            else
              Report_up(Fail);
            endif;
     endif;
  endloop;
```

An actual example of a script to perform object grasping is shown In Figure 4. Numbers after each action refer to the trimming stack associated with the grasp script.

5. Implementation

The previous sections have described some of the techniques used to perform certain desirable simplifications, such as mapping a continuous problem space into a discrete one. Now the structure and implementation of the RTC architecture will be examined.

As background, the software implementation of the RTC is on an AI VAXstation II, under MicroVMS. The software is written primarily in Ada, with some portions in C and Fortran, which are accessed with the Interface pragma supplied b/ Ada. Ada was our choice of language for the RTC deliverable software, but we perform rapid simulation and prototyping of algorithms in Prolog, Lisp, Smalltalk and a locally-written development environment called Thread. Our choice of Ada for the deliverables was made for several reasons, the most obvious being that the RTC will eventually have to comply with the Space Station all-Ada software requirement anyway. In addition, the ability to express concurrent programs directly in the language without resorting to operating system interfaces was also a desirable feature. Our strongest reason, however, was the extremely high modularity of Ada, allowing several people to work on the same software without disastrous consequences. This, coupled with the enormously high level of compile-time error checking in Ada, turns out to have been a true expectation. Our software production has been only loosely coordinated among four different people, but it has experienced no incompatibility problems and has a history of phenomenally low levels of runtime errors. We believe that Ada is a good choice for our deliverables since they are of precisely the category of software Ada was designed to fit, namely distributed embedded real-time systems. We would make the comment, however, that Ada is not a good choice of language to prototype in.

The telerobot will be used over the course of several years for many purposes, including as a testbed in which to investigate both low (e.g. feedback control laws) and medium (e.g., grasp strategies) level robotics algorithms, as well as for the targeted demonstration scenarios. For this reason, the architecture of the RTC must have the character more of a development environment rather than of a finished product. The ability to reconfigure the telerobot to perform servicing tasks on satellites other than the one targeted for testing is one of the obvious requirements. It is our opinion, however, that the many complex interactions which can take place during various types of robot operations, even in such a structured situation as the servicing of a modular satellite, would make it likely that even state-of-the-art robotics algorithms would be far too rigid to meet all needs. To insure sufficient flexibility, one must not only provide as much as possible in the way of current robotics algorithms, but also provide the ability to completely change how, when and which algorithms are applied.

The ability to do such general restructuring within a practical robotics implementation has very large software consequences since the addition of a particular algorithm to the mix used in the robot might require brand new data representations to accommodate new information relevant to previously unmodeled properties of objects, and the addition of the new algorithm could very well require completely redesigning the strategies used for applying various currently-used algorithms. An example might be the addition of a new grasp planner module, which might require that more data concerning each object to be grasped be installed in the data base and that the previously used strategy for stable grasp position determination be disabled or modified.

In order to support such a high degree of flexibility, the structure of the RTC architecture is that of a framework in which robotics algorithms can be installed, rather than a particular mix of algorithms. The RTC consists of several modules with a very rough degree of functionality assigned to each type of module, but the specific inputs, outputs and details of operation which each module performs are controlled by reconfigurable data sets supplied to each module. The rough partitioning mentioned above is relatively simple and is an ad-hoc solution to the question of how to slice up the problem of processing the commands received by the RTC. One important characteristic, namely the ability for the RTC to accept commands in parallel, had a major role in selecting the problem separation.

The RTC is required to be able to perform more than one command simultaneously, assuming that they do not conflict with each other. The task of managing access to various sharable, non-shareable and queue-based resources is fairly straightforward in the context of the telerobot, due to the fact that the autonomous system may be interrupted by the operator at any time. This makes it virtually impossible to perform any time-scheduling of resources, so the simple restriction is made that all commands must be independent of each other with respect to time shifts. This removes the necessity of performing the many complex sorts of resource allocation and activity coordination which would otherwise be required between commands. This restriction is not as crippling as it seems, however, since there is no rule which states that a single command can use only one resource. In fact, if a complex cooperative task involving two arms and a vision system is desired, it would simply need to be formulated as a single command so that the coordination operations necessary could take place within the framework of one command, without requiring propagation to other activities. This independence also greatly simplifies the task of managing response to unexpected error conditions, since all that is necessary is a simple termination of the command, or of all currently executing commands if the error is a global one. Again, this simple behavior is not restrictive, since any sort of complex error response can be produced, if desired, simply by including it as an expected possible off-nominal situation in a command. Obviously, there is a great deal of possible interaction between robot actions in the real world, and formulating two commands in such a manner as to be truly time-shift invariant might be very difficult in any given situation, but this is not really an issue. If the two operations are unconnected, they can simply be done in series, formulated as two separate commands. If they must be done cooperatively, they can be formulated as one command. The performance of dependent operations in parallel could aid efficiency, but is not a key requirement.

With the above preface, the ad-hoc decomposition of the RTC's processing of commands follows.

Command Parsing

- An incoming command from the higher-level system is first converted into some internal working data structure used by the other modules; this allows decoupling internal operations from external interfaces.

Script Elaboration

- The command is then examined, and a specific sequence of activities is decided upon to execute it, together with specific possibilities for responses to off-nominal events during execution. An example of this would be a grasp command, for which a sequence of three straight-line motions followed by a closure of the gripper was decided upon, with the off-nominal behavior of backing up to the starting point and simply repeating the entire motion if the grasp does not succeed on the first try. This sequence is specific in the number and type of motion primitives to be executed, but no details are present yet as to which trajectory is to be used or what the point of grasp on the object is to be.

Action Binding

- An elemental motion/sensing primitive in the sequence is then further processed, in order to determine the precise numerical parameters for its execution. This primitive is then sent to the appropriate subsystem, which responds with one or more reports back to the RTC providing progress/result information about the command's execution.

Report Analysis

- The returned report(s) are examined, and a determination is made whether to continue execution, abort or take some off-nominal corrective action, such as the retry option mentioned above. If the decision is made to continue, the previous step is then repeated for the next primitive in the sequence, followed by this one, until all primitives in the sequence have been executed.

The overall architecture of the RTC is one in which each of the steps outlined above has been assigned to one type of module, with each command thus being processed by one of these four types of modules at any one time. Multiple modules execute in parallel and communicate by message passing in a very simple way. New instances of each module can be created as needed to process parallel commands since each module is simply a worker which is dispatched with a job to do and then returns with a result, with no permanent memory of its own and (with a single exception) no side-effects. All information relevant to the processing and execution of the command is contained in the inputs and outputs to each module, with the exception of spatial/geometric information about the state of the world, which is contained in a global database. This database can be read by any module, but only the module assigned to analyze the reports returned from an executing subsystem has the authority to write to it, thus updating the information in the database with the data returned by the executing subsystem. This is the single side-effect present in the execution of any module, which removes the possibility of write contention and other such coordination problems.

The RTC is thus made up of zero or more copies of each of the four command-processing modules, together with several additional modules which perform various essential functions:

- An Interface Server, which serves as a communication port to the other subsystems in the telerobot.

- A Decision Unit, which is the central dispatcher/coordinator for the RTC. This is simply a finite state machine, with the four-phase command processing behavior built into it, along with the terminate command-on-error and resource-management behaviors. In the nominal case, it simply feeds the input command into successive modules, propagating one module's output to the next module's input, with minor exceptions. In anomalous situations, it simply takes several straightforward steps to insure that the executing subsystems and internal modules working on that command are shut down and sends a report of the halt back to the higher level system commanding the RTC.

Because of the simplicity of the Decision Unit (a simple decision tree), it needs only a very short period of time to execute any given response to an event, such as an incoming message. For most cases, it simply sends out the incoming data to the appropriate module. This allows very rapid response to exception conditions, such as a command from the telerobot's human operator to stop executing. The Decision Unit will almost certainly be idle within a few milliseconds of receiving the message, and can then process it, and send out subsystem halt primitives very rapidly, giving only a few tens of milliseconds of delay between the operator's commanding the halt and the arm actuation subsystem halting arm motion.

• Command Parser modules, as many as needed, which convert the input to an internal working form. It should be explained that one desirable feature of the telerobot is to let the operator intervene in execution at any level of the command hierarchy from AI (activity planning interface) down to detailed arm motions (teleoperation), so the RTC's command inputs are also of human-usable form. The command input from the higher level is thus in the form of an ascii string with fairly readable content. From the RTC viewpoint then, there is no distinction made between the human operator and the AI planner.

• Script Elaborator modules, as many as needed, which decide on the basic script to follow for executing the command. In cases where a parameter for an earlier primitive action must be determined by using a precise value for a parameter to an action to be executed later, (e.g., a precise grasp point may be needed to backtrack a trajectory to the current position of the arm) then determination of these numerical parameters is also performed at this phase, and the detailed parameters are inserted into the primitive before it is sent to the next phase. This permits the use of backtracking as a planning technique if desired, along with any other technique which requires a noncausal determination of specific parameters for actions.

• Action Binder modules, as many as needed, which determine the precise numerical values needed for each subsystem primitive to be sent out for execution. The Action Binder would be invoked once for each primitive in the script, whether the primitive was a nominal action or a response to some planned-for off-nominal condition. The stepwise invocation of the Action Binder allows the use of parameters determined at one point of the execution of the command during run-time to be used at later points in a very simple fashion. Also, as described above, a parameter for an action primitive may be already fixed when it arrives in an Action Binder, which will not disrupt the normal operation of the module.

• Analysis Unit modules, as many as needed, which examine the reports returned by the executing subsystems, update the global geometric/spatial database accordingly, and determine the recommended course of action to be followed, which would be to continue normally, abort, or take a specific foreseen corrective action. This recommendation is then sent to the Decision Unit, which may choose to follow it, if overall execution is normal, or may ignore it and choose to terminate the command, if for example the AI has sent a halt instruction to the RTC during the execution of the command. The key point concerning the AU modules is that they operate entirely on command-related information, and do not take into account such things as global anomalies, but operate as a memoryless single-input single-output system. The input is the command context together with the report sent back by the subsystem, and the output is the recommendation for action. The Decision Unit worries about coordinating any overall behavior which crosses command boundaries.

• A Database Server module, which simply behaves as a shared-memory area for the other modules. Any given location in it can be written to by only one AU at any given time, and there are various validity flags to indicate whether or not other modules should be allowed to read any given piece of information.

• In addition to these modules, which form the basic configuration of the RTC as shown in Figure 5, there are several others which will not be detailed, but which exist because of the requirement that the RTC's geometric database be accessible to any subsystem in the telerobot, to act as a central repository. Also, there is the requirement that the AI planner, in order to perform its planning, may need to use backtracking methods. To support this, the RTC contains an identical copy of the Command Parser/Script Elaborator/Action Binder modules which are used as a hypothesis-testing facility by the AI planner. The planner can simulate as much as possible of the execution of a possible RTC command, without actually commanding any subsystem activity, so as to determine whether or not a particular line of planning will be found to be infeasible at the numeric level by the RTC. This facility allows the fairly ad-hoc division in the planning made between the AI planner and the RTC to function robustly in the presence of couplings between the symbolic and numeric levels of robot activities.

The primary aspect of the modules which perform command processing is that they are implemented not as hardwired entities which operate on data, but as data-driven "interpreters," which take as input a program, as well as data. This implementation allows the modules to be easily reconfigured to use a new algorithm if at all possible. For example, adding a new algorithm may absolutely require a complete restructuring of the existing system because of robotics considerations, but such restructuring would not be necessary simply because of software difficulties. This feature allows the addition of many types of robotics and also AI techniques to the RTC if desired since there is no restriction at all on what data is passed between the modules, and only trivial restrictions on the order in which they are invoked. It is our hope that later versions of the Script Elaborator will use AI-type reasoning techniques to produce the basic scripts for use during execution, rather than the cut-and-paste/table lookup technique used now. This would give the ability to literally replan a sequence in the event of an error, instead of forcing error responses into specific scripts. Also, a prediction capability, capable of utilizing expectation information in analyzing the sensor data during execution, would be a desirable feature to add to the Analysis Unit. The ability to remove the independency restriction on separate commands and perform coordination between more than one command would also be useful in increasing system efficiency. All of these ideas have been explored in a preliminary fashion, and several fairly straightforward implementation alternatives have been found to each of them, indicating that the ability to specify module behavior at run time, rather than just its input data, is an extremely powerful feature of the RTC.

A feature of the RTC is the fact that the memoryless, dispatched-worker module format is ideal for implementation on a multiprocessor hardware architecture. Modules do not perform any significant nonlocal references (the exception is database access, which can be reduced by simply grouping read requests into bunches, rather than lots of individual read requests) and do not require any communication among themselves during execution. A preliminary study indicates that conversion of the RTC to operate on a hypercube multiprocessor would be a very straightforward task.

Another aspect of the RTC's internal operation is that it is relatively simple to treat the human operator as a subsystem to be commanded by the RTC. This allows several simple but effective solutions to the extremely difficult problems of specifying to the autonomous system what the intentions and outcomes of human teleoperation activity are. For example, if the operator intervenes into autonomous execution and picks up an object, there is no way for the autonomous system to examine arm trajectories and gripper force data to determine that the object was grasped at all, or if it was grasped what the position of it in the gripper is. This very simple example shows how difficult is the task of sharing control of an autonomous system with a human operator. One solution, which by necessity imposes a good deal of overhead and restriction on the human operator, is to specify intervention activities to the telerobot in the same form as any other command, with the exception that the performing subsystem is the operator. This paradigm would mean that precise numerical parameters would be left out of each step, but the normal sequence of subsystem primitives would be used by the RTC, and likewise the usual sequence of RTC commands would be sent out by the AI, and the operator would face the restriction of performing only that portion of the task specified by the primitive. An exploration of this method of structured operator intervention, which is one candidate for the JPL telerobot, is given below.

If an object could not be reliably grasped by the autonomous system, the operator could instruct the AI planner to

"grasp Object with Right_arm via Operator_intervention."

This would result in the command trickling down the hierarchy through the RTC and a subsystem primitive to the operator appearing on the control console. In this example, the first might be:

"move Right_arm to_neighborhood_of Object".

The operator would perform this, in teleoperation mode, and indicate that he was finished. The RTC would then send out the next primitive:

"move Right_arm to_grasp_point_of Object".

The operator would comply, seating the gripper on the object in a satisfactory configuration so that when the gripper was closed, it would grasp the object firmly, <u>without moving the object</u>. The RTC would then be able, by performing kinematics and geometric computations, to know what the grasp point reference frame for the object was and would, therefore, be able to correctly update the data base as to the position of the object after it has been grasped. The final primitive would be sent by the RTC, directing the operator to close the gripper, and the operator's response would confirm that the grasp took place as expected, without disturbing the spatial relationship set up.

This scenario demonstrates the additional overhead imposed on the operator by the necessity of maintaining the autonomous system's integrity. It is essential, however, that some form of strong restriction be placed on the operator, not only to coordinate activity with the autonomous system, but to prevent operator disruption of the autonomous system due to human oversight. It is likely that there will need to be more effort put into protecting the autonomous system from the operator than into making the autonomous system operate effectively on its own.

Another issue we believe to be of foremost importance in the telerobot design is that of detection of anomalous conditions in the world. Even in such a highly identified environment as satellite servicing, it is crucial that sensor feedback be employed as often as possible so as to prevent any cascade of errors forward through the execution of a servicing task. It is unlikely that any AI/robotics autonomous system will be able to make allowances for error propagation in its activities, within the near future. Such propagation must be eliminated if the autonomous system is not to become completely confused, with major portions of its world model invalid, which is a completely unacceptable situation owing to the large amount of time and effort which would likely be required in order to restore the world model to a correct state (work weeks).

The development of such an architecture must necessarily recognize the limitations of current science and technology in this nascent area. Early architectures focus on integrating the process-level autonomous functions into the system for simple, independently controlled arms. Operator-machine coordination is restricted to simple traded control schemes. Later architectures support more complex physical environments (more arms, redundant arms), as well as more complicated functionality such as coordinated arm motion and true shared control. Concomitant with this increased complexity and functionality is the management of complex computational architectures and the integration of more sophisticated error recovery and planning methods into the system. At the present time, the RTC of the JPL Telerobot has been implemented to contain the following capabilities:

- Command Parser -- parses ascii strings from a simple BNF form into a record data structure containing equivalent information. The BNF language in use has roughly 100 terminal symbols, 50 clauses.

- Script Elaborator -- uses a simple decision tree to splice portions of command sequences together into a coherent script, together with off-nominal scripts. Has the ability to backtrack from goal point for purposes of trajectory generation.

- Action Binder -- performs generation of joint/task space linearly interpolated trajectories in a piecewise fashion, using kinematic and load-carrying constraints of the manipulator arms, together with collision detection, to plan small arm motions for grasping purposes.

- Analysis Unit -- uses a table-driven decision tree to evaluate the outcome of an execution and makes a recommendation. If the recommendation requires corrective activities, then the script to perform it is simply looked up in a table, having been generated by the Script Elaborator.

6. Conclusion

We have described a Run-Time Control architecture for the JPL Telerobot and discussed associated issues. This work was performed at the California Institute of Technology, Jet Propulsion Laboratory under a contract from the National Aeronautics and Space Adminstration.

7. References

[1] J.S.Albus et al., "Hierarchical Control for Robot in Automated Factory." Proc. of 13th. International Symposium on Industrial Robots, Chicago, 1983, pp. 29-43.

[2] A.Lokshin, K.Kreutz "Towards a Hierarchical Robot Control Language", IEEE CS Workshop on Languages of Automation. August, 1986, Singapore.

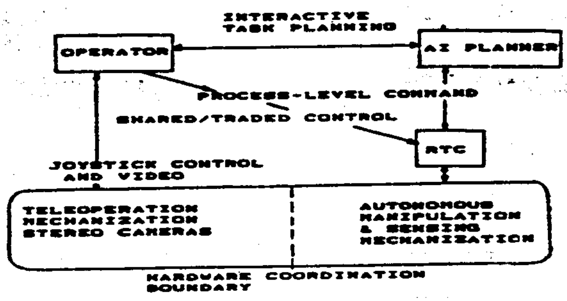

INTERACTIVE TASK PLANNING
OPERATOR
AI PLANNER
PROCESS-LEVEL COMMAND
SHARED/TRADED CONTROL
RTC
JOYSTICK CONTROL AND VIDEO
TELEOPERATION MECHANIZATION STEREO CAMERAS
AUTONOMOUS MANIPULATION & SENSING MECHANIZATION
HARDWARE COORDINATION BOUNDARY

Figure 1. Telerobot Functional Architecture

```
grasp --> GRASP goal maintain reflex
     goal --> end-effector graspable-object grasp-point
          end-effector --> LEFT-END-EFF
                           --> RIGHT-END-EFF


          maintain --> nil
                   --> MAINTAIN safe-work-range
                                avoid-task-region
                                avoid-neighbors
                                pose-during-motion
                                speed.

          reflex --> nil
                 --> GUARDED
                 --> reflex-items

             reflex-items --> nil
                          --> reflex-item reflex-items
             reflex-item --> UNLESS condition THEN action
```

Figure 2. Sample RTC Command

```
SCRIPT

        "grasp_ini"
        0 0
        "grasp_plan"
        0 0
        "active_arm JAP move_free movej"
        0 0
        "active_arm JGR move_to_contact move_joint"
        0 0
        "active_arm GP1 move_to_contact move_task"
        1 2
        "active_arm GP2 move_to_contact move_task"
        1 2
        "GP = 0.5*(GP1 + GP2)"
        0 0
        "active_arm GP move_tocontact move_task"
        0 0
        "close"
        0 0

TRIM STACK

        1       "active_object vision"
        2       "active_object grasp"
```

Figure 4. Grasp Script

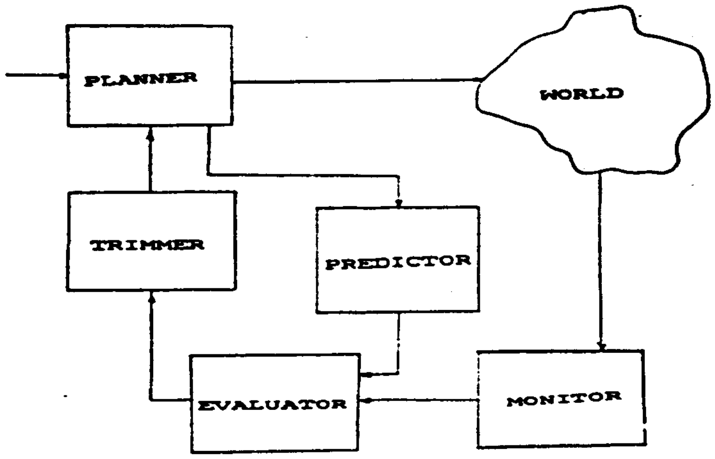


Figure 3. RTC Functional Paradigm

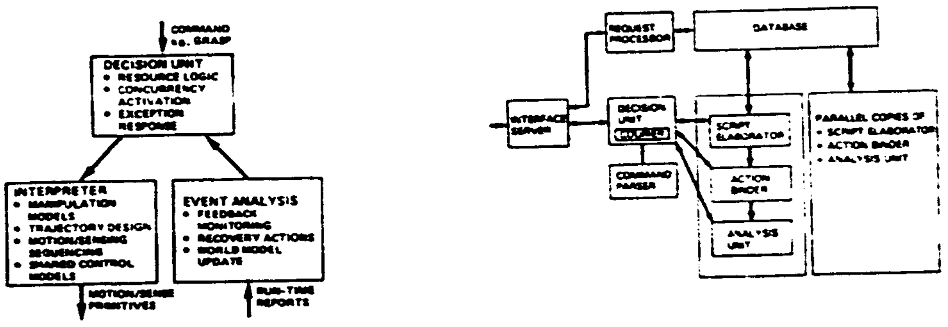


Figure 5. RTC Architecture

S24-37
9P.
188106

Distributed Control Architecture for Real-Time Telerobotic Operation

TM 031031

H.L. Martin, P.E. Satterlee, Jr., and R.F. Spille
TeleRobotics International, Inc.
Knoxville, TN 37931

1. Abstract

The emerging field of telerobotics places new demands on control system architecture to allow both autonomous operations and natural human-machine interfacing. The feasibility of multiprocessor systems performing parallel control computations is realizable. A practical distribution of control processors is presented and the issues involved in the realization of this architecture are discussed. A prototype dual axis controller based on the NOVIX computer is described, and results of its implementation are discussed. Application of this type of control system to a replicated, redundant manipulator system is also described.

2. Introduction

The development of the field of telerobotics is presently in its infancy. Driven forward by increasing application demands in space, nuclear, underwater, and battlefield activities, this new area of technology is rapidly expanding. The advent of more powerful and cost effective computing technology has provided solutions to many of the practical problems posed in the development of telerobotic controls. The development of a system that can expand with future technological progress, allow multiple programmers to simultaneously author code, and provide for autonomy as well as human control is the challenge that lies immediately ahead for NASA. To meet these challenges requires an open architecture with parallel processing performance and the ability to be organized in a logical hierarchy for future expansion. Partitioning of such a hierarchy requires that many questions related to performance, expansion path, communications, and software be answered.

The first step that must be taken to partition a telerobotic controller is to define the major control activities and information flow paths/rates that are associated with those activities. A top level listing of the activities that must be performed in a telerobotic controller is given:

1) Servomechanism control.
At the foundation of any telerobotic controller is the subsystem that must close the control loop around the encoder information and the motor drive amplifier to provide stable, responsive operation to input drive commands. Traditionally, robotic controls have utilized position as the input command and have sensed position and velocity information to accomplish closed loop control of joint location. More recently, developments have been made that allow torque control to be performed with servomechanisms. Different modes of operation may be required to optimize joint performance for a given task. This lowest level of control requires loop closure rates from 10 to 1000 Hz depending on the fidelity of control that is desired.

2) Human-machine communications.
The key to flexible and multipurpose telerobotics is the ability of the system interface to be made transparent to the human operator. Efficient operation of a telerobot in unanticipated applications will require that the human have full and natural command of all functions of the manipulation system. Improvements in graphic displays, master controllers, force-reflection capabilities, and viewing methods will be required to improve telepresence efficiency. Computational burdens associated with real-time graphic displays, controller transformations, and system diagnostics can result in slowly responding human-machine interfaces unless proper distribution of computational requirements is accomplished.

3) Sensor integration.

A primary key to successful autonomous operations is the ability to acquire and decipher sensory information from a number of different sensory systems. Acquisition and fusion of vision, tactile, force, and scanning information is computationally intensive. Such sensory information must be available at rates of 1 to 30 Hz depending on the type and quality of the information. Software to extract information from sensory data is being developed by a number of researchers at many institutions to solve specific application problems. Such diverse sensory development activities will conceivably continue requiring a flexible, but well documented sensory communications interface.

4) Activity/motion planning.

Planning is to robotics as the operator is to teleoperation. For reliable robotic operations to occur, the controller must have the ability to intelligently act on high level commands and adjust actions according to information returned from the sensory integration system. The capability to modify actions based on the condition of the environment is the key to developing a broad range of autonomous capabilities with the telerobotic controller. Equally important is the ability of the planner to know when the situation dictates that external human intervention is necessary to circumvent a difficult situation.

5) Internal operational communications/common memory manager.

Sequencing between planned activities and actual movement commands must be accomplished in a failsafe manner. The internal communications manager assures that the data transfers between the servomechanism controllers and the common memory data base occurs in an organized manner to assure that data collisions are minimized. The common memory data base serves as a documentable map of all defined sensory and control data locations. As such, it can provide the ability for a number of software developers to independently work on subsections of the code without requiring an entirely functional system. For long term evolutionary development that involves multiple software creators, the strict adherance to a documented common block of memory will save much time and effort while resulting in a flexible system.

6) External coordination communications.

To allow multiple manipulation elements to interact, it is imperative that an external communications handler be developed to sequence and transfer information from one manipulator memory common block to another manipulator memory common block. Language and communication rates must be delineated in order that the protocol for manipulator to manipulator communications may be determined.

7) Diagnostic handling.

Overseeing all of the activities that occur within the system, there must exist a diagnostic handling system. This system monitors the basic functionality of the system components on its lowest level and approves the general logic of activities on its highest level. This system diagnoses activities from the concrete (temperature, current trips, enables, etc.) to the abstract (collision between manipulators is emminent, tool not located, you are attempting to enter a restricted manipulation area, etc.) and provides the operator with condition and safety information that will protect valuable equipment.

A control system diagram that shows the interaction between these various processing centers is given in Figure 1. This control distribution process is suggested from personal experiences gained from the application and development of several control systems applied to force-reflecting teleoperation. The result of implementation of a decentralized structure is a control system that can expand as improved sensory and intelligence technologies become available. The resulting common block approach also allows activity definition at an early stage so that multiple integrators can work on the development of control system software. The activity within a given control center varies depending of the present mode of operation. Local intelligence attempts to replace human intelligence during autonomous activities.

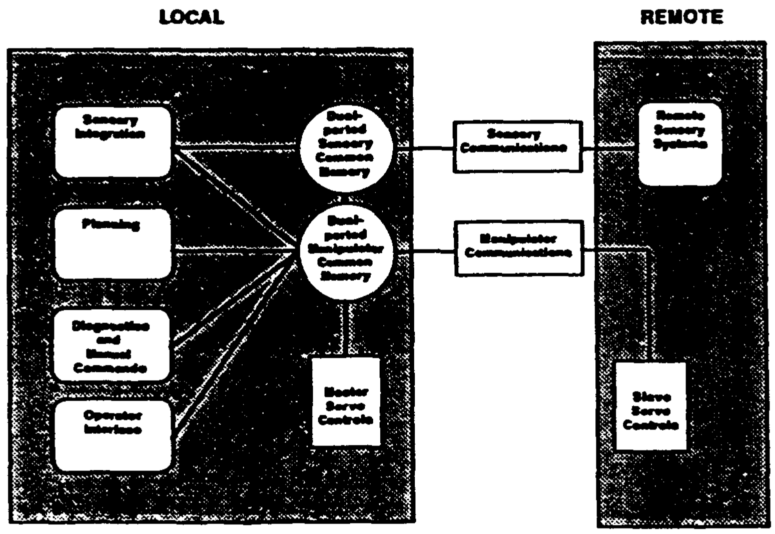


Figure 1. Conceptual layout of the principal control centers for telerobotic control.

3. Statement of the Problem

The functions described in the previous section can be accomplished in one or several processing systems. The essence of the problem is to determine the number of processors, the links between processors, the communications structure, and the upgrade development path that will provide the desired response, expandability, and computational performance required for a diverse and demanding group of control tasks to be developed over the next decade. Today's answers to these challenges are neither straightforward nor defendable as the technology will surely continue to advance. Certain approaches do have merit and should be supported in light of past experiences and anticipated advances.

The first architectural question that must be addressed is one of centralized versus distributed computation. Centralized processing refers to the use of a single, or small number of centrally located processors to accomplish all of the tasks described in Section 1. With distributed processing, a large number of less powerful processors perform multiple tasks simultaneously. There are arguments to be made for both sides. The use of centralized processors minimizes the communications and timing requirements and the synchronization that must occur if multiple processors must be utilized. Distributed, parallel processing systems provide enormous computational capabilities within a volumetrically small package. Hardware that uses both methods has been developed by the authors. Figure 2 shows the Model M2 control system that was developed at the Oak Ridge National Laboratory in 1982-84. It is a unique example of distributed digital control for force reflecting manipulator control. Utilizing over 30 microprocessors, this controller provides closed loop calculations at nearly 100 times per second. Such distribution allows online diagnostics, high speed loop closure, variable operation modes, and programmable operations to be accomplished at the servocontroller level. The system has performed very reliably for over 2 years of daily operation. The replication of an identical simple software package in each of the joints made the software development very efficient. The most complex portion of the control development was the sequencing of inter-processor communications. The greatest benefit of this form of control is the very small communications cable bundle between the master and the slave system. This system could be readily converted to wireless operation if desired. The greatest disadvantage to this implementation is the susceptability of the control components to the environment. This limitation can be addressed in future systems.

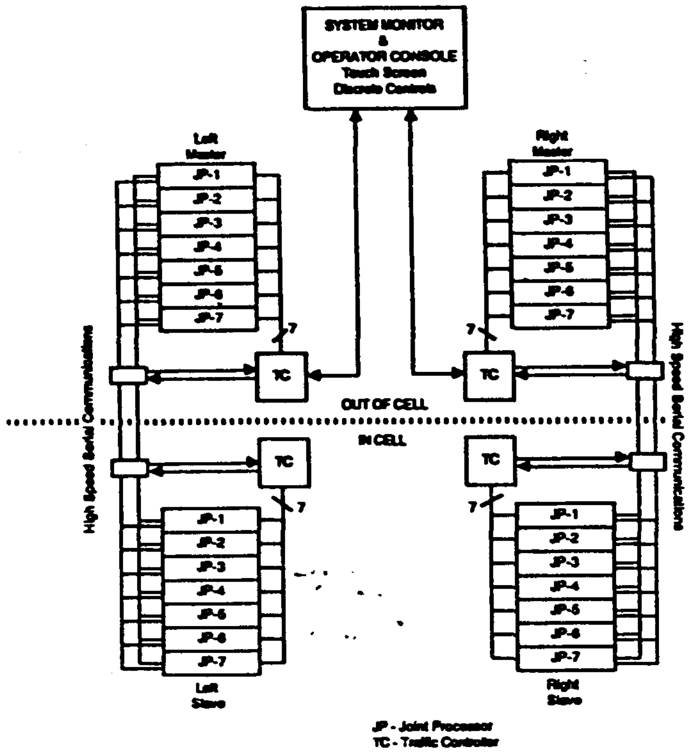


Figure 2. An example of decentralized force-reflecting manipulator control.

Figure 3 shows a general block diagram of a centralized control system. The control requirements of typical non-force-reflecting manipulators differ greatly from the M2 because the frequency response is much lower. As a result, a single host processor has the capability to sample, calculate, and control all of the functions at a much slower rate (~10 samples per second). If more diagnostic intelligence or force reflecting capabilities are to be added to this system, then the computational power must be increased. Additional processing power does not linearly improve calculational capabilities specifically because of the time required to communicate between computation centers. The REMOTEC RM-10A is an example of a non-force-reflecting centralized manipulator control system. The loop closure rate only effects the stiffness achievable in the servocontrol, but does not result in any noticable time delay between the master motion and the slave. The utilization of centralized control requires handling of a significant number of signal and power leads between the master and the slave system. The length at which this type of system will function is also limited due to lead resistance effects. The centralized control system is not amenable to wireless operation without the total redesign of the control electronics. This is not to say that the system has no merits. The centralized controller provides a cost effective, reliable means of performing limited manipulator operations.

The control systems selected for the Model M2 and the RM-10A manipulators were designed with the ultimate performance of the electromechanical system in mind. Force reflecting systems which require high throughputs of information have one level of control system calculational requirements. Non-force-reflecting systems with limited stiffness have a completely different set of control system needs. The design tradeoffs result from considerations of prototype cost, replication cost, performance needs, and reliability considerations. Both of these systems represent an implementation fit for its intended function.

To determine the duties of the control system, the next major concern that must be addressed is one of automation versus teleoperation. The course that NASA has laid out makes this determination very clear. It is a path that begins with teleoperation and evolves toward robotic operations. This means that the control system should be capable of both activities, teleoperation and robotics, and the architecture should be so devised as to allow expansion paths for future developments leading to sensory fusion and autonomous operations. The development of such a control system will result in the implementation of true telerobotic

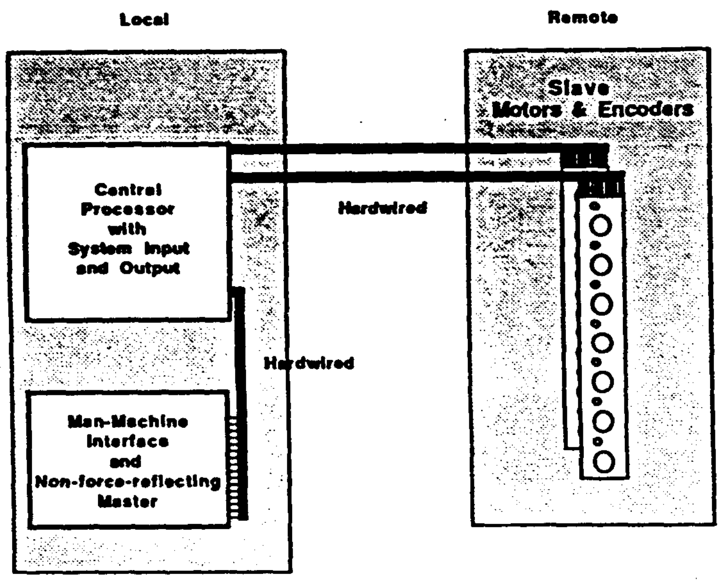


Figure 3. Example of a centralized controller for telerobotic control.

manipulation systems that work equally well under human or computer control. The elements of this type of control were shown in Figure 1. The majority of this paper will deal with those elements that are common to both approaches, the multimodal distributed servomechanism controller. By distributing the lowest function and highest throughput level of control, the distributed processing approach will assure expandability of the control system to future challenges and needs as they develop. The utilization of a common memory system allows definition of the present variable domain and can provide expansion to future variable domains.

4. Description of the Servomechanism Control System Architecture

The architecture of the servocontrollers in many ways determines the future expandability of the manipulator control system. While high level machine decisions are computationally intensive, the output data that results seldom has as high a bandwidth as the information passage between joints for master/slave operation. Sensory integration systems may have very high input data rates, but may only have limitedoutput needs. For example, a vision sensory/deciphering system has extremely high input bandwidths, but its output may be limited to geometric descriptions of the objects in the frame of view. Similarly, an operations planner may work on very large data bases, but its output results in high level commands that are not communications intensive (scan the past history of successful operations then determine the next step needed to accomplish a given task). The following is a description of the bottom-up approach focusing on the specific task of self-diagnosing servomechanism control capable of multiple modes of operation.

A prototype dual axis servomechanism controller has been developed with funding from the Department of Energy. The controller has the capability of acquiring data, receiving commands, and performing control activities for two servomotors. A detailed hardware description of the device is given in Section 5. The purpose of this development is to allow co-location of the controls with the motors to minimize cable handling problems, minimize the effects of environmental electrical noise, distribute the control complexity to its most fundamental level, and provide a system that is reliable and easily maintained. The resulting architecture is given in Figure 4. Each of the dual axis controllers shares a common serial communications bus and power bus. This yields a system that can be expanded significantly without experiencing cable handling problems that so often affect reliability. Using high speed serial communications, the serial bandwidth is sufficient to handle between 10 to 20 dual axis controllers. This architecture makes reconfigurable manipulation realizable for special applications targeted at space

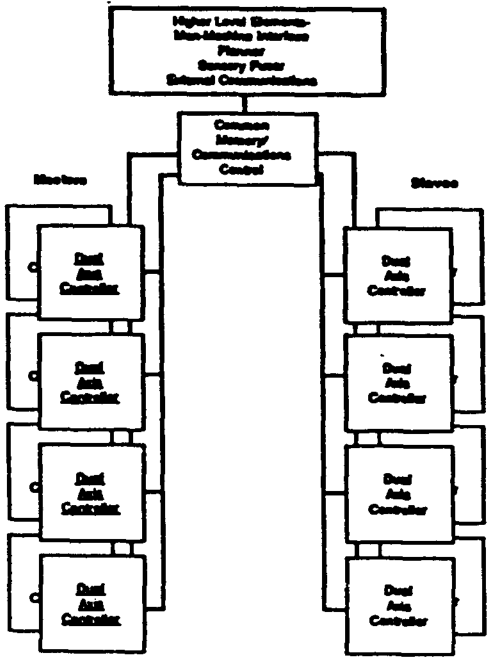


Figure 4. General architecture for manipulator control using dual axis controllers.

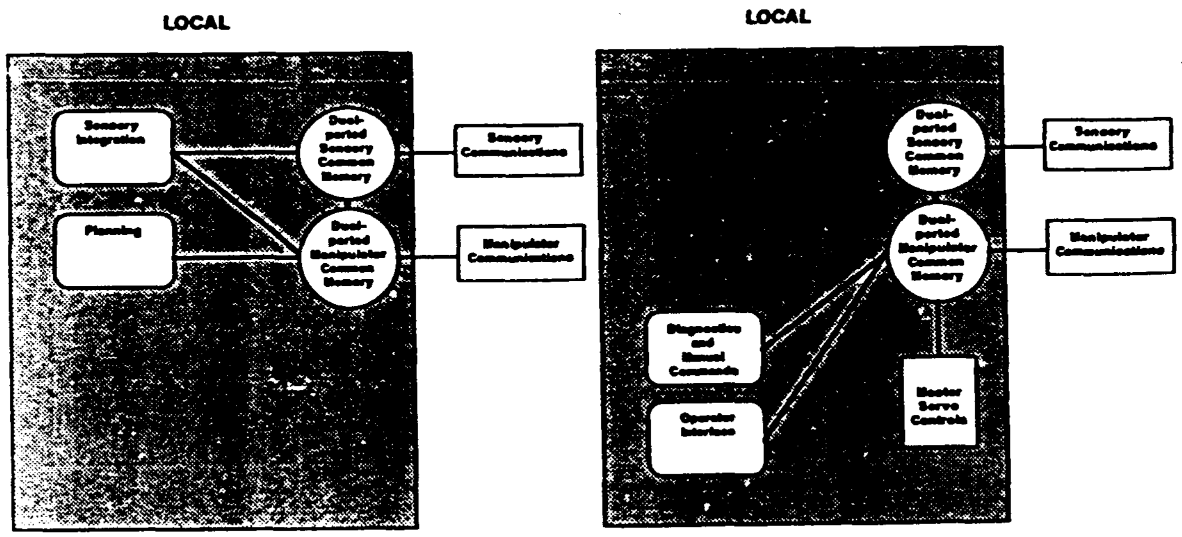
construction and ground maintenance. Major subsections of the dual axis controller are the processor, the input interface, the output interface, and the two amplifiers.

Each dual axis controller has a unique communications address and can respond independently to data requests made from a communications control processor. This processor sequences the data flow between dual axis controllers and the common memory area allowing dual ported access to the real time data collected and processed by other processing centers. The result is a high speed serial communications structure for the servomechanism control that is expandable to accomodate additional sensory and planning systems. The key to the success of this architecture is maintaining high bandwidth for the communications between dual axis controllers for teleoperation while allowing provisions for external computer control to sequence autonomous movements. The overhead associated with sequencing the different dual axis controllers must be kept to a minimum in order that the serial link between controllers hmaintain a high throughput rate.

The human-machine interface represents the high level side of the manipulation control system. During teleoperation, the human performs the functions of sensory integrator, planner, and instigator of activities while the computer performs the role of diagnostician, monitoring the condition of the system hardware. The major activities are completely inverted during autonomous activities as the computer senses, plans, and implements motions while the human is placed in a position of supervision for the activity. This inversion of responsibilities is accompanied by an inversion of the internal computer communications requirements. During teleoperation, sensory processing systems and task planners are not needed to their full extent, but the communications path between the master and the slave needs to be left unimpaired to provide responsive force reflecting operation. Figure 5 shows the differences in the principal communications flow paths depending on the type of operation that is occurring.

5. Hardware Implementation of the Dual Axis Controller

A functional block diagram of the dual axis controller is shown in Figure 6. The major components include the processing system (Novix microprocessor), the input/output interface, the communications interface, and the amplifier system. These components work together to provide stable, multimodal control of a dual axis manipulator element. Operating modes that are either functional or under development


LOCAL
Sensory Integration
Planning
Dual-ported Sensory Common Memory
Dual-ported Manipulator Common Memory
Sensory Communications
Manipulator Communications
LOCAL
Dual-ported Sensory Common Memory
Dual-ported Manipulator Common Memory
Sensory Communications
Manipulator Communications
Diagnostics and Manual Commands
Operator Interface
Master Servo Controls

Primary active elements during autonomous tasks.

Primary active elements during teleoperated tasks.

Figure 5. Main active elements for robotic and teleoperated control.

include: position-position control with velocity feedforward, position control with integration, velocity control, intermediate path generation between communication intervals, torque control, joint initialization, and internal diagnostics. In essence, this controller accomplishes all aspects of basic servomechanism operation and diagnosis. The Novix computer forms the foundation of this system, and it will be described in detail.

The Novix computer represents a new generation of microprocessor hardware. This processor is designed in its internal architecture to execute a high level language (Forth) as its "assembly" language. The result is a processing system that is extremely efficient, independent, and compact. Since other popular processors (Motorola 68000 series, Intel 8086 series, etc.) operate in assembly languages from which higher level languages are constructed, the speed with which the Novix runs Forth programs approaches an order of magnitude increase over these other systems. This result occurs even though the present Novix configuration operates at a considerably slower clock cycle (4MHz versus >10MHz). This philosophy of developing a microprocessor engine that executes a language, rather than a machine level instruction set, is a concept that will certainly spread to future microprocessor architectures.

The Novix is available on cell libraries and can be integrated with other standard cell devices to allow microcontrollers for specific applications to be developed. The potential for intelligent sensors, miniature servocontrollers, parallel processing/transputing nodes, and powerful man-machine interface drivers is quickly becoming a reality. Some of the more impressive features of the Novix microprocessor include:

1) Executes 8 millicn operations per second of high level codes.
2) One-cycle local memory access.
3) One-cycle multiplication and division instructions.
4) One-cycle subroutine (word) nesting.
5) Executes most Forth primatives in a single machine cycle.

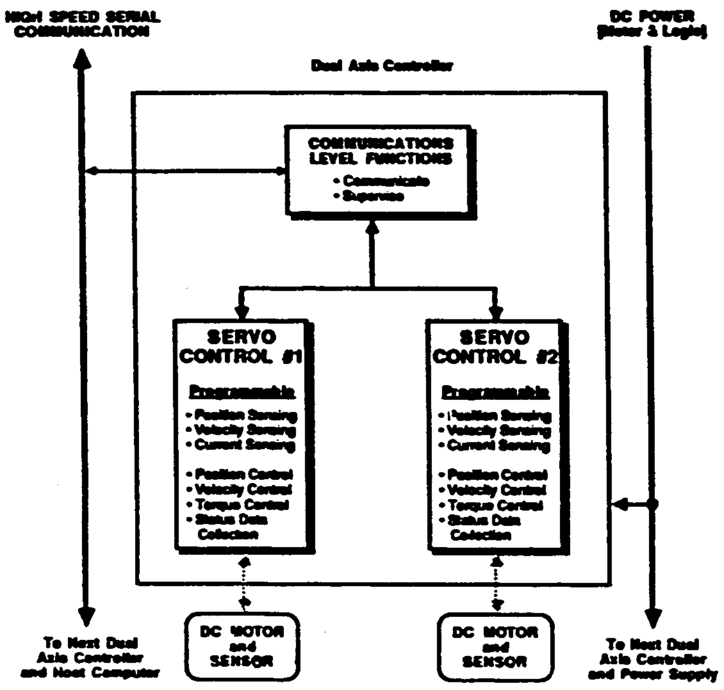


Figure 6. Dual axis controller functional diagram.

The dictionary structure of the Forth language allows the development of libraries of specific applicational words to be shared between programmers on similar systems. It also encourages top-down, bottom-up, or middle-out programming. This allows planning of the software functions to be accomplished in a number of ways allowing future software expansion capabilities. Figure 7 shows the hardware realization of this system in prototype form. A volumetric reduction of 50% will be accomplished to co-locate the motors and controllers within the joint module.

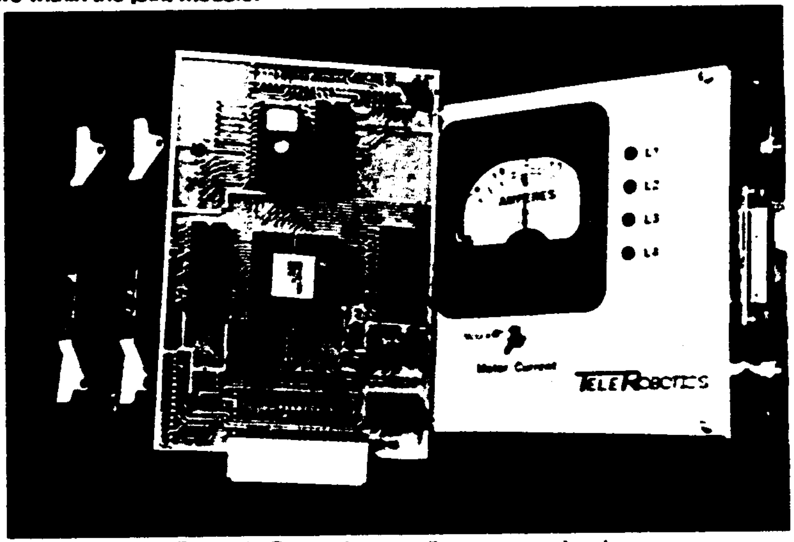


Figure 7. Dual axis controller prototype hardware.

6. Element for the Advanced TeleRobot (ATR)

The dual axis controller allows the co-location of controls and mechanisms into an entirely modular, self-contained unit. The advantages of this design and construction are numerous. First, each module can be replicated and utilized as a shoulder, elbow, or wrist joint. The shared power and communications keeps the cable handling at a minimum. The kinematic nature of the element allows reconfiguration of the joint motions to allow multiple kinematic construction to be accomplished (i.e., optimized kinematic arrangements for various task requirements). Remotely mated mechanical and electrical connections allow quick modification from one kinematic form to another. The basic mechanical element provides dual axis manipulation that can lift 30 pounds at 50 inches. Each element weighs less than 17 pounds. For ground based applications, a method of mechanical and/or electrical counterbalancing can be provided. The unit is backdrivable at approximately 20% of peak load. This allows compliant operation when the system is under external loads. The dual axis manipulator element is shown in Figure 8. The element is so versatile that it can be utilized as a camera pan/tilt device, a camera positioning device, and arm joint, or a torso positioner. More detail on both the mechanical and electrical implementations can be obtained from the reports "Analysis and Design Enhancements for the Advanced Servomanipulator" and "Using the NOVIX Computer for Control of Redundant Teleoperated and Robotic Manipulators" performed for the United States Department of Energy.

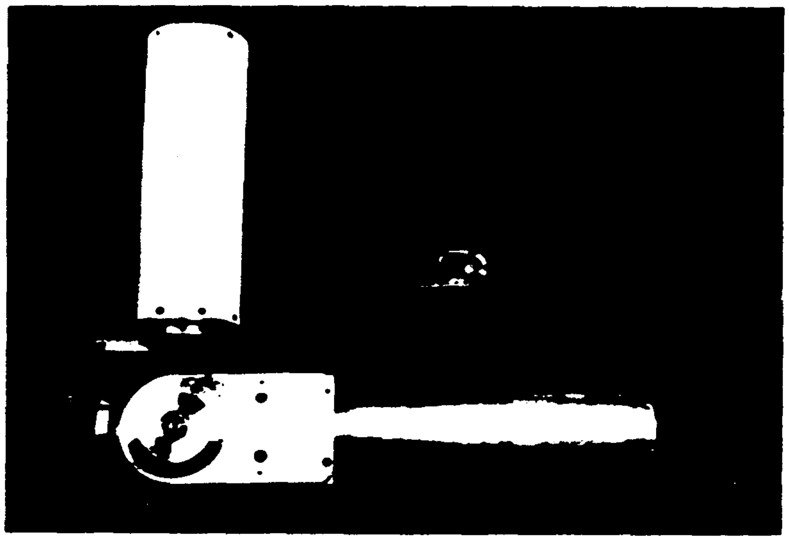


Figure 8. Dual axis manipulator element implementation.

7. Summary

A discussion of servomechanism control techniques for telerobotic systems has been presented and a brief review of two control approaches was given. The attributes of centralized and decentralized control were discussed from a perspective of applied experience. The reasoning behind the dual axis controller was developed and the general architecture reviewed. The actual implementation hardware for the dual axis controller has been accomplished and software for servocontrol has been generated. The dual axis manipulator element capable of redundant, replicated kinematic construction was introduced, and a functional prototype was described. These efforts represent a significant effort to move telerobotics ahead to meet future challenges for DOE, NASA, and the DOD. The approaches are novel and take a large step toward a reliable, inexpensive, and adaptable mechanical and electrical manipulation system. Incremental improvements were avoided, and completely new ways of accomplishing the objectives of remote manipulation were sought. The admirable performance of the prototype controls and mechanisms holds promise for future systems implementation.

S25-37
78
185107
G69 [illegible]
G7180425

High Performance Architecture for Robot Control

E. Byler
Grumman Space Systems
Bethpage, NY 11714

J. Peterson
Grumman Data Systems
Woodbury, NY 11797

1. Abstract

Are [illegible]

~~This paper discusses~~ ~~practical~~ aspects of the design and implementation of a modular, high performance, parallel computer control system for telerobots. Topics of consideration include system architecture, operator interface, and control execution. In a laboratory environment, a telerobotics test control configuration is used to obtain measurements on communications and control loop timing for use in an effective full scale operational system design. The feasibility of the selected architectural approach has been successfully demonstrated. The modularity of the software and hardware enables ease of transport for use in the operational system. A ~~major portion of this paper is devoted to the~~ distributed partioning of the control algorithms and the performance measurements acquired during control system implementation. are discussed

2. Introduction

Telerobots are manipulators designed to be controlled both directly by a computer and remotely by a human. This combination method, called supervisory control, allows the operator to apply his intelligence to the task without having to maintain continuous control. Computational requirements for robot control either limit the complexity of the control system, or require prohibitively large computers to obtain the required cycle rate. Remote operation and human control requirements create additional environmental constraints. The design of a hierarchical, parallel computer system is described. It provides the required speed within the prescribed design limits at a minimum weight and size. A prototype system was built and measurements were made to verify several performance issues.

3. System Description

The space-based telerobot for which this computer system was designed (Grumman [1]) has several aspects that drive the design of the computer system.

The first major design requirement is imposed by the robot itself (Fig. 1). There are three arms, each with seven joints and an end-effector, a torso, a vision system, and tools and test equipment. All this equipment is under computer control. Each arm must operate under control of several different types of control algorithms to provide flexibility to enhance operator productivity. Two arms must be capable of coordinated action. The manipulation requirements impose definite computation requirements.

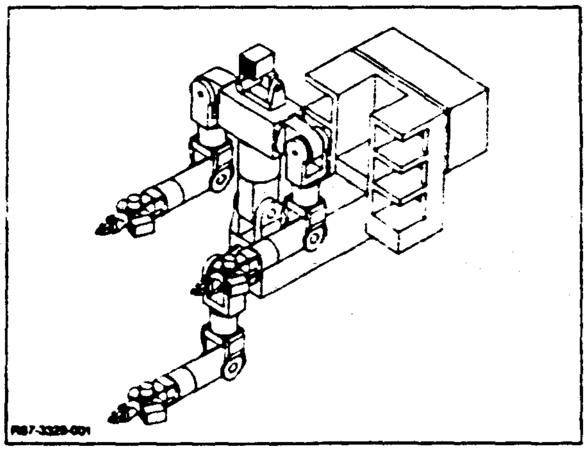


Fig. 1 Telerobot

A second major design driver results from the control devices used to operate in the various control modes. In order to maximize productivity, a variety of control inputs and information displays are needed. Input control devices include a replica master, 6DOF stick controllers, voice control, a lightpen, a keyboard, and a touchbezel. Output devices include graphics displays, monitors, and force reflection (Fig. 2). Each of these require some data processing.

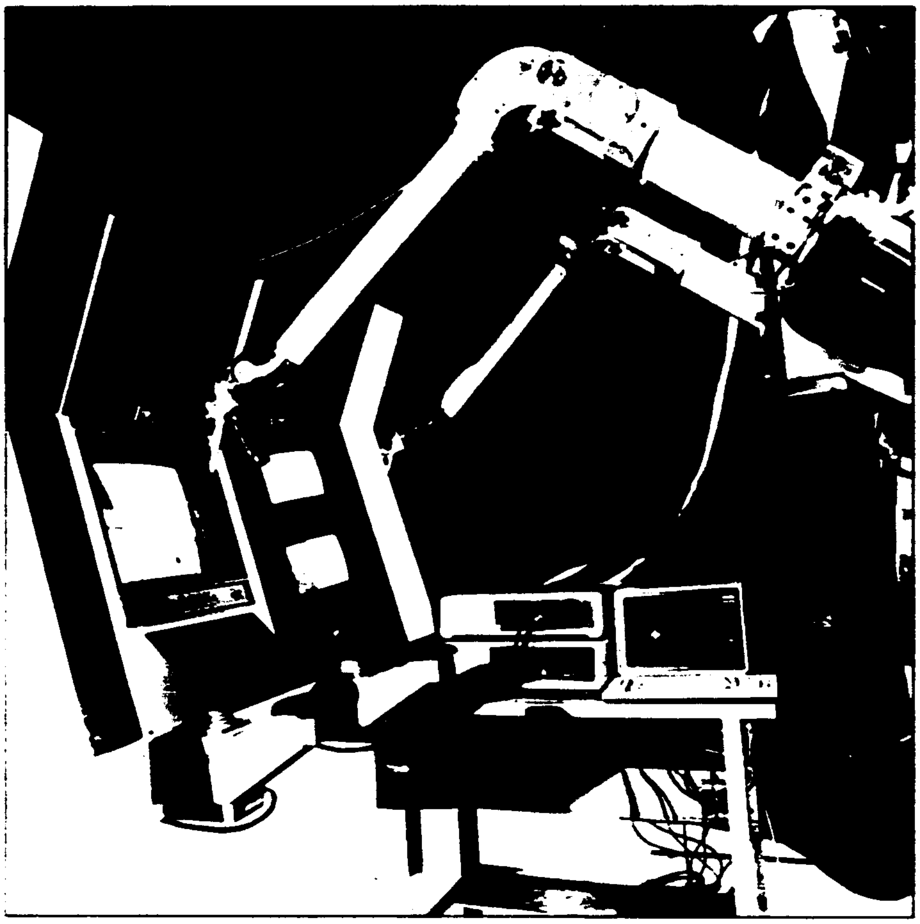


Fig. 2 Telerobot Control Station Testbed

The third major design driver is that the robot is remote from the control center. This creates a requirement for a communication link. Any communication link imposes restrictions and trade-offs on total data bandwidth.

A final design driver is the set of different control algorithms required for human control and robotic control. These impose requirements for different data and control paths. Different control algorithms have different loop rates for stability, within which all computations must be performed.

4. System Specification

The telerobot is required to run three different control algorithms: Bi-lateral Force Reflection (BFR), Resolved Rate (RR), and Active Force Control (AFC) (Fig. 3). Both BFR and RR must have human control. AFC uses computer control only. BFR has large communication requirements in that a complete set of sensor signals must be passed both from the masters to the slaves, and from the slaves to the masters. Good performance is obtained with updates of about 66 Hz. AFC has a required update for its command vector of about 400 Hz in order to maintain stability when the manipulator is in contact with elements of its environment.

Distributed processing was considered to overcome the computational bottleneck inherent in robot control. First, each algorithm was broken into a block diagram by considering which elements were on different levels, which could be run asynchronously, and which were computationally intensive, or could be split into separate variable types (Fig. 4). Then, the different algorithms were compared to determine which elements were common. Data requirements, computation times, and output for each element were tabulated.

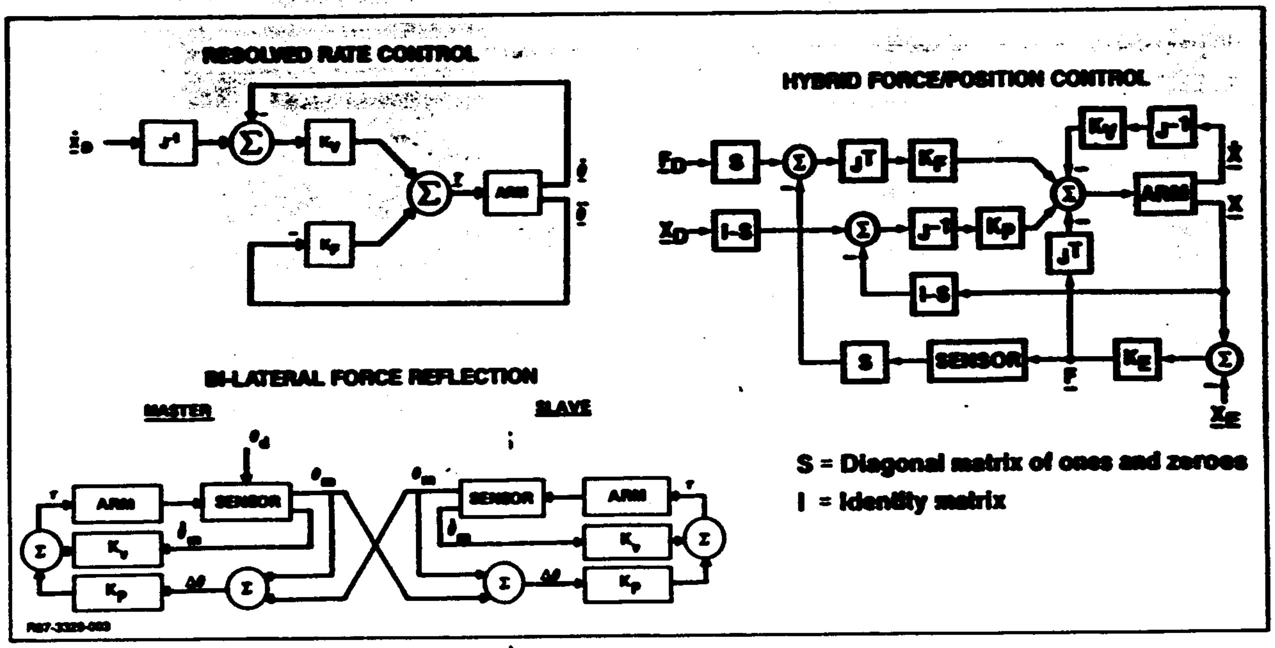


Fig. 3 Telerobotic Modes of Control

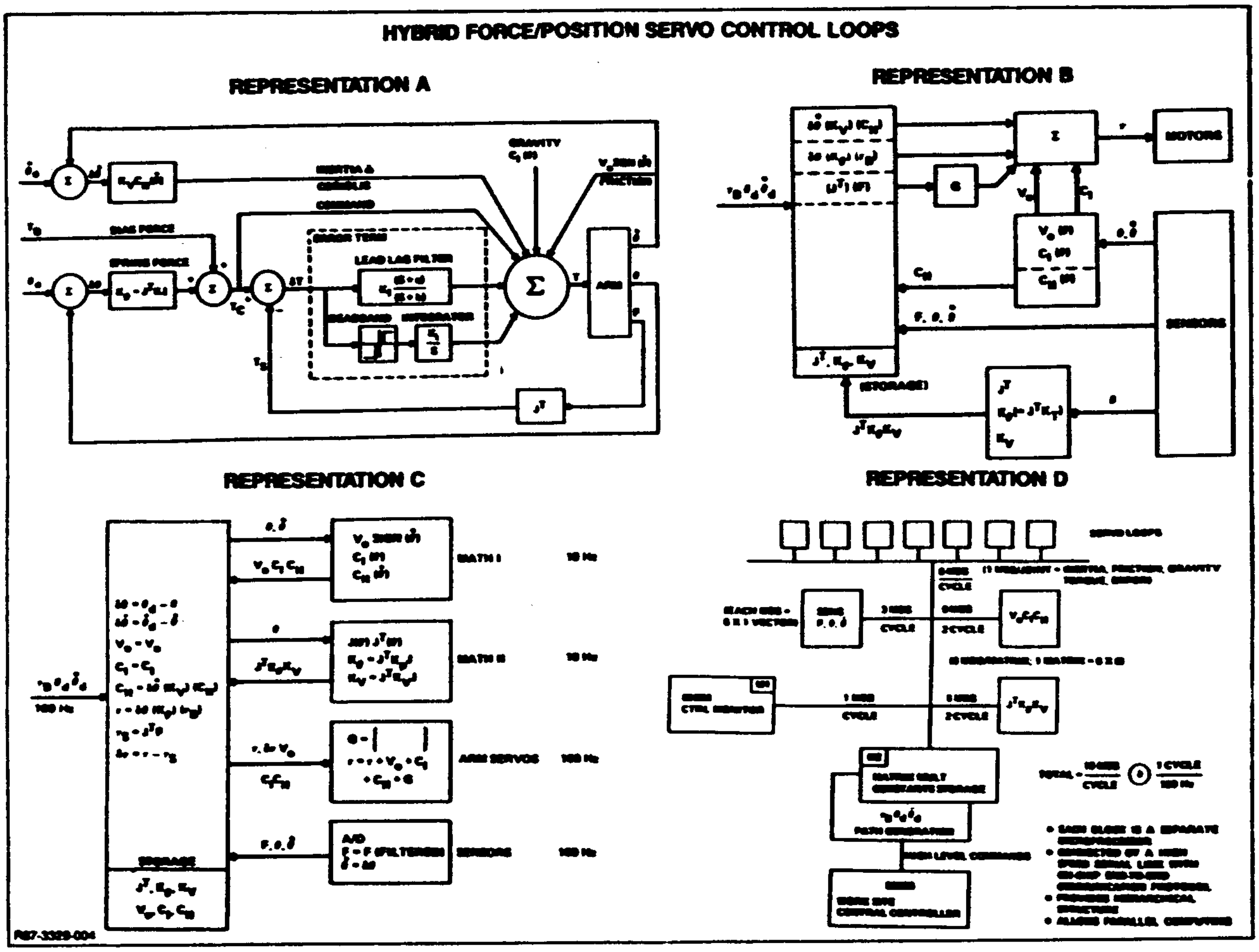


Fig. 4 Evolution of Force Control Algorithm Partitionment

Finally, these separated elements were considered in conjunction with the information flow and general system placement requirements to arrive at a coherent computer architecture (Fig. 5). Processor/algorithm granularity was driven by available commercial products and by accuracy requirements.

This architecture matches the NASA/NBS standard reference model telerobot control system architecture (NBS [2]) for levels one, two, and part of three. Servo control and parameter calculation take advantage of parallel processing techniques. Task planning and path generation are split into hierarchical processes.

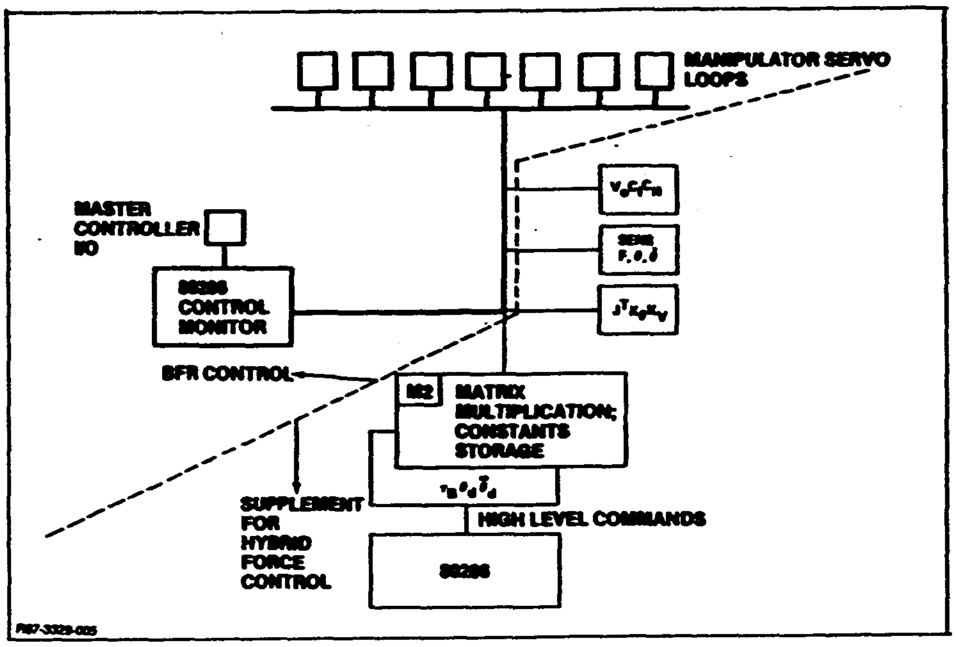


Fig. 5 Selected Computer Architecture

5. Performance Issues

Several issues were examined to verify the performance of this parallel computing system. First, central to the distributed control architecture is the correct partitionment of alogrithms among the parallel processors to properly distribute the computing loads. Second, communications bandwidth requirements of algorithm components must be considered to partition the algorithms effectively without creating bottlenecks. Third, CPU consumption by algorithm components must be analyzed. Finally, use of operating system and communication system services must also be examined to eliminate unnecessary overhead.

6. Implementation

The development environment for the digital robot control makes use of Intel microprocessor and microcontroller chip, board, and bus technology. The 80286 based System 286/380 is used to run high-level robot path planning code. The system is presently used as both a software development and target machine for implementing and running controlsite software. The 8085-based Personal Development System (PDS) is used for developing 8044 microcontroller code, and contains the necessary text editor, cross assembler, linkage editor, and monitor tools for firmware development. An EMV44 emulator provides a window to the 8044 microcontroller for debug.

The Distributed Control Module (DCM) package is the set of hierarchical, microcontroller nodes and complimentary communications software that forms the backbone of the prototype robot control system. It includes the iRCB 44/10 Remote Controller Board, the iSBX 344 Multimodule Board, the DCM Controller, and the BITBUS Interconnect. A distributed control node may be either an iRCB 44/10 or an iSBX 344 board. The DCM controller is actually an 8044 microcontroller chip with built-in SDLC communications firmware and is part of all boards. The BITBUS Interconnect is the high-speed serial communications link for the network of nodes.

The development configuration was designed specifically to be able to measure system performance. It groups six microcontrollers into one distributed network of nodes, linked together by a high speed serial communications bus (Fig. 6). The communications link implements a SDLC master slave protocol on chip and performs all message handling and error checking automatically without help from the CPU. Mastership of the network is claimed by either the Personal Development System 344 microcontroller node, or the System 286 344 node and is a function of the particular control algorithm currently executing. The 44/10 microcontroller nodes plug into a common cardframe which supplies the required power and communications wiring (Fig 7).

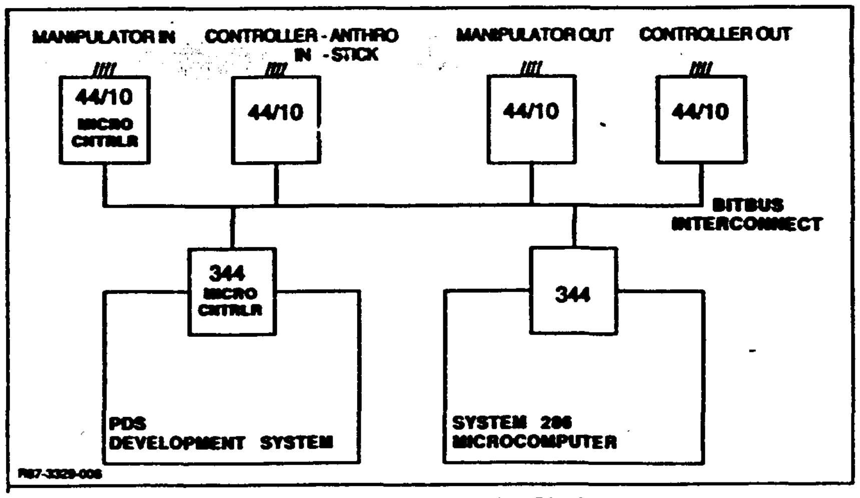


Fig. 6 Development Configuration Block Diagram

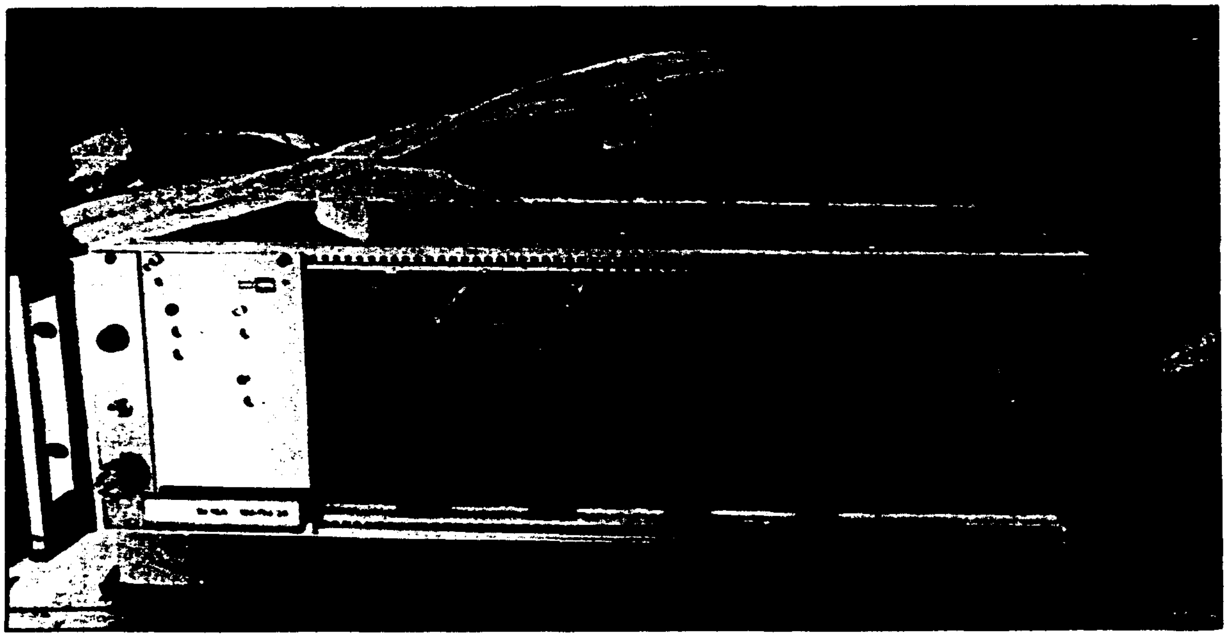


Fig. 7 Prototype Parallel Computer System for Telerobot Control

7. Measurements

Measuring the system performance is accomplished by attaching an oscilloscope to the pins on one of the dedicated parallel I/O ports (Fig. 8). The separate signal lines are toggled by the development firmware to provide a mechanism for viewing the algorithms as they run (Fig. 9). Each bit of the 8-bit port is dedicated to provide the timing for different components of the algorithms running on each processor.

Measurements were taken in three areas to examine system performance and algorithm partitionment. The three areas examined were communications, CPU utilization, and operating system.

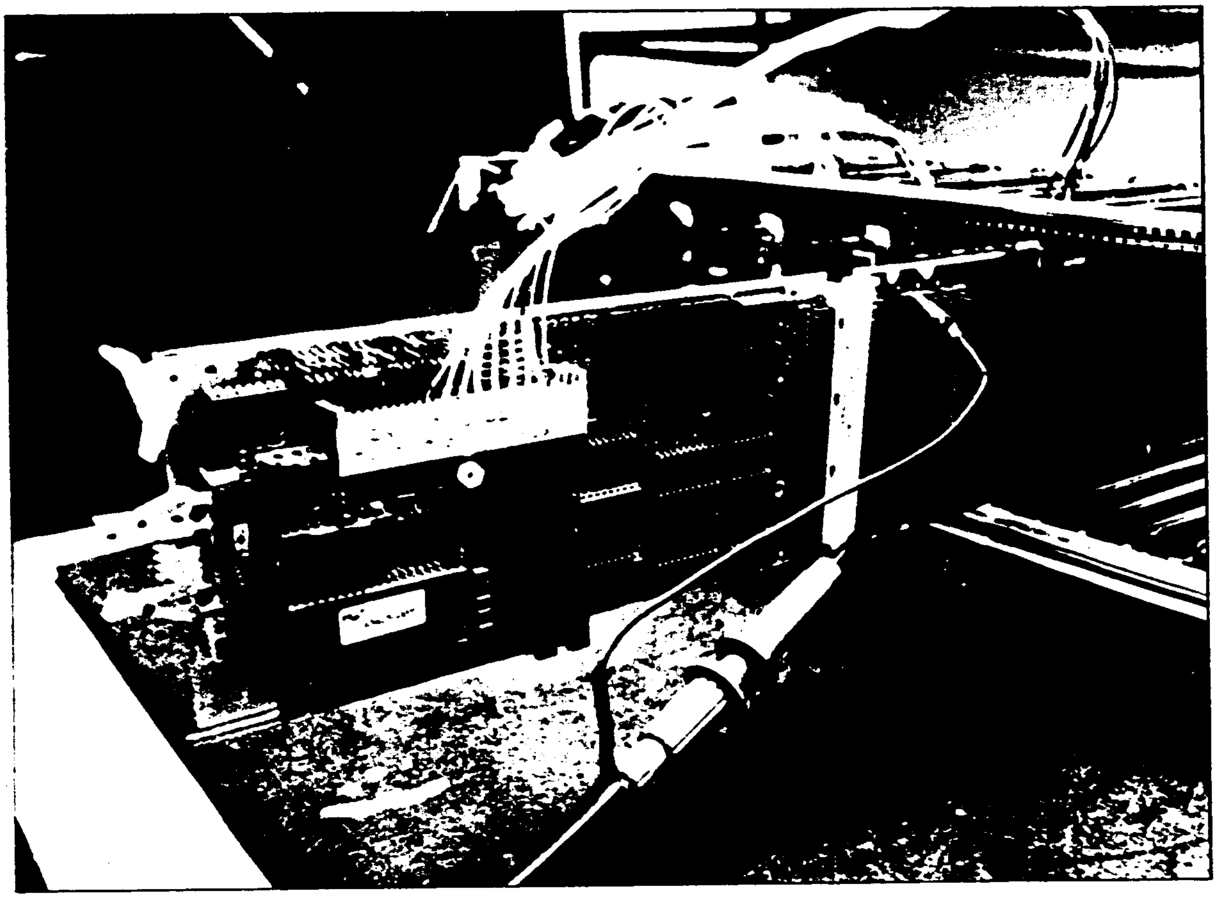


Fig. 8 Oscilloscope Attachment for Performance Measurements

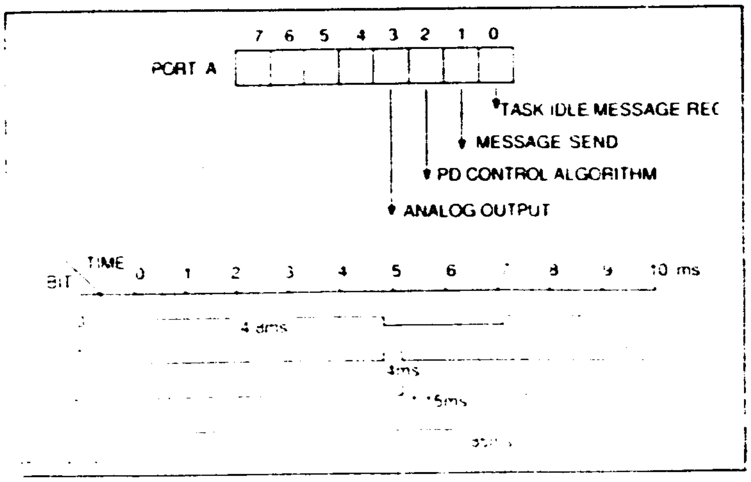


Fig. 9 Measurement Process

A variety of communications strategies have been implemented and measured for their performance. The effects of preloading the Serial Interface Unit (SIU) controller and the effects of serial and parallel message passing techniques were measured and compared. In addition, the effects of variations in asynchronous and synchronous communications bandwidth on overall timing were examined.

The preloading technique involves setting up the SIU of a slave node for a reply prior to receiving the order from the master. A time savings equivalent to the required computation time of the slave node for one control loop cycle, is observed when the SIU is preloaded.

Serial and parallel message passing techniques differ in the sequence by which the master node communicates with the slave nodes. Serial message passing techniques cause the master node to wait for replies after every message sent, which may cause other nodes to wait for their turn if the reply requires some computation. Parallel message passing techniques allow the master to have several outstanding messages, allowing slave nodes more chance to process in parallel and reduce control loop timing (Fig. 10).

The communications bandwidth has significant effect on overall timing. With the nodes jumpered for synchronous serial communications on the BITBUS interconnect, a 2.4 Mbps transmission rate is achieved as compared with the slower asynchronous rate (Fig. 11). The time savings is readily observed when the master uses serial message passing.

	SERIAL	PARALLEL
PRELOADED 4/4 NODES	7.2ms	7.2ms
HALF PRELOADED 2/4 NODES	12.0ms	8.8ms
NO PRELOADED 0/4 NODES R87-3329-010	14.2ms	9.8ms

Fig. 10 Communications Schemes

SYNCHRONOUS 2.4M CRYSTAL OSC. ALL BOARDS	5.0ms
SYNCHRONOUS 2.4M CRYSTAL OSC. MASTER ONLY	6.1ms
SYNCHRONOUS .92M ALE CLOCK	7.2ms
ASYNCHRONOUS 375K R87-3329-011	9.2ms

Fig. 11 Clock Rates

CPU utilization measurements verify how effectively the algorithms are partitioned. For example, the slave input nodes measured a 60% idle time. This reflects a capacity to handle more of the computational burden of the system. Computation measurements for different parts of the algorithm equations enable efficient redistribution of portions of the algorithm to fine tune the design. For example, the PD loop timing was approximately 0.8 ms. This measurement helped to redistribute the gain computation of the servo loop to a more appropriate node to optimize the cycle time of the loop.

Operating system calls have a significant effect on algorithm timing. For example, the preloading of a message for transmission by a slave node takes approximately 0.4 ms. Internal RAM buffer space (system pool space) must be allocated for a message dynamically, via a system call, if it is to be sent by a task on one node to a task on a different node. Since there is a limited amount of buffer space (approx. 4 20-byte buffers) available for a message send, buffer space will be unavailable until the buffers associated with previous sends are released after successful transmission. A method of reducing the allocation calls at the master node is to use the buffer allocated to hold the previously received reply message for the next order message since this buffer is automatically allocated by the system.

Use of the standard RAC communication services may also impose penalties. Passing a message through the RAC services automatically causes task context switching. This is important if the node is on the critical path either in terms of calculations or buffer allocations. The time penalties can be avoided by taking direct control of the receive buffers. The applications firmware must then provide the communication error handling capability.

8. Conclusions

Distributed processing of control algorithms is a feasible solution to improve robot control computatio speed. Measurement of system elements is important in order to optimize system CPU usage and avoid bottlenecks. Application programs must be careful to avoid unnecessary operating system use.

This architecture, as implemented, covers the National Bureau of Standards Hierarchical Control Specification levels one, two, and part of three. It is independent of path generation and task de.cription techniques and runs bilateral force reflection as easily as resolved rate. Hardware weight and power consumption are very low and are appropriate for space flight hardware.

9. References

[1] "Telepresence Work System, System Definition Study," Grumman Aerospace Corporation, October 1985.

[2] "NASA/NBS Standard Reference Model Telerobot Control System Architecture Preliminary Draft," National Bureau of Standards, September 1986.

S26-61
98
1ﾠ[illegible]

Software and Hardware for Intelligent Robots

G.N. Saridis
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

K.P. Valavanis
Northeastern University
Boston, MA 02115

R093523[illegible]
N436087

1. ABSTRACT

Various architectures and their respective software for Hierarchically Intelligent Robots are discussed in this paper. They conform to the Principle of Increasing Precision with Decreasing Intelligence by following a three-level structure. The architecture of the organization and coordination levels is presented here and their algorithms are outlined.

2. INTRODUCTION

Intelligent Robots are a special type of Intelligent Machines [5]. They may be driven by controls with special characteristics described by the methods of Hierarchically Intelligent Control Systems [3].

Hierarchically Intelligent Control is based on the Principle of Increasing Intelligence with Decreasing Precision. Intelligent Robots (and thus Intelligent Machines) may be modeled based on the constraints imposed by the Theory of Intelligent Controls. They are considered to be composed of three interactive levels, organization, coordination and execution. They utilize feedback mechanisms from the hardware processes of the execution level to the organizer selectively, by aggregation of the information at every level (Figure 1).

A three interactive level probabilistic model has already been defined [4,5] for Intelligent Robots. The organization level is modeled after a Knowledge Based System; it performs general knowledge processing tasks with little or no precision. The coordination level is composed of a specific number of coordinators, each performing its own pre-specified functions; it performs specific knowledge processing tasks. The execution level is composed of specific execution devices (hardware) associated with each coordinator. The probabilistic model is obtained by defining the various operations associated with each level of Intelligent Robots in a mathematical way and then assigning a probabilistic structure to organize the appropriate tasks for execution.

A simple but complete architectural model that can accommodate fast and reliable operation for the levels individual functions has also been derived [5]. Each level has a functional task to perform. The functional task of each level is performed in the best possible way based on accumulated information and related feedback from the lower levels. Upon completion of this functional task, a command is issued to the immediate lower level, and the functional task of the lower level is accomplished. With this structure, each level evaluates and controls the performance of the immediate lower one. The architectural model associated with top-down hierarchical knowledge (information) processing is suitable for the decision phase of Intelligent Robots. The decision phase involves formulating complete and compatible plans, deciding which one is the best to execute the requested job and how to execute it. Therefore, it may accept special simple architectures specifically designed to implement the Hierarchically Intelligent Control Algorithms.

Specific hardware units must be built within each of the higher two levels for the upgrade phase of Intelligent Robots. This phase involves the upgrade of the individual and accrued costs and probabilities associated with a particular plan. Upgrade follows plan execution (completion of the decision phase) and is performed in a bottom-up way: costs and probabilities associated with the lower level are upgraded first followed by the ones with the higher level.

The architectural model (of the decision and upgrade phases) is appropriate for both modes of operation of an Intelligent Robot, the training mode and the well-trained mode. The training mode of operation is defined as the mode in which the Intelligent Robot "explores" and "learns" its capabilities and alternative actions given the user(s) requested job(s). While in this mode, the probabilities are significantly modifiable by the learning algorithms rewarding certain plans and penalizing others [3]. The well-trained mode of operation follows the training mode and is defined as the mode in which the Intelligent Robot is well trained and knows exactly the sequence of actions necessary to complete a user requested job [6]. The well-trained mode exists only when there are not situations which might include unpredictable events.

This paper describes both models for the higher two levels of Intelligent Robots. Section three of the paper presents the pertinent definitions necessary for the derivation of the models. Sections Four and Five present the models for the decision-phase of the organization and coordination levels, while section Six explains the architectural model for the upgrade phase of Intelligent Robots. Section Seven presents the advantages of the models.

3. DEFINITIONS

For every Intelligent Robotic System define [4,5]:

1. The set of user commands $C=\{c_1,c_2,\dots,c_M;$ M fixed and finite$\}$ with associated probabilities $p(c_n)$, $n=1,2,..M$, sent to the Intelligent Robot via any remote or not channel.

2. The set of classified compiled input commands $U=\{u_1, u_2,\dots,u_M;$ M fixed and finite$\}$ with associated probabilities $p(u_j/c_n)$, $j=1,2,..M$, which are the inputs to the organization level of the system.

3. The task domain of the Intelligent Robot with the set of independent but not mutually exclusive disjoint sub-sets of non-repetitive and repetitive primitive events $E = \{E_{nr},E_r\} = \{e_1,e_2,\dots e_{N-L},e_{N-1+1},\dots e_N;$ N fixed and finite$\}$.

4. The binary valued random variable x_i associated with each e_i indicating if e_i is active $(x_i=1)$ or inactive $(x_i=0)$ given a u_j, with corresponding probabilities $p(x_i=1/u_j)$ and $p(x_i=0/u_j)$ respectively.

5. The set of the (2^N-1) activities which are groups of primitive events concatenated together to define a complex task. They are represented by a string of binary random variables $X_{jm}=(x_1,x_2,\dots x_n)_m$; $m=1,2,\dots,(2^N-1)$, which indicates which e_i's are active or inactive within an activity with a probability $P(X_{jm}/u_j)$.

6. The set of compatible ordered activities obtained by ordering the primitive events within each activity and represented by a string of compatible ordered binary random variables Y_{jmr}, where r denotes the rth ordered activity obtained from X_{jm}, with a probability $P(Y_{jmr}/u_j)$.

7. The set of compatible augmented ordered activities obtained by inserting repetitive primitive events within appropriate positions of each Y_{jmr} and represented by $Y_{jmr}(a_s)$, where a_s denotes the sth augmented activity obtained from Y_{jmr} with a probability $P(Y_{jmr}(a_s)/Y_{jmr})$.

8. The set of mask matrices M_{jmr} with associated probabilities $p(M_{jmr}/u_j)$ used to obtain the compatible ordered activities (Y_{jmr}) from the activities (X_{jm}).

9. The set of augmented mask matrices $M_{jmr}(a_s)$ with associated probabilities $p(M_{jmr}(a_s)/Y_{jmr})$ used to obtain the compatible augmented ordered activities from each Y_{jmr}.

When a user command C_n with a probability $p(C_n)$ is sent to the Intelligent Robot it is received and classified by a classifier to yield the classified command u_j; with a probability $p(u_j;/(n)$, which is the input to the organization level.

4. MODEL FOR THE ORGANIZATION LEVEL

The organization level performs five sequential functions as shown in Figure 2: machine reasoning, planning, decision making, feedback and long-term memory exchange. The last two are performed during the upgrade phase. The organizer formulates complete and compatible plans and decides about the best possible plan to execute the user requested job. This is done by associating u_j with a set of pertinent activities X_{jm} with corresponding probabilities $P(X_{jm}/u_j)$ (reasoning), and by organizing the activities in such a way (planning) to yield complete and compatible plans: The compatible ordered activities Y_{jmr} associated probabilities are: $P(Y_{jmr}/u_j= p(M_{jmr}/u_j)*P(X_{jm}/u_j)$. The compatible augmented ordered activities $Y_{jmr}(a_s)$ are obtained by inserting repetitive primitive events in appropriate positions within each Y_{jmr} and their corresponding probabilities are: $P(Y_{jmr}(a_s)/Y_{jmr})=p(M_{jmr}(a_s)/Y_{jmr})*P(Y_{jmr}/u_j)$. Every incompatible activity and incomplete plan is rejected [4]. The most probable complete and compatible plan Y^F is the final plan that is transferred to the coordination level.

After the completion of the requested job upgrade of the probabilities and stored information follows as it will be explained in section Six. Figure 3 shows the analytical probabilities illustration of the complete organization level function.

The architectural model for the machine reasoning function is shown in Figure 4. The input to the model is the (classified) compiled input command u_j and the corresponding output, Z_j^R, the set of the (maximum) (2^N-1) pertinent activities with the corresponding addresses which store the activity probabilities $P(X_{jm}/u_j)$. The reasoning block RB contains the (2^N-1) strings of binary valued random variables stored at a particular order, which represent the activities associated with any compiled input command. The corresponding addresses which store the probabilities related to these activities are transferred from the memory D^R, a part of the long-term memory of the organizer. The memory D^R consists of M different memory blocks D_1, D_2,...,D_M. One memory block, D_j, is associated with each compiled input command u_j. Once the compiled input command u_j has been recognized, realization of the switch s_1 activates (enables) the memory block D_j. Transfer of the data (addresses) contained in D_j is accomplished via the realization of the switch s_2. The switches are coupled with each other. The contents of the RB are transferred to the right most positions of the PRB (Probabilistic reasoning block) while the corresponding addresses which store the pertinent probabilities occupy the left most positions. The information stored in D^R is not modifiable by, or during the machine reasoning function. Therefore, D^R is considered as permanent memory whose values do not change during an interaction cycle, i.e., from the time the user has requested a job until its actual execution. The values of the probability distribution functions associated with the set of pertinent activities are upgraded only after the completion of the requested job through a specific hardware unit described in section Six.

The model for the machine planning function is more complicated. It is shown in Figure 5. The input to the planning model is Z_j^R. The output from the machine planning model is, Z_j^P, the set of all complete and compatible plans capable to the set of all compatible augmented ordered strings of primitive events $Y_{jmr}(a_s)$ formulated during the machine planning function. All compatible ordered activities are stored in the first planning box, PB1. Every compatible augmented ordered activity is stored in the second planning box, PB2. The two compatibility tests (one to obtain compatible ordered activities and one to obtain compatible augmented ordered activities) are performed within the boxes CPT1 and CPT2. The specific hardware unit for both compatibility tests is shown in Figure 5(b). The memory D_T, also a part of the long-term memory of the organizer, is divided into four sub-blocks, D_{T1}, D_{T2}, D_{T3} and D_{T4}: D_{T1} contains all incompatible pairs of the form (non-repetitive primitive, non-repetitive primitive), D_{T2} of the form (non-repetitive primitive, repetitive primitive), D_{T3} of the form (repetitive primitive, non-repetitive primitive) and D_{T4} of the form (repetitive primitive, repetitive primitive). Each ordered activity (augmented ordered activity) is transferred to the register R(R'). At the beginning of the compatibility test the pointer PT (PT') is at the left most position of R (R'). It transfers every pair of primitives to the two-position register R1 (R1'), which scans D_{T1} (D_{T1}, D_{T2}, D_{T3} and D_{T4}), to check if the stored pair is incompatible. If yes, the whole ordered activity (augmented ordered activity) is rejected. If not, the left most primitive event in R1 (R1') is discarded, the right most moves one position to the left and a new primitive event occupies the empty position. The test continues until the pointer has reached the last two primitive events of the activity. Therefore, the number of compatible activities is significantly reduced. The corresponding addresses of the compatible ordered activities with the mask probabilities $p(M_{jmr}/u_j)$ are transferred from the memory D_1^P via the realization of the coupled switches s_3 and s_4. The switch s_3 activates the corresponding memory block D_{jj} (once Z_j^R has been recognized), while the switch s_4 permits the transfer of data. The box PB1 now contains the compatible ordered activities with their corresponding addresses containing the probabilities $P(Y_{jmr}/u_j)$. The insertion of the repetitive primitive events is performed in the box INS while the compatible augmented ordered activities are stored in the second planning box, PB2. Their corresponding memory addresses with the augmented masks probabilities $p(M_{jmr}(a_s)/Y_{jmr})$ are transferred from D_2^P. Activation and transfer of data from the appropriate memory block D_{jjj} is accomplished via the realization of the two coupled switches s_5 and s_6 in a way similar to the ones described before. The information stored in D_1^P and D_2^P is not modifiable by, or during the machine planning function. This information is considered permanent within an iteration cycle, too. The output from the machine planning function Z_j^P is the set of all complete and compatible ordered activities that may execute the requested job. The completeness test is performed within LCMT. This test accepts every meaningful syntactically correct plan [5].

The model for the machine decision making function is shown in Figure 6. All complete and compatible plans Z_j^P are stored in MDMB (machine decision making box) and checked by pairs to find the most probable one. The most probable plan is stored in RR. If during the check, a complete plan with higher probability than the one already stored in RR is found, it is transferred to RR while the already stored one is discarded. Once the check is over, the contents of RR indicate the most probable complete and compatible plan to execute the requested job.

Every complete and compatible plan is also stored in a particular part of the long-term memory of the organizer, D_{ss}. This memory contains every complete and compatible plan related to every compiled input command as shown in Figure 6. The idea of this particular memory is very important: It represents the situation of a well-trained Intelligent Robot (under the assumption that no unpredictable events may occur). An Intelligent Robot which has reached this mode of operation associates immediately after the recognition of the compiled input command the most probable of the plans (if more than one available) stored in D_{ss}, without going through every single individual function.

5. MODEL FOR THE COORDINATION LEVEL

The coordination level is composed of a specific number of coordinators as shown in Figure 7. Its purposes to coordinate the individual tasks, select the appropriate performance requirements for the execution level, identify space limitations and assign penalty functions, optimize the performance of the overall plan and use learning for performance improvement. It issues specific commands to the execution level which is composed of a number of execution devices associated with the coordinators at the coordination level. The interaction between the three levels is represented in terms of on-line (real-time) and off-line feedback information. The on-line feedback information is communicated from the execution to the coordination level during the execution of the requested job and is used to evaluate and upgrade the information stored in the long-term memory of the organization level (functions of feedback and long-term memory exchange).

A block diagram of the coordination level architectural model is shown in Figure 8. The complete and compatible plan Y^F stored in RR is transferred to a buffer B. The complete and compatible most probable (final) plan, contains actually a sequence of addresses. Each address corresponds to a primitive event (repetitive or non-repetitive). When the final plan is loaded into the buffer the BEVENT line is activated and the pointer PT1 scans the buffer once from left to right. With the aid of the qualifier QLFR it is known how many times each coordinator will be accessed, as well as when it will be accessed. Thus, the association execution devices are activated when their coordinator is accessed via the qualifier QLFR. After the first scanning, the pointer PT1 returns to the left most position. The BEVENT line is deactivated and the qualifier is used to activate another coordinator as the pointer moves from left to right. Upon completion of its operation a BENT signal is sent to the qualifier and the buffer and the pointer moves one position to the right. When a specific coordinator completes its specific functions and the on-line feedback information has (already) been communicated to it, this information is stored in the short-term memory of the coordination level. It may be used by other coordinators to complete their functions (during the execution of a requested job) and/or to calculate the overall accrued cost associated with the coordination level, that will be communicated to the organizer after the execution of the job. Thus, the different coordinators do not communicate directly with each other. But each coordinator has access to the short-term memory for storage and retrieval of data.

The coordination level does not have a particular memory equivalent to the D_{ss} memory of the organization level. This is because the workspace environment of the coordination level is dynamic: Given a final plan Y^F, the formulation of the actual control problem regarding the way of its execution depends not only on previous experience but also on the current configuration of the workspace environment. For example, previously chosen trajectories for the different motions of the manipulator(s) should be modified by the presence of more or less obstacles and/or of additional objects.

The execution level performs the commands issued by the coordination level. Each control problem is analyzed in the light of its special requirements. Therefore, there is not one general architectural model that can include every possible operation performed at the execution level.

But although this is the case, the execution level consists of a number of devices each associated with a specific coordinator and vice versa. Each device is accessed via a command issued by its coordinator. Hence, the hierarchical structure is preserved.

6. MODELS FOR THE UPGRADE PHASE

The architectural model for the upgrade phase of Intelligent Robots includes specific hardware units that must be built within each of the higher two levels to account for both types of feedback information, on-line and off-line feedback.

The off-line feedback mechanism (from the coordination to the organization level) is activated after the execution of the requested job and is used to upgrade the probabilities distribution functions of the sets of pertinent, ordered and augmented ordered activities associated with a compiled input command.

The probabilities upgrade algorithm is given by the equation:

$$p(t+1/u_j) = p(t/u_j)+\beta_{t+1}[\xi-p(t/u_j)] \qquad (1)$$

$$\xi = \begin{cases} 1; & J = J_{min} \\ 0 & \text{otherwise} \end{cases}$$

where $p(t+1/u_j)$ denotes the corresponding probability at iteration cycle (t+1), J the actual cost of executionof the job, J_{min} the minimum cost of execution and β a coefficient that obeys Dvoketsky's conditions [3]. Each upgrade requires one addition, one subtraction and one multiplication. During this process there are no data dependencies between probabilities. A special purpose hardware unit is the one shown in Figure 9. It consists of two adders, one multiplicator unit and three registers [7]. The probability value is fetched from the memory, passes through the three arithmetic operations and is returned to the memory, overwriting the previous value. This function is performed after the execution of the requested job and calculation of the overall accrued cost associated with the coordination level.

Five such hardware units must be built in the organization level to upgrade all pertinent pdfs as shown in Figure 10.

The on-line feedback mechanism (from the execution to the coordination level) is activated during the execution of the requested job. A block diagram of this mechanism is shown in Figure 11. Solid lines represent the on-line feedback information from the execution devices (at the execution level) to the different coordinators. Individual and accrued costs are calculated within each coordinator and are transferred to the short-term memory of the coordination level where the overall accrued cost associated with the coordination level is calculated and communicated back to the organizer after the execution of the job. The dotted lines illustrate how the information from the short-term memory is used by the different coordinators and their execution devices for the completion of the job.

7. REMARKS

The models presented here have two major advantages:

1. They are applicable to any Intelligent Machine operating under the constraints of Hierarchically Intelligent Control Systems, and,
2. The same architecture may be used if one uses a linguistic approach to design Intelligent Machines. Based on the models described, one may derive the context-free grammars that satisfy the same performance requirements and criteria.

The architectural model of the organization level may be also used if one wishes to modify the machine reasoning function in the algorithm of the organization level as follows: Given a compiled input command u_j, associated with it only a subset of the (2^N-1) pertinent activities which contains only those activities with corresponding probabilities greater than a prespecified $P_{min}(u_j)$. Therefore, the machine planning and decision making functions are limited to the formulation of complete and compatible plans that originate from this subset of the (2^N-1) possible pertinent activities. This approach requires decision making from the beginning: the machine reasoning function because it eliminates every activity with corresponding probability less than $P_{min}(u_j)$.

The disadvantage of the architectural model is that it does not consider the possibility of unpredictable events during plan(s) execution.

REFERENCES

[1] Saridis, G. N., Valavanis, K. P., "Information Theoretic Approach for Knowledge Engineering and Intelligent Machines", Proceedings of the ACC Conference, Boston, MA, June 1985.

[2] Valavanis, K. P., Saridis, G. N., "Analytical Design of Intelligent Machines". Proceedings of the SYROCO Conference, Spain, November 6-8, 1985.

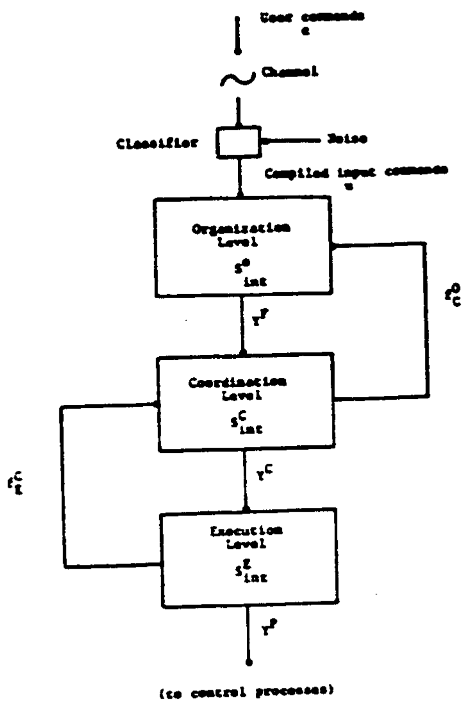
[3] Saridis, G. N., Self-Organizing Control of Stochastic Systems, NY, Marcel Dekker, 1977.

[4] Saridis, G. N., Valavanis, K. P., "A Mathematical Model for the Design of Intelligent Machines", submitted for publication.

[5] Valavanis, K. P., "A Mathematical Formulation for the Analytical Design of Intelligent Machines", Ph.D. Thesis, RAL #85, 1986.

[6] Boettcher, K. L., "An Information Theoretic Model of the Decision Maker", M. S. Thesis, LIDS-TH-1096, MIT, Cambridge, MA, June 1981.

[7] Graham, J. H., Saridis, G. N., "Linguistic Methods for Hierarchically Intelligent Control", School of Electrical Engineering, Purdue University, TR-EE 80-34, October 1980.



User commands c
Channel
Classifier
Noise
Compiled input commands u
Organization Level S^{o}_{int}
Y^{F}
f^{O}_{C}
Coordination Level S^{C}_{int}
Y^{C}
f^{C}_{E}
Execution Level S^{E}_{int}
Y^{F}
(to control processes)

Figure 1. Block diagram of an Intelligent Machine

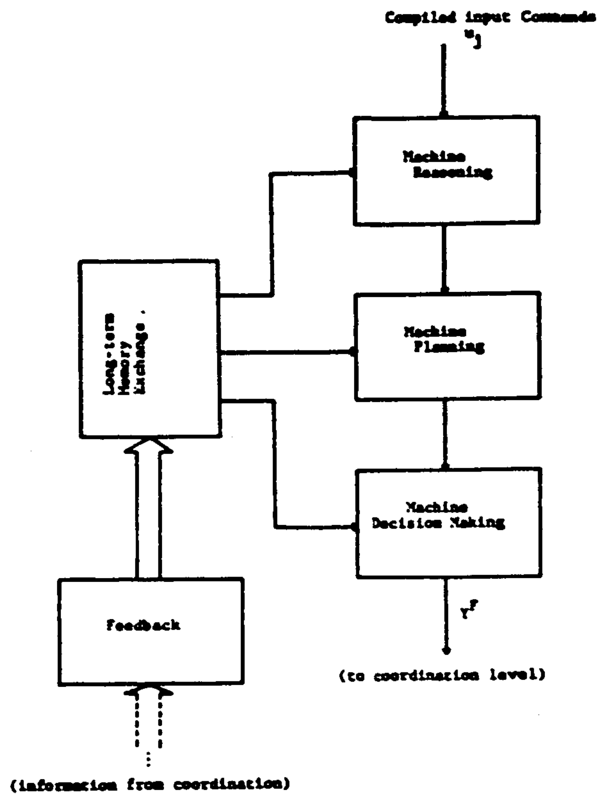


Figure 2. Block diagram of the organization level

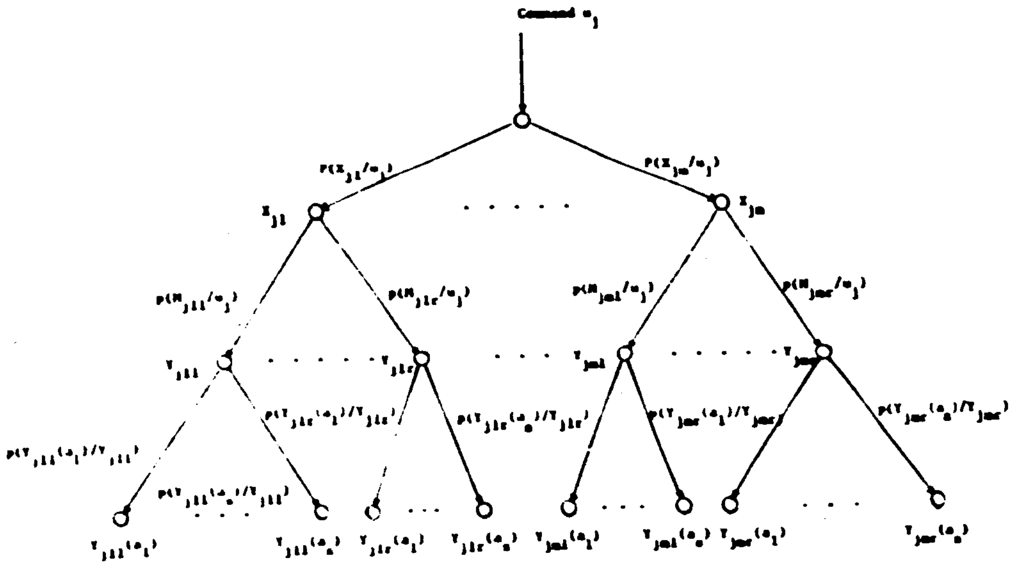


Figure 3a. Analytical illustration of the function of the organization level

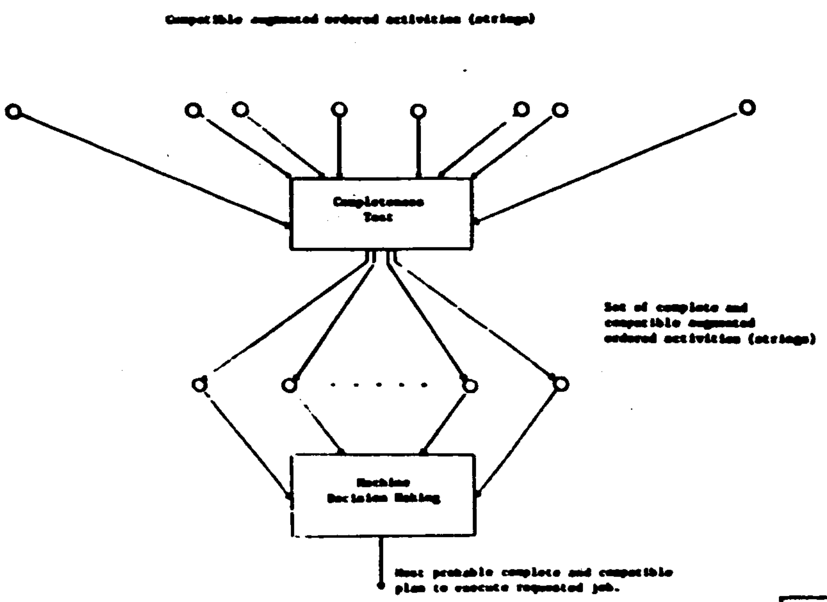

Compatible augmented ordered activities (strings)
Completeness Test
Set of complete and compatible augmented ordered activities (strings)
Machine Decision Making
Most probable complete and compatible plan to execute requested job.

Figure 3b.

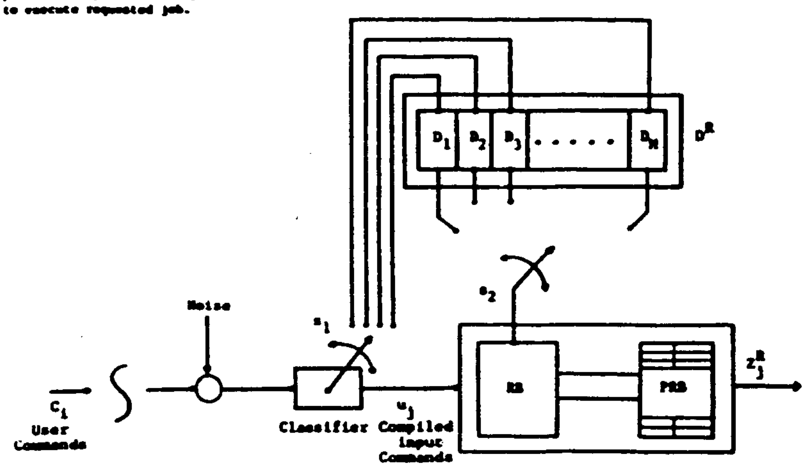


Figure 4. The Model for the Machine Reasoning Function

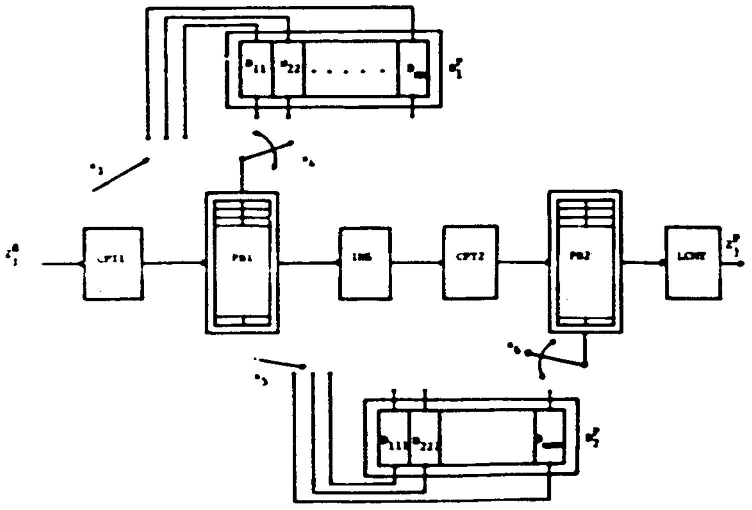


Figure 5a. The Model for the Machine Planning Function

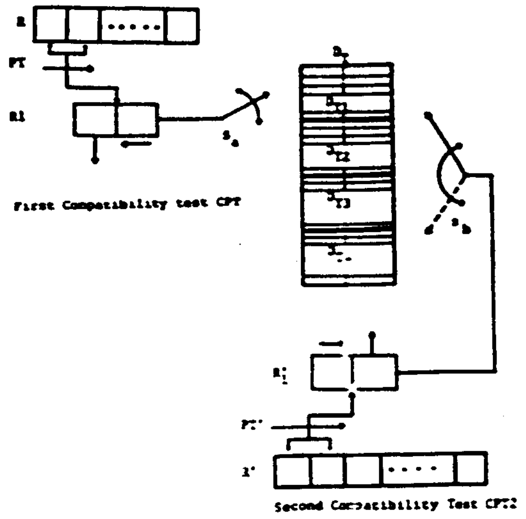


Figure 5b.

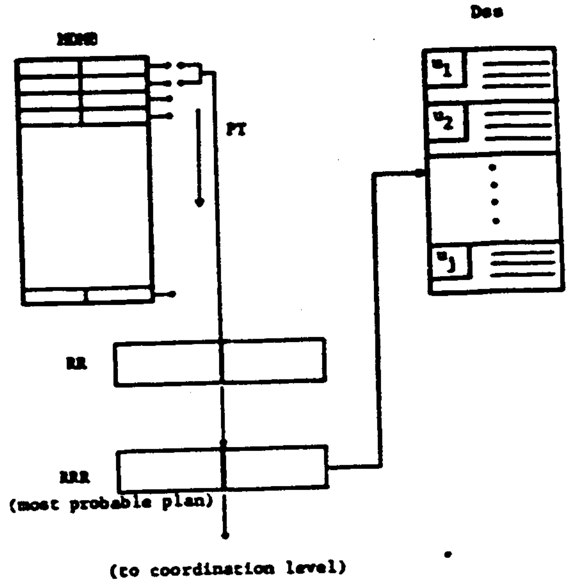


Figure 6. The Model for the Machine Decision Making Function

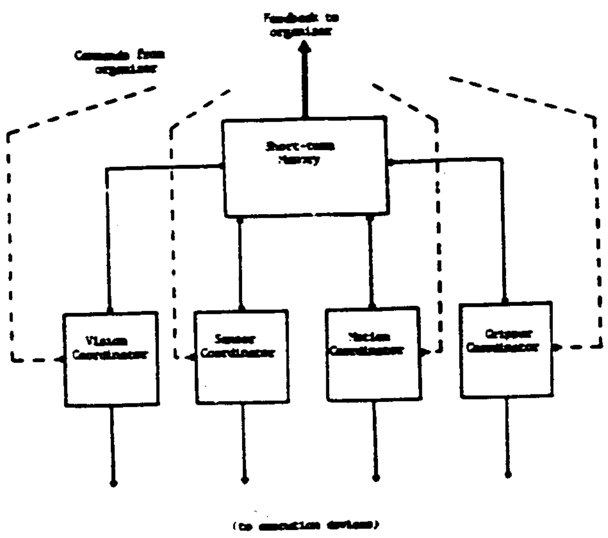


Figure 7 Block diagram of the Coordination Level

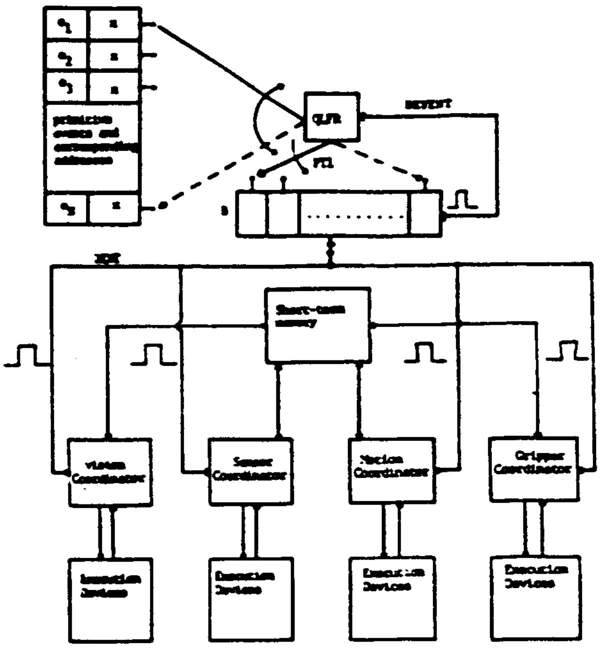


Figure 8 The Model of the Coordination Level

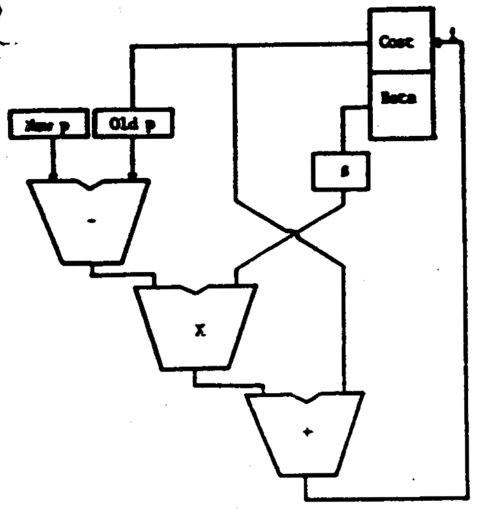


Figure 9 Probability Upgrade Hardware Unit

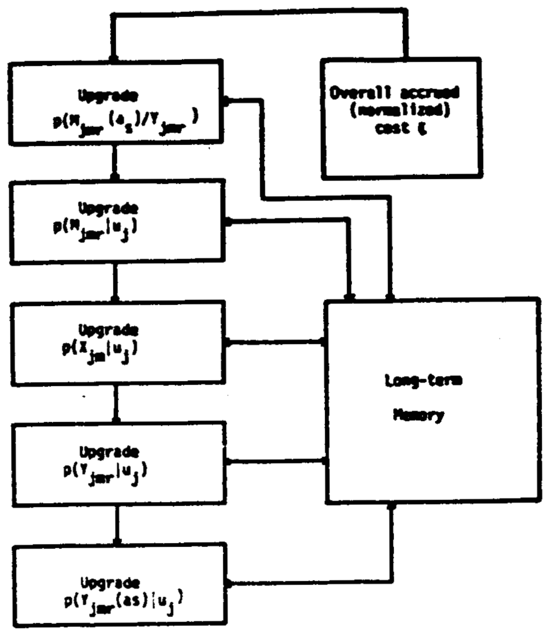

Upgrade $p(M_{jmr}(a_s)/Y_{jmr})$
Overall accrued (normalized) cost ξ
Upgrade $p(M_{jmr}|u_j)$
Upgrade $p(X_{jm}|u_j)$
Long-term Memory
Upgrade $p(Y_{jmr}|u_j)$
Upgrade $p(Y_{jmr}(as)|u_j)$

Figure 10. The Model of the Off-line Feedback Mechanism

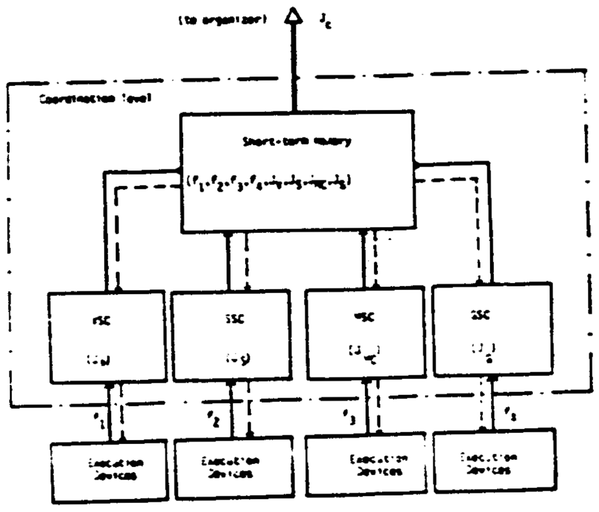


Figure 11. The Architectural Model for the On-line Feedback Mechanism

S27-60
208
188109

Nested Hierarchical Controller With Partial Autonomy

A. Meystel
Drexel University
Philadelphia, PA 19104

D4178438

Abstract. The problem of computer architecture for intelligent robots with partial autonomy is addressed. Robot with partial autonomy is considered a degenerated case of a fully autonomous robot. Thus, the problem of man-machine communication is formulated, and the conditions are determined for generating a language for such a communication. The duties of the master are determined.

Key words: *Intelligent Control, Robotics, Man-machine Operation, Hierarchical Controller, Autonomous Robots, Nesting.*

1. Introduction

Problem of partial autonomy. The problem of partial autonomy is usually considered to be a problem of enhancing the capabilities of a teleoperated pobot. Partial autonomy is meant to compensate for the deficiencies of human perception and/or performance (DHPP) when the CRT is used as a "narrow window" to the world, and no or limited other means of perception are available. After the partial autonomy emerges as a necessary property of an intelligent robot, one can expect to utilize it also for simple repetitive jobs, other routine operations, in other words, as a tool of automated operation rather then autonomous operation.

The first term (automated) means independent operation of a machine, following a set of prescribed decisions which are submitted by a human operator in advance. In other words, automation presumes existence of an explicit, and/or implicit program of operation: all decisions are already made, and all possible situations are taken in account. The second term (autonomous) presumes that there are stages of planning and decision-making performed by the machine with no human involvement. In fact, autonomy means that the set of operations to be performed (or "jobs") is not well structured, more than one possibility exist, and it is up to the robot to select which one is more beneficial.(This can happen not only when the field of view is narrow and/or cluttered by unknown objects. For example, making a step upon an unstructured surface by a multilink redundant leg, generates a huge set of ill-defined problems. Human operator cannot and need not share his attention among all these problems).

The problem of joint design of man-machine control system was first addressed in [1]. Based upon the notion of a limited "bandpass required of a man" the paper suggests that the control system should be allowed autonomy in "integration, differentiation, feedforward", and a number of other computation procedures in order to allow a man "to operate as a simple amplifier". The fact that a human operator still has to perform a unique job of information integration and decision making, at this stage was overlooked. However, very soon the human capabilities to be a universal feedback with unlimited "sensor fusion" talents were questioned in [2].

Rapid growth of the requirements to productivity and quality factors, brought DHPP mentioned above to the attention of researchers in the area of advanced control systems. The variety of teleoperated and supervised automated systems described in [3-5] concentrate upon enhancement of human capabilities by allocating some properties of autonomy within the machine control system. The state-of-the-art reflected in the literature, implies the sequence of development stages as shown in Figure 1. A need in the Interactive Manual-Automatic Control arises in a natural way due to the DHPP. According to [5], manual-automatic control allows for some motions under the manual control whereas the remaining motions are performed automatically referenced to a variety of sensor data. One of the systems created at JPL, features a menu which can be activated by an operator in modifications starting with fully manual and ending with fully automated operation.

However, the system with partial autonomy can be approached in a different way. Let us imagine that the structure of a system is available with full autonomy of operation. How does this structure look like? What are the components of this structure that can and cannot be achieved by the technological means under consideration? It may happen that some elements of the autonomous operation can never be achieved, or can be achieved in a very remote future. The set of such "unachievable-now" operations can be considered **domain of impossibility of the mission** (DIM-area).Then it would be reasonable now to make a step back from the ideal image of the autonomous machine, and build a system which rely on human participation however only within the limited part: within the DIM-area.

This approach to the problem of partial autonomy can be illustrated by Figure 2. We can see that the supervised automation is viewed as a retreat from the demand for a fully automated system. In the same vein, the partially autonomous, or supervised autonomous robot can be considered a trade-off between the demand for a fully automated system (which has never been satisfied), and a demand for a fully autonomous system (which cannot be sacisfied, at least currently).

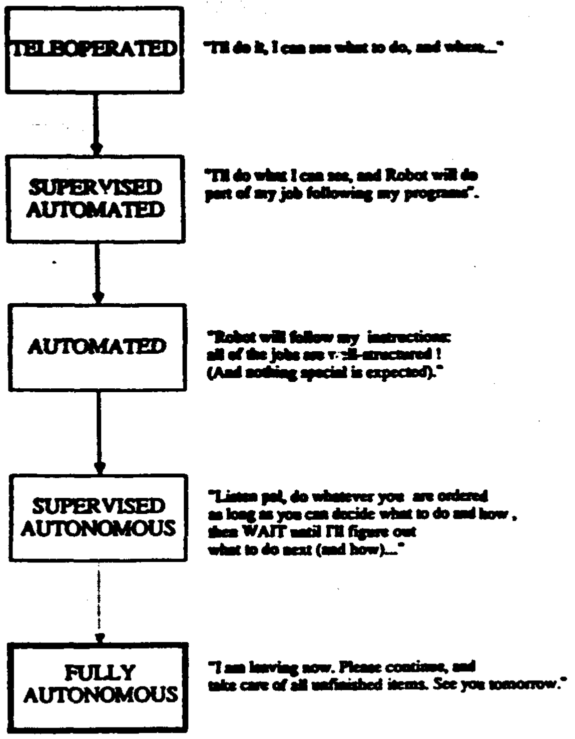

TELEOPERATED
"I'll do it, I can see what to do, and where..."
SUPERVISED AUTOMATED
"I'll do what I can see, and Robot will do part of my job following my programs".
AUTOMATED
"Robot will follow my instructions: all of the jobs are well-structured ! (And nothing special is expected)."
SUPERVISED AUTONOMOUS
"Listen pal, do whatever you are ordered as long as you can decide what to do and how, then WAIT until I'll figure out what to do next (and how)..."
FULLY AUTONOMOUS
"I am leaving now. Please continue, and take care of all unfinished items. See you tomorrow."

Figure 1. Typical approach to to the problem of partial autonomy

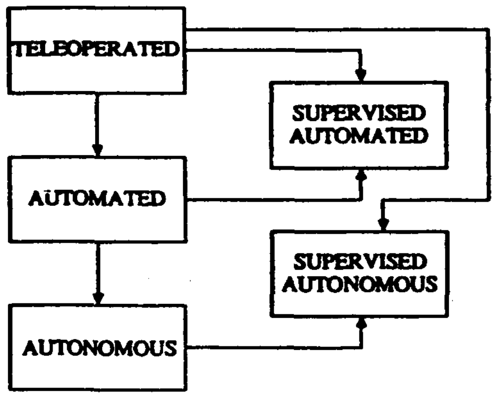


Figure 2. Our approach to to the problem of partial autonomy

Focus of concentration Problem of autonomous robot operation is often erroneously considered a problem of intelligent perception. In fact, even having the problems of perception and knowledge organization solved, the problem of motion planning and control of autonomous as well as partially autonomous robots remains unresolved. This paper attempts to formulate methods of theoretical analysis for partially autonomous planning/control processes of a class of intelligent robots equipped by a control architecture designed for fully autonomous operation. This control architecture has an important distinction which affects supervised operation with partial autonomy. Areas of motion planning, and motion control are treated separately in non-autonomous systems where such a practice is acceptable . Autonomous systems are intrinsically based upon a unified computer system jointly emulating planning-control activities as a part of a unified recursive computational process.Theory of joint planning-control systems and processes, applicable for robots equipped with Autonomous Control Systems (ACS) ascends from the theory of decision making in the framework of the control systems theory.

ACS were introduced first in [6-9], and the elements of ACS theory are presented there. ACS are to be installed in autonomous robots which should operate in unknown environment with limited human involvement or with no human involvement at all. ACS are expected to function using human-like procedures of perception. They have to maintain sophisticated information structure capable of dealing with learned knowledge, and communicating with so called, knowledge based control systems (KBCS). We make a distinction between knowledge, information, and data. Knowledge is understood as a structured information incorporating numerical data as well as linguistical, or symbolic information, which must be interpreted in a definite context. It will be demonstrated that the principle of nested hierarchies allows for an efficient knowledge organization in KBCS as well as for correspondingly efficient processes of knowledge-based perception and control.

II.Joint Planning/Control Process

Control solutions to be considered The word *control* is being used here in its initial meaning for decision making activities including *planning*, *navigation*, and *guidance*. This triad is presumed to be applicable at least, to systems for control of mechanical motion. It should provide a mapping of the a)task,b)knowledge of the system, and c) knowledge of the environment - into the output specifications.This triad is an inseparable whole, and each of the elementary parts of this triad can operate properly only in cooperation and coordination with the others.

An overview of the recent results in the area of Autonomous Control Systems for mobile robots [10], suggests that different control solutions show definite resemblance with human teams: they have a hierarchy of decision-making units even when the system is equipped by a single actuator [11]. In Figure 3, three examples of control structures are compared [12-16]. Each of these structures consists of a number of goal seeking decision units, each tends to form an intelligent module for automatic generation of control strategies, policies, and commands.

However, the most remarkable thing about all of these systems is that in all of them the scope of decision making is becoming smaller, and the resolution of decision making is becoming higher when the level of decision making is becoming lower. This sequential multiple process of decision making (replanning, and finding a new control sequence) in the same situation at different resolutions, and with different scope of attention, seems to be almost obvious. The major theoretical and design recommendations are hidden in this fact, and thus usually escape from our attention. We would assume that the underlying theory should be applicable to ACS.

The major result pronounced within the theory of team control as applicable to robotics, is related to the information structure required to run the system. The realization of this information structure is a

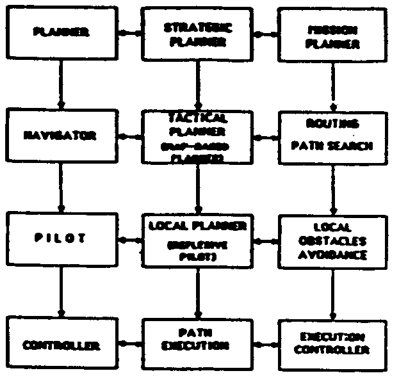


Figure 3. Comparison of control structures proposed for Autonomous Robots by the authors of [8-12].

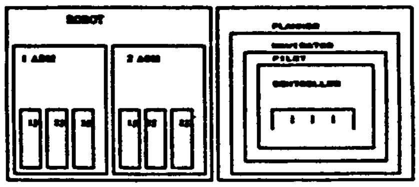


Figure 4. On comparison between nested tree-hierarchy (a), and nested stem-hierarchy (b)

special, control oriented knowledge structure. (This subject was addressed in [16], and familiarity with the treatment of the problem given there is presumed). Theory of control oriented knowledge organization as a part of the ACS, is focused upon development of models of KBCS for motion, structures of algorithms, and design of systems for optimum motion of autonomous or semiautonomous systems. Theory of ACS employs the fact that the similarities among the existing structures of control (mostly, knowledge-based) for autonomous robots reveal a number of inner mechanisms of goal oriented dealing with knowledge, as a part of ACS functioning.

Roughly speaking, any ACS is organized as a ***team of human decision-makers*** which allow for using many efficient solutions developed for human teams. Using the theory of team-control an efficient procedure of computation can be arranged which enables real-time operation of ACS. Thus, emphasis is done on solutions which are tractable, and do not lead to NP-hard problems. In the knowledge based controllers this problem gets a specific content: information structure should be suited a proper knowledge quantization. Some steps in this direction were made in [6-9].

Hierarchical structures of attention and resolution . In this subsection we will consider some of the peculiar properties of hierarchical control systems. Hierarchical decomposition of systems is a well known phenomenon, and is used usually as a method of dealing with their complexities. A system is being decomposed in parts (partitioned) when the parts contain some active elements of the systems subject to control (actuators). Thus, it is typical to consider multiactuator control systems in a form of "tree-hierarchies". Most of the statements in the theory of intelligent control, are formulated for the "tree-hierarchies" of the multiactuator systems, where the hierarchical decentralized techniques are recommended: relative independence of the levels is as important, as coordination of the branches [17-21]. The hierarchy is being preserved by the fact of resolutional as well as attentional ***nesting***. A typical decentralized control system allows for a tree-decomposition which leads to a nested hierarchy exemplified in Figure 4,a.

A degenerated case is turned out to be of substantial importance: when no multiplicity of actuators exist, and yet, a single actuator needs a nested stem-hierarchy of decision-makers in order to be properly controlled (e.g. shown in Figure 4,b). Usually, hierarchical systems are convenient as a form of organization for large systems which contain a number of goal-seeking decision units (subsystems). Thus, the problem usually includes coordinating their actions as to optimize the process of goal achievement.

Decision making processes in a nested hierarchical structure. Decision-making process which is being done upon the nested set of representations can be characterized by different frequency, accuracy, and combinational power. However, this difference can be stated only if the absolute values of these criteria are compared. Indeed, the frequency of decision-making is the lowest at the representation "a" (2.10^{-4} to 10^{-3} Hz), is higher in representations "b" and "c" (10^{-3} to 10^{-1} Hz), and is the highest in representation "d" (10^{-1} to 5Hz).

Certainly, if the zooming is being continued, the subsequent "magnifying the image" will bring attention to the processes of controller compensation, and the frequency of decision-making will grow even more. On the other hand, the accuracy of decision-making is the lowest for the representation "a": the error of decision can be in miles, and then decreases down to inches in figure "c". Certainly, on the level of controller compensation, the required accuracy might be even higher. Speaking about "combinatorial power", the solutions of representation "a" can lead to substantial differences in the overall operation, while the variety of alternatives for the narrow scope of representation "c" does not seem to be as drastic as in the case of "a".

However, all these matters assume a totally different consideration. The real difference between representations "a" through "c" is in the resolution of representation. Indeed, resolution (or the smallest distinguishable element of the image) in the case "c" is small, and in the case "a" is large, practically, the values of resolution are determined by the values of the accuracy allowed). First, we notice that the number of "objects" of the world represented in all of the images is the same due to the difference of resolution. Secondly, we can easily find that at a given frequency of decision-making at a level of consideration, the moving robot (speed is constant) will pass during the period of time between two consecutive decision-making processes, the same quantity of resolution units so that the ratio

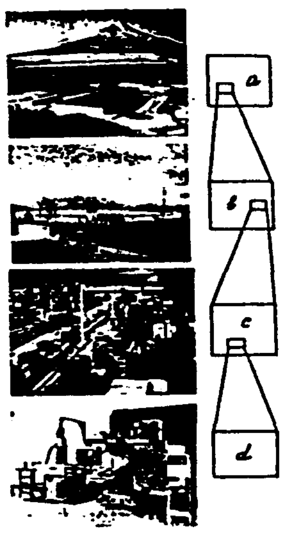


Figure 5. Nested Hierarchical Perception of the assignment for a robot

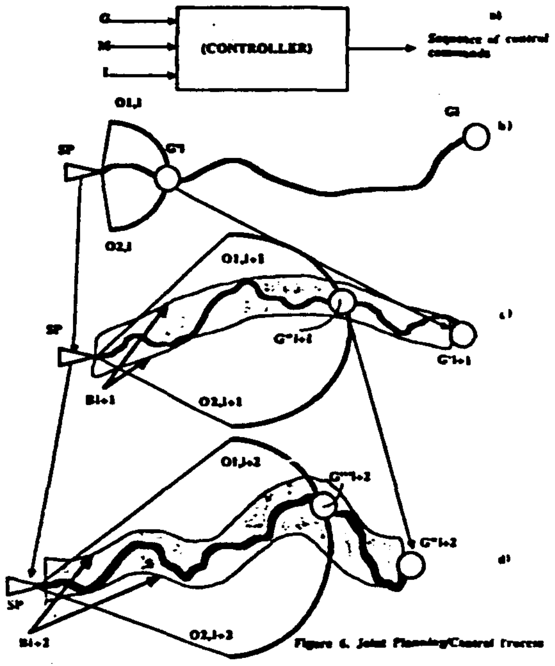


Figure 6. Joint Planning/Control Process

$$\frac{\text{(maximum state space changes per decision)}}{\text{(time interval between two consecutive decisions)}} = \text{const}$$

is constant at all levels. After simple transformation we receive that

$$\frac{\text{(time interval between 2 consecutive decisions)}}{\text{(resolution at the level)}} = \text{const}$$

for all levels of consideration.

Similar consideration brings us to conclusion that "combinatorial power" is also the same at all levels of consideration; it is just applied to different units of information, units that have different resolution. This hierarchy of representations at different resolution can be easily transformed into a hierarchy of

Decision making processes of planning-control procedures In the most general form, the controller can be represented as a box with three inputs, and only one output. These inputs can be specified as follows (see Figure 6,a):

-*Task*: the goal (G) to be achieved, (and conditions to be satisfied including the parametrical constraints, the form of the cost function, its value, or its behavior).

-Description of the "exosystem", or map of the world including numerous items of information to be taken in account during the process of control; map of the world (M) is often incomplete, sometimes, it is deceptive.

-Current information (I) is the information set delivered by the sensors in thevery beginning of the process of control, and continuing to be delivered during the process of ACS operation.

The processes within the controller are illustrated in Figure 6,b,c,d.Input T has determined where the points SP_i and G are situated within the input map M. Input I gives a limited zone $O_{1,i}$-$O_{2,i}$(in the vicinity of SP_i) in which the information set is more reliable, or even exhaustive.Thus, in the overall planned trajectory SP_i-G'_i-G_i, a part SP_i-G'_i can be determined with more reliability than the rest of it. In other words, the overall trajectory SP_i-G'_i-G_i from SP to G might be changed in the future if the input I will update the map M in the way that the computed (planned) trajectory will become not the most desirable alternative. However, SP_i-G'_i part of the plan (bold line) won't be changed: no new information is expected.

Let us concentrate now on the part of the trajectory SP_i-G'_i which is assumed to be more known than the rest of the overall plan.Label the point G'_i the "new goal", or the "goal for the lower level of the nested hierarchy". Notice that instead of the planned curve segment SP_i-G'_i the stripe is known with boundaries $B_{.+1}$.Within this stripe, a new planned motion trajectory can be determined at a higher level of resolution pertained to the level (i+1). Again, only part of the trajectory SP-G"i+1 can be refined within the sector $O_{1,i+1}$-$O_{2,i+1}$. This in turn, brings us to the more precise part SP_{i+1}-$G"_{i+1}$ which can be subsequently refined at the level (i+2). Enentually "new goals" of the adjacent levels are becoming indistinguishable and the the trajectory SP-Gn can be utilized for computing the actual control sequence. All previously determined trajectories are plans at different level of refinement.

This simple consideration includes the following components of the joint planning/control process .

1.Finding the optimum plan SP-G based upon the map M, ant the task formulation (SP, G, cost-function, constraints).
   1.1 Computing the alternatives of SP-G.
   1.2 Comparison of the available alternatives.
   1.3 Selection of the preferable alternative.
2.Updating the map information M in the vicinity of SP by using the sensor information I.
   2.1 Recognition of the set I.
   2.2 Comparison between M and I.
   2.3 Deciding upon required changes in the cases of: $M/I \neq \emptyset$, $I/M \neq \emptyset$, $I = M$.
   2.4 Creation of the new map in the vicinity of SP with required deletions and additions.
3.Refined planning the path within the updated zone of the map.
   3.1 Determining the subgoal G' (e.g. as a point of intersection of SP-G and the boundaries of updated part of the map).
   3.2 Finding the optimum plan SP-G' (which implies that the whole set of procedures mentioned in p.1, should be repeated).
4. Track the optimum path SP-G'.
5. Upon arrival to the new point of the selected trajectory (distinguishable from the initial point) loop to the position 2.

The set of procedures 1 recursive through 3 is illustrated in Figure 6,b. Easy to notice, two major recursions are presumed within the structure of accepted decision making activities. Firstly, position 3.2 generates recursion which presumes consecutive refinement of the information within the zone which allows for this refinement. Secondly, the major loop from 5 to the position 2 presumes repetition of the whole set of the procedures after each increment of the motion. Thus, the process of path planning can be substituted by a consecutive determining of subgoals located closer and closer to the current position.However, still remains unclear a) how to determine the trajectory SP-G, and b) how to determine the zone in which the refined information should be obtained. These two key elements of the overall process will determine the number of recursions to be performed for the each point of the motion trajectory.

Substitution of positioning by tracking Theory of tracking control is much better developed than the theory of positioning (or control of "pick-and-place" operations). No wonder, we are interested in reduction of positioning problem to the tracking problem. The similarity between our control and "tracking the target", is becoming even more noticeable after we realize that the tracking trajectory is being constantly recomputed in the process of the above mentioned recursion.

Obviously, our best planned trajectory can be considered as a predicted trajectory. At the each level of the system shown in Figure 6, the bold part of the trajectory is a plan per se, and the rest is just a predicted trajectory. However, for the next consecutive lower level the situation is becoming different. The plan of the upper level is becoming just a planning envelope for the lower level. Within the **planning envelope** assigned by the upper level, **an optimum motion trajectory** is being refined which in turn is considered as a plan for the adjacent level below. The initial part of this trajectory is considered the low resolution control to follow (1-st part of the plan) and the rest is becoming just a prediction (2-nd part of the plan).

Since part of SP-G is cosidered as a prediction anyway, and since the corresponding information is to be updated in the future, the problem of system's readiness to incorporate the results of updating information, can be discussed. This generates such topics as storing the alternatives of plan, evaluation of predictions by some probability measure, and synthesis of contingency plans.

Commutative diagram for a nested hierarchical controller. Hierarchical decision making process allows for using efficiently the full computer capacity which is limited at each level of such hierarchy (with no branching). In this case, the tree hierarchy of intelligent control converts into a **nested hierarchical controller** (NHC). If NHC is acting under the above mentioned constraints the process of control allows for decoupling in parts dealing separately with information of different degree of resolution (easily interpreted as certainty, belief, etc.) This means that the degree of "fuzziness" is different at different levels of the hierarchy, and in the nested hierarchy of the fuzzy-state automata each automaton of the lower level (automaton-child) is enclosed in the corresponding parent-automaton, and serves for clarification of its uncertainties.

Considering subsystems as categories C, and the interaction among them as functors F, the commutative diagram of the ACS can be shown in Figure 7 (indices mean: s-sensing, p-perception, k-knowledge, pc-planning/control, a-actuation, w-world). Feedbacks are not shown: boxes are connected by "functors" which characterize the structure conservation in a set of mappings of interest. The bold horizontal line separates two major different parts of the system: what is below, is a world of real objects, and what is above the bold line, is the world of information processing. All of the "boxes" in Figure 7 are fuzzy-state automata. They are easily and adequately described in terms of the automata theory, and provide consistency of the descriptions, computer representations, and control operation. Then, the search can be done by combining A* and dynamic programming, discretization of the space is being determined by the level of resolution, and the rules which are formulated within the given context.

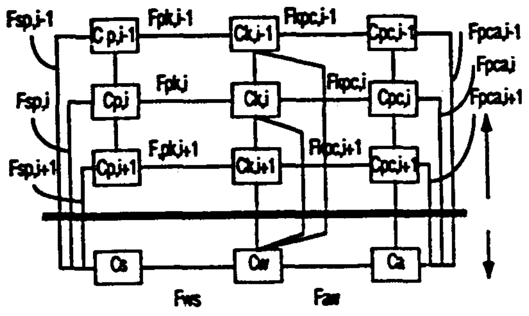


Figure 7. Category-theoretical Diagram of Nested Hierarchical Controller

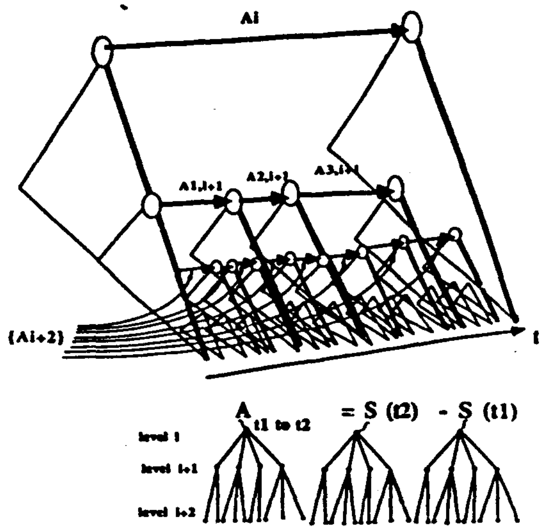


Figure 8. On the relations between representation and control (S-states, A-actions)

Nested dynamic programming The principle of optimality of Bellman [22] can be stated as follows for stochastic problems: at any time whatever the present information and past decisions, the remaining decisions must constitute an optimal policy with regard to the current information set [23].Thesequence of "planning-navigating-piloting (or guidance)-actuator control (or execution)" appears as a direct result of nested hierarchical search in the structure of information under consideration. Method of Nested Dynamic Programming (NDP) follows from the commutative diagrams and analysis given in [6-9]. It states that the optimum control should be found by consecutive top-down and bottom-up procedures, based on the following rules.

Rule 1. NDP should be performed first at the most generalized level of information system with complete (available) world representation.

Rule 2. NDP is being performed consecutivelv level after level top down. The subspace of the search at each of the consecutive lower levels is constrained by the solution at the preceding upper level recomputed to the resolution of the next lower level.

Rule 3. When during the actual motion, due to the new information, the optimum trajectory determined at a given level must violate the assigned boundaries, this new information should be submitted to the upper level (proper generalization must be performed, and the information structure must be updated). This generates a new top-down NDP process.

Rule 4. When arrival of the new information is bounded (e.g. by a "limit of vision"), then the recursion of nested process of planning is being done with consecutive process of subgoals creation.

Control Stratified by Resolution into a Joint Multilelel Planning/Control Frocess. The nested hierarchy of maps $\{m_i\}$,i=1,2,... is the input for planning/control system. Actually, this nested hierarchy is being generated in the process of interaction between the subsystems M and C. Let us show how this process of interaction is going

$$\begin{tikzcd}[column sep=small]
 & \overset{fa}{(m_1}\in \overset{g}{M_1)} \supset \overset{fa}{(m_2}\in \overset{g}{M_2)} \supset\ldots\overset{g}{\supset} & \overset{fa}{(m_{i-1}\in M_{i-1})} \overset{g}{\supset} & \overset{fa}{(m_i\in M_i)} \\
\text{line1:} \arrow[r, phantom] & {} \arrow[d, "fa"] & {} \arrow[d, "fa"] & {} \arrow[d] \\
\text{line2:} & NDP(J_1,M_1)=m_1 \ \ldots\ \ldots \arrow[d] & NDP(J_{i-1},M_{i-1})=m_{i-1} \arrow[d] & NDP(J_i,M_i)=m_i \arrow[d] \\
\text{line 3:} & \text{output} \ \ldots \arrow[d] & CH(m_{i-1})=M_{i-2} \arrow[d] & CH(m_i)=M_{i-1} \arrow[d] \\
 & \text{to controller} \ \ldots & \text{to (i-2)-th} & \text{to (i-1)-th} \\
 & & & \text{l e v e l s}
\end{tikzcd}$$

Line 1 shows two nested hierarchies: one of them by generalization (of maps $\{M_i\}$) and another by focus of attention (of maps $\{m_i\}$). Hierarchy of sets is obtained from the hierarchy of sets by applying NDP-algorithm per level (line 2). In order to do this, a hested hierarchy is added to the nested hierarchy . In order to compute a set ,the results of applying NDP per level, are enhanced up to the meaningful consistent map partition; one of possible algorithms for this is "convex hull" (CH). So this system is closed loop level to level top-down and after convergence, the system of controller commands is obtained.

This planning/control algorithm represents actually a **hierarchy of blackboards** (HBB) in which the communication is being rendered among interacting subsystems of the intelligent module.**Main navigator** takes this plan as a suggestion of the milestones, and determines a string of subgoals at the way to the first of milestones. This string is sent to the **pilot** which also has nested structure.The first **subgoal** in this string might be invisible for the ACS, the visible goal is to be determined by the **planner within the pilot**. Then **pilot's navigator** can compute accurate enough trajectory of motion to be executed. This trajectory is submitted to **controller** which in turn, has nested structure.Its **planner** determines the next tarcking command, then its **navigator** determines the execution sequence, etc.

Built-in combinatorial operators:team of decision-makers Not only strings or combinations of variables (words) as well as combinations of mappings (clauses) will be considered, but also combinations of these combinations. Search by scanning, and inclusion when the desirable property is met, is one of the straightforward algorithms of combination generation. If the results of search are constantly enhancing the input vocabulary, then during the exhaustive search with recursive enhancement of the vocabularies, all possible unions, intersections, and complementations of the sets are being obtained. The final vocabulary forms a field (F) which means that it contains all set of subsets constructed upon the initial set including the results of applying all acceptable combinatorial operations, (one of these operations should be later used for decision making process: combinatorial search). Using focus of attention we receive a special category of **limited fields** which do not contain all combination of components which is required by a formal definition of field, and yet gives more meaningful recommendations without introduction of complicated algebraic structures.

Using search for alternatives generation. Any combinatorial algorithm is an operator of generating plans or **solution alternatives** for a decision-making process. Consider A*-algorithm for the search of minimum path trajectory [29,32,35], or any type of conventional or 'enhanced" dynamic programming [22,23,31,33]. Then a number (value) is assigned to each of the combinations generated (preferability, closeness, propensity, cost-effectiveness, etc.) which will enable the decision-maker to make his choice under the **accepted strategy of decision making**. According to the existing terminology, the chain of consecutive decisions named **policy** of decision-making, or policy of control

Since we have received already a hierarchical structure of representation, and N levels are considered which are representing the same world with different resolution, the following situation should be considered. Given a vocabulary consisting of N elements at the level "i" (words) w_{1i}, w_{2i}, ... , w_{Ni} the set S_i of admissible decisions (or decision N-tuples) determined within the boundaries determined by the upper level $B\{w_{i-1}\}$

$$S_i=w_{1i}\otimes w_{2i}\otimes\ldots\otimes w_{Ni}=\{w_i\},$$

and assume the cost-functionals (e.g. presented as distances)

$$J_i:\{w_i\},B\{w_{i-1}\}\rightarrow\rho_i$$

then the obtained N-tuple (a string $w_1w_2\ldots w_n$) is the i-level hierarchical solution. This string can be interpreted as the next subsequent state of the world, or the change leading to the next subsequent state of the world (action). In the latter case, the subactions at various levels of the hierarchy, show the overall action which leads from one substate to another. Clearly, any decision making process is a result of decision making activities of a couple of adjacent levels, which means that the planning/control procedure in a result of a recursive computation upon the whole hierarchy of representation. Strictly speaking, the decision making at a level require involvement of at least two adjacent levels. Thus, the structure of the levels representation is becoming of importance.

II. Representation System

Information set contained in the representation system. The information set at time k is assumed to be the past measurements and controls for all levels of our hierarchy nested by

a)generalization:

$$(I_i(k_i) \overset{g}{\supset} I_{i+1}(k_{i+1}) \overset{g}{\supset} I_{i+2}(k_{i+2}) \overset{g}{\supset} \ldots) \therefore$$

$$\{Y_i(k_i) \overset{g}{\supset} Y_{i+1}(k_{i+1}) \overset{g}{\supset} \ldots, U_i(k_{i-1}) \overset{g}{\supset}$$

$$U_{i+1}(\overset{g}{k}_{i+1}\overset{t}{-1)} \supset \ldots \overset{g}{\}} \supset I_i(k_{i-1}) \overset{g}{\supset} I_{i+1}(k_{i+1}-1) \supset \ldots$$

b)focus of attention

$$(I_i(k_i) \overset{fa}{\supset} I_{i-1}(k_{i-1}) \overset{fa}{\supset} I_{i-2}(k_{i-2}) \overset{fa}{\supset} \ldots) \therefore$$

$$\{Y_i(k_i) \overset{fa}{\supset} Y_{i-1}(k_{i-1}) \overset{fa}{\supset} \ldots, U_i(k_i-1) \overset{fa}{\supset}$$

$$U_{i-1}(k_{i-1}-1) \overset{fa}{\supset} \ldots \} \overset{t}{\supset}$$

$$I_i(k_i-1) \overset{fa}{\supset} I_{i-1}(k_{i-1}-1) \overset{fa}{\supset} \ldots$$

It is shown in [5-9] that the phenomena of nesting by generalization and focusing the attention can be characterized best by the operators of inclusion which presumes some procedures unusual for the conventional approaches to control systems. Indeed, nesting by generalization is obtained as a result of a)grouping the object of the i-th level by the class-generating feature, b)assigning a label to this group, c)placing this laber into the (i-1)-th level as a single object, d)repeating the same for the words of the (i-1)-th level,and e)finally, obtaining a new word at the (i-2)-th level.Then, any class of the (i-1)-th level is nested by generalization in its much broader representation at the i-th level.On the other hand, after a solution is obtained as a string within the class of the (i-1)-th level, this solution will automatically "carve out" a subset within the corresponding much broader representation at the i-th level, of the same (i-1)-th level class in which the solution was determined.The result of such "carving out" is nested by attention within its own class of the i-th level.

Partitioning and nesting of representations. Decomposition of the categories of representation is done through decomposition of objects and morphisms represented in this category [24]. Decomposition imlies dividing objects and relationships in parts, so the components of objects are becoming of interest which was not the fact before the process of decomposition. This, in turn, implies higher resolution of world representation than before the process of decomposition. If we continue with decomposition, the further representations are expected to disclose new details about the objects and their parts, as well as about relations and their parts. So, in the hierarchy of decompositions, all of the levels describe the same world with higher and higher level of resolution.

Changes in time are represented by sequences of the snapshots in domain of representation describing sequences of the world states. Thus, change in the time-scale is intrinsically linked with the value of resolution. Certainly, different levels of the hierarchy can be characterized by different time-scales. Allthese phenomena are illustrated in Figure 8.

Nested Tesselation of Knowledge. This system of nested variables and nested mappings is described in a Hausdorff nested H-space which means that any two different n-dimensional points in it have at each level of nesting, disjoint neighborhoods. One may assume that our space is obtained as a result of natural, and or artificial discretization (tesselation, partitioning) from an initial continuous space of the isomorphical world description. This space can be considered as a nested state-space for a dynamical system in which the trajectory of motion can be represented. This space is considered for a given moment of time (snapshot of the world). Then, a sequence of snapshots can be obtained. The conventional state-space is a superposition of all snapshots which describe the process.

Two types of discretization (natural, and artificial) should be understood as follows. After the problem is formulated, it is already based upon some world discretization since the context is represented by a semantic network discretizing the world in a "natural" way. However, when the higher level of resolution is desired, the natural discretes can appear to be insufficient. Then another "artificial" type of discretization is applied, and the space is tesselated by (say) a "mechanical", quantitative division of larger bodies into the smaller parts.

This tesselation is done in such a manner that the "points of interest" (symbolizing the variables, words, or wwf's) are placed to the center of the "tile" of tesselation at a level (elementary segment, grain, discrete, pixel, or voxel of the space). Such terms as segmentation, granularity, discretization, and tesselation can be used intermittently. Property of being centered, holds if the tile can be derived as a system of subsets with unempty intersection. This property should be considered rather in a symbolic way: in fact, we cannot discern any another point within the tile, this is a minimum grain at this level of resolution.

Resolution of Knowledge In other words, the tile of the tesselation determines the **resolution of knowledge** which is defined as a minimum discrete of information, or minimum wwf which can be stated unmistakably.The minimum centered tile will have diameter ε and the net of centers emerging from this tesselation is named ε-net [25].Analytical details can be found in [6-9].

The idea of nested tesselations is coming together with a definition of a single tile $T(\varepsilon)$ based upon nested sphere theorem which can be rephrased as nested tile theorem which defines a chain of inclusionsfor the hierarchy of tesselations. Thus, ε -net is considered to be a set of elementary tiles satisfying a condition

$$T(x_0, \varepsilon_0) \supset T(x_1, \varepsilon_1) \supset T(x_2, \varepsilon_2) \supset ... \supset T(x_n, \varepsilon_n)$$

where $x_1, x_2,..., x_n$ are the coordinates of the centers of the tiles,

$\varepsilon_1, \varepsilon_2, ..., \varepsilon_n$ are the radii of the nested tiles.

In the equation of relationships among the tiles $$\varepsilon_0 = \frac{\varepsilon_1}{\sigma_1} = \frac{\varepsilon_2}{\sigma_2} = ... = \frac{\varepsilon_n}{\sigma_n}$$

Pairs of adjacent levels. Hierarchies created in this way, satisfy the following principle: **at a given level, the results of generalization (classes) serve as primitives for the above level.** Then each level of the hierarchy has its own classes and primitives; thus, it has its own variables (vocabulary), and the algorithms assigned upon this vocabulary can be adjusted to the properties of the objects represented by the vocabulary. This determines the mandatory rule of combined consideration of two (at least) adjacent levels of the hierarchy. Set of relationships among the variables at the i-th level describes the implications which are being utilized for decision-making at this particular level.

On the other hand, the set of relationships at the (i+1)-th level describes the relationships among classes and thus can be characterized as set of **metaimplications** (metarules, metaclauses) which are governing the process of applying the implications (rules) at the i-th level. In the light of this consideration, each two adjacent levels can be understood as a complete E. Post production system (analog to "general problem solver" or "knowledge based controller") in which the metarules applied to the alphabet of the lower level act as a semi-Thue process of the grammar [26].

"natural" (qualitative) way (based upon "words" of the context), and the lower level is discretized in an "artificial" (quantitative) way (based upon grids, etc.). The process of decision making in this case depend on the ability of these two levels to communicate efficiently despite of the different representation of the elementary "tiles". The role of human "master" or "supervisor" is critical for operation of such adjacent levels since he is the real carrier of the context knowledge which is required for the adequate translation from the language of the (i-1)-th level into the language of i-th level and vice versa.

Resolution of representation. Labeling the class presumes dealing with this class as a primitive at the given level of consideration. Moreover, this class (now, also a primitive) is being again clustered in classes of the higher level. In order to deal with class as with a primitive we have to neglect the inner content of this class (which might be reflected as new properties of the new primitive but with no mentioning the initial primitives with their properties). The levels of the hierarchy of representation (since they are created by a mechanism of generalization) are dealing with the same world given with different level of richness in submitting specific details, level of coarseness (fineness). We will name this characteristics of world representation at the level, **resolution** which is a measure of distinguishability of the vectors in the state space.

Accuracy versus resolution. It is clear that after assigning to the cluster a new class-label (a word to be included to the vocabulary), this class-label is becoming a new primitive with its own properties, the numerical values are assigned to these new properties, and these numerical values are characterized by the accuracy depending on the accepted evaluation theory for this particular property (including probabilistic approaches, fuzzy set theory, etc.). Clearly, this accuracy evaluation is formally independent from the act of neglecting the inner contents of the new primitive. This means that accuracy and resolution are formally independent. The word "formally" means: in the view of procedure to be performed.(Consider digitized images (e.g. any "map" of the world) with different base discrete of the image. The bigger this discrete is, the less is the resolution of image. However, the accuracy of representation at this level can be the same provided the accepted value of error upon the base discrete).

Thus, accuracy presumes the error evaluation in terms of the existence of difference between the world and its description **within the accepted concept of the model.** The smaller the vocabulary Σ_i is, the more different phenomena are neglected. This neglect may entail the increase in error and may not. However, the smaller $\mathrm{Card}(\Sigma_i)$, or size of the vocabulary is, the higher is the level of generalization, and the larger is the radius of the tile in the ε-net. Thus, the following relation should hold

$$\rho = \frac{1}{\mathrm{Card}(\Sigma_i)} > \Delta$$

where ρ determines the value of allowable error (inaccuracy) and $\mathrm{Card}(\Sigma_i)$ determines the value of resolution.

Thus, the information of the world which is required for decision-making, should be explicated and prepared for the subsequent decision-making processes. Map is defined as a subset of the state-space which is to be taken in consideration during the process of decision-making. Thus, map is a part of policy delivered from the leader to the follower. Map of the upper level contains the maximum subset given at the lowest resolution. Maps for the next levels of the nested map structure (top down) are obtained using the second apparatus of nesting: *focusing of attention*. Thus, only one part of the map is being activated: which is required for the current decision-making.

The system of maps has dual nesting:

firstly, we have a subsystem with nesting by generalization

$$M_1 \overset{g}{\supset} M_2 \overset{g}{\supset} \dots \overset{g}{\supset} M_{i-1} \overset{g}{\supset} M_i$$

which we name "CARTOGRAPHER", and assign to this subsystem maintenance of the information rather than active participation in the process of decision-making.

and secondly, we have nesting by focusing attention superimposed upon the nested hierarchy by generalization , and this information extracted from the cartographer $\{m_i\}, i=1,2,...$ is delivered to the "C" system (Figure 7) in a form of "maps for decision making".

$$(\overset{fa}{m_1}\in M_1)\overset{g}{\underset{fa}{\supset}}(\overset{fa}{m_2}\in M_2)\overset{g}{\underset{fa}{\supset}}\dots\overset{g}{\underset{fa}{\supset}}(\overset{fa}{m_{i-1}}\in M_{i-1}\overset{g}{\underset{fa}{\supset}}\quad(\overset{fa}{m_i}\in M_i)$$

All fa-predicates are performed on the basis of NDP-algorithm results and thus belong to the subsystem of control, all g-predicates are prerogative of the system of map maintenance. Later we will show how these predicates are built in the algorithms of planning/control and map maintenance.

The problem of map maintenance is of scientific and practical importance. The upper level map ("planner's map") should be maintained for a long time due to the largest scope, and the "slow rhythm" of this level. Changes in the upper level map are not frequent. Maps of the subsequent levels are to be regularly updated with increasing frequency but decreasing volume. At the level of "pilot's" map, most of the information might be considered of little importance in long terms, and the map is an "ephemeral" one. The lowest level map ("controller's map") may or may not need even an ephemeral map maintained as a part of the nested hierarchy. (Actually, from our first experience of dealing with ACS we found that intelligent module cannot afford maintenance of the pilot map , i.e. of the lowest level of world representation), and therefore all processes related to the real time operation have ephemeral structure indeed, with a number of logical filters determining whether this ephemeral information contains anything to be included in the maps of the upper level.

The rules of assigning for the information to be retained in details, retained in generalization, or dropped, cannot be formulated consistently for all possible situations. How long the information will be needed can probably be determined only by a **master**.
Generalization: belonging to a meaningful class. So, knowledge in a context can be represented as an ε -net at a definite resolution, and as a system of nested ε -nets with different scale where scale is defined as a ratio between resolution at a level and resolution at the lowest level. Clearly, each of the larger tiles at the upper level is a "**generalization**" for the set of smaller tiles at the lower level of the hierarchy. Selection of one of the tiles at the upper level (*focusing attention*) entails selection of a definite subset of smaller tiles at the lower level.

We would like to stress the fact that the inclusion $X \supset X \supset x$ shown in this hierarchy of the tile embeddings has more important and broadmeaning than just "scaling of the size". The inclusion predicate $\supset$ has a meaning of "belonging to a class" (since it is assumed that the objects are unified into a set by some class generating property, or feature). One can talk about state space, space of weighted properties, and so on, and the notion of "belonging to a class of come spatial neighborhood" is becoming closer to a meaning of "generalization" as if it is understood in the discipline of logic. Then dicretization of the state space will contribute simultaneously to a) minimum required interval of consideration, b) hierarchy of **classes of belonging to a meanigful neighborhood.**

From the theory of pattern analysis and image recognition we know that selection of the meaningful neighborhood is a difficult issue, and it cannot be addressed exhaustively unless a human operator, or **master** is involved in the procedure of the feature assignment.

Look-up Tables. The prespecified lists of mappings upon a vocabulary, can be organized in **look-up tables** (LUT's) for convenience. For example, a control "u" can be selected from this prespecified set if LUT contains clauses (e.g. "**If S &C then u**" where S-is a state description, C-strategy selected, or cost-functional agreed upon). Minimum look-up table can contain only one mapping.

Sets of variables to be considered simultaneously are named **vectors**, or **strings** . A single LUT, or a set of LUT's are named **grammar (operators, transformations)** if they can be utilized to generate output strings from input strings. Input and output vocabularies together with the grammar are named **language** (over a finate alphabet), or **automaton** (free monoid generated by the alphabet). Clearly, vectors or strings are **statements** in the above mentioned language (built upon words of this language). The rules of combining variables into vectors, or srings, or strings of vectors, etc. are being determined by LUTs.

Special role of attention When a part X_p is detached from the "whole" X (which may happen to be category, object, and/or morphism) and the relation of inclusion holds

$$X \supset X_p$$

this separation of X_p from X we will name **focusing attention** upon X_p . Sampling is one of the common methods of focusing attention.Usually we focus the attention when the subset of attention can be considered important, or is typical for the whole set. The latter case links focusing attention with mechanism of generalization.

Special role of attention is determined by its close links with the overall goal of operation. Attention always presume sacrifice of some information in favor of the information set which is considered to be more important. Thus, attention almost always rely on a set of preferences that can be ultimately formulated only by a **master**.

Generalization oriented tesselation We would like to stress the link between the generation of entries for LUT as an approximation in the space of consideration, and the problem of **generalization oriented tesselation** (discretization of the space). At the level of discretization determined by the context, i.e. determined by the words of the semantic network, the results of tesselation are always determined by the vocabulary of the **master**.

The subsequent problems of "digitization" and "quantization", or quantitative tesselation at the lower level are being solved currently by introduction of the "shah function", and other mechanisms of sampling which has been proven to be applicable in decades of development of communication and control systems. Some difficulties, such as aliasing, and similar, lead to algorithmic aggravations at the output. In the context of this paper, discretization of the space does not allow for the problem of aliasing because there is no information between the adjacent tiles of tesselation (digitization, quantization), and the information about the tile properties (values for them) is the set of average average values over the tile. Clearly, the term "average" in this context, acquires a somewhat unusual meaning of "**class generating property**".However, in most of cases the"automatic" tesselation of the space creates discrepancies when the correspondence between the two adjacent levels is to be determined. These discrepancies might be eliminated by the interference of the **master**.

Complexity of tesselation: ε-entropy .Complexity of LUT is being evaluated by computing the ε-entropy (by A.N. Kolmogorov [25]) for a space where the corresponding ε-net has been constructed

$$H_\varepsilon(S)=\log N_\varepsilon(S)$$

where S-is a space represented by LUT,with ε-net assigned,

N_ε-number of elements (nodes) in ε-net, or tiles in the tesselation.

For the system with combinatorial search as a tool for planning, the category of consideration can be represented as a power set, then trivially

$$H_\varepsilon(S) = N_\varepsilon(S)$$

Various theories of planning are based upon the idea of space tesselation (e.g. configuratin space, etc.). A multiplicity of different strategies of space tesselation is created, and numerous techniques of evaluating the distance, and the complexity are known from the literature. It is essential to understand that all of these techniques ascend to the theory and formalisms of ε-nets. In our case, minimization of the ε-entropy allows for determining an optimum number of the levels of the hierarchy of a control system.

Accuracy of representation. To all words and all edges with the corresponding strength of the bond, the values of accuracy are assigned. Accuracy is understood as the value of the relative error of the quantitative characteristics assigned to the node or the edge of the graph. This evaluation can be based on estimation of probabilities, zones of uncertainty, and/or in any another way. The approach to the world representation in this paper does not depend on the particular way of dealing with uncertainty evaluation.The value of uncertainty is assigned to the numerical values associated with the properties of the tile of representation which is considered as a minimum unit at a level.

Evaluation of the size of a tile. This is similar to assigning a definite ε-net. However, this also suggests that the minimum value of the radius of the tile, can be time dependent. In order to avoid this predicament in the following treatment, we will try not to get involved with the particular stochastic and dynamical microstructure of the ε-tile at the level of the representation hierarchy. We will define the radius of the tile by the width of the fictitious uniform distribution which has the same value of the conditional entropy as the real distribution of the variable x. Then, using the value of conditional entropy

$$H = -\int_{-\infty}^{+\infty} p_i(x) \ln p_i(x)\, dx$$

for a Gaussian the diameter of a tile is determined

$$\Delta = \sigma\, (\pi e/2)^{.5}$$

Referring to the definition of ε-net, we would suggest that this value of Δ is the minimum size of the tile radius to be assigned if the net has the same law of distribution in all of its nodes. For systems with stationary random processes, evaluation will be sufficient with no involvement in analysis of stochastic processes anymore. Certainly, this depend on the nature of what is considered to be a random component of the variable.

Nevertheless in real situations, it is very difficult to construct a level with the same law of distribution in all of its nodes. The role of master here, is either to determine the "average" law of distribution, or to allow for a net with different sizes of tiles. Based upon condition for Δ, the problem of different laws of state-space tesselation known from the theory of motion planning (e.g. polyhedra, etc.) is virtually eliminated. The geometry of a single tile becomes irrelevant, and yet the space is fully covered by these "fuzzy" tiles.

III. Hybrid Master-Dependent Representation

Knowledge Bases: Semantic Networks with context oriented interpretation Knowledge Bases (KB) are considered to be a collection of well-formed formulas (wff's) for man-machine as well as for autonomous decision-support systems of different kinds. So, it is clear that knowledge consists of linguistical discretes (entities) which are later referred to as **units of knowledge**. When we are talking about **knowledge**, the **represented knowledge** is actually meant, (i.e. we are not interested here in discussing the nature of the reality represented within the system of world representation). Nesting is being introduced often in a semantic way , and reflects semantic discretization of resolution in representation reflecting human experience (which sometimes is irrelevant).Thus, in a nested system, a nested hierarchy of semantic networks can be formulated. This semantic nesting can be interpreted within a context only under human supervision since the subtleties of the context knowledge must be properly transformed into statements of existence or absence of relationships among the entities.

Thus, under master's supervision we receive, two types of nested hierarchies for declarative wff's: existential nesting (nested statement of existence) including nested statements of objects and relations among them, and transitive nesting (nested statement of change). Both of these nested units of knowledge are presented in implicative form (nested clauses). It seems reasonable to describe the set-theoretical on one hand, and on the other hand, vector-analytical manners of describing situations (which do not differ in essence). This another form of representation in turn, can be divided in two different kinds of representation: continuous (using differential equations) and discrete (using difference equations and/or finite state machines (see Figure 9). In both cases, the property of nesting is applied.

If a hierarchy is built using the existing semantic network, them each (i-1)-th level of the hierarchy will contain generalizations of the words of the i-th level. This means that the solutions found within the (i-1)-th level will automatically "carve out" subsets within the classes of i-th levels in which the process of combinatorial search can be repeated, and a new solution can be found with higher level of

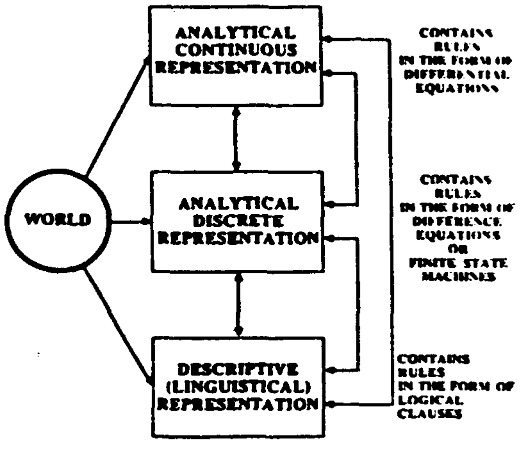

WORLD
ANALYTICAL CONTINUOUS REPRESENTATION
ANALYTICAL DISCRETE REPRESENTATION
DESCRIPTIVE (LINGUISTICAL) REPRESENTATION
CONTAINS RULES IN THE FORM OF DIFFERENTIAL EQUATIONS
CONTAINS RULES IN THE FORM OF DIFFERENCE EQUATIONS OR FINITE STATE MACHINES
CONTAINS RULES IN THE FORM OF LOGICAL CLAUSES

Figure 9. Different Categories for World Representation

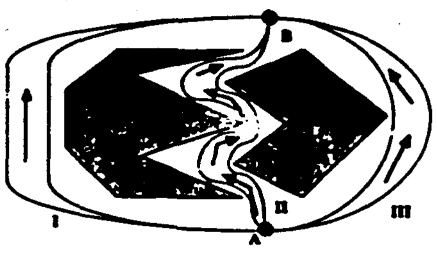


Figure 10. General solutions preceding the particular ones

resolution. However, the i-th level should understand the (i-1)-th message on the boundaries of the search.

Postulate of multiple representation . So, the same world can be represented within the same levels of resolution in a different way. Representation is defined as a structure (e.g. algebraic, or information structure) which is homomorphic to the world, to the structure of reality (or a domain of reality). Representation consists of both numerical as well as descriptive information about the objects and systems, and is assumed to be obtained from prior experience, and or derived theoretically (based upon multiplicity of existing and possible tools of logical inference).

The phenomenon of multiple world representation presumes that homomorphisms exist between various representations.Currently, various bodies and techniques of world representation are used in the practice of control, based upon different elements and rules of information organization. In this paper, knowledge bases as a technique of world representation, are being focused upon. Knowledge Bases as a model of information system, is a system of world representation which is considered to be **homomorphic** to other systems of world representation such as systems of difference, or differential, or integral equations, and others (see Figure 9).

Nondeterministic nested referencing. Accuracy and resolution can mistakenly be understood as nondeterministic properties of representation. We will separate the characteristics of accuracy and resolution from the nondeterministic information. Let the vector of observations be

$$X_o = X_{od} + \xi$$

where X_{od}- deterministic model after recognition,

ξ -stochastic component after observation.

Definition. A component of observation is named stochastic component if it is

a) not identified (yet) with any of models stored in the knowledge base,

b) substantially larger than measure of accuracy and resolution ($|\xi| > \rho$),

c) presumed to potentially affect the results of decision-making.

Our approach differs from the classical only in a sense of (11) which implies two recommendations.

Recommendation 1. Control problem is to be solved as a deterministic problems dealing with models X_{od} and estimating likelihood or plausibility of the decision, or the policy by measure of uncertainty linked with the set of $\{\xi_i\}$.

Recommendation 2.. Learning will be understood as a tool for extracting new recognizable models from $\{ \xi_i \}$ rather than for updating knowledge of probabilistic characteristics of the set $\{ \xi_i \}$.

The following structure of dealing with unrecognized (unmodelled) information is implied by these two recommendations. Initial decomposition should be be repeated recursively for the nested structure of information. At each level the component is being decomposed in two parts: which can be recognized and included in the deterministic part of the next level, and which at the next level remains still unrecognizable (with $E[\xi_i]=0$)

$$X_{0i}= X_{odi} + \xi_{r,i+1} + \xi_i$$

where X_{odi}- deterministic model after recognition at the level i,

$\xi_{r,i+1}$-part of the stochastic component at the i-th level which will be recognized after observation at the (i+1)-th level.("trend").

ξ_i -part of the stochastic component at the i-th level which remains unrecognized, $E[\xi_i]=0$.

This recursive analysis of the stochastic information can be illustrated as follows

$$X_{0i}= X_{odi} + \xi_{r,i+1} + \xi_i \qquad X_{0,n-1}= X_{od,n-1} + \xi_{r,n} + \xi_{n-1}$$

$$X_{0,i+1}= X_{od,i+1} + \xi_{r,i+2} + \xi_{i+1} \qquad X_{0,n}= X_{od,n} + \xi_n$$

----

and n is the level where the recursion stops (no consequtive levels are expected to be built).

This decomposition of information (which is possible within the nested hierarchical structure) allows for multiple reference system. The key motivation for the multiple referencing is simplification of information representation per level. Multiple referencing is indirectly presented in the requirement that $E[\xi_i]=0$. This means that the origin is placed in a point in the state space as to provide $E[\xi_i]=0$. Then, the rest of the information allocated for decision-making at this level is referenced to these origin.

Another important implication of **multiple referencing** in dealing with nondeterministic information, is related to the topic of learning. As mentioned above, the system is supposed to deal with partially, or completely unknown world. Thus, learning is presumed. Any learned information is being identified with memory models (patterns) which determine the initial referencing. The residual information is supposed to be collected, and later it is expected to generate a new pattern upon the multiplicity of realizations. If generation of a new pattern seems to be impossible (no regularities are discovered), the change in the initial referencing might be undertaken. This philosophy of dealing with new information is to be utilized for procedures of **map updating**.

We can see also within the body of this problem of nested referencing, a direct link among the quantitative characteristics of the system and its linguistical description, and the components of this description. At this time, however, we will restrain from further statements on these links since we do not have enough factual observations.

Recognition and Identification. All of the above statements are based upon *Definition* containing undefined word "identified". In further considerations, the word "recognition" is used. Some list of *patterns* is presumed which should be used for comparison and identification. Recognition is also loosely defined but some freedom seems to be allowed for possible enhancement of the list of existing patterns. Most of these matters deserve a special and detailed treatment, they are not discussed in this paper. However our attitude toward this domain should be at least illustrated by an example.

An example of interpreting the situation with identification, recognition, and learning the new patterns of the entities of the world, is shown in Figure 10.If A is a robot location, and B is the goal, then minimum time path can be sought in one of the following areas: I, II, and III. It can be demonstrated geometrically, that the minimum distance path can be found in the area II. However, since each turn within this "slalom-type" area is linked with the losses of time (speed at the turning point should be reduced to avoid skidding), then the minimum time trajectory seems to be within one of the areas I or III.

It seems, that even before we start to compute all and possible trajectories within these areas, they can be analyzed as some entities with common properties. Indeed, area I can be described as an area which has "obstacle 1" on the right of the moving robot all over the motion, area IIi has "obstacle 2" on the left of the moving robot, and area II has obstacles on both sides. Areas I and III do not seem to be different when robot is moving far from the obstacle, but they differ substantially if we try to make the length of the trajectory as small as possible: the trajectory in the are I will have 5 linear segments and four turns while the trajectory III will have only two linear segments and only one turn.

Further analysis shows that the cost-functions have different form for all three areas. Moreover, within one particular area cost-function can be considered monotonoc in the sense of [32]. These kind of areas are named "topoways" [33], and they can be considered entities for minimum time problem solving at the higher level of planning/control system. Rules can be formulated in which the condition part will describe the relative obstacle location while the consequent part will be related to the most general description of the topoway recommended for further consideration.

Clearly, rules of this level of generality have to be recognized by a master, however they can be utilized with no master's direct participation.

On the equivalence between the difference equation model, and the production rules. Difference equations can be easily transformed into production rules, Indeed, instead of the form

$$x(k+1)= A(k)\,x(k) + B(k)\,u(k)$$

where k-is the number of the stage of recursion,
A(k) and B(k) are the matrices of system parameters,
x-is a state, and
u-is a control.

Instead of having this information in a form of an equation one can state

$$\text{IF } \{x(k)\wedge A(k)\wedge B(k)\wedge u(k)\} \to \text{ THEN } \{x(k+1)\}$$

which actually describes the causality exposed by a particular system : "if present state is given, and the parameters are known, and some definite control is applied, then the state will change as follows". In most of the real systems causalities obtained experimentally and/or from the other models of representation can be inverted in a form of rules-prescriptions [28]

$$\text{IF}\{x(k)\wedge x(k+1)\wedge A(k)\wedge B(k)\} \to \text{ THEN } \{u(k)\}$$

which can be interpreted as follows: "if present state is given, and the the following state is required, while the parameters of the system are known, then apply this control".

In some cases even the form of trajectory can be utilized for rules formulation. Indeed, the form

$$x(k) = F(k,0)\ x(0) + \sum_{l=0}^{k-1} F(k,\ l+1)\ B(l)\ u(l), \text{ and}$$

$$F(k+1,l+1) = A(k)\ F(k,\ l+1)$$

implies the following ways of subsequent planning/control activities:

1) solution trajectory on-line computation, submitting to the execution controller, and storing in memory for the subsequent use in a similar situation,
2) retrieving the previously stored solution for the same or similar conditions, obtained from off-line training, or from on-line experience.

Applying the production rules at different resolution levels. The above consideration can be expanded if the prior behavior is taken in consideration in order to estimate how the reality differs from what was expected and provide correctives for the control signal, or "tuning" for the matrices A and B (using available techniques of extrapolation).
Thus, the following sets of information can be considered initially known:

S-states : $\{x(0),\ x(-1),\ x(-2),...\}$
P-parameters : $\{A(0),\ B(0),\ A(-1),\ B(-1),\ ...\}$
G-goal states : $\{x(k),\ x(k-1),...\}$.

The general form of a control-generating rule can be represented as follows

$$\text{IF } [P\wedge S\wedge G]^* \to \text{ THEN DO } U$$

where U -is a string $\{u(1),\ u(2),...,\ u(k-1),\ u(k)\}$,
*-means that the solution is Pareto-optimal.

Solution can be found by one of the selected search procedures. There is a set of optimum solutions, the cost difference between them can be found only at higher resolution.NDP is done a the following sequence of procedures:

SUBSTITUTE THE "OUT-OF-REACH" GOAL BY AN ACHIEVABLE GOAL
FIND A CONTROL-GENERATING RULE
IF THE CONTROL-GENERATING RULE IS NOT FOUND, APPLY SEARCH

SUBMIT THE SOLUTION TO THE NEXT LOWER LEVEL
IF THE SOLUTION IS NOT FOUND, REPORT THE PREDICAMENT TO THE NEXT
UPPER LEVEL

This approach is very general and produces the trajectories of motion in all cases: when the general rule exist and is known to the planning/control system, or when it does exist but is unknown. All algorithms of path-search in 2D binary world start with a statement **IF THERE IS NO OBSTACLE BETWEEN THE ROBOT AND THE GOAL, GO DIRECTLY TO THE GOAL, ELSE DO THE SEARCH** [29]. In fact, this rule must be proven by geometrical methods (i.e. at the higher level) or foung by a search after discretization and putting the problem to the lower level. This is the way of finding the trivial trajectories A and B in Figure 11,a.After the trajectories A and B are obtained as a result of the search procedure, the corresponding strings of commands can be stored for subsequent using them as solutions in corresponding rules [37].

In the case shown in Figure 11,b we have a less trivial situation. Indeed, the lower level search can lead us in a straightforward manner to the trajectory A. However, when the back-up motion is allowed, at least one additional trajectory should be added: a backing up motion B and then motion forward C. This new opportunity immensely reduces the productivity of search. One would probably prefer to first select one of these opportunities (A or B+C) at the higher level, and then compute the actual trajectory.

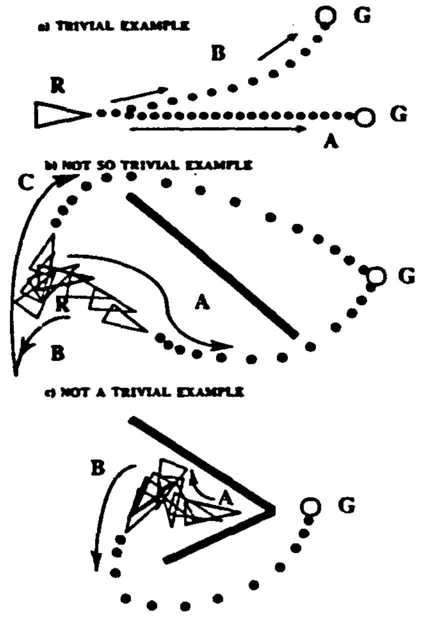
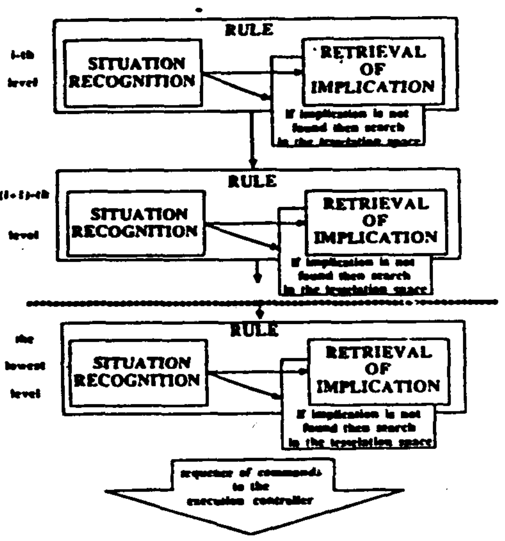


Figure 11. Examples of solving typical motions


RULE
i-th level
SITUATION RECOGNITION
RETRIEVAL OF IMPLICATION
If implication is not found then search in the templation space
RULE
(i+1)-th level
SITUATION RECOGNITION
RETRIEVAL OF IMPLICATION
If implication is not found then search in the templation space
RULE
the lowest level
SITUATION RECOGNITION
RETRIEVAL OF IMPLICATION
If implication is not found then search in the templation space
sequence of commands to the execution controller
Figure 12. Sequence of Rules-Search Application in the Nested Hierarchical Controller

```
IF ROBOT IS BECOMING CLOSE TO AN OBSTACLE (DISTANCE IS LESS THAN D)
                              (ON THE RIGHT, OR ON THE LEFT)
THEN PUT MESSAGES TO PILOT AND EXECUTION CONTROLLER TO 1 PRIORITY
AND
    IF DISTANCE IS LESS THAN D* AND MORE THAN D**
    THEN REDUCE SPEED TO V* AND MAKE A TURN
                              (LEFT, RIGHT)
    AND
        IF DISTANCE IS LESS THAN D**
        THEN STOP AND REPLAN
```

One can see that these rules do not refer to any specific global coordinates of the trajectory to be executed, and the local coordinates (position of the obstacle relative to the robot) are given with low accuracy using linguistic variables as it was done in [28]. Similar rules are formulated for a more subtle maneuver when after the local goal was determined in a particular situation, it turned out to be in an inconvenient location, e.g. the actual location of G^{**}_{i+2} (see Figure 6) cannot be achieved in a near-minimum time fashion by a simple continuation of the motion ahead.

```
IF G(X) IS LESS THAN $R^T_{MIN}$
  AND G(Y) IS LESS THEN 2 $R^T_{MIN}$
    AND ORIENTATION IS LESS THEN |90°|

THEN BACK-UP AND TURN IN THE DIRECTION OPPOSITE TO THE GOAL
```

(G(X) and G(Y) are the "x" and "y" coordinates of the local goal in a local reference frame, i.e. when the coordinate system is attached to the robot, R^T_{MIN} is the minimum radius of turn).

Tis rule allows to perform a nontrivial motion shown in Figure 11,c which reproduces a very anthropomorphic way of dealing with this problem.No wonder - it was introduced by a human "masrer".We do not have any experience in formulating rules like this automatically.It seems that using the human experience and intuition in formulating rules would be beneficial for dealing with predicaments arising during processes of operation.

The overall process of using the rules previously formulated and stores, and the search when the rule is not found in the list, is illustrated in Figure 12.

Learning of rules. Thus, if the rule is not on the list, the recommendation is obtained using the search in the state space, or in other words, by solving the automaton of representation, or by solving difference equations of representation via (say) enhanced dynamic programming [31, 33, 36]. The results of this search can be stored and later used as a rule. Then the number of rules will grow tremendously, and instead of spending time for search the system will spend time for browsing the lists of rules. Probably, there exist an optimum number of rules depending on the descriptive properties of the space as well as the characteristics of the computer system utilized in a specific intelligent robot.

On the other hand, if a group of rules can be generalized and pushed to the upper level of the hierarchy, then the overall time of search can be reduced. This generalization can be done as follows

```
    DETERMINE FEATURES OF CONDITION PARTS OF RULES IN STORAGE
    FIND SIMILARITIES AMONG THE FEATURES OF RULES IN STORAGE
    GROUP THE CONDITION PART OF RULES IN STORAGE
    GROUP THE CONSEQUENT PARTS OF RULES WITHIN THE GROUP OF
RULES
        WITH SIMILAR CONDITION PARTS
    SUBSTITUTE ALL CONSEQUENT PART BY ONE (AVERAGE) SOLUTION
    TO THE CORRESPONDING LIST OF RULES
```

The process of learning with new rules generation is illustrated in Figure 13. The stage of "generalization" can be performed so far only with "master" direct participation. Indeed, the control strings found as a response to the concrete situations should be stored but the subsequent analysis should be done by a "master" unless the reliable algorithms of generalization are developed.

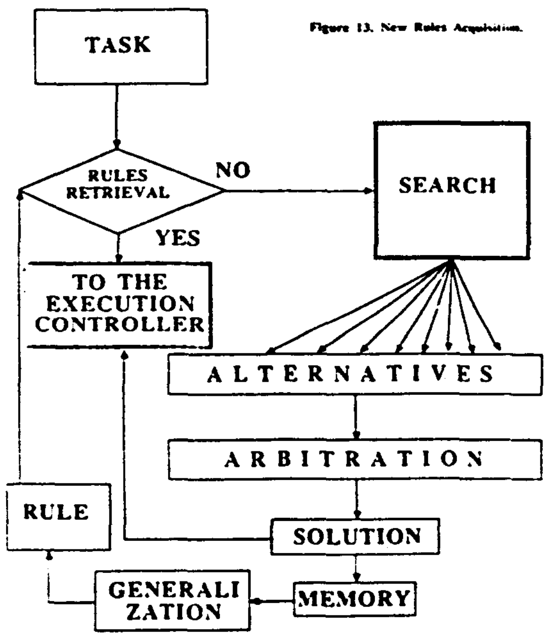
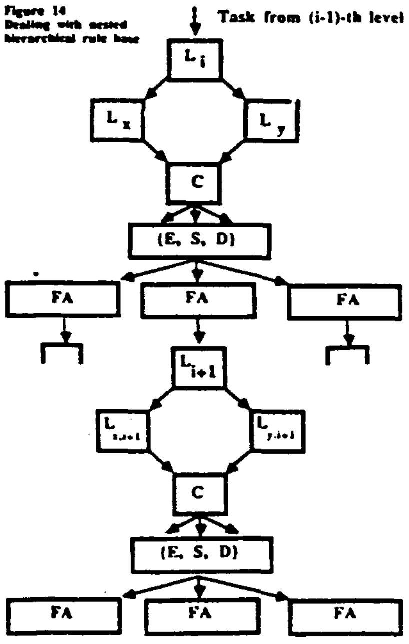


Figure 13. New Rules Acquisition.

Figure 14 Dealing with nested hierarchical rule base



In the meantime, the other types of learning (primarily, inductive learning) can be excercised with no human involvement. They include memorizing the snapshots of the world, analysing the evolution of the snapshots, and computation of correctives for the matrices A and B as well as for the string of U.

Retrieval of rules. The first experience of operation of autonomous mobile robot Drexel Dune-Buggy has demonstrated that usin the lists of rules is inefficient, and the hierarchical structure increases the productivity of the rules retrieval. The structure is shown in Figure 14. It is based upon the theory presented in [38]. L_x is the language of conditions consisting of their vocabulary and the weighted relationships among them. L_y is the language of consequents also consisting of the vocabulary and the grammar of weighted relationships among the words of this vocabulary. The matching algorithm is based upon the grammar translator consisting of the weighted relationships among the words of L_x and L_y. Tme matching process C ends with obtaining several "best" machings, then evaluation, selection, and decomposition of the proper rule is done ({E,S,D}).

After decomposition is completed, focusing of attention should pe performed (FA), after this the search of the proper rule of the next level can be performed. The process is not fully automated since the process of focusing attention not always can be done with no master involvement.

IV. Conclusions

In existing systems the function of "master" is working in the DIM-area. This presumes that master's participation is expected to be required at the stages of

context selection,
arranging the blackboards for the specific context,
assignment of the search envelope,
supervised learning.

To be more specific his functions will include (but not limited to) the following activities:

1. Making available the "universe of discourse" knowledge.
2. Proper selection of the context subset.
3. Monitoring the processes of image recognition especially while sensor integration.
4. Final judgment on the space discretization (uniform, nonuniform,etc.) based upon master's opinion on what is "meaningful neighborhood".
5. Monitoring the process of cost-function assignment.
6. Monitoring the process of vocabulary determining for the adjacent levels.
7. Elimination of discrepancies in world representation at the adjacent levels.
8. Monitoring the assignment of the relations of nesting by generalization.
9. Determining the scope of attention according to the general assignment of the operation.
10. Final decision on the duration of retaining the stored information especially for the levels of ephemeral maps.
11. Determiniming the rules of the high level of generality
12. Arbitration among the multiple alternatives.
13. Direct dealing with the problem of the lower level if the upper level vocabulary is poor.
14. Participation in the processes of new rule recognition.
15. Doing the job of linguistical tesseletion: determining the groups of entities belonging to the same level of discretization.

Expectantly, the language for communication with the "master" will be different from the language to be utilized for dealing with the other robots of the cooperative team.

V.References

1. H. P. Birmingham, F. V. Taylor, "A Design Philosophy for Man-Machine Control Systems", Proc. of the IRE, Vol. XLII, No.12, December 1954

2. N.H. Mackworth, "Researches on the Measurement of Human Performance", from Medical Research Council Special Report Series 268, London, GB 1950, Published in *Selected Papers on Human Factors in the Design and Use of Control Systems*, ed. by H. Wallace Sinaiko, Dover, New York, 1961

3. Bejczy, A.K., "Smart Sensors for Smart Hands", in Progress in Austronautics and Aeronautics, Vol. 67, Publ. AIAA, New York, 1979

4. Bejczy, A.K., "Effect of Hand-Based Sensors on Manipulator Control Performance", in Mechanism and Machine Theory, Vol. 12, Pergamon Press, 1977

5. Bejczy, A.K., Control of Remote Manipulators, in Handbook of Industrial Robotics, ed. by S. Y. Nof, Wiley, New York, 1985

6. A. Meystel, "Planning in a Hierarchical Nested Controller for Autonomous Robots", Proc of the 25-th IEEE Conference on Decision and Control, Athens, Greece, 1986

7. A. Meystel, "Nested Hierarchical Controller for Intelligent Mobile Autonomous System", Proc. of the International Congress on Intelligent Autonomous Systems, Amsterdam, Netherlands, 1986

8. A. Meystel, "Nested Hierarchical Intelligent Module for Automatic Generation of Control Strategies", Proc. of the NATO International Advanced Research Workshop on Languages for Sensor-based Control in Robotics, Ed. by U. Rembold, K. Hormann, Karlsruhe, FRG, 1986

9. A. Meystel, "Hierarchical Algorithm for Suboptimum Trajectory Planning and Control", in Recent Trends in Robotics, eds. M. Jamshidi, J.Y.C. Luh, M. Shahinpoor, North-Holland Elsevier, New York, 1986

10. A. Meystel, "Autonomous Mobile Device: A Step In the Evolution", in Applications in Artificial Intelligence, Ed. by S. Andriole, Petrocelly Books, Princeton, NJ, 1985

11. A. Meystel, "Intelligent Control of a Multiactuator System", in IFAC Information Control Problems in Manufacturing Technology 1982, ed. by D.E. Hardt, Pergamon Press, Oxford, 1983

12. A. Meystel, G. Hoffman, E. Koch, T. Looney, J. McKisson, M. Roberts, "Anthropomorphical Control of Intelligent Mobile System", Proc. of IEEE Southeastcon-82, Destin, Fl, 1982

13. G. Giralt, R. Chatila, M. Vaisset, "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots", In Robotics Research, Ed. by M. Brady and R. Paul, MIT Press, Cambridge, MA 1984 9. A.M. Parodi, J.J. Nitao, L.S. McTamaney, "An Intelligent System for Autonomous Vehicle", Proc. of IEEE Intl Conf. on Robotics and Automation, San-Francisco, CA 1986

14. A.M. Parodi, J.J. Nitao, L.S. McTamaney, "An Intelligent System for Autonomous Vehicle", Proc. of IEEE Intl Conf. on Robotics and Automation, San-Francisco, CA 1986

15. D.W. Payton, "An Architecture for Reflexive Autonomous Vehicle Control", Proc. of IEEE Intl Conf. on Robotics and Automation, San-Francisco, CA 1986

16. A. Meystel, "Knowledge-based Controller for Intelligent Mobile Robots", in Artificial Intelligence and Man-Machine Systems, ed. H. Winter, series "Lecture Notes in Control and Information Sciences, 80, Springer-Verlag, Berlin, 1986

17. G.N. Saridis, Self-Organizing Control of Stochastic Systems, Marcel-Dekker, New York, 1977

18. G.N. Saridis, "Toward the Realization of Intelligent Controls", Proseedings of IEEE, 67, August, 1979

19. G.N. Saridis, "Intelligent Robotic Control", IEEE Transactions on Automatic Control, Vol. AC-28, No. 5, 1983

20. G.N. Saridis, J.H. Graham, "Linguistic Decision Schemata for Intelligent Robots", Automatica, Vol. 20, NO 1, 1984

21. G.N. Saridis,"Foundations of the Theory of Intelligent Control", Proc. of the IEEE Workshop on Intelligent Control, Troy, NY, 1985

22. R. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ 1957

23. Y. Bar-Shalom, "Stochastic Dynamic Programming: Caution and Probing", IEEE Transactions on Automatic Control, Vol. AC-26, NO. 5, Oct. 1981

24. H. Herrlich, G.E. Strecker, Category Theory, Allyn and Bacon, Inc., Boston, 1973

25. A.N. Kolmogorov, S.V. Fomin, Introductory Real Analysis, Prentice Hall, Englewood Cliffs, NJ 1970

26. M.D. Davis, E.J. Weyuker, Computability, Complexity, and Languages, Academic Press, NY, 1983

28. C. Isik, A. Meystel, "Knowledge-based Pilot for an Intelligent Mobile Autonomous System", Proc. of the First Conference on Artificial Intelligence Applications", Denver, CO, 1984

29. E. Koch, C. Yeh, G. Hillel, A. Meystel, C. Isik, "Simulation of Path Planning For a System With Vision and Map Updating", Proc. of IEEE Int'l Conf. on Robotics and Automation. St. Louis, MO. 1985

30. R. Chavez, A. Meystel "Structure of Intelligence For An Autonomous Vehicle", Proc. of the IEEE Int'l Conf. on Robotics and Automation, Atlanta, GA, 1984,

31. A. Guez, A. Meystel, "Time -Optimal Path Planning and Hierarchical Control via Heuristically Enhanced Dynamic Programming: a Preliminary Analysis", Proc. of the Workshop on Intelligent Control, Troy, NY, 1985

32. P.E. Hart, N.J. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum-Cost Paths", IEEE Transactions on Systems, Science, and Cybernetics, Vol. SSC-4, No. 2, July 1968

33. A.Meystel, A. Guez, G. Hillel, "Minimum Time Path Planning for a Robot", Proc. of the IFEE Conf. on Robotics and Automation, San Francisco, CA, 19867

34. P. Graglia, A. Meystel, "Minimum Time Path Planning in the Traversability Space of an Autonomous Robot" Proc. of the IEEE Symposium on Intelligent Control, Philadelphia, PA, 1987

35. D. Gaw, A. Meystel, "Minimum-Time Navigation of an Unmanned Mobile Robot in a 2 1/2 World with Obstacles", Proc. of IEEE Conf. on Robotic and Automation, San Francisco. CA, 1986

36. A. Meystel, Primer on Autonomous Mobility, Drexel University, Philadelphia, PA, 1986

37. R. Bhatt, D. Gaw, L. Venetsky, D. Lowing, A. Meystel, "Dynamic Motion Guidance and Trajectory Tracking for an Autonomous Vehicle", Proceedings of the IEEE Intl. Symposium on Intelligent Control, Philadelphia, PA, 1987

38. A. Meystel, "Purposeful Arrangements via Search in the Version Space", Proceedings of the IEEE Intl. Conference on Systems, Man, and Cybernetics, Halifax, Nova Scotia, Canada, 1984

N91-17

AUTONOMOUS SYSTEMS

S28-37
108

158110

3-D World Modeling With Updating Capability Based on Combinatorial Geometry

M. Goldstein, F.G. Pin, G. de Saussure, and C.R. Weisbin
Oak Ridge National Laboratory
Oak Ridge, TN 37831

ΦA 789470

1. Abstract

This paper describes a 3-D world modeling technique using range data. Range data quantify the distances from the sensor focal plane to the object surface, i.e., the 3-D coordinates of discrete points on the object surface are known. The approach proposed herein for 3-D world modeling is based on the Combinatorial Geometry (CG) Method which is widely used in Monte Carlo particle transport calculations. First, each measured point on the object surface is surrounded by a small sphere with a radius determined by the range to that point. Then, the 3-D shapes of the visible surfaces are obtained by taking the (Boolean) union of all the spheres. The result is an unambiguous representation of the object's boundary surfaces. The "pre-learned" partial knowledge of the environment can be also represented using the CG Method with a relatively small amount of data. Using the CG type of representation, distances in desired directions to boundary surfaces of various objects are efficiently calculated. This feature is particularly useful for continuously verifying the world model against the data provided by a range finder, and for integrating range data from successive locations of the robot during motion. The efficiency of the proposed approach is illustrated by simulations of a spherical robot in a 3-D room in the presence of moving obstacles and inadequate pre-learned partial knowledge of the environment.

2. Introduction

An autonomous robot must have sensory capability to deal with unknown or partially known environments. The sensor derived data need to be processed to an appropriate internal representation of the external world. The external world to be described is fundamentally three-dimensional, involving object occlusion. Most computer vision research performed during the past twenty years has concentrated on using intensity images as sensor data. The imaging hardware (cameras) for these studies typically project a three-dimensional scene onto a two-dimensional image plane, thus providing a matrix of gray level values representing the scene from a given viewpoint. These values indicate the brightness at points on a regular spaced grid and contain no explicit information about depth. Methods that use intensity information only for deriving 3-D structure are usually computationally intensive. This computationally expensive processing arises due to the fact that correspondence of points between different views must be established and a complex system of nonlinear equations must be solved([1]-[5]).

In recent years digitized range data have become available from both active and passive sensors, and the qual'ty of these data has been steadily improving([6]-[8]). Range data quantify the distances from the sensor focal plane to an object surface. Since depth information depends only on geometry and is independent of illumination and reflectivity, intensity image problems with shadows and surface markings do not occur. Therefore, the process of representing 3-D objects by their shape should be less difficult in range images than in intensity images. The problem addressed by this paper is the external world modeling using range data. Unique requirements for such a model are:

a) Allow representation of a general 3-D unknown or partially known environment, based on range data.
b) Allow for minimal fast memory for storage.
c) Allow the introduction of a feedback loop for continuous verification of the world model against the data provided by the sensor (efficient distance calculation).
d) Allow for efficient integration of the range data from multiple views.
e) Allow for efficient navigation and manipulation.

A wide variety of techniques have been developed for representing 3-D objects for digital computing purposes. There are methods which describe the surface boundary and methods which represent the object's volume. The simplest boundary representation is using n-sided planar polygons (triangles, quadrilaterals, etc.) which can be stored as a list of 3-D node points along with their relationship information. Arbitrary surfaces are approximated to any desired degree of accuracy by using many planar polygons. This type of representation is popular because model surface area is well defined and all object operations are carried out using piecewise-planar algorithms. The next step in generality is obtained using quadric surface boundary representation. More advanced techniques for representing curved surfaces with higher order polynomials or splines are mentioned in the computer graphics and CAD literature([9]-[12]). There are many different techniques of this type; however

†On leave from the Nuclear Research Centre-Negev, Israel.

they are generally not very compact in terms of data storage, nor are they computationally efficient in calculating distances to boundary surfaces[13]. The best known volumetric representations are the oct-tree[14] generalized cylinder[15] and multiple 2-D projection views[16]. The generalized cylinder approach is well suited to many real world shapes. However, it becomes just about impossible to use this representation for large, thin objects. The multiple 2-D projection view is not a general purpose technique since different objects may have similar 2-D projections. The oct-tree representation is used in many world models. However, the indexing problem[17] is seriously affecting the efficiency of this tecnnique. In conclusion there is a need for a fast and robust technique for building 3-D models of arbitrarily shaped objects.

In Chapter 3, a proposed external world modeling procedure and an efficient distance calculation algorithm are presented. A technique for integrating the range data from multiple views and a continuous verification procedure of the world model versus the range data provided by the sensor is illustrated in Chapter 4. Finally, the feasibility of the proposed approach is illustrated in Chapter 5 by simulations of a spherical robot navigating in a 3-D room in presence of static and moving obstacles and inadequate pre-learned partial knowledge of the environment.

3. Representing 3-D Surfaces Using the Combinatorial Geometry

The basic problem addressed in this paper is one of representation. The proposed approach is based on the Combinatorial Geometry (CG) method[18] which is widely used in Monte Carlo simulation of particle transport in 3-D geometries. In CG (also known as Constructive Solid Geometry (CSG) in computer graphics and CAD literature) solids are represented as combinations of primitive solids or "building blocks" (i.e., spheres, cylinders, boxes, etc.) using the Boolean operations of union, intersection and difference. The storage data structure is a binary tree where the terminal nodes are instances of primitives and the branching nodes represent Boolean operators. Any 3-D known object can be represented by a (Boolean) combination of primitive solids or deformed superquadrics[19]. This representation is especially suitable for describing pre-learned partial knowledge of the environment. An example of describing an object composed of two boxes, one of them with a cylindrical hole is illustrated in Fig. 1. The result is a concise, unambiguous and complete representation of the object volume and boundary surface.

ORNL-DWG 86-16942

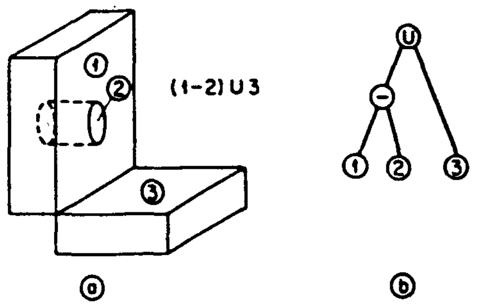


Fig. 1. Representing a 3-D object using Combinatorial Geometry.
a - given object and its CG representation.
b - the storage data tree.

The result of a range scan is a matrix of distances from the sensor focal plane to an object surface. In other words, the coordinates of discrete points on the "visible" parts of the boundary surfaces of different objects in the external world of the robot, are known. Let α be the (small) angle between two successive "reading" directions of the sensor. First, each discrete point i, is surrounded by a small solid sphere with a radius, $r_i = \max(R_i \sin \alpha, \Delta R_i)$, where R_i is the range to point i, and ΔR_i is the associated measurement error. Then, the approximate 3-D shape of the visible boundary surfaces is obtained directly by taking the union of all the spheres (see Fig. 2).

ORNL-DWG 86-16943

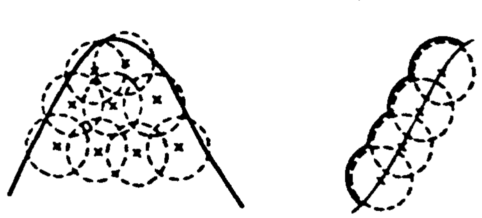


Fig. 2. Describing the shape of 3-D objects using spheres.

The reason for using spheres is to keep the representation as compact as possible. Describing the sphere for a particular discrete point in space means adding only one additional parameter (the radius) to the coordinates of the discrete point which are provided by the sensor. The radius r_i is defined as $r_i = \max(R_i \sin \alpha, \Delta R_i)$ to avoid the appearance of "holes" in the geometry and to take into account the range uncertainty. Using this definition for r_i, neighbor spheres overlap one another and the boundary surface of the union of all spheres is continuous (without holes) from the robot's point of view. Finally, it is obvious that using the "sphere" procedure, the shape of the boundary surfaces is distorted. However, the distortion is practically proportional to the range to each point since the range uncertainty is relatively small. In other words, the resolution of the model is improved as the range to the surface is decreased.

3.1. Distance Calculation in CG Representation

A very useful feature of the CG representation is its efficiency in calculating distances to 3-D surfaces in a desired direction. Observing discontinuities in the range data greater than the maximum size of the robot, the scene is partitioned in many different zones. A zone is defined as the union of small spheres located between two successive discontinuities in the range data. Using the storage data structure mentioned in Chapter 3, two tables are defined: the first one includes the spatial location of the small spheres; the second one identifies the different zones in terms of these spheres. The distance to 3-D surfaces in a desired direction from a given point, is calculated in two steps:

a) Each zone is surrounded tightly by a box (rectangular parallelepiped). Since the boxes are approximate bounding configurations, intersections of a given ray with a box does not necessarily imply intersection with any particular sphere. In addition, the different orientations of the sphere clusters imply that bounding boxes can intersect and therefore multiple boxes may have to be checked for penetration by a given ray. The box (boxes) penetrated by the ray is determined by calculating the intersection points between the boxes and the ray. A list consisting of the boxes physically penetrated by the ray, is defined. The corresponding list of zones is used to determine the penetration point.

b) Determine the small sphere penetrated by the ray and calculate the penetration point. This is done by considering only the spheres included in the zones listed in the first step. Using this approach, only a small number of spheres are checked for penetration, and therefore significant computation time is saved.

It should be mentioned that the boxes surrounding the zones are used only internally during distance calculations and they are not affecting the geometric description of the 3-D surfaces. During path planning, "tentative paths" are checked for potential collision by calculating the distances to object surfaces from scattered points on the robot's surface in the desired direction. These distances can be effectively calculated by using the CG representation, and the procedure outlined above.

4. Updating the World Model

Automatic construction of 3-D models of objects from multiple views is an important problem in computer vision. In the past, a number of different techniques have been used for representation and modeling of 3-D objects for computer vision applications([20]-[27]). However, there is an absence of a fast and robust technique for building 3-D models of arbitrarily shaped objects. The process of constructing 3-D models for objects involves integrating the range data from multiple views. In general, the integration process performs matching to establish correspondence between the views, determines the interframe transformations to register the views in a common reference coordinate system and then merges the data. The difficult and time consuming step in the above process is the matching step required to establish a correspondence. Much of the previous research efforts have been directed toward solving the difficult correspondence problem. The algorithm presented in this paper, does not require any correspondence between different views, because the world model uses a universal coordinate system with the origin arbitrarily located at the robot's initial position. According to the representation algorithm described in Chapter 3, the accuracy with which a certain point in space may be observed by the robot depends upon the distance between the robot and the point. This fact is translated to the radius of the sphere surrounding a particular point in the CG representation. Therefore, a point in space should be kept in memory along with the most accurate information (shortest observation distance). In other words, for each "measured point" in space, the shortest observation distance in the robot's history should be determined. The main problem in implementing this approach is the fact that since the sampling procedure of range data is discrete, the probability of a particular point to be sampled from two different positions of the robot is zero. In other words, each "measured point" is sampled just once during the robot history. The solution implemented in our approach follows an iterative algorithm using the "old data" acquired before the current scan and the "new data" acquired during the current scan:

a) The "old data" is checked from the current position of the robot. Using the world model based on the "old data", distances to 3-D surfaces from the robot's current position in the direction of points in the "old data" are calculated. If the distance to 3-D surfaces is smaller than the euclidean distance to the sphere surrounding the point then this particular point cannot be seen from the current position of the robot and the point representation is kept unchanged. Otherwise, the "old" radius of the surrounding sphere is compared with the "new" radius determined by the euclidean distance from the current position of the robot and the smallest radius is chosen between the old and the new radii.

b) The "new data" acquired from the current position of the robot is checked against the "old data". If the "new" point (provided by the sensor) is located within the world model based on the old data (within a sphere surrounding an "old" point) then the new point is rejected and therefore no new sphere is added to the world model.

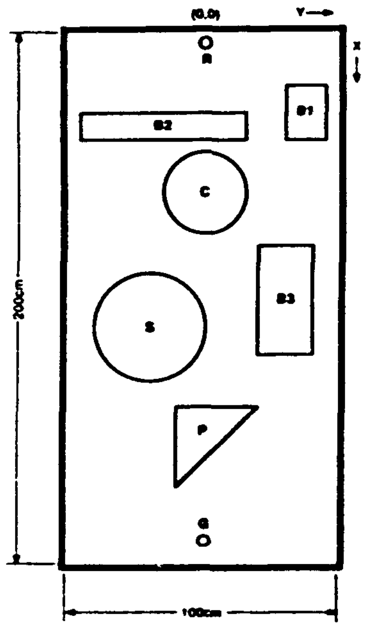
If the "new" point is outside the old world model, then the distance to 3-D surfaces in the "new" point direction is calculated (using the old world model). If the obtained distance is greater than the range to the "new" point (provided by the sensor), the "new" point is added to the world model as a sphere with a radius determined by the range to the point. Otherwise (the obtained distance is smaller than the range to the point) the "old" point (located approximately in the same direction, but closer to the robot) is erased from the model and the "new" point is added to the world model. This is the case of moving objects, in which the "old" data should be continuously verified and updated.

c) Verification of pre-learned geometric knowledge of the environment.

The sensor derived data is compared with the calculated distances obtained from scanning the pre-learned geometric environment. The pre-learned data is represented in a very concise way using the CG representation. If the "real" range in a certain direction is found to be similar (within the uncertainty of the pre-learned data) to the calculated range in the same direction, the representation is kept unchanged and no point is added to the world model. If the real range is smaller than the calculated range, the new real point is added to the world model. Finally, if the real range is greater than the calculated range, the entire pre-learned object is removed from the world model and the "real" point is added to the representation.

5. Sample Problems

The efficiency of the proposed world model is illustrated in several simulations of a spherical robot navigating in a 3-D room in presence of static and moving obstacles and inadequate pre-learned partial knowledge of the environment. The robot is assumed to move in a plane parallel to the floor, along straight lines. The origin of coordinates is arbitrarily located at the robot's starting position. The goal coordinates are known a-priori. The external world geometry, the robot starting position and the goal location are illustrated in Fig. 3. The radius of the spherical robot is 3 cm. The plane of motion is 30 cm off the floor. The navigation algorithm used in the sample problems is described in detail in Ref. 28.



R	- The initial position of the robot (4,0,30)
G	- The goal position (190,0,30)
B1	- BOX; dimensions (20 x 15 x 40); Center at (30,37.5,20)
B2	- BOX; dimensions (10 x 60 x 90); Center at (-15,35,45)
B3	- BOX; dimensions (40 x 20 x 90); Center at (100,30,45)
C	- CYLINDER; Center of basis at (60,0,0) Height 60, Radius 15
S	- SPHERE; Center at (110,-20,30) Radius 20
P	- PRISM; dimensions (30 x 30 x 90); Vertex at (140,-10,0)

Room dimensions: 200 x 100 x 100

All dimensions are in centimeters

Fig. 3. The geometry of the room.

To illustrate the efficiency of the proposed technique for building the world model, four sample problems have been considered. In the first problem the 3-D environment is completely unknown and the robot is representing the surrounding environment using the range data provided by the sensor. Figures 4-8 illustrate the plane of motion during the robot's journey from its initial position to a final position where he can directly "see" the goal. The world representation is continuously updated using the information provided by the sensor from different reading positions of the robot. It can be seen that as the robot proceeds to the goal the world representation becomes more complete.

In the second problem (Figs. 9-13), the box B1 and the prism P are provided a-priori to the robot (pre-learned knowledge). The robot is verifying the accuracy of the pre-learned information and after finding it correct, is representing the two objects using two CG primitives (Box and Prism), without using the sphere type of representation. The representation of the overall 3-D world is thus more concise than in the first problem, in which the sphere procedure was used to represent all objects.

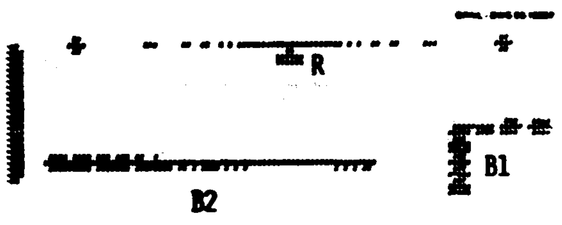


Fig. 4

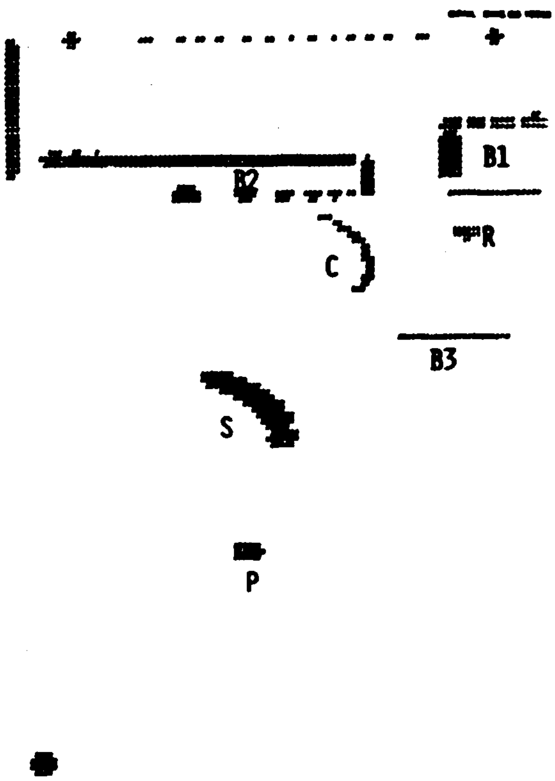


Fig. 5

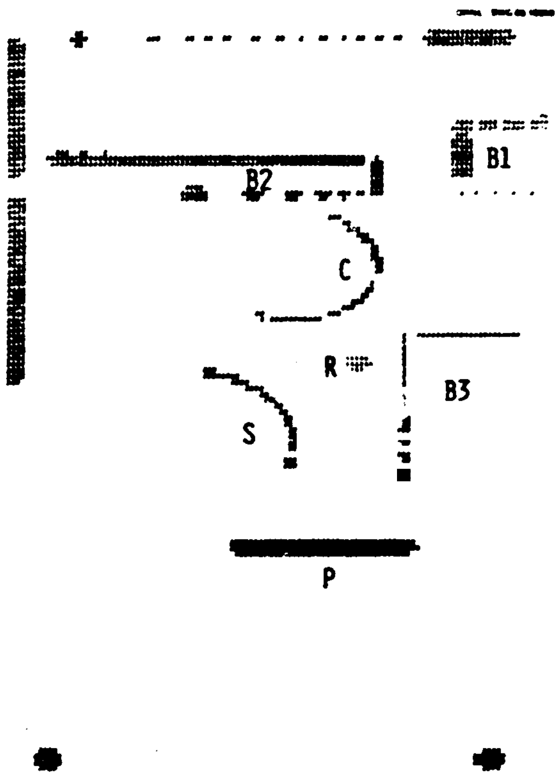


Fig. 6

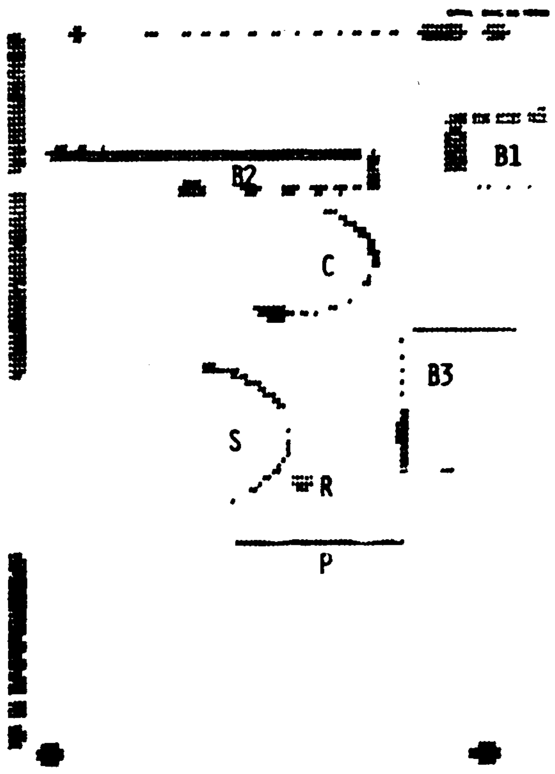


Fig. 7

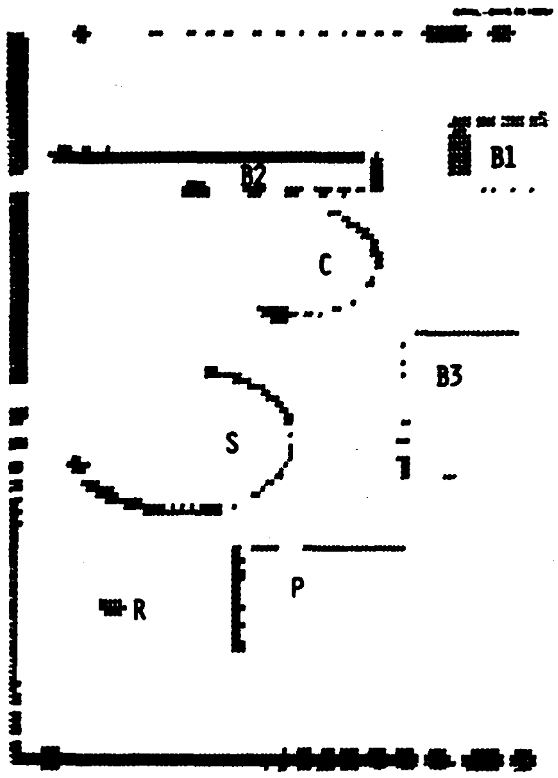


Fig. 8

The external world geometry considered in the two last problems is similar to the geometry of the first two problems, except that the boxes B1, B2 and the cylinder C are removed from the scene. In the third problem the box B3 and the prism P are defined as pre-learned information which (intentionally) was provided inaccurately to the robot. From Figs. 14-17 it can be seen that the robot is verifying the pre-learned data and finding that the box is inaccurately positioned (Fig. 14) is using the sensor (real) data only to represent it (Figs. 15 and 16). At a later stage, (Fig. 17) the robot can directly check the pre-learned information for the prism (which was previously occluded) and finding it incorrect is removing the prism from memory. The prism is then accurately represented using the data provided by the sensor.

In the last example, the box and the prism are correctly provided as pre-learned information, and the "unknown" sphere is moving forward and backward between successive positions of the robot. Figure 18 illustrates the environment with the sphere at its initial position. While the robot is moving to the second position (Fig. 19) the sphere is moving forward. The previous information about the sphere is then checked, found incorrect and removed from the robot's memory. Finally, when the robot is reaching the next (third) position, the sphere has moved back to its initial position. It can be seen (Fig. 20) that the robot is keeping the previous information about the sphere, since it is now occluded by the "real" data and therefore cannot be verified. If at a later stage the robot is again in a position to directly "see" the "old" position of the sphere, this previous information will be checked and eventually removed from the world model.

These and the following figures shown in this paper have been produced using a computer printer and a very simple plotting routine. Since the maximum resolution of the printer along the Y axis (across the page) is 130 characters, certain existing spheres having diameters smaller than the printer resolution are not printed and therefore some "holes" may artificially appear on surfaces which are in fact continuous.

6. Conclusion

The proposed approach for modeling the external world using the Combinatorial Geometry was found promising. The range data from successive locations of the robot during motion can be effectively combined and given an adequate world representation. The pre-learned knowledge and moving objects in the scene can be effectively verified and represented in the world model. The computation time per "picture" including the simulated range scan, modeling the geometry, trajectory planning and plotting the plane of motion was 30 s to 1 min CPU time of VAX-8600 Computer, depends upon the scene complexity and the number of tentative paths considered. More than 50% of the computation time is used for plotting the plane of motion and for calculating distances in a given

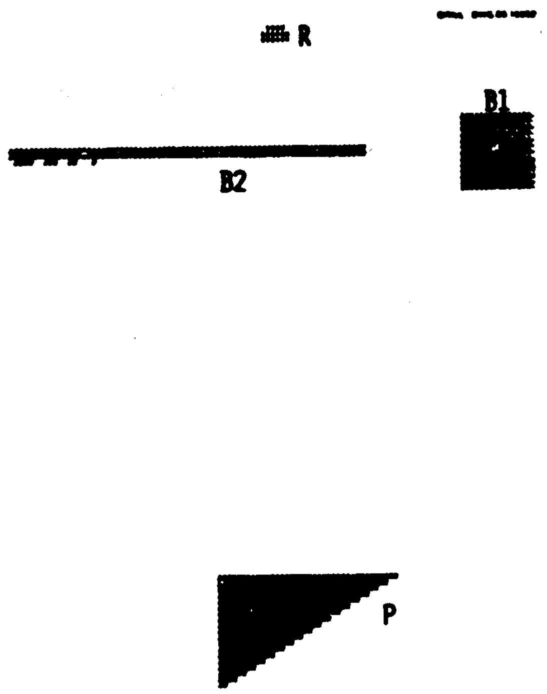


Fig. 9

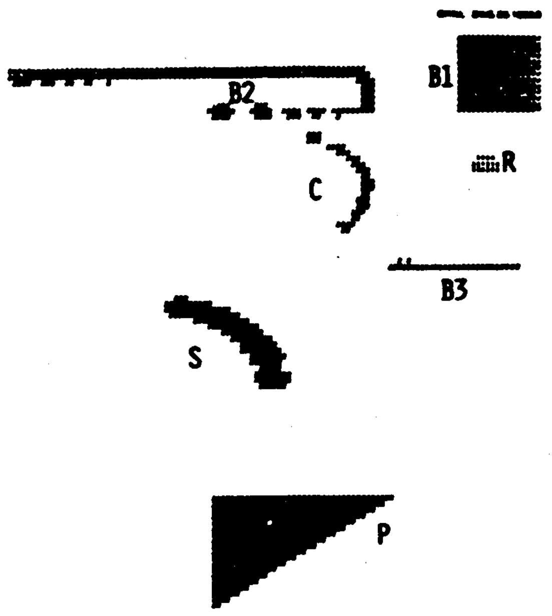


Fig. 10

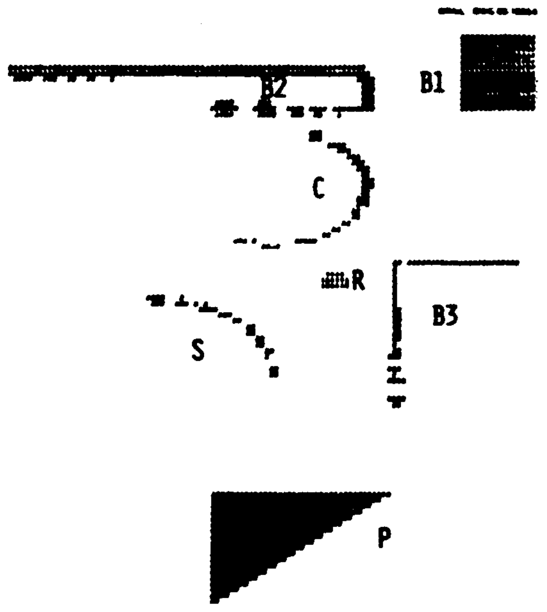


Fig. 11

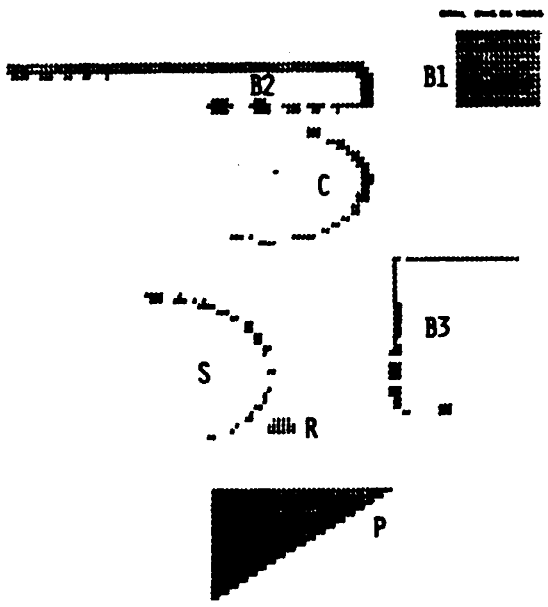


Fig. 12

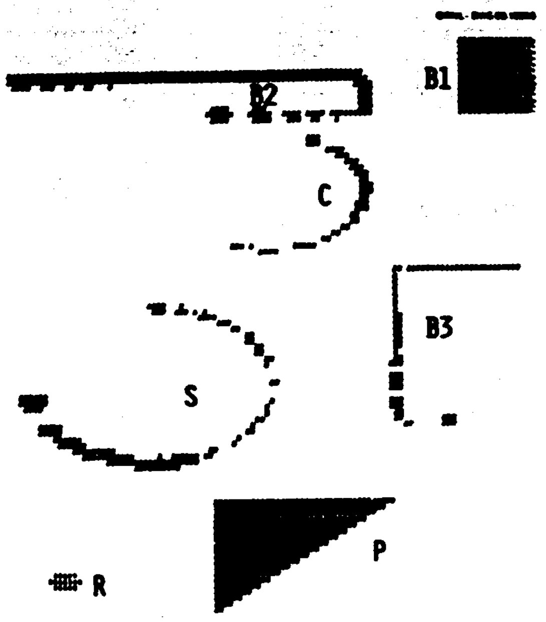


Fig. 13

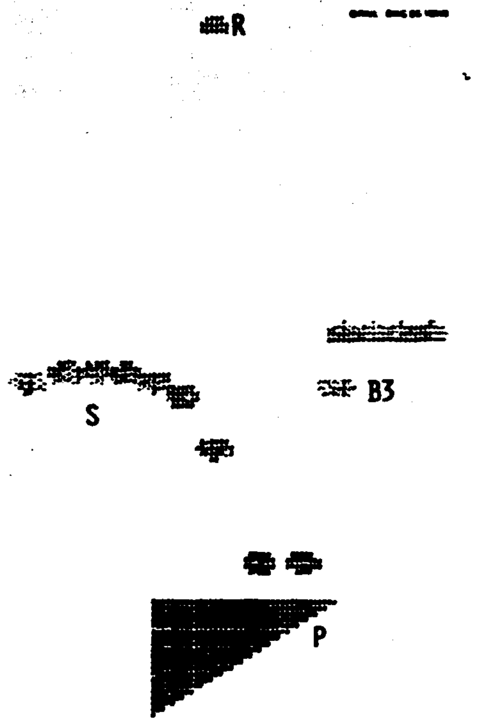


Fig. 14

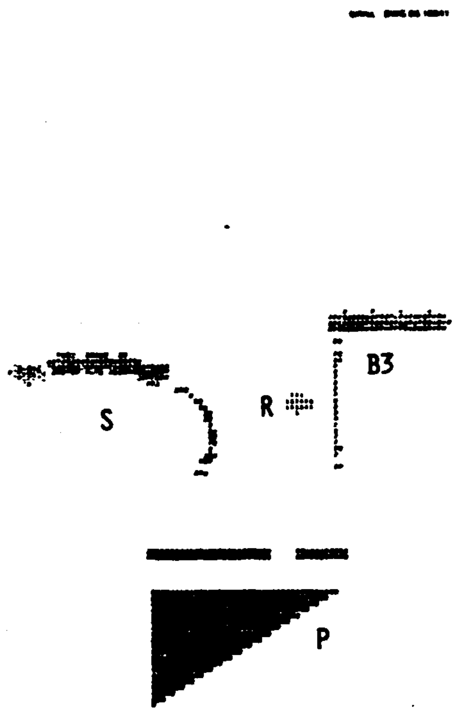


Fig. 15

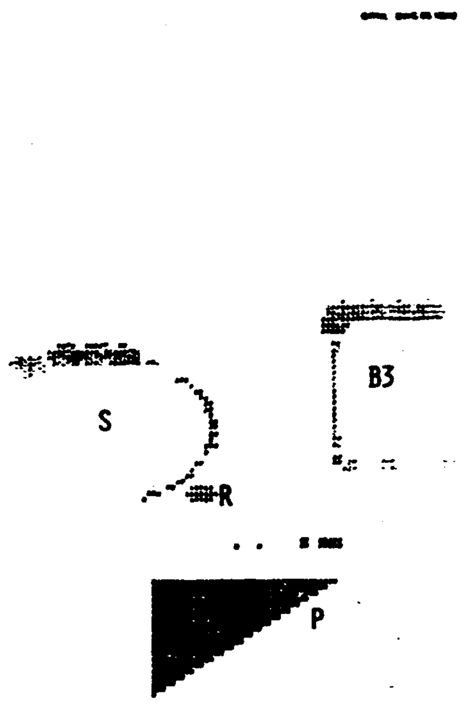


Fig. 16

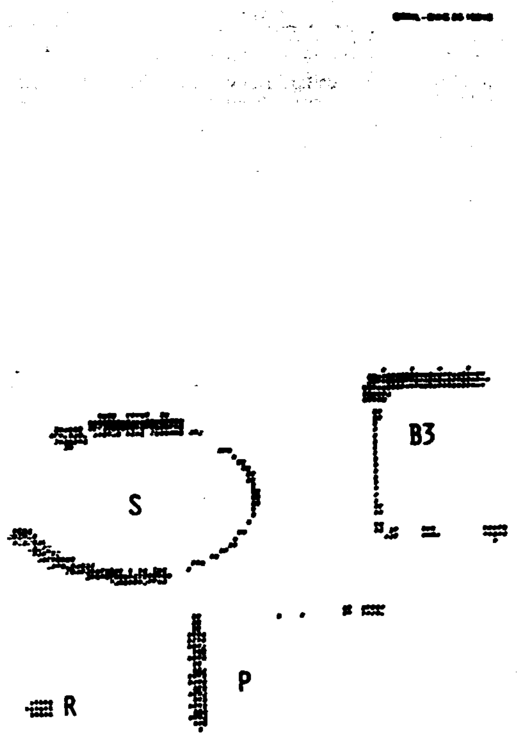


Fig. 17

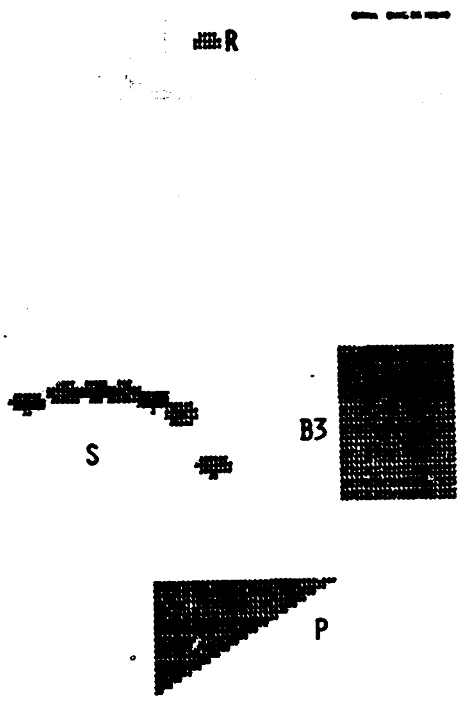


Fig. 18

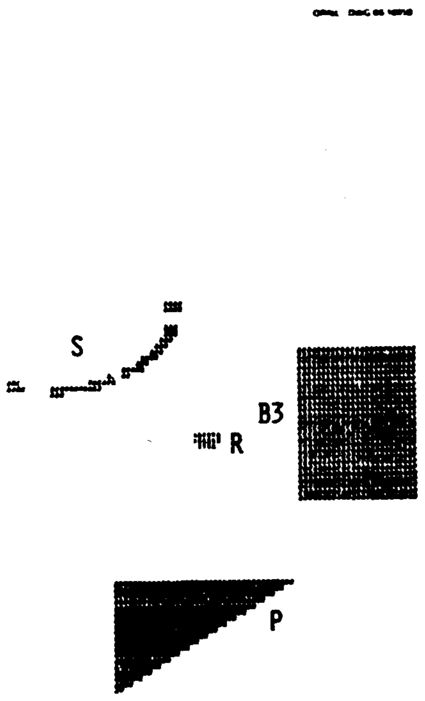


Fig. 19

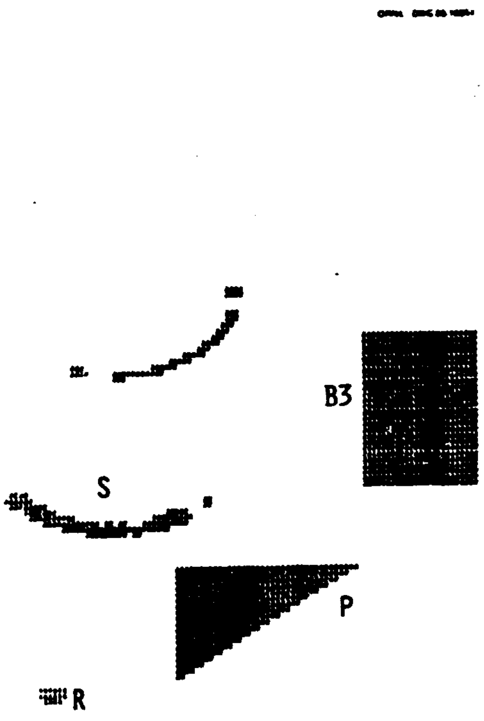


Fig. 20

direction from discrete points. These calculations can be executed independently and therefore, performing the same calculations on a parallel or concurrent computer may significantly reduce the computation time. Future work using the proposed external world modeling approach will focus on the following issues: scene segmentation into objects, feature point extraction, recognition of 3-D objects from range data, replacing the sphere representation with a more concise CG volumetric representation of the recognized objects and finally implementation of this method on the NCUBE Machine and experimental verification using the HERMIES-II robot.

7. Acknowledgements

The first author is grateful to the Nuclear Research Centre-Negev, Israel for the support and encouragement during the preliminary stage of this work. The authors gratefully acknowledge the support provided by Dr. O. Manley of the Department of Energy Office of Basic Energy Sciences, Dr. H. Alter of Office of Nuclear Energy, Dr. D. Bailey of Spent Fuel Management and Reprocessing Systems and to Karen Harber for preparing the manuscript. This research was performed under DOE contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

8. References

[1] R. O. Duda and P. E. Hart, "Pattern Classification and Scene Analysis," Wiley Interscience, New York (1972).
[2] S. Ullman, "The Interpretation of Visual Motion, MIT Press, Cambridge, Massachusetts (1979).
[3] J. W. Roach and J. K. Aggarwal, "Determining the Movement of Objects from a Sequence of Images," IEEE Trans. Pattern Anal. Machine Intell., PAMI-2. 554-564 (November 1980).
[4] H. H. Nagel, "Representation of Moving Rigid Objects Based on Visual Observations," Computer 14, 29-39 (August 1981).
[5] H. P. Moravec, "Rover Visual Obstacle Avoidance," in Proc. 7th Int. Conf. Artif. Intell., Vancouver, Canada, August 24-28, 1981.
[6] M. J. Magee and J. K. Aggarwal, "Determining Motion Parameters Using Intensity Guided Range Sensing," in Proc. 7th Int. Conf. Pattern Recogn., pp. 541-583, Montreal, Canada, July 30-August 2, 1985.
[7] D. Laurendeau and D. Poussart, "3-D Model Building Using a Fast Range Finder," IEEE Proc. of Comp. Vision and Pattern Recogn., pp. 424-426 (June 1986).
[8] M. Hebert, "Outdoor Scene Analysis Using Range Data," Proc. 1986 IEEE Int. Conf. on Robotics and Automation, 3, 1426-1432 (April 1986).
[9] R. T. Farouki and J. K. Hinds, "A Hierarchy of Geometric Forms," IEEE Comp. Graphics Appl. 5, 51-78 (May 1985).
[10] I. D. Faux and M. J. Pratt, "Computational Geometry for Design and Manufacture," Ellis Horwood, Chichester, U.K. (1979).
[11] W. Tiller, "Rational B-Spline for Curve and Surface Representation," IEEE Comp. Graphics Appl. 3, 61-69 (November 1983).
[12] T. W. Sedenberg and D. C. Anderson, "Steiner Surface Patches," IEEE Comput. Graphics Appl. 5, 23-36 (May 1985).
[13] P. Besl and R. Jain, "An Overview of Three-Dimensional Object Recognition," The University of Michigan, RSD-TR-19-84 (December 1984).
[14] D. J. Meager, "Efficient Synthetic Image Generation of Arbitrary 3-D Objects" IEEE Conf. Proc. Patt. Recogn. and Image Proc., pp. 473-478, 1982.
[15] J. G. Agin and T. O. Binford," Computer Description of Curved Objects," Proc. IJCAI-3, 629-640, 1973.
[16] J. J. Koenderink and A. J. Van Doorn, "Internal Representation of Solid Shape with Respect to Vision," Biological Cybernetics 32, 211-216, 1979.'
[17] G. B. Smith and T. M. Strat, "A Knowledge-Based Architecture for Organizing Sensory Data, Proc. of Int. Conf. Intelligent Autonomous Systems, pp. 106-117, Amsterdam, The Netherlands, December 1986.
[18] W. Guber, "The Combinatorial Geometry Technique for Description and Computer Processing of Complex Three-Dimensional Objects," MAGI Project 6915, MR-7004/2 (March 1970).
[19] A. P. Peutland, "Perceptual Organization and the Representation of Natural Form," AI Journal. 28(2), pp. 1-33, June 1986.
[20] B. Bhanu, "Representation and Shape Matching of 3-D Objects," IEEE Transactions on Pattern Analysis and Machine Intelligence, May 1984, PAMI-6(3), 340-351.
[21] B. Bhanu, T. C. Henderson and S. Thomas, "3-D Model Building Using CAGD Techniques," Proceedings CVPR, 1985, San Francisco, CA, pp. 234-239.
[22] M. Potmesil, "Generating Models of 3-D Objects by Matching 3-D Surface Segments," Proc. 8th IJCAI, August 1983, Karlsruhe, West Germany, pp. 1089-1093.
[23] F. P. Ferrie and M. D. Levine, "Piecing Together the 3-D Shape of Moving Objects: An Overview," Proceedings CVPR, 1985, San Francisco, CA, 574-584.
[24] B. A. Boyter and J. K. Aggarwal, "Recognition with Range and Intensity Data," Proc. of the Workshop on Computer Vision: Representation and Control, 1984, Annapolis, Maryland, pp. 112-117.
[25] M. J. Maggee, B. A. Boyter, C. H. Chien and J. K. Aggarwal, "Experiments in Intensity Guided Range Sensing Recognition of Three-Dimensional Objects," IEEE Trans. on Pattern Analysis and Machine Intelligence, November 1985, PAMI-7(6), pp. 629-637.
[26] B. A. Boyter and J. K. Aggarwal, "Recognition of Polyhedra from Range Data," IEEE Expert, Spring 1986 1(1), pp. 47-59.
[27] B. C. Vemuri and J. K. Aggarwal, "3-D Model Construction from Multiple Views Using Range and Intensity Data," Proc. CVPR, 1986, Miami Beach, Florida, pp. 435-437.
[28] M. Goldstein, F. G. Pin, G. de Saussure and C. R. Weisbin, "3-D World Modeling Based on Combinatorial Geometry for Autonomous Robot Navigation," to be published in Proc. of 1987 IEEE Int. Conf. on Robotics and Automation, March 1987.

S29-63
68.
188111

On Autonomous Terrain Model Acquisition by a Mobile Robot

N.S.V. Rao and S.S. Iyengar
Louisiana State University
Baton Rouge, LA 70803

C.R. Weisbin
Oak Ridge National Laboratory
Oak Ridge, TN 37831

L52?2337
ΦA 78440

ABSTRACT

In this paper we consider the following problem: A point robot is placed in a terrain populated by unknown number of polyhedral obstacles of varied sizes and locations in two/three dimensions. The robot is equipped with a sensor capable of detecting all the obstacle vertices and edges that are visible from the present location of the robot. The robot is required to autonomously navigate and build the complete terrain model using the sensor information. We establish that the *necessary* number of scanning operations needed for complete terrain model acquisition by any algorithm that is based on 'scan from vertices' strategy is given by $\sum_{i=1}^{n} N(O_i) - n$ and $\sum_{i=1}^{n} N(O_i) - 2n$ in two and three dimensional terrains respectively, where $O = \{O_1, O_2, \cdots, O_n\}$ is the set of the obstacles in the terrain, and $N(O_i)$ is the number of vertices of the obstacle O_i.

Keywords and Phrases:

Path Planning, Terrain Acquisition, Collision Avoidance.

1. INTRODUCTION

In recent years there has been an enormous amount of research activity generated in the area of *navigation and path planning* for mobile robots. Much of this work could be thought of as an offshoot of the pioneering works of Lozano-perez and Wesley [1], Reif [2], Schwartz and Sharir [3], and O'Dunlaing and Yap [4]. In this work the robot is located in a terrain whose model is precisely known. A path ha to be planned to navigate a robot from a specified point to a specified destination point (if such path exists). A comprehensive survey of these and related techniques for robot path planning is available in Whitesides [5]. Another important problem is the *navigation in unexplored terrains*. Here the robot is equipped with a sensor with which the robot scans the terrain, and a navigation path is planned based on these sensor readings. In general several sensor operations are needed for planning a navigational course. Lumelsky and Stepanov [6] present nice solutions to a restricted version of this problem. Iyengar et al [7] and Rao et al [8] present a technique that utilizes the sensor readings to construct a world map through *incidental learning*. Oommen et al [9] presents a more formal treatment for the case of convex polygonal obstacles. In these approaches the terrain model acquisition is purely *incidental* i.e., the construction of the terrain model is only secondary and scanning is performed for the purpose of navigation.

Another important problem in the navigation in unexplored terrains is the *Terrain Acquisition Problem* in which the robot is required to autonomously navigate and build the complete terrain model through the sensor readings. In this paper we consider the following version of terrain acquisition problem: A point-sized robot M is placed in a two/three dimensional obstacle terrain O. The terrain O is populated by the set of obstacles $\{O_1, O_2, \cdots, O_n\}$, where O_i is a polyhedron. We assume that O is finite, i.e., O can be inscribed in a circle/sphere of finite radius in two/three dimensions. Furthermore each O_i is finite and had a finite number of vertices. Initially the sizes and locations of the obstacles are totally unknown to the robot. The robot M is equipped with an ideal sensor system capable of detecting all edges and vertices visible to the robot from its current position. The robot is required to autonomously navigate in the terrain and acquire the *complete* obstacle terrain model, i.e. obtain the locations of all edges and vertices of each obstacle of O. The main motivation for this problem stems from the fact that after terrain acquisition phase, the future navigation of the robot can be carried out without sensor operations using the techniques for navigation in known terrains. In many cases navigational path can be made optimal in terms of the distance to be traversed by the robot.

A solution to this problem is given by Rao et al [10] based on the incremental construction of the visibility graph of the terrain. The same technique is extended to a finite-sized robot in plane by Rao et al [11]. The algorithm of [10] is guaranteed to acquire the complete terrain model in finite time. The algorithm terminates when a scan operation is performed from each vertex of every obstacle and consequently the number of scanning operation required is $\sum_{i=1}^{n} N(O_i)$, where $N(O_i)$ is the number of vertices of the obstacle O_i. However, this is only a sufficient condition on the number of scan operations. In this paper we establish that for any terrain acquisition algorithm (based on scan from vertex strategy) there exists a terrain O such that the necessary number of scan operations is given by $\sum_{i=1}^{n} N(O_i) - n$ and $\sum_{i=1}^{n} N(O_i) - 2n$ respectively for two and three dimensional terrains. In other words, no more than one (two) scan operations per obstacle can be skipped in two (three) dimensional terrains. We also show that a strategy that randomly skips one vertex (two vertices) per obstacle will not acquire the complete terrain model in two (three) dimensional terrains. We then list a number of issues for future research.

The organization of the paper is follows: In section 2, we briefly discuss the issues involved in the terrain acquisition problem and also the algorithm of [10]. In section 3, we present the bound on the necessary number of scan operations.

2. TERRAIN ACQUISITION METHODOLOGY

During the terrain acquisition the robot M is required to plan and execute a *navigational course*; robot stops at certain points, called the *sensing points*, on the path to carry out the scan operations. The terrain model is reconstructed by integrating the scanning information obtained from the individual scan operations. In general, the navigational path could only be planned in an incremental manner by utilizing the scan information because the terrain is unexplored. The main requirement on the terrain acquisition algorithm is that the *complete* terrain model should be acquired in a finite amount of time.

Here we deal with *vertex-based* terrain acquisition methods where the sensing points are always vertices of the obstacles, i.e., every scan operation is performed from an obstacle vertex. The robot M moves from vertex to vertex during the navigational course. The algorithm of [10] is based on this strategy. There are two key issues that are important for a terrain acquisition algorithm:

(a) Computing the next vertex to be visited,

(b) Detecting the completion of terrain acquisition (termination of the algorithm).

We now briefly discuss the terrain acquisition algorithm of [10]. Let $VER(O_i)$ denote the set of vertices of O_i. Let $V = \bigcup_{i=1}^{n} VER(O_i)$ be the set of all vertices of the obstacles. The *visibility graph* of the terrain O, denoted by $VG(O)$, is a graph (V,E), where an edge $(v_1,v_2) \in E$, $v_1,v_2 \in V$, exists if and only if (v_1,v_2) is either an edge of an obstacle or v_1 is visible from v_2 and vice versa. In Fig.1. an obstacle terrain populated by three obstacles O_1, O_2 and O_3 is shown and its visi-

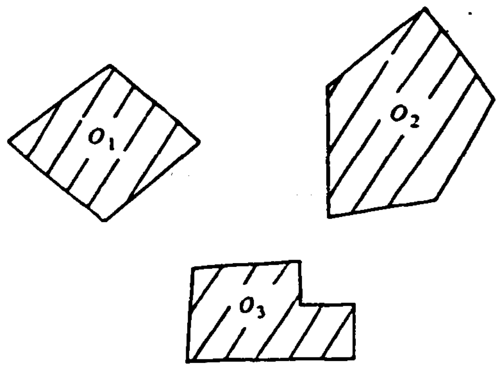


Fig. 1. Obstacle terrain

bility graph is shown in Fig.2. A vertex is said to be *explored* if a scan operation is performed from v, and otherwise v is said to be *unexplored*. Once v is explored then the adjacency list of v in the visibility graph is known. The robot M is initially placed at a point in the obstacle terrain. Then M scans and moves to a vertex. From this point the terrain acquisition algorithm, called algorithm ACQUIRE, of [10] is invoked. Let M start at vertex $v_0 \in V$. A scan is performed and the adjacency list of v_0 is stored. Then M moves to an adjacent unvisited vertex and recursively applies this method. When an unexplored vertex is visited it is pushed onto a stack called *path-stack*. Let M be located at a vertex v from which it performed a scan operation. Then M moves to a nearest unexplored vertex adjacent to v if one exists. The M can move to this chosen vertex in a straight line because it is seen from v. If all the vertices adjacent to v are visited then the path-stack is used to obtain the next sensing point. The top of the path-stack is recursively popped till a node v_1 with unvisited adjacent nodes is found. Shortest paths to all the unvisited adja-

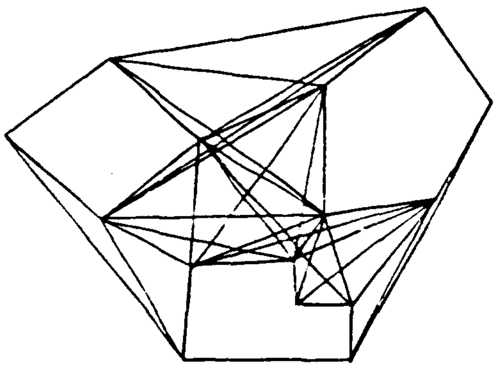


Fig. 2. The visibility graph for the terrain of Fig.1.

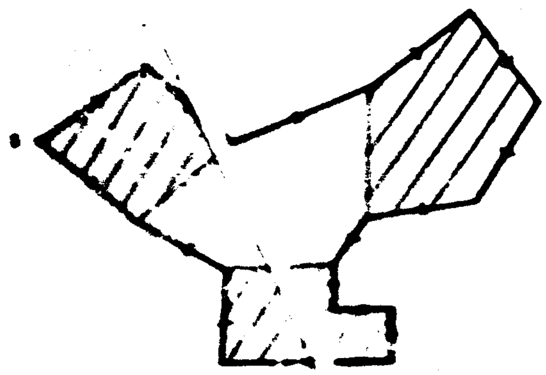


Fig.3. Navigational path taken [illegible] terrain acquisition

[illegible] the starting [illegible] in dark [illegible] of [illegible] using [illegible] shortest path algorithm on the [illegible] graph. The path of minimum distance is chosen among [illegible] paths, and M moves to the appropriate vertex [illegible] the algorithm is recursively applied. In Fig [illegible] the path taken by M is shown when M [illegible] the terrain of Fig.1. In this example the M moves [illegible] the starting point. An [illegible] stage of terrain [illegible] is shown in Fig.4.

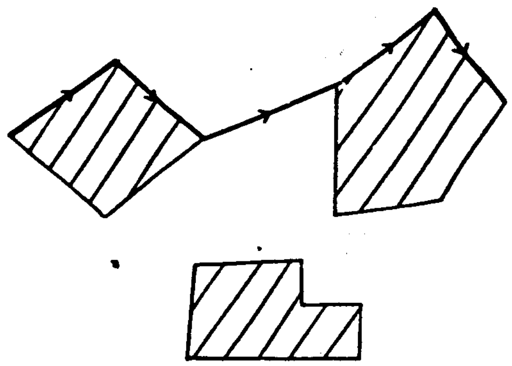
The correctness of the algorithm [illegible] demonstrating the following.

1) The visibility graph [illegible] there exists a path between [illegible]

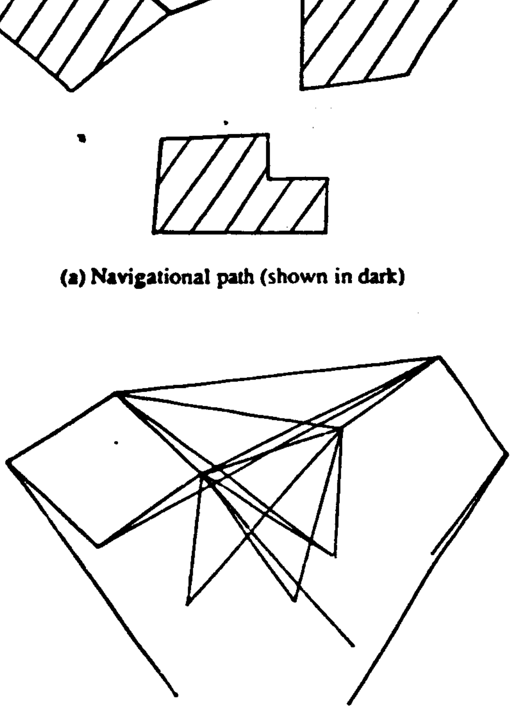
2) The manner in which [illegible] of the obstacles are explored [illegible] out a depth first search [illegible]

The [illegible] all vertices are explored in a [illegible] as a depth first search of a con[illegible] all vertices in finite amount of time). [illegible] the free space is seen by at least one scan operation [illegible] on a vertex. Thus the terrain acquisition will be complete in finite amount of time. Clearly the number of scan operations carried out by M is $\sum_{i=1}^{n} N(O_i)$. This is only a sufficient condition on the number of scan operations.

In the next section we show that for any vertex-based terrain acquisition algorithm there exists a terrain such that the necessary number of scanning operations is given by $\sum_{i=1}^{n} N(O_i)-n$.



(a) Navigational path (shown in dark)

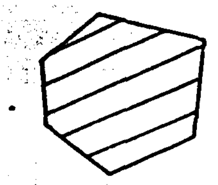


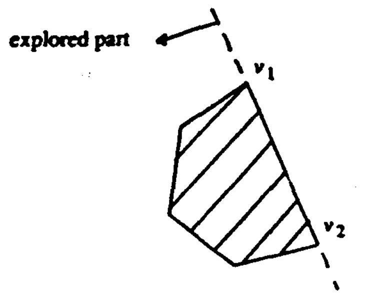
(b) Partially built visibility graph

Fig.4. Intermediate stage of exploration

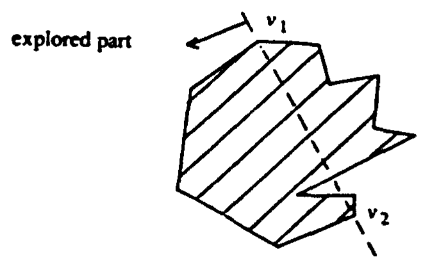
3. NUMBER OF SCAN OPERATIONS

Consider a vertex-based terrain exploration algorithm (and algorithm of [10] is one such). The algorithm performs scans and detects newer vertices which will be explored in subsequent scans. During terrain exploration by a vertex based algorithm no more than *one* vertex per obstacle can be left unexplored in two dimensional terrain constructed as explained below. For three dimensional terrains no more than two vertices per obstacle can be left unexplored in our specially constructed terrain. The basic idea is illustrated in Fig.5. We consider a single convex polygonal obstacle in Fig.5(a). If M starts at a vertex it detects one new vertex with one exploration (except when the first vertex is explored) of a vertex as the robot moves along the circumference of the obstacle. In other words at no point of time the terrain acquisition could be declared complete if there are two unexplored vertices say v_1 and v_2. This is because the robot does not, in general, know what lies on the hinder (unexplored) side of the line joining v_1 and v_2. There could a single vertex or a number of edges on the other side of the line joining v_1 and v_2 as in Fig.5 (b) and (c). For three dimensional terrains, no more than *two* vertices per obstacle can be left unexplored. This is because if three vertices (say v_1, v_2 and v_3) are left unexplored then the information on the hinder side of the plane formed by the vertices v_1,v_2 and v_3 is not known in general. The hidden side of the obstacle can be either a simple plane or composed of a a number of planes as shown in Fig.6 (a) and (b).


(a)

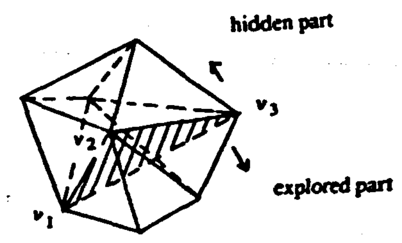


(b)



(c)

Fig.5. Two dimensional case



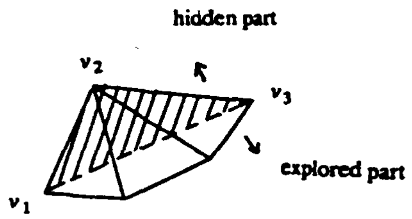


Fig.6. Three dimensional case

Theorem 1:

For a vertex-based terrain acquisition algorithm and given positive integer n there exists a terrain $\{O_1,O_2,\cdots,O_n\}$ of n polyhedral obstacles such that the necessary number of scan operations is

$\sum_{i=1}^{n} N(O_i)-n$ for two dimensional terrain

$\sum_{i=1}^{n} N(O_i)-2n$ for three dimensional terrain

Proof: We use induction on the number of obstacles in the terrain. Consider $n=1$. In two dimensional terrains consider a convex polygon as in Fig 5(a). Note that from a vertex v_2, we can only see two vertices that are adjacent to v. Apart from the first scan, no more than one unexplored vertex can be seen in any scan operation. From the discussion above M has to carry out scanning till no more than one vertex is unexplored. Thus $N(O_1)-1$ is the necessary number of scan operations for two dimensional terrains. By similar arguments we can show that the necessary number of scan operation is $N(O_1)-2$. Hence the claim is true for $n=1$.

Assume that the claim is true for $n=k$. There exist a terrain of k obstacles with the necessary number of scan operations given in the theorem. Now construct a terrain of $k+1$ obstacles as follows: In two dimensions add a big polygon O_{k+1} outside the circle inscribing the terrain that satisfies the induction hypothesis as shown in Fig.7. The $k+1$th polygon has a long edge joining v_1 and v_2 that obscures the remaining edges of the polygon from the scan operations carried out in the terrain of k obstacle. Thus the scan operations needed during the exploration of the $k+1$th obstacle is $N(O_{k+1})-1$. Hence total number of necessary scan operations for two dimensional terrains is given by $\sum_{i=1}^{k+1} N(O_i)-(k+1)$. For three dimensional terrains the obstacle O_{k+1} is such that a plane formed by three vertices v_1, v_2 and v_3 obscures the rest of the obstacle from a scan in the terrain of k obstacles as in Fig.8. The O_{k+1} lies outside the sphere the encloses the terrain of k obstacles. Using the arguments similar to two dimensional case we can show that the necessary number of scan operations to acquire O_{k+1} is $N(O_{k+1})-2$. Thus the theorem follows by mathematical induction. □

In the above theorem we have seen that no more than one (two) vertices per obstacle can be left unexplored in two (three) dimensional terrain. The natural question is to ask if we can always skip one (two) vertices per obstacle for two (three) dimensional terrains. The answer is no as the vertices

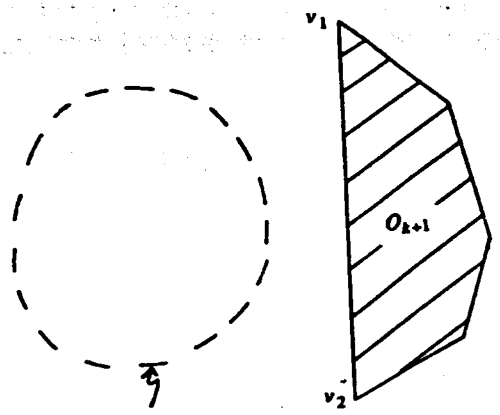


Fig.7. Two dimensional case - Addition of O_{k+1}

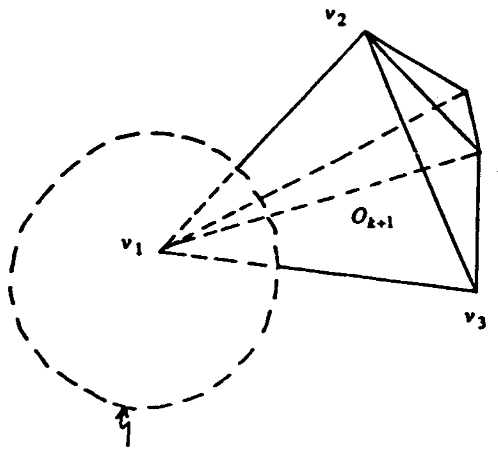


Fig.8. Three dimensional case - Addition of O_{k+1}

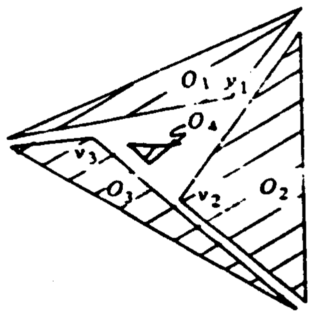


Fig. 9. Configuration - two dimensional case

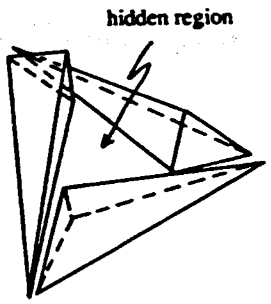


Fig. 10. Configuration - three dimensional case

are to be randomly skipped. This is illustrated in Fig. 9 and Fig. 10. In two dimensions the if the robot skips the vertices v_1, v_2 and v_3 then the obstacle O_4 will not be detected. Fig.10 shows a three dimensional example. The configurations such as shown in Fig.9 and 10 can be formed with any (finite) number of obstacles which could be other than triangles or tetrahedrons. Fig.11 shows one such example. It is open at this point to design a vertex-based terrain acquisition algorithm (or show algorithm does not exists) that skips one (two) vertices for each obstacle and guaranteed to acquire the complete obstacle terrain model.

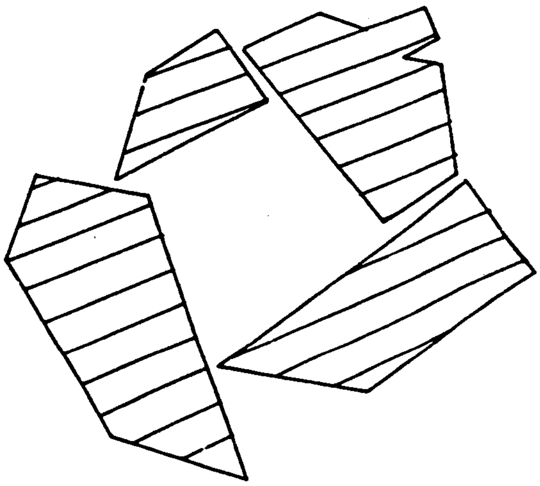


Fig. 11. A general configuration

4. CONCLUSIONS

In this paper we have shown that for any vertex-based terrain acquisition algorithm there exists a terrain such that the necessary number of scan operations is given by $\sum_{i=1}^{n} N(O_i)-1$ and $\sum_{i=1}^{n} N(O_i)-2$ respectively for two and three dimensional obstacle terrains. In other words, we do not expect to design a vertex-based terrain acquisition algorithm that has complexity lower than the above stated sums (in terms of the sensor operations). There exists a terrain acquisition algorithm with the number of sensor operations given by $\sum_{i=1}^{n} N(O_i)$ [10]. It would be interesting to see if there exists a terrain acquisition algorithm with only the necessary number of sensor operations given in this paper.

REFERENCES

[1] LOZANO-PEREZ, T., and M. A. WESLEY, An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles, Commun. ACM, Vol. 22, No. 10, October 1979, pp. 560-570.

[2] REIF, J., Complexity of the mover's problem and generalizations, *Proc. 20th Symp. on Foundation of Computer Science*, 1979, pp. 421-427.

[3] SCHWARTZ, J.T., and M. SHARIR, On the piano movers' problem I: The special case of a rigid polygonal body moving amidst polygonal barriers, **Communications Pure Applied Mathematics**, Vol. 36, 1983, pp. 345-398.

[4] O'DUNLAING, C., and C. YAP, A 'Retraction' method for planning the motion of a disc, **Journal of Algorithms, Vol.6, 1985, pp.104-111.**

[5] WHITESIDES, S.H., Computational Geometry and Motion Planning, *Computational Geometry*, Editor G.T. Toussaint, Elsevier Science Publishers B.V. (North-Holland), 1985, pp 377-427.

[6] LUMELSKY, V.J., and A.A. STEPHANOV, Effect of Uncertainity on Continuous Path Planning for an Autonomous Vehicle, *Proc. 23rd Conf. on Decision and Control*, Las Vegas, Nevada, December 1984.

[7] S.S. IYENGAR, C.C. JORGENSEN, S.V.N. RAO and C.R. WEISBIN, Robot Navigation Algorithms Using Learned Spatial Graphs, **Robotica**, Vol. 4, 1986, pp. 93-100.

[8] N.S.V. RAO, S.S. IYENGAR, C.C. JORGENSEN and C.R. WEISBIN, Robot Navigation in an Unexplored Terrain, **Journal of Robotic Systems**, Vol. , 1986, pp.

[9] B.J. OOMMEN, S.S. IYENGAR, N.S.V. RAO, R.L. KASHYAP, Robot Navigation in Unknown Terrains Using Learned Visibility Graphs. Part I: The disjoint Convex Obstacle Case, **IEEE Transactions on Robotics and Automation**, 1987, to appear.

[10] N.S.V. RAO, S.S. IYENGAR, B.J. OOMMEN and R.L. KASHYAP, Terrain Acquisition by Point Robot Amidst Polyhedral Obstacles, *Proc. The 3rd Conf. on Artificial Intelligence Applications*, Orlando, FL, Feb. 22-28, 1987, to appear.

[11] N.S.V. RAO, S.S. IYENGAR, C.C. JORGENSEN and C.R. WEISBIN, On Terrain Acquisition by a Finite-sized Mobile Robot in Plane, *Proc. of 1987 The IEEE Int. Conf. on Robotics and Automation*, Raleigh, North Carolina, March 30 - April 3, 1987, to appear.

S30-60
12p
188112

SOLON: An Autonomous Vehicle Mission Planner

M.J. Dudziak
Martin Marietta Baltimore Aerospace
Baltimore, MD 21220

MI 2/11/2/2

Presented at the Workshop on Space Telerobotics,
Jet Propulsion Laboratory, Jan. 20-22, 1987

S30-60
128
188112

I) Introduction:

The SOLON Planner provides an architecture for effective real-time planning and replanning for an autonomous vehicle. The acronym, **SOLON**, stands for: **State-Operator LOgic MachiNe**; the highlights of the system, which distinguish it from other AI-based planners that have been designed previously, are its hybrid application of a state-driven control architecture and the use of both schematic representations and logic programming for the management of its knowledge base (1).

SOLON is designed to provide multiple levels of planning for a single autonomous vehicle which is supplied with a skeletal, partially-specified mission plan at the outset of the vehicle's operations. This mission plan consists of a set of objectives, each of which will be decomposable by the planner into tasks. These tasks are themselves comparatively complex sets of actions which are executable by a conventional real-time control system which does not perform planning but which is capable of making adjustments or modifications to the provided tasks according to constraints and tolerances provided by the Planner.

The ~~Our~~ current implementation of the SOLON is in the form of a real-time simulation of the Planner module of an Intelligent Vehicle Controller (IVC) on-board an autonomous underwater vehicle (AUV). The simulation is embedded within a larger simulator environment known as ICDS (Intelligent Controller Development System) operating on a Symbolics 3645/75 computer (2).

II) Real-Time Operational Context

Many planning systems have been developed over the years of AI research, but few have been built expressly for use in the control of an autonomous vehicle which will be operated in extremely remote, inaccessible environments (3). Most autonomous vehicles to date have been land-based,and the focus of research has been upon problems of mobility and obstacle avoidance. We recognize that the AUV problem includes all of the same issues faced in the design of any autonomous vehicle but in our case long-range goals dictate the need for modularity and interchangeability of system components (e.g., sensor subsystems, manipulators, photographic equipment, etc.) and the integration of these subsystems with the controller responsible for moving the actual vehicle. Furthermore, an onboard IVC must be capable of performing in real time, without the typical options that are generally available to terrestial vehicles. An AUV has additional directional degrees of freedom than a terrestial vehicle as it moves through 3-space but it has less operational freedom to stop or to undo certain motions. For example, with the exception of certain experimental-only vehicle architectures, an AUV cannot retrace its path exactly and bringing the vehicle to a halt cannot make use of braking systems or high degrees of friction.

The underlying philosophy of the SOLON planner includes the following basic premisses:

i) Both the knowledge base and the work (the processing) must be distributed, allowing for concurrent processing by different components, in order to solve the real-time bottleneck;

ii) The architecture must be dynamic and flexible to allow for the fact that different environmental and decision-making situations will require a different balancing of the processing loads;

iii) Any intelligent controller must evolve in its design from the simple to the complex

and that vehicle-specific and mission-specific requirements which are not presently foreseeable should not require major restructuring of a system design or implemented, operable software.

III) High-Level Architecture

SOLON is composed of five logical processes (logprocs) which operate as cooperative but autonomous agents in the Planner. These are:

i) *Strategist* - central decision-making; selection and scheduling and detailing of objectives and tasks within objectives;

ii) *Event Assessor* - processing of events and conditions reported by other IVC modules about outside environment or internal vehicle systems;

iii) *Tactician* - interfacing between Planner and lower-level vehicle control systems responsible for driving actuators and effectors;

iv) *Plan Simulator* - evaluating alternative plans in cases where Strategist's first-cut logic cannot derive a clear and distinct best choice;

v) *Plan Registrar* - central editing and modification of the Active Plan.

Each logproc communicates with others by means of an message-passing system which is also the structure employed for inter-modular communications within the whole IVC. The logprocs may be multiple processes which share a single-processor machine such as the Symbolics or they may be implemented in a distributed processing environment where each logproc has its own physical processor. Their relationships are described in a data flow diagram *(Figure 1)* and their functions are described in more detail later in this paper.

IV) Distribution of Intelligence and Decision-Making

A major stumbling block to the implementation of real-time AI systems has been in the area of performance. Typically a system is bound with achieving a particular goal and events occur which cause one of the following types of situations:

i) The original problem that the system is working upon is no longer relevant

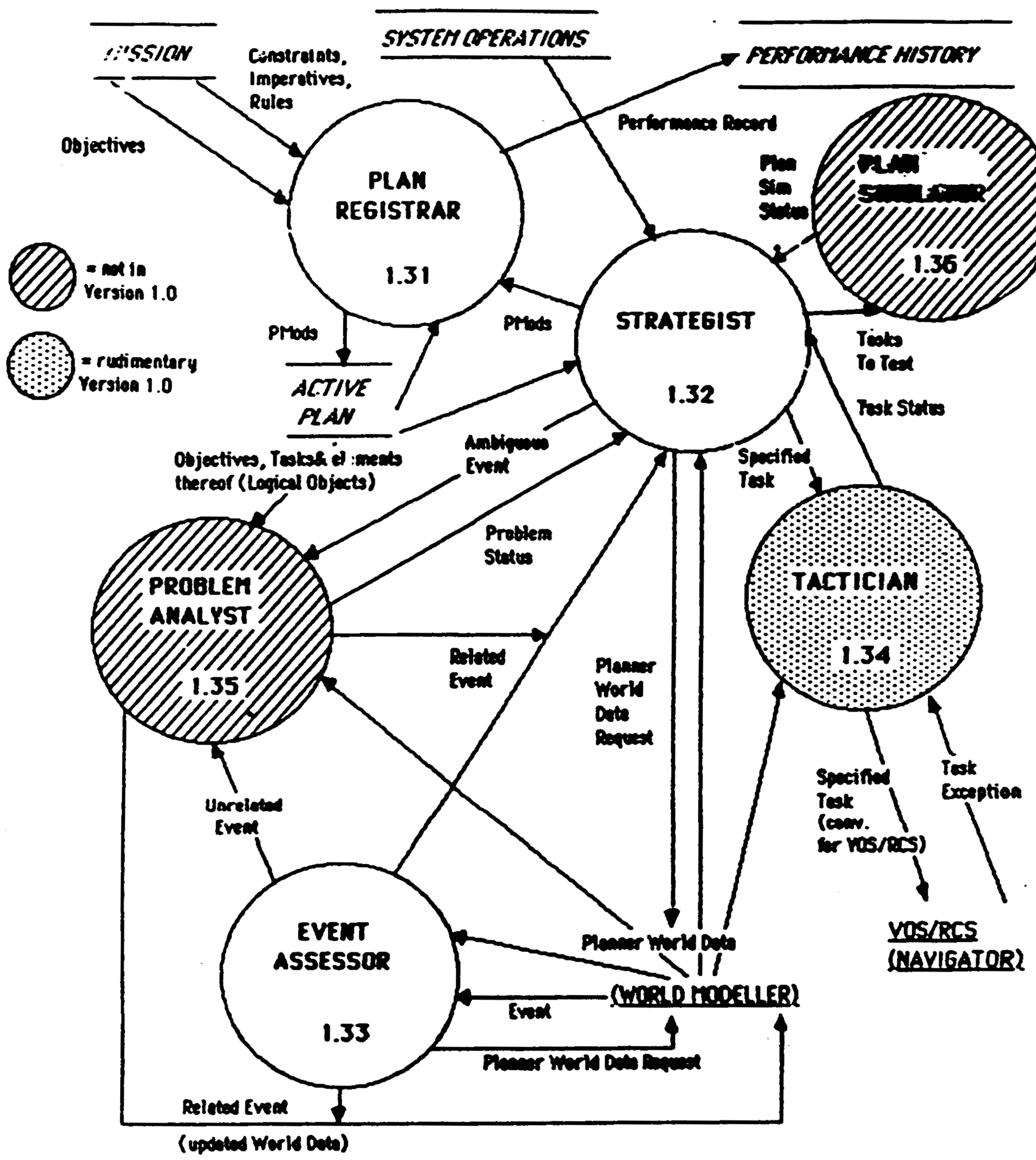
ii) The data being used by the system in its current problem-solving has been changed

iii) A new problem has higher priority and should take precedence over the current activities of the system.

This is more than a problem of conflicting goals in which the system must decide to satisfy one. Rather, we are faced with goal conflicts that may not arise until after a given attempt to satisfy a goal is underway and which may not be communicated to the system to satisfy the goal. The cost in time and computation resources for determining how to alter the current activity and how to respond to a new situation may be prohibitive given the other requirements of the control system.

The approach undertaken in SOLON distributes the work among the multiple logical processes which can proceed independently. The performance of the Plan Simulator is not impacted by the Strategist, which may be in an idle state or else busy running PROLOG inferences, other than by the constraints imposed by the scheduler if both logprocs are running off the same physical processor. Obviously there is signficant advantage to a logproc having its own processor. At any given time, messages can be sent between logprocs in order to interrupt or provide additional information. There are a finite number of message types (potential-obstacle, task-completed, relevant-object, task-status, etc.) which may be

FIGURE 1 - High-Level Planner Architecture



sent between logical processes but the schema-based representation structure allows for this number to grow as the complexity of the Planner implementation increases.

When a message is sent, for instance, from the Tactician to the Strategist, the message is sent with the priority that has been assigned by the sender. This indicates how significant or urgent the sender believes the message to be. That will not necessarily be the way the message is treated by the receiver, which may be working on a task that is the result of a previous message and which may be rated with a much higher priority.

When the receiver gets a message, it is not automatically interrupted from its work. Instead, each process follows a cycle of execution followed by checking for messages. This has been implemented for the Strategist and is described further in section X. If the receiver is at a point where it can be interrupted, then it will determine if there is any message in its queue that demands a jump to a new task. The receiver process uses the Process-Msg function to determine what message is the most critical one in the queue. Currently Process-Msg is a LISP function that employs several simple rules but in future implementations it will be a compact expert system in its own right within the Planner.

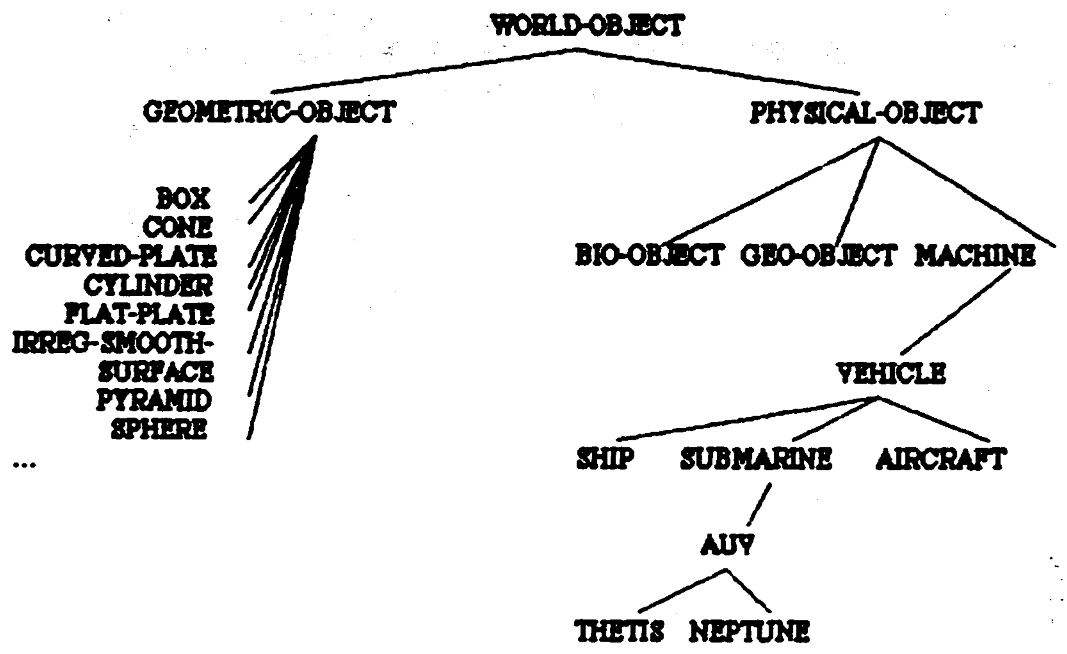
What becomes of the work that a logproc was doing if and when it is redirected by a message to tackle some other task? There are two possible approaches. One is to interrupt the original task, save a history of what was going on and possibly resume it at a later time. However, this poses many problems in truth maintenance and non-monotonic reasoning because the situation that gave rise to the original task may be altered as a result of the new operations being executed. Rather than attempt to keep track of all changes and determine that the original task should be resumed or not, the approach in SOLON is to simply drop the original task altogether. If in fact the original task should be resumed, then the conditions which led to that task will surface again in the form of a message from one logproc to another. Otherwise the system will deal with the highest priority messages that currently exist. The Planner is always responding to the latest, most current state of affairs, even if this means doing some extra work or duplicating a few steps to determine what should be done next. In other words, one may ask the question "What is the most pressing business right now?" more often, but one will always have the best and most up-to-date answer to that question, given the rules that have been built into the Planner through its functions and specific PROLOG axioms.

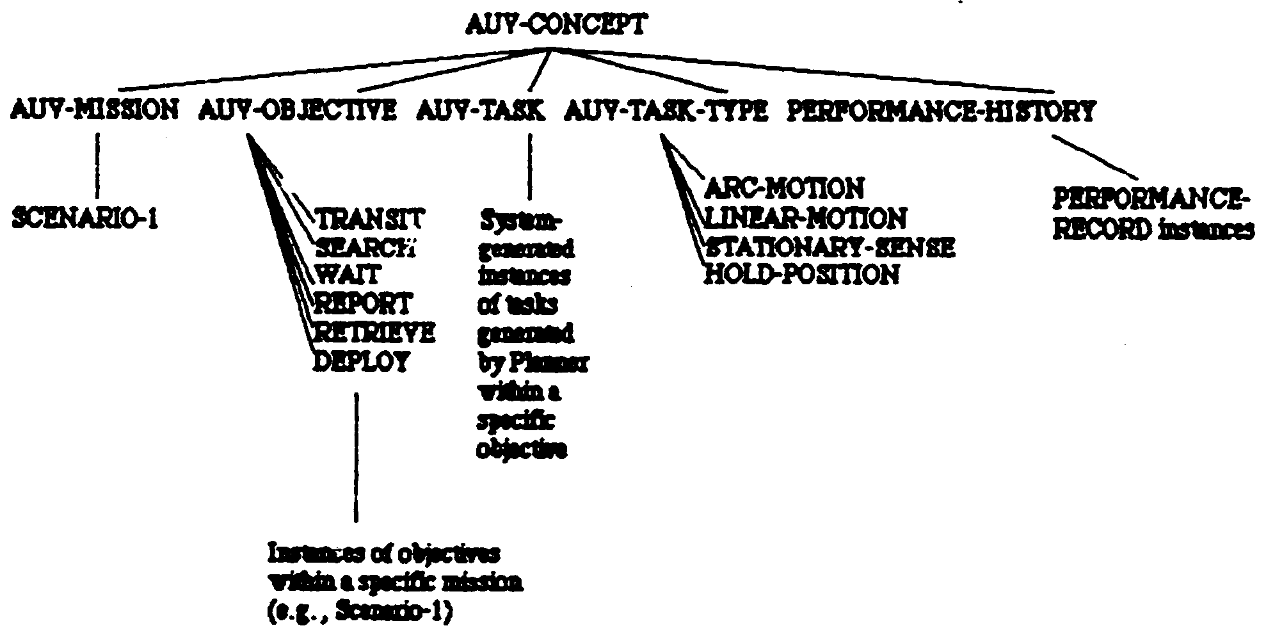
VI) Multiple Forms of Knowledge Representation

Factual knowledge of the environment and the vehicle is maintained in a database which employs the schema-slot-value structure as implemented in Knowledge Craft, a Common LISP based system tool developed by Carnegie Group, Inc. There is a fundamental taxonomy of world objects and relations which hold between various objects, as indicated in *Figure 2*. Generally, knowledge about "what is" is stored in this schematic representation, whereas "how to" knowledge, rules and operational guidelines, is represented in PROLOG axiomatic expressions. The PROLOG mechanism was deemed to be suitable for making queries and deriving solutions that could best be found using backward-chaining inference processes. However the PROLOG form was not deemed sufficient to be used as the only form of knowledge representation.

The power of the schematic representation is that the facts are easily stored in a well-structured format and the information is easily accessible. Moreover, from a

FIGURE 2 - Fundamental Objects and Relations in the SOLON World




AUV-CONCEPT
AUV-MISSION
AUV-OBJECTIVE
AUV-TASK
AUV-TASK-TYPE
PERFORMANCE-HISTORY
SCENARIO-1
TRANSIT
SEARCH
WAIT
REPORT
RETRIEVE
DEPLOY
System-generated instances of tasks generated by Planner within a specific objective
ARC-MOTION
LINEAR-MOTION
STATIONARY-SENSE
HOLD-POSITION
PERFORMANCE-RECORD instances
Instances of objectives within a specific mission (e.g., Scenario-1)

experimentalist point of view it is a good representation for building a prototypical system, one that is open-ended and intended to evolve beyond one's current expectations.

Vehicle missions consist of human-specified skeletal plans, the elements of which are known as objectives - high-level types of operations such as transiting to a destination, searching a region for a given set of objects, following a moving object, etc. Objectives are deocmposable into tasks, more elementary types of activity (transit to waypoint, hold position, send transmission, manipulate arm or sensor platform, etc.). Mission specifications are, therefore, short program-like structures that provide the Planner with a first-cut set of objectives to perform in a given temporal sequence with some parameters of those objectives detailed and others left for the Planner to address at run-time. The sequence of objectives and the parameters (e.g., bounds of a region to search, starting position for a search operation) are the default or first-choice values for the Planner to use as optimal guidelines. Through the built-in logic of the Planner and rules that are specified within the mission (see next section) the sequence and parameters and the choice of tasks for satisfying each objective are modified in response to events that occur both in the external environment and within the vehicle.

VII) Hierarchy of Operational Rules

A considerable database of operational rules is implemented using PROLOG. Certain rules are applicable to any mission for any type vehicle and these may be considered the most basic planning rules, modification of which constitutes a redesign of the planning algorithms. These rules are stored in a structure called Act-Prolog and are always present in the working set of PROLOG axioms available for inferencing operations. Other rules are specific to the vehicle but apply to any mission that such a vehicle might undertake. They are known as Sys-Oper rules and will also be available during the entirety of a given mission but may be replaced at pre-deployment time without disturbing the Act-Prolog body of rules. Obviously, this type of partitioning is principally for the benefit of easy system maintenance and modification and does not affect the actual Planner operations.

A similar body of rules are global for a given mission and are accessible at all times during the Planner operations for any objective that is a component of the mission. Next there are three classes of rules which are not accessible uniformly; these include:

i) rules that apply during a specific type of mission objective; i.e., a special rule for search operations, which does not apply to transit or escort operations;

ii) rules that apply for a specific type of task within an objective; i.e., a special rule that applies to 'waiting' tasks and does not apply to motion-oriented tasks;

iii) rules that apply for a specific objective within a mission; i.e., a special rule for when the vehicle is searching area B for a sunken object and only for that objective, having no applicability to other objectives within that mission.

The purpose of this subdivision of rules is to allow efficient entry of new rules into the system and to manage the PROLOG inferencing process so that the system never has to deal with more than those rules which could possibly apply at a given instance. The aim is to reduce the size of the rulebase wherever possible, reducing the number of rules that must be examined and eliminating from consideration those rules that could not possibly have any relevance at a given instance.

VIII) The Strategist

The Strategist is clearly the most complex logical process within the Planner, and it is within the Strategist that the state-operator architecture is employed most fully. It is also the logical process that makes most extensive use of the message processing and evaluation logic which is a critical element in the SOLON design. Messages which arrive from other Planner logprocs are processed according to an algorithm which attends to both the sender's and the receiver's evaluation of the message priority and significance for the current state of the vehicle's mission plan. The algorithm we have initially employed is a simplified version of what we expect will evolve into an expert system in its own right, a clear place-holder for future machine learning studies. Depending on the message chosen to address, the Strategist moves into one of several state-operator functions or processes (currently all are implemented as LISP functions) and future actions of the Strategist are then governed by two factors:

i) The results of function evaluations and hypothesis generations within the current state-operator;

ii) The appearance in the Strategist's message-queue of a "critical message" demanding immediate attention and the overriding of current replanning activities.

Examples of current state-operators include:

i) Analyze-Next-Scheduled-Objective
ii) Expand-Objective-Into-Tasks
iii) Examine-Task-Status-Msg
iv) Determine-Plan-Change-Directive

Within each state-operator, the main activities consist of determining what is the appropriate hypothesis to test and then making queries into the PROLOG-based knowledge base. This consists of a dynamic set of axioms (modifiable facts and constant rules). Through the PROLOG mechanism employed within our implementation (CRL-PROLOG) these have full access to the major body of represented knowledge about the vehicle, mission plan and environment, which is in the form of schema-slot-value-relation data structures.

The basic algorithm of the Strategist top-level function is presented in *Figure 3*.

FIGURE 3 - Strategist Top-Level Algorithm

```
[Outer loop]
    IF (Critical Message Received)
    THEN (Make the Critical Message the Current Message)
    ELSE (Get the Most Relevant Message From the Queue if there is one)

    Based on the type of message [related-event, obstacle, task-status, etc.]
    received, (Select Appropriate State-Operator To Activate)

    IF (Strategist is in wait-state)
    THEN (Put Strategist process into wait state)
    ELSE
        [Inner loop]
            UNLESS (Strategist is directed into wait-state)
                OR (Critical Message Received)
            (Execute the Selected State-Operator)
```

IX) A Working Example

The operation of the Strategist may best be described through the use of an example scenario which will illustrate the manner in which the state-operator functions and PROLOG rules operate to control the vehicle and replan its mission. We will consider a simple search mission as described by *Figure 4*. The tentative path of the AUV, were it to execute each of the initial scheduled objectives in sequence and without any replanning due to new events, is indicated by *Figure 5*.

Initially the Strategist is given a start-mission message which activates the state-operator Analyze-Next-Scheduled-Objective. The obvious first step is to consider what is next on the list of scheduled objectives given by Mission Control. A series of Prolog queries are generated to determine if there are sufficient reasons for undertaking this objective and no overriding reasons to avoid this objective at this time.

Assume that the first objective is to move the vehicle to a given point A. This objective is represented as Transit-A and it inherits all the default characteristics associated with transit operations, as well as any special rules that may have been specified by Mission Control in the skeletal plan. These defaults include the decomposition of the objective into component tasks. For point-to-point transits, there is only one elementary task, that of moving the vehicle in 3-space. However, before asserting a new transit-type current-task, checks are made to determine that there are no known obstacles (which may have been detected by the sensors) in the vehicle's path. Using an algorithm which treats potential obstacles as expanded spheres and represents the vehicle as a point, lines of tangency are computed which provide possible new waypoints for the vehicle to use in navigating around the obstacles. An elementary set of rules determines which is the best set of alternative paths and these paths become the new component tasks for the current objective.

Suppose that the vehicle is now engaged in a search objective, where there are a specified number of objects (cylinders or curved surfaces with a radius > 3m but < 10m) which are deemed relevant to the search and one particular object (a pipeline on the seafloor) which is the goal object for the search. Specific rules provided with the mission plan indicate the actions to be taken if and when various relevant objects are detected. These are in the database of rules which are active while the vehicle is executing its search objective. Having received a message about a cylindrical object at point A from the World Modeller, the Event Assessor determines that the object is relevant to the current objective and the Strategist is notified. The latter responds by determining an appropriate change that applies to the current state of the Active Plan. This response, based upon rules input with the mission specification, may be to initiate a new objective, a spiral search operation centered upon the newly discovered object. Before actually changing activities, the Strategist invokes a state-operator which explores "what if" type queries to determine if instituting the change of plan would cause conflicts for other objectives of the mission. If there are significant conflicts, then the Strategist returns to the state-operator charged with determining an appropriate change of plan given the new event (object). Otherwise, the new objective is treated as the next scheduled objective (as if it were part of the original schedule) and the state-operator charged with checking out all next-scheduled objectives is then activated.

FIGURE 4 - Search Mission Scenario

SCENARIO-1 Test Mission—

TRANSIT-A (move vehicle in straightest possible path to point (200 200 200))

LADDER-SEARCH-A (search 200m x 200m region for sunken submarine vessel, using ladder-like motion pattern; relevant objects include smooth, curved surfaces and metallic objects)

TRANSIT-B (move from end of search operation to point (1200 1200 400))

LADDER-SEARCH-B (search 400m x 400m region for same submarine)

TRANSIT-C (move to point (2000 800 800))

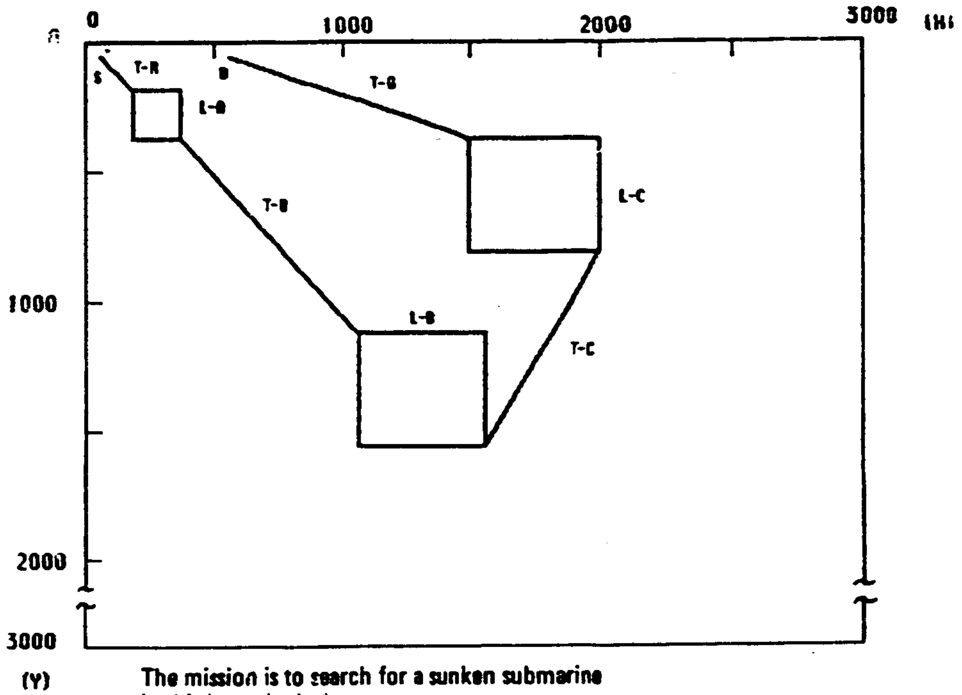
SPIRAL-SEARCH-C (search 600m radius region starting from current position, moving counter-clockwise in a spiral motion pattern)

TRANSIT-D (move to point (400 60 0))

REPORT-TO-BASE (transmit data from surface using radio)

FIGURE 5 - AUV Scenario Path

The world is a 3 km by 3 km square of coastal ocean of varying depths.



The mission is to search for a sunken submarine in this hypothetical ocean space.

The in'.'al mission specification given to the AUU consists of eight objectives: Transit-A (go to pt.A), Ladder-Search-A (search an area using a ladder-like motion pattern), Transit-B, Ladder-Search-B, Transit-C, Ladder-Search-C, Transit-D and Report-To-Base.

When the next-scheduled objective "checks out" as being satisfiable and not in conflict with other objectives (according to the rules provided for determining conflict and satisfiability), a state-operator is activated which "expands" the objective into appropriate component tasks. In the case of transiting operations, these tasks will generally all be motion-oriented, but in the case of a search objective there may be an interspersing of different types of tasks among the motion segments required to move the vehicle around. These include performing intensive scanning of a region while the vehicle is at an intermediary waypoint in the search, manipulating objects if the vehicle is equipped with appropriate devices, and so forth. Once the new current objective has been "expanded" into an initial set of tasks, these tasks are sequentially processed in a similar fashion - the next scheduled task is "checked out" just prior to execution for consistency with the actual state of affairs (which may have changed significantly since the objective was expanded into a list of tasks) and if the new task is approved, it is then transmitted via the Tactician to the Vehicle Operating System, a control program whose function is, akin to the mechanical engineering staff on a ship, to carry out the high-level commands and operate the servos and actuators of the vehicle. Communications from the Vehicle Operating System back to the Planner consist of messages indicating the status of the given task - either that it has been completed or that it cannot be completed, given the constraints specified (e.g., maximum time to perform a transit, maximum deviation from a given course).

Fundamental to the SOLON architecture is this partitioning or distribution of jobs among many different agents, the state-operator functions (4) of the Strategist. Many independent specialists, as it were, handle their particular tasks, without burdening each other or the higher-level modules in the Planner. However, with the message-passing system that runs throughout SOLON, the higher-level modules, like the captain and officers of a ship, have access to the activities of the lower-level units and are able to make changes which can include changing the tasks of those lower, simpler units.

X) Conclusion

The SOLON Planner provides several new features which we believe are important for planning and [illegible]ning, particularly in a real-time mobile context. First, it provides a mechanism f[illegible]ributing or partitioning the knowledge required for high-to-medium-level vehicle control into a number of different representation schemes (rather than just one method) and into a number of independent but communicating databases. Secondly, SOLON provides a mechanism for distributing the work of evaluating alternatives and selecting sequences of objectives and tasks among several agents (logprocs) which can readily be implemented in a concurrent, MIMD-type machine architecture. Thirdly, SOLON operates by "default reasoning" principles - the network of state-operators is such that problem-solving is attempted first using the simplest, most probable or most expected searches and queries. When the default methods fail, more complex logic is invoked. There is a definite redundancy built into SOLON; certain logprocs will receive messages and initiate activity which may turn out to be not required because of solutions implemented by another logproc. (This is particularly true in some cases of obstacle avoidance). This redundancy, we feel, is not only admissible but important. The justification for such mechanisms and for a good part of the state-operator architecture lies not only with the failure of many previous planning systems but the obvious historical success of biological planning in humans and animals.

(1) Foundations and influences for our research include work which is described in the following sources:

Brooks, Rodney, "Long-Term Autonomy for Mobile Robots", *1986 MIT Sea Grant College Program Lecture and Seminar Series*, MIT, Cambridge MA, October 1986

Durrant-Whyte, Hugh F., "Distributed Coordination and Decision Making in a Robot Sensing-Action System", PhD. dissertation, Univ. of Pa., Philadelphia PA (private circulation)

Guha, A. K. & Dudziak, M. J., "Knowledge-Based Controllers for Autonomous Systems", *Proceedings of the 1985 IEEE International Symposium on Intelligent Control*, Troy NY, August 1985

Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S. & Cammarata, S., "Modeling Planning as an Incremental, Opportunistic Process", RAND Corporation, Santa Monica CA, June 1979

Shank, Roger & Abelson, R. P., *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum, Hillsdale NJ, 1977

Sowa, J. F., *Conceptual Structures*, Addison-Wesley Systems Programming Series, Reading MA, 1984

Wesson, R. & Hayes-Roth, F., "Dynamic Planning: Searching Through Space and Time", RAND Corporation, Santa Monica CA, (Report P-6266)

(2) The overall IVC and the ICDS simulator are described in more detail within a related paper: Dudziak, M., Hall, M. & Zaret, D., "IVC : An Intelligent Vehicle Controller with Real-Time Strategic Replanning", *Proc. of the IEEE 1987 International Symposium on Intelligent Control*, Phila PA, Jan. 1987

(3) Principal foundations for our approach to autonomous-vehicle-specific problems include:

Harmon, S. K., "Planning for Transit in Unknown Natural Terrain", *Proceedings of the DOE CESAR Workshop on Planning and Sensing for Autonomous Navigation*, Los Angeles CA, August 1985

Kanayama, Yutaka & Yuta, Shin-Ichi, "Concurrent Programming of Intelligent Robots", *Proceedings of the 8th Int'l. Joint Conf. on Artificial Intelligence*, August 1983

Orlando, Nancy E., "Interfacing Intelligent Software to Robotic Peripherals", *Proceedings of the First International Conference on Applications of Artificial Intelligence to Engineering*

Problems, Southampton Univ., England, April 1986

(4) In our design the state-operators are entities represented using the common schema-based structure employed throughout SOLON. They are program objects which have among their attributes (slots) a list of LISP functions (usually only one) to be evaluated when the state-operator has been activated. These functions can, in a multiple-processor ("parallel machine") environment, easily be converted into individual processes running on their own CPUs. In this way the SOLON system is evolvable in two important respects:

First, new state-operators may be added and integrated into the overall state-operator network structure without disruption to other parts of the net, as it is deemed necessary to subdivide jobs among state-operators or handle new dimensions of the planning problem;

Second, state-operators which are currently functions or processes sharing a single CPU resource may be moved off to other processor machines as the work-load grows to an point where multiple processors become important for maintaining required real-time performance.

S31-63

4P.

188113

Mission Planning for Autonomous Systems

G. Pearson
FMC Corporation
Santa Clara. CA 95052

FY 563409

[illegible]

1. Abstract

Planning is a necessary task for intelligent, adaptive systems operating independently of human controllers. ~~This paper reports on~~ a mission planning system that performs task planning by decomposing a high-level mission objective into subtasks and synthesizing a plan for those tasks at varying levels of abstraction. We use a blackboard architecture to partition the search space and direct the focus of attention of the planner. Using advanced planning techniques, ~~we~~ can control plan synthesis for the complex planning tasks involved in mission planning.

Research Key

2. Introduction

Planning is a necessary task for intelligent, adaptive systems operating independently of human controllers. Autonomous systems need to plan their actions and adapt themselves to environmental changes for survival. Given a high level mission specification, the mission planning module needs to synthesize a sequence of actions to achieve mission goals. This requires advanced techniques that reason about constraints, granularity of search, spatial configurations, levels of abstraction and temporal orderings. Robotic systems benefit from these planning techniques by increasing their independence from mission control. As a result, expanded missions with reduced supervisory control can be conceived and executed[1].

Various approaches are being used in planning systems. STRIPS uses means-ends analysis in robot problem solving by identifying differences from a state description to a goal and selecting forward production rules to reduce them[2]. NOAH uses levels of abstraction and a least-commitment strategy to generate parallel plans hierarchically for an indoor mobile robot[3]. OPM uses opportunistic reasoning to solve the errand planning problem by combining both top-down and bottom-up planning activities[4]. MOLGEN uses skeletal refinement to plan experiments in molecular biology by instantiating specific operations from a sequence of generalized plan steps[5]. PROTEAN uses a blackboard architecture to configure protein structures by reasoning about the solution to the problem as well as the problem-solving process[6]. Our mission planner builds on these earlier systems and uses key aspects from each. It is implemented within the BB1 blackboard architecture[7] and uses the features offered by BB1.

3. Mission Planning

The overall control of autonomous systems requires the management of multiple subsystems cooperating to achieve mission goals. One such subsystem is mission planning. This paper reports on a mission planning system built for the autonomous operation of FMC's autonomous land vehicle, an M113 tracked vehicle.

Mission planning is the process of synthesizing a sequence of actions to satisfy goals and constraints posed by the mission manager. Mission plans are specified at varying levels of abstraction, with mission profiles at the higher levels and command sequences at the lower levels. Command sequences are fixed to perform specific tasks given the vehicle's operating characteristics; as such, they are best sequenced by computer programs that perform table lookups. Contention among these commands can usually be resolved at the higher levels, thus interaction among commands is minimal. At the other end of the planning hierarchy, mission profiles specify objectives and time tables for accomplishing the objectives. The vehicle achieves these objectives by executing command sequences downloaded by Mission Control at the appropriate times specified in the mission profile. Mission profiles lend themselves to template or script planning because they are specified at a level of detail higher in the abstraction hierarchy where interaction among the objectives is minimal.

Tasks, on the other hand, are synthesized into plans by considering the current state of the mission. Tasks consist of command sequences an autonomous vehicle executes to achieve some part of the overall mission. The planner performs task planning by decomposing the high-level mission objective into subtasks and synthesizing a plan for those tasks at varying levels of abstraction. Intermediate tasks must be selected and sequenced in such a way that subsequent goals can be achieved. An exemplary task performed by the planner is to develop a plan for conducting reconnaissance in a particular area specified by the mission commander. For example, a mission to conduct area reconnaissance is necessary when the commander desires specific information about certain locations or facilities within a defined area. To accomplish this mission, the planner must find overwatch positions for reconnoitering the targets, establish routes sequencing these positions, retain stealth of the operation and report all information rapidly and accurately[3].

Tasks refer to the intermediate abstraction levels in the planning hierarchy, those levels where interaction among planning decisions is the highest. Interaction among tasks either by sequencing tasks with prerequisite and postrequisite conditions or by decomposing tasks into subtasks makes up the interesting planning problem. These interactions occur in a dynamically changing environment and create a combinatorial explosion of the planning space; the search through this dynamic planning space is a key issue for mission planners.

4. Blackboard Architecture

We are implementing our mission planner using BEDS, a Blackboard Event Driven System[9], a version of the BB1

PAGE 302 INTENTIONALLY BLANK

blackboard architecture[7]. As such, it defines problem-solving knowledge sources for synthesizing plan steps, a multi-level solution blackboard for recording partial plans and a flexible control structure for controlling the expansion of the planning space.

Using a blackboard, a hierarchy of abstraction levels where each level represents a partial state description of the world at some time, we can partition the search space and direct the focus of attention of the planner. We map the problem space onto the blackboard by specifying abstraction levels in the plan hierarchy. These levels represent both spatial and conceptual abstractions for the mission planning problem. For the mission of area reconnaissance, we generate a visibility map by creating boundary regions that contain locations visible to the target—a spatial abstraction. For the path planning task of the mission, we generate one type of non-trafficable region by creating water boundaries—a conceptual abstraction. Data abstractions help control the exponential search process required in planning by establishing planning islands; areas where local search can find plan anchors for attaching the remainder of the plan. The more independent the planning islands, the better the planner controls its planning space by relying on local search. The blackboard structures the planning space in the problem domain. To this structure, we apply the problem-solving strategy of skeletal plan refinement.

When a mission is specified, the planner chooses a general design. We specify a design with only the essential detail necessary to direct the initial search of the plan. This least-commitment strategy is maintained throughout the plan refinement process. The design specifies spatial configurations for plans and partitions the planning space into plan segments. Once these segments are found, the planner successively refines its plan by instantiating plan steps at the lower levels. Plan instantiations occur by creating planning elements using the correct data abstraction with the current plan abstraction. At the design level, the planner cannot use low-level data to form decisions. Instead, it uses high-level symbolic objects that represent the relationships between the tasks that make up the mission.

For example, consider a plan for a reconnaissance mission that synthesizes a sequence of tasks in both time and space such that the final configuration satisfies overriding objectives. A good design specifies the spatial orientation for each of the tasks. Finding this design depends on the reconnaissance tasks involved and their relationships to each other. At this level of abstraction, the planner reasons about the target location, the type of reconnaissance mission, visibility maps, non-trafficable regions, military strategy and communication requirements. Only after refinement of the design can plan steps involving exact task locations be instantiated using pixel data represented as coordinate triples.

4.1. Controlling Plan Synthesis

Plan synthesis occurs when knowledge sources instantiate plan steps recorded in the blackboard hierarchy. Without controlling plan synthesis, the planning system would exhaustively create the solution space of possible plans. While this works for simple planning problems, in mission planning, as the complexity of the mission increases, the number of tasks grows and the number of potential plans grows exponentially. We use a three-tiered structure for varying control over the execution of knowledge sources in the mission planning system that consists of *establishing focus decisions, executing strategies* and *ranking knowledge sources.* During problem solving, knowledge sources create decision elements in the plan hierarchy—as planning proceeds, more knowledge sources are activated and become available for execution. A controller rates these knowledge sources using focus decisions, strategies and rankings, and a scheduler selects a knowledge source to execute by choosing the one with the highest rating.

Focus decisions represent collections of heuristics against which knowledge sources are rated[6]. These decisions establish criteria used to evaluate the utility of knowledge sources. For each knowledge source the controller calculates a utility value by summing together, for each focus decision, the products of a focus weight representing the value of a focus decision and a satisfaction level, the degree to which a knowledge source satisfies a focus decision. This calculation results in ratings that prioritize the knowledge sources so a scheduler can select the knowledge source with the highest rating. Focus decisions are created during problem solving in response to changes in planning and reflect the general behavior of the system. They add high-level control decisions that the controller uses to direct the generation of plan steps.

Strategies provide a rigid control structure that directly controls the execution of knowledge sources. They permit the execution of a strict sequence of knowledge sources. A strategy represents a procedure for achieving a particular goal and consists of a goal, a status, a rationale and a list of strategies and tactics. The goal denotes what the strategy will accomplish when its status becomes operative, and the rationale describes what the strategy accomplishes. The ordered list of strategies and tactics defines the specific subgoals that make up the procedure. When strategies are operative, knowledge sources that achieve the same goals of the operative strategies receive higher priorities than ones that achieve different goals. Focus decisions are used to differentiate between knowledge sources with the same goals.

```
STRATEGY: FIND-LOCATION
   goal = FIND-LOCATION
   status = OPERATIVE
   rationale = "Instantiate best location for
                    performing a task"
   strategy&tactic = (INSTANTIATE-LOCATION
                      RATE-LOCATION
                      CHOOSE-LOCATION)

STRATEGY: FIND-R1
   goal = FIND-R1
   status = OPERATIVE
   rationale = "Controls search for R1"
   strategy&tactic = (INSTANTIATE-LOCAL-AREA-R1
                      FIND-LOCATION)
```

Figure 4-1: FIND-LOCATION and FIND-R1 strategies

Figure 4-1 illustrates the structure of two strategies used by the Mission Planning System. The first strategy, FIND-LOCATION, consists of three tactics: INSTANTIATE-LOCATION, RATE-LOCATION and CHOOSE-BEST-LOCATION. This strategy finds a location by creating instances of locations, rating them and choosing the best one. The second strategy, FIND-R1, consists of the tactic INSTANTIATE-LOCAL-AREA-R1 and the strategy FIND-LOCATION. This recursive definition facilitates creating new strategies from existing ones. This strategy finds a position for performing reconnaissance by creating instances of local areas then finding locations within these areas.

The third level of control in this three-tiered structure, ranking knowledge sources, overlaps with the proceeding two. Ranking prioritizes knowledge sources that are grouped together because of similarities in function or strategy. During system design, knowledge sources are ranked to differentiate between subtleties in their performance characteristics. Usually, performance refers to processing speed with processing speed determining the granularity of search. Ranking gives the controller a discriminating factor when it chooses among knowledge sources with the same rating. Thus, ranking discriminates between knowledge sources that would otherwise be considered equal. Knowledge sources with the special rank of IMMEDIATE bypass the controller and execute immediately.

5. Constraint-based Reasoning

Another technique for controlling search is using constraints to limit the number of acceptable plans. Our planner uses constraints based on terrain feature information, resource limitations, vehicle limitations and military doctrine. In this way, the space of possible planning solutions is constrained by the specifications of the mission requirements. A mission must meet certain objectives while satisfying constraints that limit the success of an operation. The harder* the constraints, the less flexible the plan and the easier it is to confine the search. As constraints become softer** they contribute less to confining the space of possible plans. Our mission planner uses hard constraints to limit the number of acceptable plans by reasoning about terrain feature information, resource limitations, vehicle limitations and military doctrine during the planning process.

Mission planning is data intensive; therefore, search must be performed using different levels of granularity. We use a strategy that satisfies hard constraints before considering the soft constraints. Failure to satisfy any of the hard constraints results in the planner either terminating its search or backtracking by considering new potential solutions. Our planner performs simple backtracking by expanding its search through the data base. By continually increasing the resolution of the search, the planner increases the number of data points that it considers during planning. This technique allows it to make uniform cuts through the planning space as it refines plans top-down through the plan hierarchy. The planner performs more complicated backtracking by modifying constraints, thereby achieving the objective but with some loss of optimality. The planner relaxes constraints by propagating hard constraints. In our example of a reconnaissance mission, an original constraint is to use the reconnaissance technique of triangulation. However, when this constraint cannot be satisfied, the system relaxes it into one that allows straight reconnaissance. The constraint remains a hard constraint; it must be satisfied to complete the mission, but a plan allowing for straight reconnaissance is less desirable than one that uses triangulation.

Our planner works first from hard constraints to find a solution and backtracks only when necessary. It uses soft constraints, but considers them with less priority. Using constrained search, we confine the planning space to one that satisfies the hard constraints then find a solution that satisfies most of the soft constraints. As an example, consider the problem to place different sized objects in a leather pouch. A effectively simple strategy places the larger items in first, then squeezes in the smaller ones. This strategy works well because the planning space is characterized by the flexibility of the leather pouch. Mission planning has a similar flexible planning space because missions are defined to encompass changing conditions that occur during the operation of a mission. Thus, we use a strategy that satisfies the hard constraints, much like placing the larger items in the leather pouch first, then squeezes the soft constraints into place.

* Hard constraints refer to those constraints that must be satisfied when finding a valid solution.

** Soft constraints refer to those constraints that may or may not be satisfied when finding a valid solution.

6. Mission Planning Results

The Mission Planning System is written in ZetaLisp on a Symbolics Lisp Machine and interfaces to other software modules needed to control FMC's autonomous land vehicle. The planning system consists of 105 domain knowledge sources, 8 control knowledge sources, 27 strategies and 17 tactics.

Figure 6-1 shows the planning state of the Mission Planning System in the intermediate stages of finding a plan for conducting reconnaissance. In this example, the planner constructs a plan to reconnoiter an area using triangulation techniques that gather information about the target. It synthesizes a plan by executing knowledge sources and posting information on the blackboard. The blackboard levels are shown from the Strategy level down to the Route level. Strategy, Tactics and Foci make up the Control Blackboard, while Mission, Design, Region, Script, Lot, Location and Route make up the Domain Blackboard. Values are depicted in one of four states: underlined values represent operative criteria that the planner considers when making decisions; values shown in reverse-video denote the current activated subgoals in the planning space; boxed values are the selected positions and routes that make up the final plan; and shaded values (see figure 6-2) depict goals that have been accomplished.

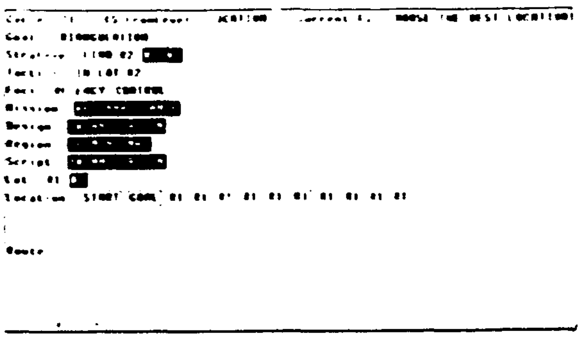

```
Cycle 21   KS triggered: [illegible]   Current KS: [illegible] THE BEST LOCATION
Goal: TRIANGULATION
Strategy: [illegible] R2
Tactics: [illegible]
Foci: [illegible] CONTROL
Mission
Design
Region
Script
Lot  01
Location  START COM  01 01 01 01 01 01 01 01 01 01
Route
```

Figure 6-1: Mission Planning State on Cycle 21

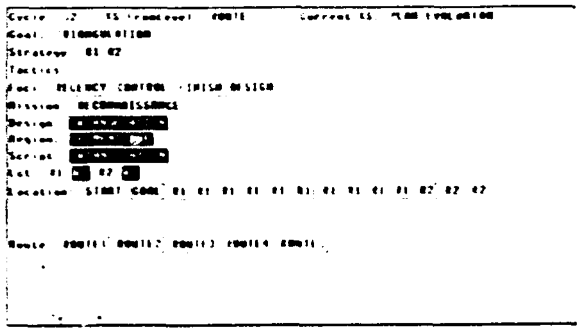

```
Cycle 32   KS triggered: ROUTE   Current KS: PLAN EVALUATION
Goal: TRIANGULATION
Strategy  R1 R2
Tactics
Foci  AGENCY CONTROL  [illegible] DESIGN
Mission  RECONNAISSANCE
Design
Region
Script
Lot  01  02
Location  START COM  01 01 01 01 01 01 01 01 01 01 02 02 02
Route  ROUTE1 ROUTE2 ROUTE3 ROUTE4 ROUTE5
```

Figure 6-2: Mission Planning State on Cycle 32

After receiving a mission statement from the commander of the vehicle, the planning system begins to find a possible plan. It starts by creating focus decisions that provide top-down control. This behavior changes as the system operates and creates other control decisions. A *design* is selected based on the requirement for triangulation that identifies the tasks necessary to accomplish the mission. These tasks are confined to a *region* that is computed based on vehicle and terrain constraints. With the region computed, the *start* and *goal* locations are posted for reference by the planner. A *script* resulting in a reconnaissance mission using triangulation is selected and a *strategy* for instantiating the script is generated. Here, the planner implements the R1-R2 strategy.

Having generated a strategy, the planner can instantiate *lats*, local areas where the planner searches for reconnaissance locations. It then *finds locations* and *routes* and sequences them into the final plan.

Figure 6-2 shows the planning state after the planner has found one possible plan for performing area reconnaissance using triangulation. The final plan is represented as nodes at the Location and Route levels. In this final plan, the vehicle travels along ROUTE1 from its starting position to the first reconnaissance location, R1. After reconnoitering the target, it travels along ROUTE5 to the second reconnaissance location, R2. At this point, the vehicle triangulates data acquired from the first reconnaissance task and completes the mission by moving to its final destination along ROUTE2.

7. Conclusion

We have built a Mission Planning System capable of sequencing tasks to achieve higher level mission objectives. We have built this system using a blackboard architecture that defines knowledge sources, a multi-level blackboard and a flexible control structure. Using this architecture integrated with other planning techniques, we have some degree of control over the explosive search space inherent in mission planning problems.

8. Acknowledgements

I am indebted to Perry Thorndyke and Shawn Amirsardary for helpful comments on the content of this paper, with special thanks to Jolan Yao for her software support of the Mission Planning System. Barbara Hayes-Roth was responsible for the BB1 system and made it available to FMC.

References

1. Thorndyke, P., McTamaney, L., Kuan, D. & Pearson, G., "The Autonomous Land Vehicle", *Defense Science & Electronics*, Vol. 1 November 1985.

2. Nilsson, N., *Principles of Artificial Intelligence*, Tioga Publishing Co., 1980.

3. Sacerdoti, E. D., "A Structure for Plans and Behavior", Tech. report 109, SRI, AI Center, 1975.

4. Hayes-Roth, B. & Hayes-Roth, F., "A cognitive model of planning", *Cognitive Science*, Vol. 3 1979, pp. 275-310.

5. Friedland, P. & Iwasaki, Y., "The Concept and Implementation of Skeletal Plans", *Journal of Automated Reasoning*, Vol. 1 1985, pp. 161-208.

6. Hayes-Roth, B., et. al., "A Layered Environment for Reasoning about Action", Tech. report KSL86-38, Stanford University, 1986.

7. Hayes-Roth, B., "A Blackboard Model of Control", Tech. report HPP 83-38, Stanford University, Computer Science Department, 1984.

8. Pearson, G. & Kuan, D., "Mission Planning System for an Autonomous Vehicle", *The Second Conference on Artificial Intelligence Applications*, 1985, pp. 162-167, Miami Beach, Fla.

9. Pearson G., "Mission Planning within the Framework of the Blackboard Model", *To appear in Proceedings from Expert Systems in Government Conference*, 1985, McLean, Va.

532-37

63

188114

Certainty Grids for Mobile Robots

H.P. Moravec
Carnegie-Mellon University
Pittsburgh, PA 15213

CH 1805[illegible]

1 Abstract

A numerical representation of uncertain and incomplete sensor knowledge we call Certainty Grids has been used successfully in several of our past mobile robot control programs, and has proven itself to be a powerful and efficient unifying solution for sensor fusion, motion planning, landmark identification, and many other central problems. We propose to build a software framework running on processors onboard our new Uranus mobile robot that will maintain a probabilistic, geometric map of the robot's surroundings as it moves. The "certainty grid" representation will allow this map to be incrementally updated in a uniform way from various sources including sonar, stereo vision, proximity and contact sensors. The approach can correctly model the fuzziness of each reading, while at the same time combining multiple measurements to produce sharper map features, and it can deal correctly with uncertainties in the robot's motion. The map will be used by planning programs to choose clear paths, identify locations (by correlating maps), identify well known and insufficiently sensed terrain, and perhaps identify objects by shape. The certainty grid representation can be extended in the time dimension and used to detect and track moving objects. Even the simplest versions of the idea will allow us fairly straightforwardly to program the robot for tasks that have hitherto been out of reach. We look forward to a program that can explore a region and return to its starting place, using map "snapshots" from its outbound journey to find its way back, even in the presence of disturbances of its motion and occasional changes in the terrain.

2 Introduction

Robot motion planning systems have used many space and object representations. Objects have been modelled by polygons and polyhedra, or bounded by curved surfaces. Free space has been partitioned into Voronoi regions or, more heuristically, free corridors. Traditionally the models have been hard edged - positional uncertainty, if considered at all, was used in just a few special places in the algorithms, expressed as a gaussian spread. Partly this is the result of analytical difficulty in manipulating interacting uncertainties, especially if the distributions are not gaussian. Incomplete error modelling reduces positional accuracy. More seriously, it can produce entirely faulty conclusions: a false determination of an edge in a certain location, for instance, may derail an entire train of inference about the location or existence of an object. Because they neglect uncertainties and alternative interpretations, such programs are brittle. When they jump to the right conclusions, they do well, but a small error early in the algorithm can be amplified to produce a ridiculous action. Most artificial intelligence based robot controllers have suffered from this weakness.

We've built our share of brittle controllers. Occasionally, however, we stumble across numerical (as opposed to analytic) representations that seem to escape this fate. One is deep inside the program that drove the Stanford Cart in 1979 [6]. Each of 36 pairings of nine images from a sliding camera produced a stereo depth measurement of a given feature, identified by a correlator, in the nine images. Some pairings were from short baselines, and had large distance uncertainty, others were from widely separated viewpoints, with small spread. The probability distributions from the 36 readings were combined numerically in a 1000 cell array, each cell representing a small range interval (Figure 1). Correlator matching errors often produced a multi-peaked resultant distribution, but the largest peak almost always gave the correct range. The procedure was the most error tolerant step in the Cart navigator, but it alone did not protect the whole program from brittleness.

A descendant of the Cart program by Thorpe and Matthies contained a path planner [5] that modelled floor space as a grid of cells containing numbers representing the suitability of each region to be on a path. Regions near obstacles had low suitability while empty space was high. A relaxation algorithm found locally optimum paths (Figure 2). The program represented uncertainty in the location, or even existence, of obstacles by having the suitability numbers for them vary according to extended, overlapping, probability distributions. The method dealt very reliably and completely with uncertainty, but also suffered from being embedded in an otherwise brittle program.

Our most thorough use of a numerical model of position uncertainty is a sonar mapper, map matcher and path planner developed initially for navigating the Denning Sentry [4, 2, 3]. Space is represented as a grid of cells, each mapping an area 30 (in some versions 15) centimeters on a side and containing two numbers, one the estimated probability that the area is empty, the other that it is occupied. Cells whose state of occupancy is completely unknown have both probabilities zero, and inconsistent data is indicated if both numbers are high. Many of the algorithms work with the difference of the numbers. Each wide angle sonar reading adds a thirty degree swath of emptiness, and a thirty degree

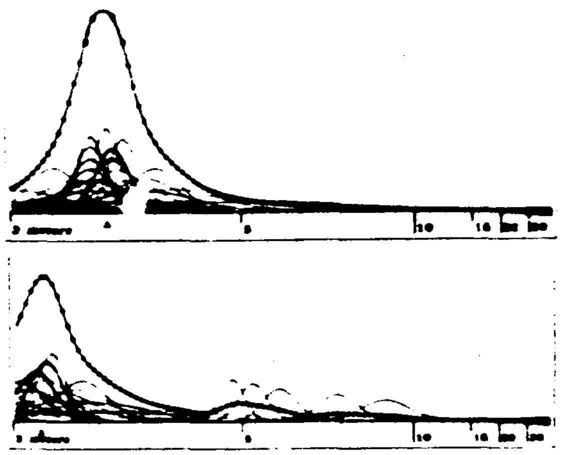


Figure 1: **Nine Eyed Stereo** - Identifications of a point on an object seen in nine different images taken as a camera traversed a track at right angles to its direction of view. Each pairing of images gives a stereo baseline, some short some long. Long baselines have less uncertainty in the calculated distance. The distributions for all 36 possible pairings are added in a one dimensional "certainty grid", and the peak of the resultant sum taken as the actual distance to the object. The top graph is for a case where are nine identifications of the point in the images are correct. The bottom is a case where one image is in error. The error produces eight small peaks at incorrect locations, but these are no match for the accumulation of the correct values.

arc of occupancy, by itself a very fuzzy image of the world. Several hundred readings together produce an image with a resolution often better than 15 centimeters, despite many aberrations in individual readings (Figure 3). The resiliency of the method has been demonstrated in successful multi-hour long runs of Denning robots around and around long trajectories, using three second map building and three second map matching pauses at key intersections to repeatedly correct their position. These runs work well in clutter, and survive disturbances such as people milling around the running robot.

Ken Stuart of MIT and Woods Hole has implemented a three dimensional version of the sonar mapper for use with small submersible craft. Tested so far only in simulation, but in the presence of large simulated errors, Stuart's program provides extremely good reconstructions, in a 128 × 128 × 64 array, of large scale terrain, working with about 60,000 readings from a sonar transducer with a seven degree beam. Running on a Sun computer, his program can process sonar data fast enough to keep up with the approximately one second pulse rate of the transducers on the two candidate submersibles at Woods Hole.

Recently Serey and Matthies demonstrated the utility of the grid representation in a stereo vision based navigator [1]. Edges crossing a particular scanline in the two stereo images are matched by a dynamic programming method, to produce a range profile. The wedge shaped space from the camera to the range profile is marked empty, cells along the profile itself are marked occupied. The resulting map is then used to plan obstacle avoiding paths as with the stereo and sonar programs mentioned above (Figure 4).

Despite its effectiveness, in each instance we adopted the grid representation of space reluctantly. This may reflect habits from a recent time when analytic approaches were more feasible and seemed more elegant because computer memories were too small to easily handle numerical arrays of a few thousand to a million cells. I think the reluctance is no longer appropriate. The straightforwardness, generality, and uniformity of the grid representation has

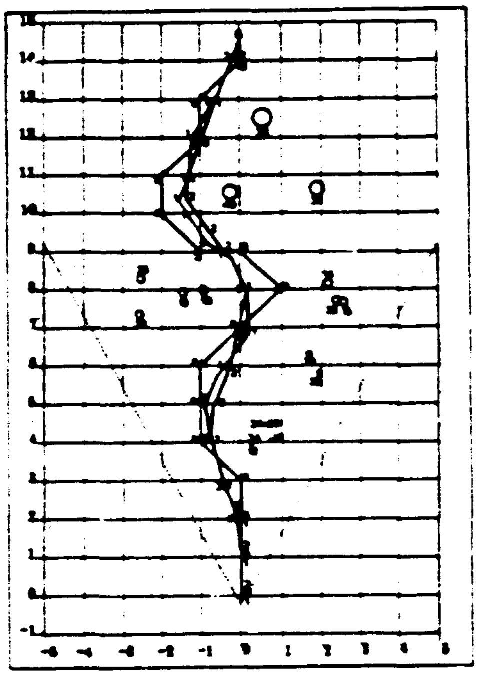


Figure 2: **Relaxation Path Planner** - A path is chosen that minimizes a given cost function in a Certainty Grid. Small perturbations are made in the vertices of the path in directions that reduce the cost.

proven itself in finite element approaches to problems in physics, in raster based approaches to computer graphics, and has the same promise in robotic spatial representations. At first glance a grid's finite resolution seems inherently to limit positioning accuracy. This impression is false. Cameras, sonar transducers, laser scanners and other long range sensors have intrinsic uncertainties and resolution limits that can be matched by grids no larger than a few hundred cells on a side, giving a few thousand cells in two dimensions, or a few million in three dimensions. Since the accuracy of most transducers drops with range, even greater economy is possible by using a heirarchy of scales, covering the near field at high resolution, and successively larger ranges with increasingly coarser grids. Besides this, the implicit accuracy of a certainty grid can be better than the size of its cell. The grid can be thought of as a discrete sampling of a continuous function. Extended features such as lines (perhaps representing walls) may be located to high precision by examining the parameters of surfaces of best fit. The Denning robot navigator mentioned above convolves two maps to find the displacement and rotation between them. In the final stages of the matching correlation values are obtained for a number of positions and angles in the vicinity of the best match. A quadratic least squares polynomial is fitted to the correlation values, and its peak is located analytically. Controlled tests of the procedure usually give positions accurate to better than one quarter of a cell width.

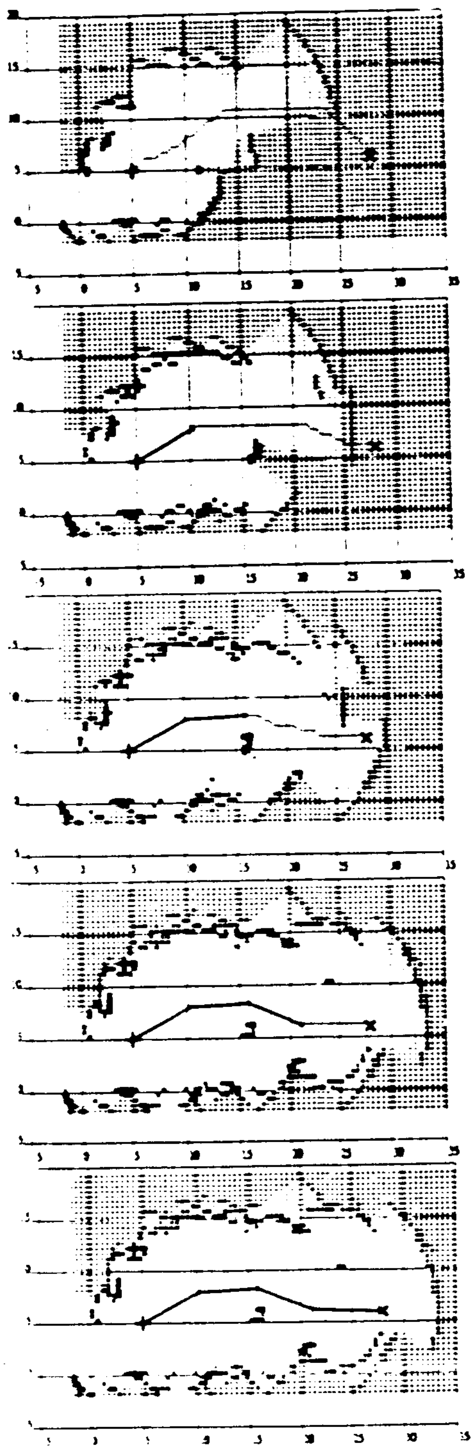


Figure 3: **Sonar Mapping and Navigation** - Plan view of the certainty grid built by a sonar guided robot traversing our laboratory. The scale marks are in feet. Each point on the dark trajectory is a stop that allowed the onboard sonar ring to collect twenty four new readings. The grid cells are white if the occupancy probability is low, dots if unknown, and x if high. The forward paths were planned by a relaxation path planner working in the grid as it was incrementally generated.

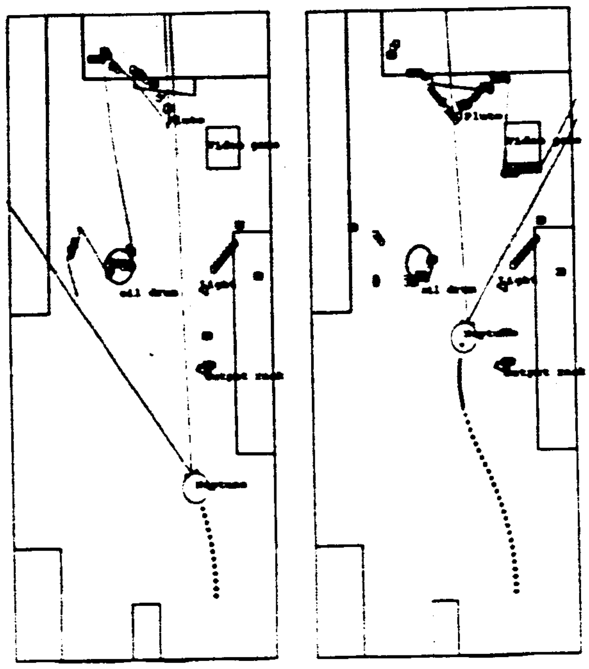


Figure 4: **Stereo Mapping and Navigation** - Plan view of the certainty grid built by a stereo guided robot traversing our laboratory. The situation is analogous to the sonar case of Figure 3, but the range profiles were gathered from a scanline stereo method using two TV cameras rather than a sonar ring.

Our results to date suggest that many mobile robot tasks can be solved with this unified, sensor independent, approach to space modelling. The key ingredients are a robot centered, multi resolution, map of the robot's surroundings, procedures for efficiently inserting data from sonar, stereo vision, proximity and other sensors into the map, other procedures for updating the map to reflect the uncertainties introduced by imprecise robot motion, and yet others to extract conclusions from the maps. We've already demonstrated procedures that produce local and global navigational fixes and obstacle avoiding paths from such maps. Other tasks, such as tracking corridors, finding vantage points with good views of unseen regions, and identification of larger features such as doors and desks by general shape seem within reach.

3 The Representation

The sonar mappers mentioned above are our most thorough use to date of the certainty grid idea. Although our original implementations used two grids to represent occupancy knowledge (labelled $P_{occupied}$ and P_{empty}), Stuart's 3D system uses only one. An analysis of the steps in our code reveals that one grid will indeed suffice, and this simplification makes clear several puzzling issues in the original formulation.

Before any measurements are made, the grid is initialized to a background occupancy certainty value, C_b. This number represents the average occupancy certainty we expect in a mature map, and encodes a (very) little bit of a-priori information we have about the world. In our lab a good C_b seems to be about the number of cells in the perimeter of the grid divided by the total cells ($4 \times 32/(32 \times 32) = 1/8$) in the case of the Denning code. If the space is very cluttered, C_b should be larger. As the map is used, values near C_b will stand for regions whose occupancy state is essentially unknown, while those much nearer zero will represent empty places, and those much nearer unity are likely to be occupied. Most of the planning algorithms that use the grid will be better off if they do not make sharp distinctions, but instead numerically combine the certainty values from various cells to produce "goodness of fit" numbers for their various hypotheses. In this way the essential uncertainties in the measurements are not masked, and the algorithms do not jump to unnecessary, possibly false, conclusions.

4 Inserting Measurements

The readings of almost any kind of sensor can be incorporated into a certainty grid, if they can be expressed in geometric terms. The information from a reading can be as minimal as a proximity detector's report that there is probably something in a certain region of space, or as detailed as a stereo depth profiler's precise numbers on the countours of a surface.

The first step, in general, is to express the sensor's measurement as a numerical spatial certainty distribution commensurate with the grid's geometry. For an infrared proximity detector this may take the form of set of numbers P_x in an elliptical envelope with high certainty values in a central axis (meaning detection is likely there) tapering to zero at the edges of the illumination envelope. Let's suppose the sensor returns a binary indication that there is or is not something in its field of view. If the sensor reports a hit, cells in the certainty grid C_x falling under the sensor's envelope can be updated with the formula

$$C_x := C_x + P_x - C_x \times P_x$$

which will increase the C values. In this case the P values should be scaled so their sum is one, since the measurement describes a situation where there is something somewhere in the field of view, probably not everywhere. If the reliability of the sensor is less than perfect, the normalization may be to a sum less than unity. If, on the other hand, the detector registers no hit, the formula might be

$$C_x := C_x \times (1 - P_x)$$

and the Cs will be reduced. In this case the measurement states that there is nothing anywhere in the field of view, and the P values should reflect only the chance that an object has been overlooked at each particular position; i.e. they should not be normalized. If the sensor returns a continuous value rather than a binary one, perhaps expressing some kind of rough range estimate, a mixed strategy similar to the one described below for sonar is called for.

A Polaroid sonar measurement is a number giving the range of the nearest object within an approximately thirty degree cone in front of the sonar transducer. Because of the wide angle, the object position is known only to be somewhere on a certain surface. This range surface can be handled in the same manner as the sensitivity distribution of a proximity detector "hit" above. The sonar measurement has something else to say, however. The volume of the cone up to the range reading is probably empty, else a smaller range would have been returned. The empty volume is like the "no hit" proximity detector case, and can be handled in the same fashion. So a sonar reading is like a proximity detector hit at some locations, and increases the occupancy probability there, and like a miss at others, where it decreases the probability. If we have a large number of sonar readings taken from different vantage points (say as the robot moves), the gradual accumulation of such certainty numbers will build a respectable map. We can, in fact, do a little better than that. Imagine two sonar readings whose volumes intersect. And suppose the "empty" region of the second overlaps part of the range surface of the first. Now the range surface says "somewhere along here there is an object", while the empty volume says "there is no object here". The second reading can be used to reduce the uncertainty in the position of the object located by the first reading by decreasing the probability in the area of the overlap, and correspondingly increasing it in the rest of the range surface. This can be accomplished by reducing the range surface certainties R_x with the formula $R_x := R_x \times (1 - E_x)$ where E_x is the "empty" certainty at each point from the second reading, then normalizing the Rs. This method is used to good effect in the existing sonar navigation programs, with the elaboration that the Es of many readings are first accumulated, and then used to condense the Rs of the same readings. (It is this two stage process that led us to use two grids in our original programs. In fact, the grid in which the Es are accumulated need merely be temporary working space.)

The stereo method of Serey and Matthies provides a depth profile of visible surfaces. Although, like a sonar reading, it describes a volume of emptiness bounded by a surface whose distance has been measured, it differs by providing a high certainty that there is matter at each point along the range surface. The processing of the "empty" volume is the same, but the certainty reduction and normalization steps we apply to sonar range surfaces are thus not appropriate. The grid cells along a very tight distribution around the range surface should simply be increased in value according to the "hit" formula. The magnitude and spread of the distribution should vary according to the confidence of the stereo match at each point. The method used by Serey and Matthies matches edge crossing along corresponding scanlines of two images, and is likely to be accurate at those points. Elsewhere it interpolates, and the expected accuracy declines.

If the robot has proximity or contact sensors, its own motion can contribute to a certainty grid. Areas traversed by the robot are almost certainly empty, and their cells can be reduced by the "no hit" formula, applied over a confident sharp edged distribution in the shape of the robot. This approach becomes more interesting if the robot's motion has inherent uncertainties and inaccuracies. If the certainty grid is maintained so it is accurate with respect to the robot's present position (so called robot co-ordinates), then the past positions of the robot will be uncertain in this co-ordinate system. This can be expressed by *blurring* the certainty grid accumulated from previous readings in a certain way after each move, to reflect the uncertainty in that move. New readings are inserted without blur (essentially the robot is saying "I know *exactly* where I am now; I'm just not sure where I was before). The track in the certainty grid of a moving robot's path in this system will resemble the vapor trail of a high flying jet - tight and dense in the vicinity of the robot, diffusing eventually to nothing with time and distance.

5 Extracting Deductions

The purpose of maintaining a certainty grid in the robot is to plan and monitor actions. Thorpe and Elfes showed the way to plan obstacle avoiding paths. Conceptually the grid can be considered an array of topographic values - high occupancy certainties are hills while low certainties are valleys. A safe path follows valleys, like running water. A relaxation algorithm can perturb portions of a trial path to bring each part to a local minimum. In principle a decision need never be made as to which locations are actually empty and which are occupied, though perhaps the program should stop if the best path climbs beyond some threshold "altitude". If the robot's sensors continue to operate and update the grid as the path is executed, impasses will become obvious as proximity and contact sensors raise the occupancy certainty of locations where they make contact with solid matter.

As indicated in the introduction, we have already demonstrated effective navigation by convolving certainty grids of given locations built at different times, allowing the robot to determine its location with respect to previously constructed maps. This technique can be extended to subparts of maps, and may be suitable for recognizing particular landmarks and objects. For instance, we are presently developing a wall tracker that fits a least squares line to points that are weighted by the product of the occupancy certainty value and a gaussian of the distance of the grid points from an a-priori guess of the wall location. The parameters of the least squares line are the found wall location, and serve, after being transformed for robot motion, as the initial guess for the next iteration of the process.

For tasks that would benefit from an opportunistic exploration of unknown terrain, the certainty grid can be examined to find interesting places to go next. Unknown regions are those whose certainty values are near the background certainty Cb. By applying an operator that computes a function such as

$$\sum (C_x - Cb)^2$$

over a weighted window of suitable size, a program can find regions whose contents are relatively unknown, and head for them. Other operators similar in spirit can measure other properties of the space and the robot's state of knowledge about it. Hard edged characterizations of the stuff in the space can be left to the last possible moment by this approach, or avoided altogether.

6 A Plan: Awareness for a Robot

Uranus is the CMU Mobile Robot Lab's latest and best robot and the third and last one we intend to construct for the forseeable future. About 60 cm square, with an omnidirectional drive system intended primarily for indoor work, Uranus carries two racks wired for the industry standard VME computer bus, and can be upgraded with off the shelf processors, memory and input output boards. In the last few years the speed and memory available on single boards has begun to match that available in our mainframe computers. This removes the main arguments for operating the machine primarily by remote control. With most computing done on board by dedicated processors, enabling very high bandwidth and relaible connection of processors to sensors and effectors, real time control is much easier. Also favoring this change in approach is a realization by us, growing from our experience with robot control programs from the very complex to the relatively simple, that the most complicated programs are probably not the most effective way to learn about programming robots. Very complex programs are slow, limiting the numebr of experiments possible in any given time, and they involve too many simultaneous variables, whose effects can be hard to separate. A manageable intermediate complexity seems likely to get us to our long term goals fastest. The most exciting element in our current plans is a realization that certainty grids are a powerful and efficient unifying solution for sensor fusion, motion planning, landmark identification, and many other central problems.

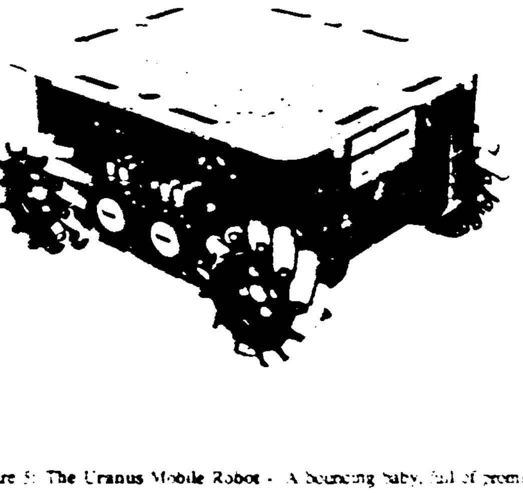


Figure 5: The Uranus Mobile Robot - A bouncing baby, full of promise.

As the core of the robot and the research we will prepare a kind of operating system based on the "certainty grid" idea. Software running continuously on processors onboard Uranus will maintain a probabilistic, geometric map of the robot's surroundings as it moves. The certainty grid representation will allow this map to be incrementally updated in a uniform way from various sources including sonar, stereo vision, proximity and contact sensors. The approach can correctly model the fuzziness of each reading, while at the same time combining multiple measurements to produce sharper map features, and it can deal correctly with uncertainties in the robot's motion. The map will be used by planning programs to choose clear paths, identify locations (by correlating maps), identify well known and insufficiently sensed terrain, and perhaps identify objects by shape. To obtain both adequate resolution of nearby areas and sufficient coverage for longer range planning, without excessive cost, a hierarchy of maps will be kept, the smallest covering a 2 meter area at 6.25 cm resolution, the largest 16 meters at 50 cm resolution (Figure 6). This map will be "scrolled" to keep the robot centered as it moves, but rotations of the robot will be handled by changing elements of a matrix the represents the robot's orientation in the grid. The map forms a kind of consciousness of the world surrounding the robot - reasoning about the world would actually be done by computations in the map. It might be interesting to take one more step in the hierarchy, to a one meter grid that simply covers the robot's own extent. It would be natural to keep this final grid oriented with respect to robot chassis itself, rather than approximately to the compass as with the other grids. This change of co-ordinate system would provide a natural distinction between "world" awareness and "body" or "self" awareness. Such encoding of a sense of self might even be useful if the robot were covered with many sensors, or perhaps were equipped with manipulators. We have no immediate plans in that direction, and so will pass by this interesting idea for now.

Our initial version will contain a pair of two dimensional grid sets, one mapping the presence of objects at the robots operating height of a few feet above ground level. The other will map the less complex idea of presence of passable floor at various locations. The object map will be updated from all sensors, the floor map primarily from downward looking proximity detectors, though possibly also from long range data from vision and sonar. The robot will navigate by dead reckoning, integrating the motion of its wheels. This method accumulates error rapidly, and this uncertainty will be reflected in the maps by a repeated blurring operation. Old readings, whose location relative to the robot's present position and orientation are known with decreasing precision, will have their effect gradually diffused by this operation, until they eventually evaporate to the background certainty value.

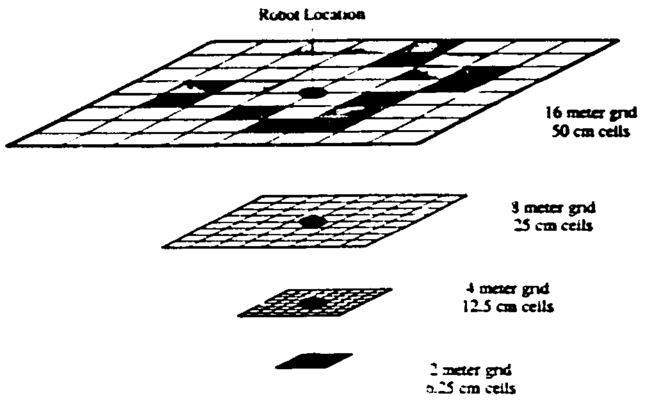


Figure 6: **Map Resolution Heirarchy** - Coarse maps for the big picture, fine ones for the fiddly details in the immediate environment. All the maps are scrolled to keep the robot in the center cells.

It would be natural to extend the two-grid system to many grids, each mapping a particular vertical slice, until we have a true three dimensional grid. We will do this as our research results, and processing power permit. The availability of single board array processors that can be installed on the robot would help this, as the certainty grid operations are very amenable to vectorizing. The certainty grid representation can also be extended in the time dimension, with past certainty grids being saved at regular intervals, like frames in a movie film, and registered to the robot's current co-ordinates (and blurred for motion uncertainties). Line operators applied across the time dimension could detect and track moving objects, and give the robot a sense of time as well as space. This has some very thrilling conceptual (and perceptual) consequences, but we may not get to it for a while.

Even the simplest versions of the idea will allows us fairly straightforwardly to program the robot for tasks that have hitherto been out of reach. We look forward to a program that can explore a region and return to its starting place, using map "snapshots" from its outbound journey to find its way back, even in the presence of disturbances of its motion and occasional changes in the terrain. By funneling the sensor readings through a certainty grid, which collects and preserves all the essential data, and indications of uncertainties, and makes it available in a uniform way, we avoid the problem we've had, that for each combination of sensor and task a different program is required. Now the task execution is decoupled from the sensing, and thus becomes simpler.

7 References

1. Serey, B. and L. H. Matthies, **Obstacle Avoidance using 1-D Stereo Vision**, CMU Robotics Institute Report, November, 1986.

2. Elfes, A. E, **A Sonar-Based Mapping and Navigation System**, Workshop on Robotics, Oak Ridge National Lab, Oak Ridge, TN, August, 1985. (invited presentation), in the proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, April 7-10 1986 also to appear as an invited paper in IEEE Transactions on Robotics and Automation.

3. Kadonoff, M., F. Benayad-Cherif, A. Franklin, J. Maddox, L. Muller, B. Sert and H. Moravec, **Arbitration of Multiple Control Strategies for Mobile Robots**, SPIE conference on Advances in Intelligent Robotics Systems, Cambridge, Massachusetts, October 26-31, 1986. In SPIE Proceedings Vol 727, paper 727-10.

4. Moravec, H. P. and A. E. Elfes, **High Resolution Maps from Wide Angle Sonar**, proceeding of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, March, 1985, pp 116-121, and proceedings of the 1985 ASME conference on Computers in Engineering, Boston, August, 1985.

5. Thorpe, C. E., **Path Relaxation: Path Planning for a Mobile Robot**, CMU-RI-TR-84-5, Robotics Institute, Carnegie-Mellon University, April, 1984. also in proceedings of IEEE Oceans 84, Washington, D.C., August, 1984 and Proceedings of AAAI-84, Austin, Texas, August 6-10, 1984, pp. 318-321.

6. Moravec, H. P., **Robot Rover Visual Navigation**, UMI Research Press, Ann Arbor, Michigan, 1981. also available as **Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover**, Stanford AIM-340, CS-80-813 and CMU-RI-TR-3.

S33-37
248

18 8/15

Environmental Modeling and Recognition for an Autonomous Land Vehicle

D.T. Lawton, T.S. Levitt, C.C. McConnell, and P.C. Nelson
Advanced Decision Systems
Mountain View, CA 94040

AC [illegible]

1. ABSTRACT

~~We present~~ an architecture for object modeling and recognition for an autonomous land vehicle. Examples of objects of interest include terrain features, fields, roads, horizon features, trees, etc. The architecture is organized around a set of data bases for generic object models and perceptual structures, temporary memory for the instantiation of object and relational hypotheses, and a long term memory for storing stable hypotheses that are affixed to the terrain representation. Multiple inference processes operate over these databases. ~~We~~ describe these particular components: the perceptual structure database, the grouping processes that operate over this, schemas, and the long term terrain database. ~~We conclude with~~ a processing example that matches predictions from the long term terrain model to imagery, extracts significant perceptual structures for consideration as potential landmarks, and extracts a relational structure to update the long term terrain database. given

2. INTRODUCTION

Terrain and object models for autonomous land vehicles (ALVs) are required for a wide range of applications including route and tactical planning, location verification through the recognition of terrain features and objects, and acquiring new information about the environment as it is explored. The following lists important criteria for terrain and object modeling capabilities.

Descriptive Adequacy: The modeling technique should be capable of describing the objects and situations in the environment necessary for the vehicle to function. This includes representing natural as well as man-made objects. It should be a consistent representation that supports modular system development and uniform inference procedures that can operate over different types of objects at different levels of detail. Uniform shape, object subpart and surface attribute affixments are necessary to do this.

Recognition Adequacy: Much of the activity of an ALV is concerned with determining where it is and what is around it. Terrain models should be manipulable for determining the sensor-based appearances of world objects and for controlling recognition processing. This involves the formation of general predictions of sensor derived features from the terrain model. Such predictions will often be uncertain and qualitative due to incomplete prior knowledge of the terrain.

Handling Uncertainty: The existence and exact environmental location of objects will often not be known with complete certainty. Locations will often be determined relative to other known locations and not with respect to a globally consistent terrain map. This is true, for instance, when the sensor displacement parameters are not well determined. It is necessary to represent this uncertainty explicitly in the terrain model so incrementally acquired information can be used for disambiguation.

Learning: A vehicle will learn about the environment as it moves through it. Associating new information with the terrain representation should be straightforward. This is difficult to do, for example, by changing values in a raw elevation array. Types of information to be affixed to the terrain representation include newly discovered objects, details of expected objects, and the processing used in object recognition.

Fusion of Information: The ALV must build a consistent environmental model over time from different sensors. As an object is approached, its image appearance and scale will change considerably, yet it has to be recognized as the same object, with newly acquired information associated with the unique instance of the general object type. In a typical situation, a distant dark terrain patch will be partially recognized based upon distinctive visual characteristics, but may be either a building or a road segment. As it is approached, its image appearance changes considerably, making disambiguation possible. This requires the representation of multiple hypotheses, each formated with respect to the properties of the potential world objects. The structure of the object description should direct the accumulation of information.

A further consideration in developing and evaluating terrain modeling capabilities is that there is not a single ALV. Instead, there are a wide range of autonomous vehicles, indexed by a diverse range of active and passive sensors and assumptions about a priori data. There is a continuum from systems having a complete initial model of the terrain and perfect sensors to those with no a priori model, and highly imperfect sensors. For example, a robot with no a priori data and only an unstabilized optical sensor will probably model the environment in terms of a sequence of views related by landmarks and distinct visual events embedded in a representation that is more topological than metric. An ALV solely dependent on optical imagery will have to deal with the huge variability in the appearance of objects. Experience has shown that even road surfaces have highly variable visual characteristics. Alternatively, a few pieces of highly preselected visual information can serve to verify predictions from a reliable and detailed terrain model and precise position and range sensors.

We call a general object model a schema. A schema can represent perceived, but unrecognized, visual events, as well as recognized objects and their relationships in environmental scenes. The architectural design is focused about the representation, instantiation, and inference over schemas developed by the ALV as it moves through the environment. Schemas are related to similar concepts found in Hanson et.al. - 78 and Ohta - 80. The short term terrain representation consists of schema instantiations that represent accumulated perceptual evidence for objects as attributes and relations that are hypothesized with varying levels of certainty.

Object models are used to organize perceptual processing by integrating descriptive representations with recognition and segmentation control. One aspect of this is the use of different types of attributes and inheritance relations between generic schemas for representation in IS-A and PART-OF hierarchies. A particular object attribute relates three dimensional world properties of an object and sensor dependent view information, either by a set of generic views or viewing procedures. These viewing attributes are also inherited and modified according to different object types. In many systems, objects are treated as lists of attributes that are matched against extracted image features. Here they are treated as specifying an active control process that directs image segmentation by specifying grouping procedures to extract and organize image structures.

Another critical aspect of the architecture is the various types of spatial, localization relations that deal with uncertainty and learning by associating different types of perceptually derived information with terrain models. For example, local (multi-sensor) viewframes affix sets of schemas and un-recognized perceptual structures into local "robot's-eye" views of an ALV's environment. Path-affixments between local viewframes support fusion of information in time without necessarily corresponding to locations in an a priori grid.

This effort has developed an architecture for terrain and object recognition compatible with the wide range of potential sensor configurations and the different qualities of a priori data.

There has been work in artificial intelligence, computer vision, and graphics that satisfy the individual requirements for object modeling capabilities, but little has been done to integrate them. To date, there is no vision system that can interpret general natural scenes, although some can deal with restricted environments Hanson et.al. - 78 while other systems are restricted to artificial objects and environments. Brooks' Brooks - 84 representation based on generalized cylinders meets, or could be extended to deal with, many of these functions. It has well defined shape attribute inheritance between a set of progressively more complex object models, and affixment relations that could be generalized to handle uncertainty. It can also be used to generate constraints on image features from object models. Nonetheless, the system built around this representation has had limited success beyond dealing with essentially orthographic views of geometrically well defined man-made objects. This appears to be partially because the constraints on image structures generated from the abstract instances of object models are too general to generate initial correspondences between models and image structures. Brook's system also used an impoverished set of image descriptions, and the object models could not direct the segmentation process directly during their instantiation. The majority of work in terrain modeling deals with how well a representation can realistically model three dimensional terrain, but not how it is used for recognition. The simplicity of a model that is described by a few parameters is not useful for recognition unless it can direct constrained searches against image data. For example, Pentland's Pentland - 83 use of fractals satisfies aspects of descriptive adequacy for natural terrain, but has been less effective for recognition. Kuipers Kuipers - 82 has produced an interesti[illegible] terrain model for learning and handling uncertainty, but it is non visual. R[illegible]ated to this is Kuan's Kuan - 84 object based terrain representation for planning that is organized in terms of distinct, modifiable objects, but is also not associated with sensor derived processing results.

3. ARCHITECTURE OVERVIEW

The system architecture consists of several databases and inference processes. The inference processes transform the databases, creating additional data structures, and modifying the existing ones. The task interface focuses attention in system processing and monitors progress toward system task goals. This high level architecture is depicted in Figure 1. The boxes with square corners in this figure represent databases, the ellipses represent inference processes, and arrows indicate dataflow.

3.1 SYSTEM DATABASES

At the highest level there are three databases. These are the short term memory (STM), long term memory (LTM), and generic models.

The STM acts as a dynamic scratchpad for the vision system. It has two sub-areas, a perceptual structures database (PSDB) and a hypothesis space. The PSDB includes incoming imagery from sensors, immediate results of extracting image structures such as curves, regions and surfaces, spatial temporal groupings of these structures, and results of inferring 3D information.

The hypotheses space contains statements about objects and terrain in the world. A hypothesis is represented as an instantiated schema. The schema points to the various perceptual structures in the PSDB that provide evidence that the object represented by the schema (such as a terrain patch, road, tree, etc.) exists in the world. Other types of hypotheses include grids, viewframes, and viewpaths. Grids are a special type of terrain representation that contain elevation information and are derived from range data or successive depth maps from motion stereo. Viewpaths, as partially ordered sequences of viewframes, give space time relationships between hypotheses. Viewframes are sets of hypotheses that correspond to what can be seen from a localized position. A hypothesis with no associated perceptual structures is a prediction. As structures and localization are incrementally added to a hypothesis, it progresses on the continuum from predicted to recognized. Hypotheses that have enough evidence associated with them to be considered recognized and stable, are moved to the LTM.

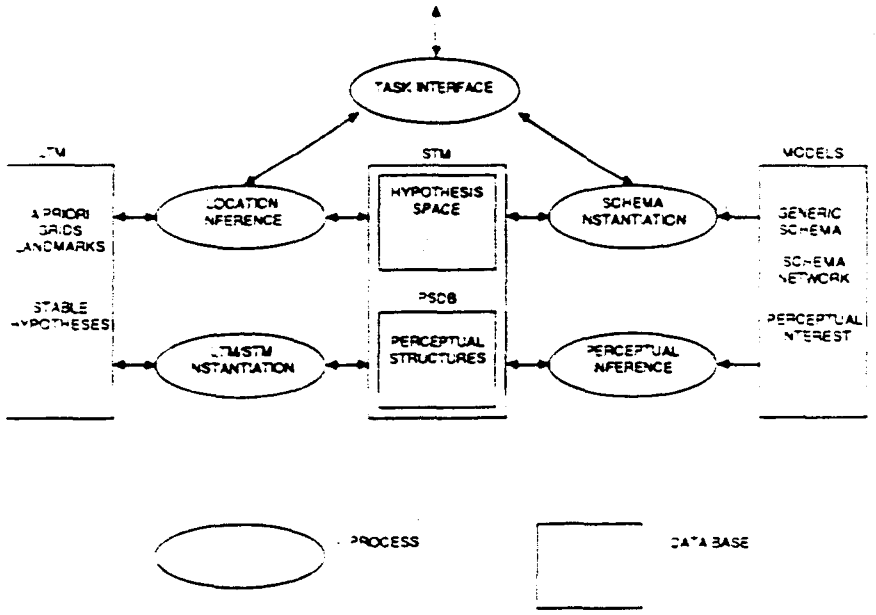


Figure 1: Terrain Modeling and Recognition System Architecture

The LTM stores a priori terrain representations, the long term terrain database, and hypotheses with enough associated evidence to be considered visually stable. A priori data concerning elevation and terrain type information, as well as knowledge of specific landmarks are stored in the LTM. A viewframe, representing a certain location in the world is stored in the LTM if the evidence associated with it could be re-used to recognize the local environment if it was re-encountered. Consistency of one hypothesis with another is not required for storage in the LTM.

The model space stores generic object models, the inheritance relations of the (model) schema network, and a set of image structure grouping processes and rules for evaluating image structure interestingness. Generic models are used dynamically to instantiate and guide search processes to associate evidence to an object instance. Inheritance relations are used by various schema inference procedures to propagate structures, attributes and relations between object instantiations. For instance, the generic two-lane-road schema has an "IS-A" relationship to the generic road schema. It follows, based on the inheritance models, that an instantiation of the two-lane-road schema will inherit the more general characteristics of the generic road schema that in turn inherits the more general characteristics of a terrain patch. Unlike the STM and LTM, the model space is not modified by inference processes.

3.2 INFERENCE PROCESSES

At the highest level, there are five different sorts of inference processes in the vision system. These are perceptual inference, location inference, object instantiation, LTM/STM instantiation, and the task interface.

The PSDB is initialized with the output of standard multi-resolution image processing operations for smoothing, edge extraction, flow field determination, etc. Much subtler inference is required for grouping processes that produce connected curves, textures, surfaces, and temporal matches between image structures. These grouping operations are typically model guided. There are generic models (which may be task dependent) of what constitutes "interestingness" of an image structure.

The hypothesis inference processes produce tasks for the perceptual processes. These may be satisfied by simple queries over the PSDB such as "find all long lines in this region of image", where "long", "line" and "region" are suitably interpreted. Queries can be more complex, requiring, for instance, temporal stability, such as "find all homogeneous green texture regions that are matched (i.e., remain in the field of view) over at least two seconds of imagery", where, again, qualitative descriptors are rigorously defined. Alternatively, the requested perceptual structures may be dynamically extracted. In this case, a history of the processing attempts and results are maintained. If similar requests are made later, such as if we were to view the same environment from a different perspective, these processing histories could be used to recall a processing sequence that produced successful results.

Location processes include a number of different modes of spatial location representation and inference. While exact location information is used when it is available, a key concept is the qualitative representation of relative location. This is fundamental, because the problem of acquiring terrain knowledge from moving sensors involves handling perceptual information that arises from multiple coordinate systems that are transforming in time. The basic approach to location inference is to represent the location of world objects in a qualitative manner that does not require the full knowledge of continuous transformations of sensor coordinates relative to the vehicle the sensors are mounted on, or of transformations of vehicle coordinates relative to the terrain.

The main structures involved in location inference are viewframes, viewpaths, and grids. Viewframes represent both metric location information about world objects derived from range sensors and view-based location information about the directions in which objects are found derived from passive sensor data.

Generic schemas are models of world objects that include information and procedures on how to predict and match the object models in the available sensor data. Besides representing 3D geometric constraints, 2D-3D sensor view appearance including effects of change in resolution and environmental effects such as season, weather, etc., schemas also indicate contextual relationships with other objects, type and spatial constraints, similarity and conflict relations, spatial localization, and appearance in viewframes.

Object schema instantiation may occur by model-driven prediction from a priori knowledge, or directly from another instantiation and a PART-OF relation. The other instantiation process may also occur by matching a distinctive perceptual structure to a schema appearance instance. This sort of "triggering" is more common in situations where there is little a priori information to guide prediction. Object instantiations generate queries to the PSDB grouping process in order to complete matching.

A key idea in object instantiation processing is inference over the model schema network hierarchies. Direct representation and inference over a large enough body of world objects to accomplish outdoor terrain understanding requires very large memory and proportionately lengthy inference procedures over that memory space. Hierarchical representation makes a significant reduction in storage requirements; furthermore, it lends itself naturally to matching schema to world objects at multiple levels of abstraction, thus speeding the inference process. Two basic hierarchies are the IS-A and PART-OF trees.

IS-A hierarchies represent the refinement of object classification. Figure 2 shows part of an IS-A hierarchy for terrain representation. The level of terrestrial-object tells us that we will not see evidence of any schema instance below this node as perceptual structures surrounded by sky. At the level of terrain-patch we pick up the geometric knowledge of adherence to the ground plane, while information stored at the level of a road schema constrains the boundaries of a terrain patch to be locally linear (with other constraints). Types beneath road add critical appearance constraints in color and texture, while the final refinement level in the IS-A hierarchy, the number of lanes, further constrains size parameters inherited from the road schema.

PART-OF hierarchies represent the decomposition of world objects into components, each of which is, itself, another world object. Figure 3 shows a PART-OF hierarchy decomposition for a generic 2-lane-road. PART-OF hierarchies contain relative geometric information that is useful in prediction and search.

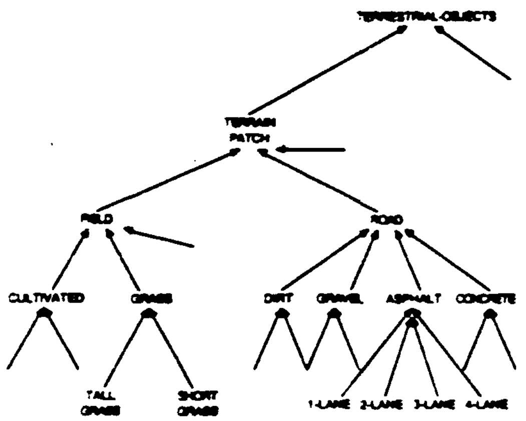


Figure 2: IS-A Hierarchy

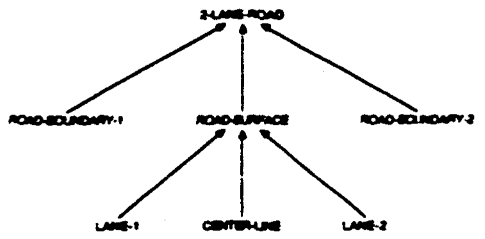


Figure 3: Part of Hierarchy

As object instantiation inference reasons up and down schema network hierarchies, incrementally matching perceptual structures and other data to instances of object appearance in the world, a history mechanism records the inference processing steps, parameters and results. This dynamic data structure is called the schema instantiation structure. One important aspect of this structure is that it can used to extract the inference and processing sequence(s) that worked earlier to see the same object, or ones that are similar. This accounts for the fact that distinctiveness in image appearance is an idiosyncratic process that depends upon many factors which are difficult to model and control, such as current motion, wind, varying outdoor illumination, etc.

4. PERCEPTUAL PROCESSING AND THE PSDB

Perceptual processing is concerned with organizing images into meaningful chunks. The definition of "meaningful" and the development of explicit criteria to evaluate segmentation techniques involves, from a data-driven perspective, that the chunks have characterizing properties, such as regularity, connectedness, and not tending to fragment the image. From a model-driven point of view, segmentation appropriateness corresponds to the extent to which the pieces can be matched to structures and predictions derived from object models. From either perspective, a basic requirement is that image segmentation procedures find significant image structures, independent of world semantics, in order to initialize and cue model matching. This allows for the extraction of world events such as surfaces, boundaries, and interesting patterns independent of understanding perceptions in the context of a particular object. These, in turn, are useful abstractions from image information to match against object models or describe the characteristics of novel objects.

The Perceptual Structure Data Base (PSDB), conceptualized in Figure 4, contains several different types of information. These are classified as images, perceptual objects, and groups. Images are the arrays of numbers obtained from the different sensors and the results of low level image processing (such as contour extraction and region growing routines) that produce such arrays. It is difficult for the symbolic relational representations used for object models, such as schemas, and the processing rules in computer vision systems, to work directly with an array of numbers. Therefore, there are many spatially-tagged, symbolic representations used in image understanding systems that describe extracted image structures such as the primal sketch [Marr - 82], the RSV structure of the VISIONS system [Hanson et.al. - 78], and the patchery data structure of Ohta [Ohta - 80]. We found it useful to build such a representation around a set of basic perceptual objects corresponding to points curves, regions, surfaces, and volumes.

Groupings are recursively defined to be a related set of such objects. The relation may be exactly determined, as in representing which edges are directly adjacent to a region, or they may require a grouping procedure to determine the set of objects that satisfy the relationship. Groupings can occur over space, e.g., linking texture elements under some shape criteria such as compactness and density, or over time, as in associating instances of perceptual structures in successive images. We stretch the concept a bit, so that groupings also refer to general non-image registered perceptual information, such as histograms.

4.1 INITIALIZATION OF THE PSDB

Whenever new sensor data is obtained, a default set of operations are performed to initialize the PSDB. Edges are extracted at multiple spatial frequencies and decomposed into linear subsegments. The edges are extracted into distinct connected curves, and general attributes such as average intensity, contrast, and variance are associated with them. Similar processing is performed for regions extractions. Histograms are computed with respect to a wide range of object based and image based characteristics in a pyramid like structure. These default operations are used to initialize bottom-up grouping processes and schema instantiations. These, in turn, determine significant structures using heuristic interestingness rules to prioritize the structures for the application of grouping processes or object instantiations.

4.2 IMAGES

Images are the data arrays derived from the optical and laser range sensors and the results of image processing routines for operations including histogram-based segmentation, different edge operators, optic flow field computations, and so forth. Associated with images are several attributes for time of acquisition, relevant sensor parameters, etc. Processing history is maintained in the processing relationship structure that keeps track of the processing history of all objects in the PSDB.

4.3 PERCEPTUAL OBJECTS

Points, curves, regions, surfaces, and volumes are basic types of perceptual structures that are accessible to object instantiations and grouping processes. An example instance of a curve structure is shown in Figure 5. This figure shows many common representational characteristics of perceptual objects. There are default attributes associated with particular objects, such as endpoints, length and positions for a curve. There is also an associated attribute-list mechanism for incorporating more general properties with an object. This list is accessible

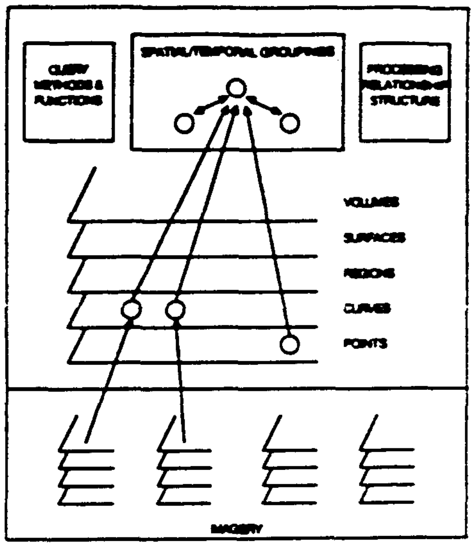
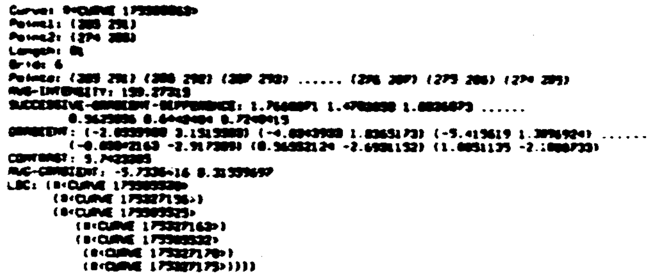

QUERY METHODS & FUNCTIONS
SPATIAL/TEMPORAL GROUPINGS
PROCESSING RELATIONSHIP STRUCTURE
VOLUMES
SURFACES
REGIONS
CURVES
POINTS
IMAGERY

Figure 4: Perceptual Structure Data Base (PSDB)



```
Curve: 0<CURVE 1753[illegible]>
Point1: (285 291)
Point2: (274 28[illegible])
Length: [illegible]
Grid: 6
Points: (285 291) (286 292) (287 293) ...... (276 287) (275 286) (274 285)
AVG-INTENSITY: 159.27319
SUCCESSIVE-GRADIENT-DIFFERENCE: 1.76[illegible] 1.47[illegible] 1.0[illegible] ......
        0.3625856 0.64[illegible] 0.724[illegible]15
GRADIENT: (-2.0[illegible] 3.1315[illegible]) (-4.0[illegible] 1.0363173) (-5.415619 1.3[illegible]) ......
        (-0.0[illegible]2163 -2.917[illegible]) (0.3[illegible]2124 -2.6[illegible]152) (1.0[illegible]1135 -2.1[illegible]733)
CONTRAST: 5.7[illegible]
AVG-GRADIENT: -5.7336-16 0.31539697
LBC: (0<CURVE 1753[illegible]>
      (0<CURVE 1753271 56>)
      (0<CURVE 1753[illegible]25>
        (0<CURVE 1753271 62>)
        (0<CURVE 1753[illegible]32>
         (0<CURVE 1753271 70>)
         (0<CURVE 1753271 75>))))
```

Figure 5: Curve Example

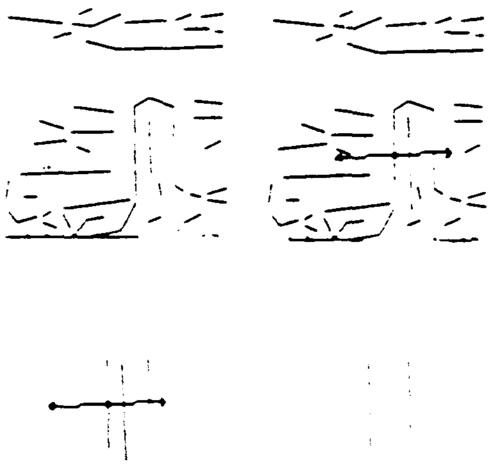


Figure 6: Parallel Grouping

by keywords and a general query mechanism using methods specific to the particular associated attribute. The associated attributes in the example are shown in capital letters. There are many types of attributes that can be consistently associated with a curve using this mechanism.

A useful representation for performing geometric operations and queries over objects is the OBJECT LABEL-GRID (or GRID: in the example curve. The number 6 indicates the index of this structure). This is an image where each pixel contains a vector of pointers back to the set of perceptual objects and groups which occupy that position. This allows geometric operations to be performed directly on the grid. Filtering operations can be applied to the OBJECT LABEL GRID to restrict processing based upon attributes associated with objects. Various types of masks can be associated with objects to reflect a directional or uniform neighborhood to determine object relationships in the OBJECT LABEL GRID.

4.4 GROUPS

A group is a set of related perceptual objects. The relation can be determined directly by a query over an object and those surrounding it, as in finding the set of curves within some distance of a given region. Alternatively, it may require a search process to find the set of objects meeting some, potentially complex, criteria. For example, an ordered set of curves can be grouped together using thresholds on allowable changes in the average contrast and orientation of successive elements. By expressing the grouping process as a search over a state space of potential groups, each group becomes a potential hypothesis in the PSDB. Groups can also reflect temporal relationships; this occurs in matching structures in successive images. A relational grouping procedure is shown in Figure 6 for the determination of nearby parallel lines with opposite contrast directions. This is done for a linear segment by first extracting nearby neighbors using a narrow mask oriented perpendicular from the segment at its mid-point. The intersection of this mask with points in the label grid are determined, and then each candidate is evaluated by checking if it is within allowable thresholds for length, contrast, and orientation. It is then ordered with respect to the smallest magnitude of the difference vector computed from the average gradients. The grouping processes can either produce the best candidate as a potential grouping, or some set of them.

Two different types of grouping processes have been developed: measure-based and interestingness-based. The measure based grouper is a generalization of established edge and region linkers [Martelli - 76]. It uses a measure consisting of:

1) some value to be optimized, such as length, minimal curvature, compactness, or a composite scalar value

2) local constraints on allowable changes in attributes

3) global thresholds on attributes

The measure and associated constraints are optimized by a best first search returning several ordered candidate groups. The measure to be used can be associated with a prediction from an object model for substance or shape characteristics. The measure to be optimized can also be determined directly from initially extracted objects by selecting those that are extreme in some attribute or are correlated with the attributes of surrounding objects to derive a measure to be optimized.

The measure based grouper is currently being generalized into one based on interestingness. It involves the basic processing loop shown in Figure 7. Initially, basic perceptual objects including curves, regions, junctions and their associated attributes are extracted using conventional techniques. Extracted objects are represented in label grids to express spatial neighborhood operations over the objects. A uniform neighborhood is established for each object, and directed relations are formed with the adjacent objects in each neighborhood. These relations are represented in a small number of types of match relationships that contain descriptions of the correlation of attributes, subcomponent matching, and composite properties.

Selected attributes of the extracted perceptual objects and the match structures are then sorted into lists with pointers back to the associated objects. These lists are for attributes such as size, average feature values, variance of feature values, compactness, the extent of correlation between the components and attributes of different structures, and the number of groups an object is involved in. These different rankings are then combined using a selection criteria to choose the set of interesting perceptual objects and relationships. The selection criteria sets the required position in different subsets of the sorted attribute lists. An example is to find 100 largest objects in the top 10 of any of the attribute correlation lists. The selection criteria is modifiable during processing and is meant to reflect the influence of model-based predictions.

Interestingness is used to focus the application of grouping rules to a selected set of objects and relations between objects indicated in match structures. The grouping rules then combine perceptual objects to form new perceptual objects, or groups, based upon the type of relation between the objects. Neighborhoods are established with respect to these derived groups to form new relationships. These in turn are sorted in the attribute lists with respect to the previously extracted perceptual objects. In addition to the relations established in uniform neighborhoods, for some groups, non-uniform relations are also established. Processing can continue indefinitely as less and less interesting relations become candidates for the application of grouping rules. Explicit criteria are needed to stop processing; e.g., we can limit processing time, determine when there is a uniform covering of the image with extracted groups, or when structures belong to unique groups.

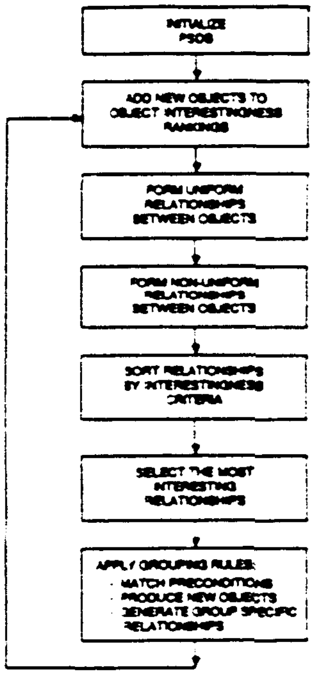

INITIALIZE PSOS
ADD NEW OBJECTS TO OBJECT INTERESTINGNESS RANKINGS
FORM UNIFORM RELATIONSHIPS BETWEEN OBJECTS
FORM NON-UNIFORM RELATIONSHIPS BETWEEN OBJECTS
SORT RELATIONSHIPS BY INTERESTINGNESS CRITERIA
SELECT THE MOST INTERESTING RELATIONSHIPS
APPLY GROUPING RULES:
- MATCH PRECONDITIONS
- PRODUCE NEW OBJECTS
- GENERATE GROUP SPECIFIC RELATIONSHIPS

Figure 7: Grouping Processing Flow

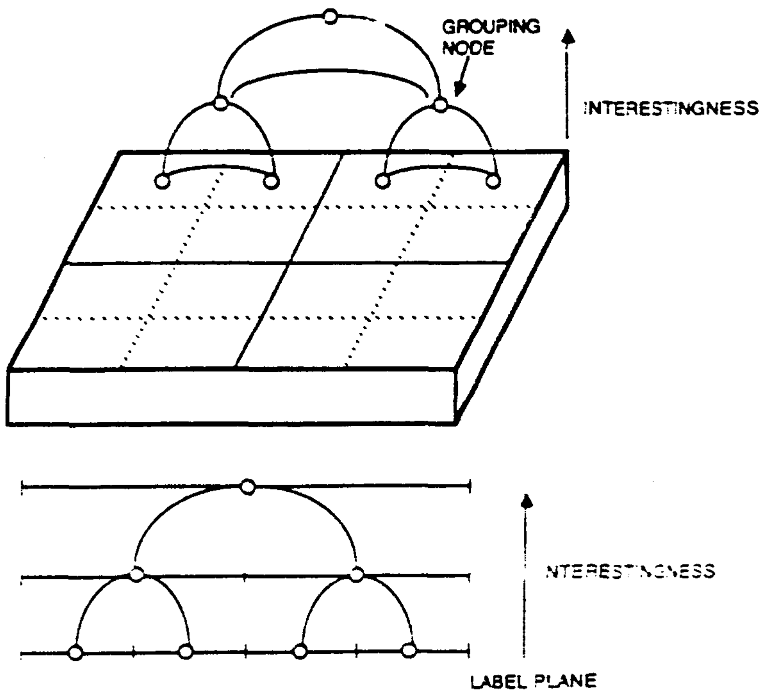


Figure 8: Grouping Architecture

These operations are performed by virtual processors called grouping nodes. Grouping nodes are seen as covering regular and adjacent portions of an image area (not necessarily of a single image, because there can be multiple images in a motion sequence). The image area contains some portion of a label plane for accessing the objects based upon their spatial dispositions as well as object-based associated attributes. The grouping nodes are further organized in a hierarchical pyramid shown in Figure 8. Each node is connected to its adjacent neighbors and has a parent and descendants. The transfer of information between nodes at different levels is based upon interestingness. Lower level processes send their most interesting structures up the hierarchy. There are several effects of this. One is that it allows a uniform processing to occur at different levels, so grouping rules can be applied to objects at different levels of interestingness. It also allows relations between nonspatially adjacent structures to be handled in a uniform architecture. It also partitions perceptual structures in a way that corresponds to different levels of control in instantiation of object models.

Organizing segmentation in terms of grouping processes has many advantages for a model based vision system. The grouping processes can be run automatically from extracted significant structures based upon perceptually significant, though non-semantic criteria. Thus, connected curves of slowly changing orientation or compact, homogeneous regions can be extracted purely on perceptual criteria. These image structures correspond to world structure and events, and they are useful for initializing schema instantiations. They correspond to the qualitative image predictions associated with more general schemas. An inference process for compilation from an object model into grouping processes, allows model based vision to have a very active character quite different from single-level attribute matching.

5. SCHEMAS

Schemas represent hypotheses about objects in the world. The process of schema instantiation creates an instance of a schema together with evidence for that schema. Evidence consists of structures in the PSDB, a priori knowledge stored in the LTM, predictions derived from location inference, and relations to already instantiated schema.

Table 1 shows the various slots and relationships in a generic schema. Although this data structure has a frame-like appearance, it is useful to view the schema as a semantic net structure, with slots representing nodes in the net and relationships representing arcs. Schema instantiation inference reasons from a (partially) instantiated node, follows arcs, and infers procedures to execute from the sum of its acquired information in order to obtain more evidence to further instantiate the schema.

The schema network is a generic set of data structures that indicate the a priori relationships between schemas. A key part of this network is the inheritance hierarchies that indicate which descriptions and relationships can be inherited from schema to schema. Inheritance hierarchies allow efficient matching of objects in the world against sensor evidence from progressively coarser to finer levels. As reasoning moves from coarser to finer levels of description in model-based schema instantiations, the schemas inherit descriptive bounds and add new descriptions, and also add constraints to inherited ones. For example, the system

Table 1: Generic Schema Data Structure

- SCHEMA TYPE
- SCHEMA NAME
- SCHEMA INSTANTIATION STRUCTURE
- 3D DESCRIPTION
  - SHAPE
  - SIZE
  - COLOR
  - TEXTURE
  - INDEX TO SENSOR VIEWS
- SENSOR VIEWS
  (FOR EACH SENSOR)
  (FOR EACH VIEW)
  - PROJECTION RELATIONS
    - PROJECTION FUNCTION
    - 3D BACK CONSTRAINTS
  - DISTINCTIVE IMAGE BASED EVIDENCE
  - PERCEPTUAL STRUCTURE
- COMPONENTS
  - MUST HAVE
  - MAY HAVE
  - 3D SPATIAL RELATIONSHIPS
  - VIEW DEPENDENT RELATIONSHIPS
- PART OFS
- CLASSIFICATIONS
  (POINTS UP THE IS-A HIERARCHY ONE LEVEL)
- CONTEXTUAL RELATIONSHIPS
  - ALWAYS OCCURS WITH
  - SOMETIMES OCCURS WITH
  - NEVER OCCURS WITH
  - CONFUSED WITH
  - SIMILAR TO
- LOCATIONAL INFORMATION
  - LOCAL MULTI-SENSOR FRAME AFFIXMENTS
  - GRID AFFIXMENTS
  - 3D SPATIAL RELATIONSHIPS WITH
- RECOGNITION STRATEGIES

may first recognize an object as a terrain patch (because it lies on the ground plane). A road is a type of terrain patch (see Figure 1, that adds linear boundary description, and constrains the visual image appearance of the terrain patch schema in the color and texture descriptors. The two basic types of schema network inheritance hierarchies are IS-A and PART-OF.

Below is a brief explanation of each of the slots and relationships in the generic schema data structure. Schema type refers to the generic name of the schema in the IS-A hierarchy. Schema name is the identification of the schema instance, e.g., if the schema type is "road" then the schema name might be "highway 101". The schema instantiation structure maintains the control history of the schema recognition inference processes for this schema.

The 3D description is an object-centered view of the world object represented by the schema. It includes its 3D geometry and shape description, actual size, and inherent color and texture (as opposed to how its color and texture might appear to a particular sensor). Note that this is the description that matches the schema-object before looking at its structure refined into components. For example, the 3D geometric description of a tree schema does not separate the canopy from the trunk, but gives a single enclosing volume as its representation. The volumetric descriptions of the trunk and canopy appear as the 3D descriptors on their schema further down the PART-OF hierarchy. Thus, inferring down the PART-OF hierarchy corresponds to increasing the resolution of the view of the object represented by the schemas.

The sensor views are descriptions of the stable or frequently occurring appearances of the schema object in imagery. This description is intended to be used for image appearance prediction, evidence accrual for instance recognition, 3D shape inference, and location inference. The reason for storing or runtime generation of explicit (parametrized) image views is that the perceptual evidence matches to these descriptions, not to the three dimensional ones.

The distinctive image appearance slot holds descriptions of perceptual structures that are likely to occur bottom-up in the PSDB. They provide coarse triggers for instantiating the schema object hypothesis without prediction.

The perceptual structure is the dynamically created PSDB query history generated by the schema instantiation as it attempts to fill in evidence matching the various schema slots and relations. The instantiator can re-use successful branches of perceptual structures to improve its recognition speed as it continues to view other instances of the same generic schema type.

Components are pointers to other schema that represent sub-parts of the schema object. They are finer resolution description of the schema, one level down on the PART-OF hierarchy. The MUST-HAVE components are assumed to be parts the represented object must have to exist, although the schema may be instantiated without observing them all. Occasionally occurring components, such as center-lines on roads, can be stored in the MAY-HAVE slot. Spatial relationships between components as they make up the schema object are listed at this level also. Relationships can also be stored on a view dependent basis. These relationships access the sensor-view dependent data in that slot. PART-OF's point upward one level on the PART-OF hierarchy, indicating that this schema is a component of another schema.

Classification points upward and downward one level on the IS-A hierarchy. There may be more than one such pointer, which is to say that the IS-A hierarchy may be partially ordered.

Contextual relationships indicate spatial/temporal consonance or disconsonance between groups of schema types, omitting those which are already indicated in the PART-OF and IS-A hierarchies. Schema that ALWAYS or never-occur with the given one can be used strongly for belief or dis-belief in the schema instance and as focus of attention mechanisms within the instantiation process. SOMETIMES occurs with relationships that are used to store the spatial-temporal aspects of schemas relative appearance in the viewed environment.

CONFUSED-WITH and SIMILAR-TO relationships indicate schema that may be mistaken for the given one, but for different reasons. One schema may be confused with another because they share common evidence pieces, but for which there are sufficient descriptors to disambiguate. Two schema are similar if there is sufficient ambiguity in their appearances, and therefore the available perceptual evidence, that they may be indistinguishable without contextual reasoning. For example, tall grass may be confused with wheat from coarse shape and texture evidence, but can often be disambiguated by color descriptors or finer resolution examination of structure (because of wheat berries, for example). However, roads are similar to runways because they cannot necessarily be distinguished by their intrinsic appearance, no matter how detailed or accurate the descriptors and evidence. Contextual reasoning, e.g., the presence of aircraft on the runway, global curvature of the road, etc. is required.

Locational information points at the various viewframes the schema appears in and inferred 3D relationships with other world objects.

Recognition strategies are prioritization cues for the schema instantiation processes that suggest inference chains likely to pay off to match this schema instance against sensor evidence.

The recognition strategies slot in the schema data structure prioritizes inference approaches relevant to this schema. These approaches include search for components, search for part of schema instance, search on weaker classification, relations with other schema instances, and PSDB matching.

Search for COMPONENTS and search for PART-OF are both inferences along the PART-OF hierarchy in different directions. The instantiator searches the relevant slot to see if there are components to search for or another object of which this schema is a component. If the COMPONENT or PART-OF schemas exist, they can be accessed to continue the inference. Otherwise, each causes an instantiation of the missing schema to be generated as a prediction. Instantiation control can be transferred at this point to the COMPONENT or PART-OF schema. The schema inference process maintains its thread of reasoning relevant to the schema in the schema instantiation structure slot.

8. LONG TERM TERRAIN DATABASE

The long term terrain database is part of LTM. It stores the data necessary for a mobile robot to perform vision-based navigation and guidance, predict visual events, such as landmarks and horizon lines, and to update and refine maps.

The long term terrain database contains a priori map data including government terrain grids, elevation data, and schemas representing instances of stable visual events recorded while traversing paths in the environment. The use of a priori map and grid data to predict percepts and to help guide image segmentation is shown in Section 5. The following presents a summary of a structure for spatial representation and inference that enables a robot to navigate and guide itself through the environment.

We first define the notion of a geographic "place" in terms of data about visible landmarks. A place, as a point on the surface of the ground, is defined by the landmarks and spatial relationships between landmarks that can be observed from a fixed location. More generally, a place can be defined as a region in space, in which a fixed set of landmarks can be observed from anywhere in the region, and relationships between them do not change in some appropriate qualitative sense. Data about places is stored in structures called viewframes, boundaries and orientation regions.

Viewframes provide a definition of place in terms of relative angles and angular error between landmarks, and very coarse estimates of the absolute range of the landmarks from our point of observation. Viewframes allow the system to localize its position in space relative to observable local landmark coordinate systems. In performing a viewframe localization, observed or inferred data about the approximate range to landmarks can be used. Errors in ranging and relative angular separation between landmarks are smoothly accounted for. A priori map data can also be incorporated. A viewframe is pictured in Figure 20.

A viewframe encodes the observable landmark information in a stationary panorama. That is, we assume that the sensor platform is stationary long enough for the sensor to pan up to 360 degrees, to tilt up to 90 degrees (or to use an omni-directional sensor [Cao et.al. - 86]), to recognize landmarks in its field of view, or to buffer imagery and recognize landmarks while in motion.

A sensor-centered spherical coordinate system is established. It fixes an orientation in azimuth and elevation, and takes the direction opposite the current heading as the zero degree axis. Then two landmarks in front of the vehicle, relative to the heading, will have an azimuth separation of less than 180 degrees. If we assume that no two distinguished landmark points have the same elevation coordinates (i.e., no two distinguished points appear one directly above the other) then a well-ordering of the landmarks in the azimuth direction can be generated. We can speak of the landmarks as being "ordered from left to right". The relative solid angle between two distinguished landmark points is now well defined.

Under the above assumptions, the system can pan from left to right, recognizing landmarks, L_i, and storing the solid angles between landmarks in order, denoting the angle between the i-th and j-th landmarks by Ang_{ij}. The basic viewframe data are these two ordered lists, $(L_1, L_2, \ldots)$ and $(\mathrm{Ang}_{12}, \mathrm{Ang}_{23}, \ldots)$. The relative angular displacement between any two landmarks can be computed from this basic list. In [Levitt et.al. - 87] we show how to use this data to essentially parametrize all possible triangulations of our location relative to a set of simultaneously visible landmarks. This localizes the robot's position in space relative to a local landmark coordinate system.

Viewframes contain two basic dimensions of data: the relative angles between landmarks, and the estimated range (intervals) to the landmarks. If we drop the range information, we are left with purely topological data. That is, it is impossible, using only the relative angles between landmarks, and no range, map or other metric data, to determine the relative angles between triples of landmarks, or to construct parametric representations of our location with respect to the landmarks. Nonetheless, there is topological localization information present in the ordinal sequence of landmarks: there is a sense in which we can compute differences between geographic regions, and observe which region we are in.

The basic concept is to note that if we draw a line between two (point) landmarks, and project that line onto the (possibly not flat) surface of the ground, then this line divides the earth into two distinct regions. If we can observe the landmarks, we can observe which side of this line we are on. The "virtual boundary" created by associating two observable landmarks together thus divides space over the region in which both landmarks are visible. We call these landmark-pair-boundaries (LPB's), and denote the LPB constructed from the landmarks L_1 and L_2 by $\mathrm{LPB}(L_1, L_2)$.

Roughly speaking, if we observe that landmark L_1 is on our left hand, and landmark L_2 is on our right, and the angle from L_1 to L_2 (left to right) is less than 180 degrees, then we denote this side of, or equivalently, this orientation of, the LPB by $[L_1\ L_2]$. If we stand on the other side of the boundary, LPB(L_1,L_2), "facing" the boundary, then L_2 will be on our left hand and L_1 on our right and the angle between them less than 180 degrees, and we can denote this orientation or side as $[L_2\ L_1]$ (left to right).

More rigorously, define:

$$\text{orientation-of-LPB}(L_1,L_2)$$
$$= \text{sign}(\pi-\Theta_{12}) = \begin{array}{l} -1 \text{ if } \Theta_{12} < \pi \\ 0 \text{ if } \Theta_{12} = \pi \\ -1 \text{ if } \Theta_{12} > \pi \end{array}$$

where Θ_{12} is the relative azimuth angle between L_1 and L_2 measured in an arbitrary sensor-centered coordinate system. Here, an orientation of -1 corresponds to the $[L_1\ L_2]$ side of LPB(L_1,L_2), -1 corresponds to the $[L_2\ L_1]$ side of LPB(L_1,L_2) and 0 corresponds to being on LPB(L_1,L_2). It is a straightforward to show that this definition of LPB orientation does not depend on the choice of sensor-centered coordinate system.

LPB's give rise to a topological division of the ground surface into observable regions of localization, called orientation regions. Crossing boundaries between orientation regions leads to a qualitative sense of path planning based on perceptual information. The three levels of spatial representation given by map or metric data, viewframes and orientation regions are pictured in Figure 9. A

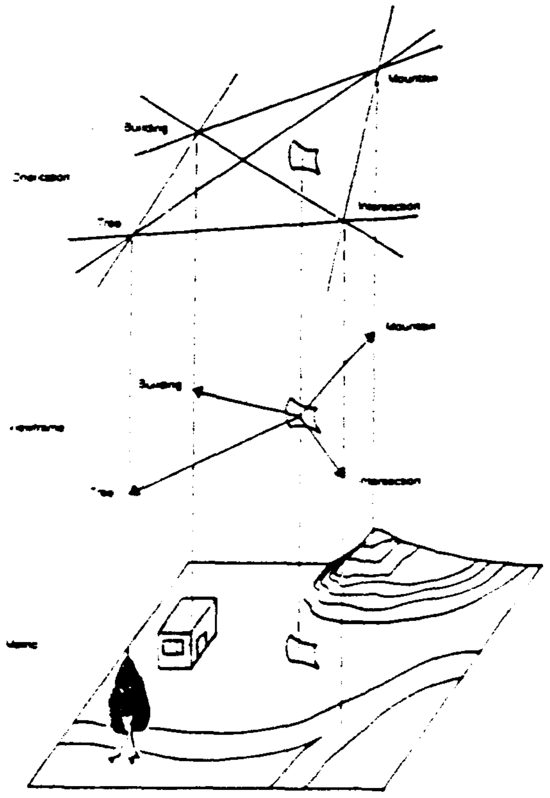


Figure 9: Multiple-Levels-of-Spatial Representation

natural environmental representation based on viewframes recorded while following a path is given by two lists, one list of the ordered sequence of viewframes collected on the path, and another of the set of landmarks observed on the path. We call the viewframe list a viewpath. The landmark list acts as an index into the viewpath, each landmark pointing at the observations of itself in the viewframes. For efficiency, the landmark list can be formed as a database that can be accessed based on spatial and/or visual proximity. Visual proximity can be observed, or computed from an underlying elevation grid and a model of sensor and vision system resolution.

The first occurrence of a landmark points at the instantiated schema or perceptual structure in the vision system database that was used to gather evidence in the landmark recognition process. After that, all recognized re-occurrences of this landmark point back at this initial instance. The same is true for the first occurrences and successful re-recognition of LPB's and viewframes. This mechanism allows multiple visual path representations, built at different times, to be incrementally integrated together as they are acquired by using a common landmark indexing/pointer list.

We use an environmental representation for orientation-region reasoning that is a list of oriented LPB's encountered and crossed in the course of following a path. We call such a list an orientation-path. As with viewpaths, there is an associated landmark list that indexes into the orientation-path.

A dynamically acquirable environmental representation that merges the representations for viewpaths and orientation-paths consists of an ordered list interspersing viewframes, LPB crossings, and appearance and occlusion (or loss of resolution) of landmarks, as well as recording the headings taken in the course of following the path over which the environmental map is being built. Thus, we can integrate the representations required for viewframe and orientation region based reasoning with heading and landmark information to formulate an environmental representation that supports hybrid strategies for navigation and guidance. The representation is formed at runtime and consists of multiple interlocking lists of sequential, time ordered, lists of visual events that include those necessary for the navigation and guidance algorithms presented in Levitt et.al. - 87.

7. PROCESSING EXAMPLE

The following processing example demonstrates the behavior of some implemented system components. These include the format of predictions from the long term terrain model, the extraction of perceptually significant groupings from the PSDB, how an instantiated schema uses grouping processes and queries over the PSDB, and extracting relevant cues for making viewframe localizations in the long term terrain representation.

Figure 10 shows the elevation contours and road network in the a priori terrain data from the Martin Marietta ALV test site in Denver which was supplied by the U.S. Army Engineer Topographic Laboratories (ETL). The vehicle position on the road is indicated by the arrow in the figure. From this, we are able to roughly determine the correspondence between an image taken from the road

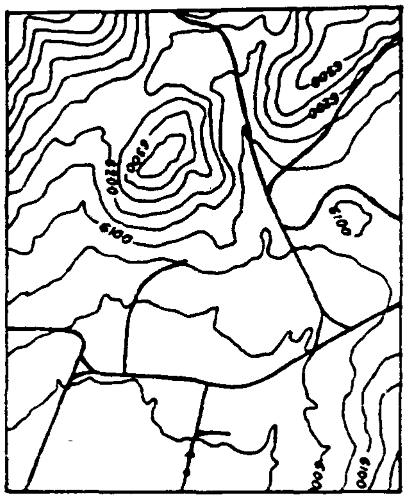


Figure 10: Terrain Data

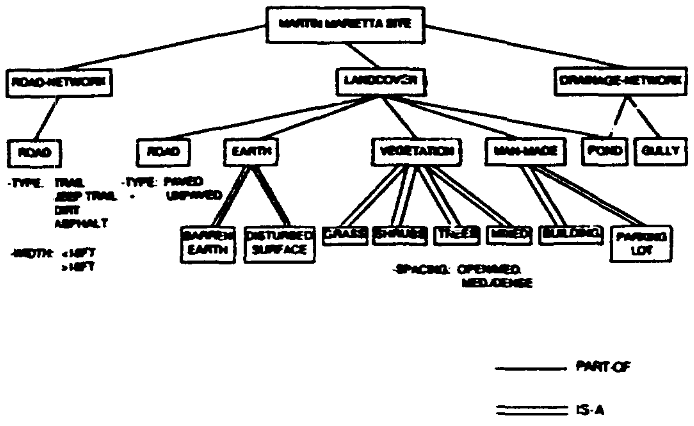


Figure 11: A Prior Terrain Type Classification

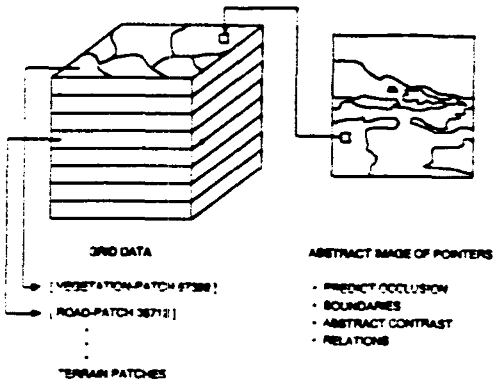


Figure 12: Predicted Segmentation From Grid Data

and the terrain data. (the relevant sensor parameters were not available). Figure 11 shows the terrain and feature classification supplied with the a priori data. These correspond to sets of image overlays in register with the elevation data. The road network is stored as a set of curve objects that is decomposed into linear segments with supplied attributes, such as road material and width. Terrain patches are extracted as regions from terrain type information and parametric surface fits to the a priori elevation data.

Figure 12 shows how the grid registered terrain data is instantiated into STM to form a predicted segmentation. The grid data regions from connected analysis correspond to schema instances in the Long Term terrain memory. Established surface display techniques are used to project the elevation with the associated schema instances to form a predicted view. Image positions are then labeled with their associated schema instances. Additionally, there are many schema instances, ordered by depth, at the corresponding image locations. The resulting predicted segmentation is processed as an abstract image where critical perceptual events are determined by size, adjacencies across occlusion boundaries, or types of terrain with high semantic contrast, such as water, fields, or man-made structures. The perceptual structures are merged together based upon distances and semantic type to yield predictions at different resolutions.

Figure 13 shows the predicted terrain patches for the vehicle positioned with respect to the terrain in Figure 10. Figure 14 shows the predicted segmentation after filtering to pull out the horizon line and road/terrain discontinuities for roads near the vehicle. This data is quite coarse (30m sampling), and image areas in the foreground are highly composite containing instances of road and the adjacent grassy fields. Nonetheless, the predicted segmentation yields a qualitative description of predicted image features that is sufficient to initialize and direct grouping processes to find corresponding image features and relationships. The key characteristics of the predicted segmentation are that the vehicle is on a flat plane, and that its field of view consists of road and grassy field terrain patches with some mountains in the distance. Predictions of the dirt road off to the right and the intersection are made from the road-network and the elevation information stored along with it. The predictions are in terms of constraints on region adjacencies across boundaries, and the shape and attributes, such as color contrasts, of the boundaries themselves. The horizon line constraints are that it will tend to have smoothly changing orientation and be adjacent to a large homogeneous region (the sky). In general, the predicted features are described with constrained attributes determined from the visibility components of schemas.

Figures 15 and 16 show some of the contour related structures in the initialized PSDB. Figure 15 shows the edges extracted at one spatial resolution using the Canny edge operator [Canny - 83]. We have found it useful not to apply noise suppression to extracted segments in order to base filtering on structural properties of the contours, including linear deviation and relationships to other image structures. Different linear segment fits for this extracted edge images are shown in Figure 16.

Figure 17 shows the results of grouping processes applied to a set of selected curves in Figure 12 with multiple associated attributes for orientation and color contrasts. The grouping processes were constrained by the predicted segmentation in Figure 14 using constraints on allowable color contrasts, changes in linear segment orientation, and rough image position and extent. Multiple groups are obtained for each predicted image event. Selection of one, or maintaining multiple alternative groups, is explicitly represented in the schema instantiation structure. Here, groups were selected based upon length and uniformity of composite attributes.

Figure 18 shows the results of a road schema instantiation based upon matches to extracted road boundaries in accounting for road surface properties through PART-OF relations. Texture elements adjacent to the road boundary which are consistent with a road surface, such as low contrast, parallel edges corresponding to tread marks, are used to direct queries to instantiate potential road area. Queries are also used to determine the presence of anomalous structures in the road such as anything which is high contrast or oriented perpendicular to the road direction. Such structures require disambiguation through instantiation of another schema (it could be a road marking) cued by the anomaly or elevation estimates derived from motion displacements or range sensing.

Significant image structures near the horizon line are particularly important for landmark extraction. Figure 19 shows extracted interesting perceptual groups near and above the horizon line. Figure 20 shows an extracted viewframe representing the relative visual spatial relationships between some of the objects extracted from this field of view.

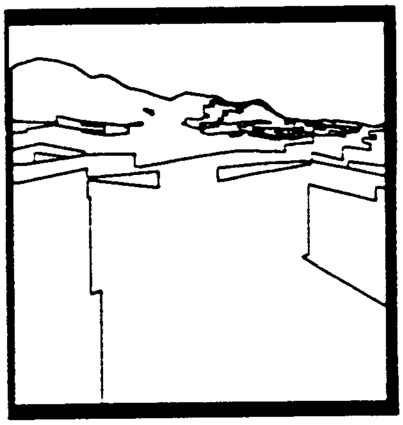


Figure 13: Terrain Patches

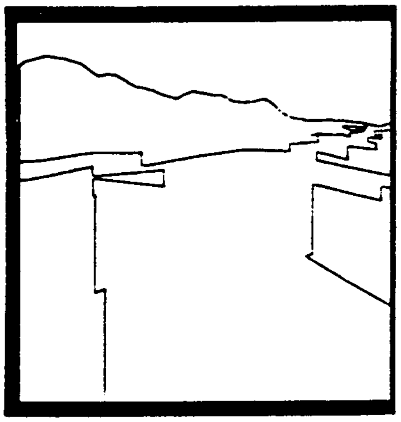


Figure 14: Merged Terrain Patches

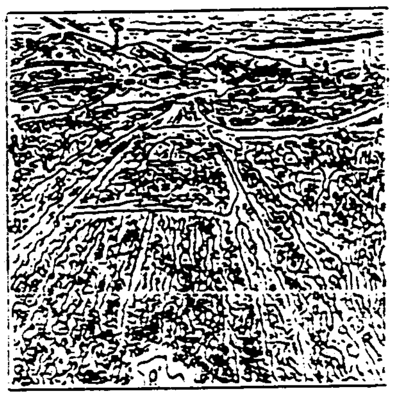


Figure 15: Canny Operator

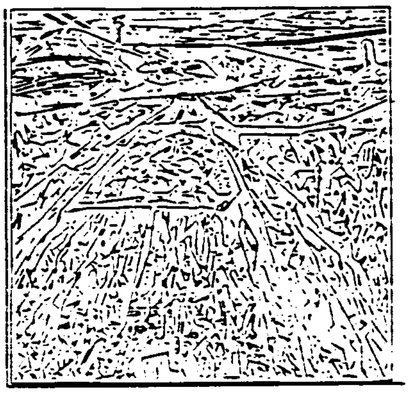


Figure 16: Linear Segment Fits

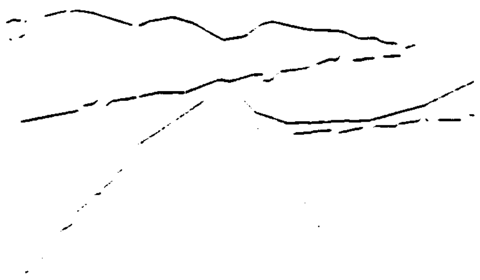


Figure 17: Contour Groupings

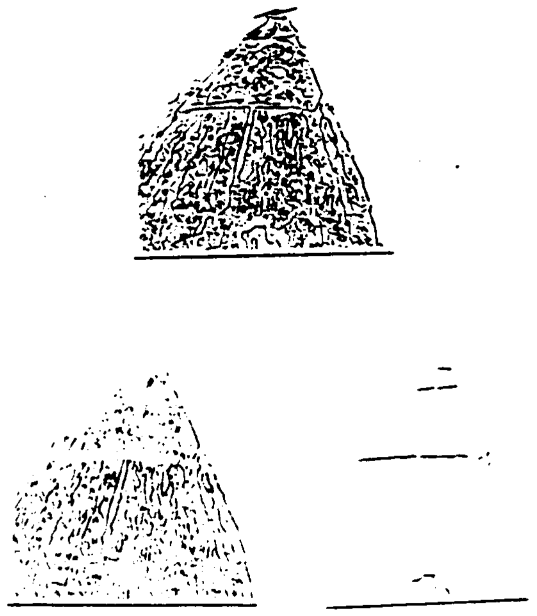


Figure 18: Road Schema Instantiation

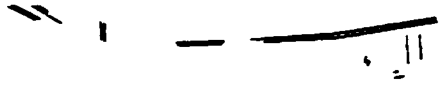
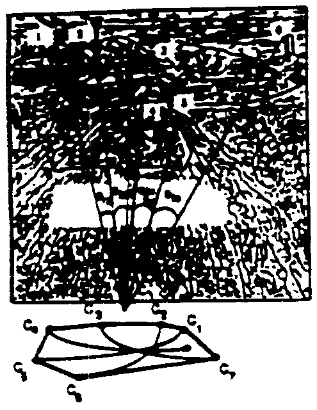


Figure 19: Significant Perceptual Groups



3D-ANGLES

[illegible]

[illegible] FOV)

RANGES

PEAK	[illegible]	based on LOS correspondence with map
POLE	[illegible]	[illegible]
NOTCH	[illegible]	based on LOS correspondence with map
[illegible]	[illegible]	[illegible]
INTERSECTION	[illegible]	[illegible]
BUILDING	[illegible]	[illegible]

LOCAL-COORDINATE-SYSTEM: (POLE, NOTCH, INTERSECTION, BUILDING)

PERCEPTUAL-LOCALIZATION-REGION: [illegible]

Figure 20: Viewframe Instance

8. SUMMARY

The architecture we have developed, using terrain and road schemas with implemented system components for perceptual processing and manipulating long term terrain data, has been successfully used in tasks for ALV navigation and scene interpretation.

9. ACKNOWLEDGEMENTS

This document was prepared by Advanced Decision Systems (ADS) of Mountain View, California, under contract number DACA76-85-C-0005 for the U.S. Army Engineer Topographic Laboratories (ETL), Fort Belvoir, Virginia, and the Defense Advanced Research Projects Agency (DARPA), Arlington, Virginia. The Contracting Officer's Representative at ETL is Ms. Rosalene M. Holecheck. The DARPA point of contact is Lt. Col. Robert Simpson.

The authors wish to thank Jeanne Gadol and Angela Erickson.

10. REFERENCES

Brooks - 84 - R.A. Brooks, "Model-Based Computer Vision", Computer Science: Artificial Intelligence, No. 14, UMI Research Press, 1984.

Canny - 83 - J. Canny, "A Variational Approach to Edge Detection", In Proceedings of the National Conference on Artificial Intelligence (AAAI-83), pp. 54-58, August 1983.

Cao et.al. - 86 - Z. Cao, S. Oh, and E. Hall, "Dynamic Omnidirectional Vision for Mobile Robots", Journal of Robotic Systems, Vol. 3, No. 1, 1986, pp. 5-17.

Hanson et.al. - 78 - A.R. Hanson and E.M. Riseman, "VISIONS: A Computer System for Interpreting Scenes", In Computer Vision Systems, Academic Press, 1978.

Kuan - 84 - D.T. Kuan, "Terrain Map Knowledge Representation for Spatial Planning", 1st National Conference on AI Applications, pp. 578-584, 1984.

Kuipers - 82 - B.J. Kuipers, "Getting the envisionment right", In Proceedings of the National Conference on Artificial Intelligence (AAAI-82), Pittsburgh, Pennsylvania, August 1982.

Levitt et.al. - 87 - T. Levitt, D. Lawton, D. Chelberg, and P. Nelson, "Qualitative Navigation", Defense Advanced Research Projects Agency Image Understanding Workshop, Los Angeles, California, February 23-25, 1987.

[Marr - 82] - D. Marr, "Vision", W.H. Freeman, San Francisco, 1982.

[Martelli - 76] - A. Martelli, "An application of heuristic search methods to edge and contour detection", Commun. ACM, Vol. 19, No. 2, February 1976, pp. 73-83.

[Ohta - 80] - Y. Ohta, "A Region-Oriented Image-Analysis System by Computer", Ph.D. Thesis, Kyoto University, Department of Information Science, Kyoto, Japan, 1980.

[Pentland - 83] - A. Pentland, "Fractal-Based Description of Natural Scenes", In Proceedings Image Understanding Workshop, Arlington, Virginia, June, 1983, pp. 184-192.

GEOMETRIC REASONING

S34-63
12P.

188116

Telerobot Task Planning and Reasoning: Introduction to JPL Artificial Intelligence Research

D.J. Atkinson
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

JJ574450

A. Introduction

The goal of building robots with increasing amounts of autonomy from direct human control or supervision is a long term goal of the Telerobot project. Among the primary missions for such robots are tasks in the areas of on-orbit satellite servicing, inspection, and various assembly operations associated with the space station. A research and development program with a series of demonstrations of increasing robot autonomy has been planned. This paper discusses some of the technical, methodological, and logistical issues of producing the artificial intelligence capabilities required by the long-term (1993 through 2000-era) demonstrations of these robot systems. A substantial research effort must begin now if the demonstration objectives are to be met on schedule. It is no coincidence that many of the same objectives for Telerobot artificial intelligence research are common to both the Telerobot and System Autonomy programs. Many of the capabilities and associated research required for autonomy transcend the particular application.

The planned series of Telerobot demonstrations provides a relatively clear progression of increasing sophistication in the artificial intelligence capabilities required. As a directed research and development program, these demonstrations provide considerable *applications pull* in the kind of research issues which may be addressed. However, applications pull alone can easily become overly restrictive to new research ideas and also unrealistic about existing or projected technological capability. Basic research, on the other hand, provides the *technology push* towards new ideas. Ultimately, this push results in demonstrations and application of more powerful capabilities. Technology push by itself runs the risk of being distractable and undirected towards economically realistic and desirable application goals. Clearly, a mixture of both applications pull and technology push is necessary. Achieving a sensible balance between these two opposing forces will be major criteria of on-going and overall success of the Telerobot project.

Due to the scope of the artificial intelligence technology required by Telerobot, one of the conclusions to draw from this document is that our goals for Telerobot artificial intelligence will be difficult to achieve unless we make maximal use of all available research resources. NASA probably does not have the desire or capability to fund all of the artificial intelligence research which will be required by

PAGE 338 INTENTIONALLY BLANK

Telerobot, nor should NASA duplicate research conducted under other auspices. We will need to draw extensively upon research results generated by the Systems Autonomy Program at the NASA Ames Research Center laboratory, and externally funded laboratories at major universities and within the AI industry.

B. THE LONG-RANGE GOAL: CAPABILITIES

This section details the specific capabilities which will be required in the area of Artificial Intelligence for the long-term Telerobot demonstrations leading to operational 2000-era autonomous robots. These capabilities should be regarded as *necessary* for the specific objectives of the demonstration program. However, they should not be construed to be a *sufficient* set of capabilities. This can only be established by continuing research, experimentation, and the demonstrations themselves. Subsequent sections will provide detail on some of the technical issues which must be addressed.

B.1 TASK PLANNING AND SCHEDULING

A number of processes must operate at the **Task Level** to robustly control the achievement of goals. One of these is task planning and scheduling. Its function is to select and schedule activities whose collective effects logically achieve the desired goals. In the Telerobot, some of the planning and scheduling capabilities required are:

- **Use of highly structured procedures.**
  Many of the tasks which the robot will carry out have been engineered beforehand, e.g., there are satellite servicing procedures which must be followed in a definite, prescribed sequence. The robot should have the ability to select the appropriate procedure and combine it with other elements of its activity plan. It will be a rare case when predefined procedures can be simply "invoked" without regard to plan interactions or the potential for error due to real-world uncertainty.

- **Creation of ad hoc routines.**
  The robot must be able to reason about the task requirements of novel goals. All situations in which the robot is expected to behave cannot be conceived or engineered beforehand. There will undoubtedly be circumstances when a structured procedure has not been developed. This will certainly be the case when a structured procedure fails and error recovery processes must create appropriate plans to handle the novel circumstances. The robot must be able to freely mix planning using structured procedures and ad hoc routines.

- **Management of task resources.**
  In the accomplishment of any goal, the robot can draw only upon limited resources. There are a finite number of effectors available, a finite number of sensors to employ, a finite number of interchangeable or unique tools or replacement parts, etc. Management of resource constraints is a fundamental component of creating viable plans.

- **Planning with uncertainty.**
  The ability to formulate robust plans when there is considerable uncertainty about the run-time environment, the robot's ability to affect it, or the knowledge used in planning (see **Uncertainty management** below).

- **Multi-agent cooperation.**
  The ability to formulate plans which provide for specific tasks to be accomplished in a coordinated fashion by other intelligent agents, robot or human. Also, the ability to formulate plans which are robust in circumstances where the actions of other intelligent agents are not prescribed or are otherwise unpredictable.

B.2 SPATIAL PLANNING AND REASONING

The function of spatial planning and reasoning processes is to bring the logical procedures of activity planning into the real-world by making geometric, physical, or temporal constraints on activities an integral part of the task level control process. This may require a theory of manipulative processes which support planning for handling, service, repair, construction, and inspection. Some of the capabilities which are required are:

- **Reasoning about robot, workspace, and workpiece geometry.**
  Geometric constraints on tasks can play a significant role in the selection and sequencing of appropriate actions by the robot. Given a device to be disassembled for repair, the selection and sequencing of manipulations is dependent on the geometric configuration of the device, e.g., you can't remove an orbital replacable unit (ORU) without removing an access panel. Contact and attachment constraints easily generate other examples where geometric reasoning is important.

- **Reasoning about physical processes.**
  The ability to plan for or around the effects of physical processes is important. When a tool is released in zero-g, the robot must understand how physical processes such as inertia will affect the state of the tool, e.g., will it float away and if so, where and at what speed? Often only qualitative answers are required in the planning process (e.g., the tool will rotate, float left, etc.), but the robot must be able to utilize precise quantitative information where necessary (e.g., to direct the movement of a manipulator towards a wrench which is floating away). Another example would be reasoning about the flow of fluid, such as fuel, through a hose in a refueling operation. The planning process must take into account physical constraints on planned activities.

- **Planning for workpiece and robot positioning.**
  The robot must be able to reason about how to position itself and the workpiece to enable required operations, and when this is impossible, to provide the appropriate constraints for activity planning. In zero-G, without the default constraints and orientation provided by Earth gravity, a mobile robot has considerable latitude in this area which should be exploited.

- **Planning for robot movement.**

The ability to plan reasonable and safe manipulator trajectories through a cluttered workspace is a critical component of task level control. Collision avoidance with fixed obstacles, stationary but movable obstacles, and moving obstacles is a significant problem.

- **Mobility.**
  The ability to plan the movement of the robot and navigate in real-time through a 3-dimensional environment. The robot should have the ability to maintain its position relative to moving workspaces (i.e., station keeping) as well as plan its movements so as not to conflict with other mobile objects (e.g., other robots, astronauts on EVA, or spacecraft).

- **Temporal Reasoning.**
  Reasoning about the duration and timing of activities and physical processes introduces additional constraints on the task planning process. The scheduling of activities to accommodate real-world events, including the actions of other agents, will rely on an effective temporal reasoning ability.

- **World modelling.**
  The ability to maintain a coherent geometric and physical model of the world as it dynamically changes as a result of robot actions, the actions of other agents, and controlled or uncontrolled physical processes is critical. Virtually all task level control processes must have access to a (reasonably) consistent set of beliefs about the state of the world in order to make assumptions for planning, diagnosis, error recovery, and other functions. When conflicting information and ambiguity are introduced, this can become a serious problem.

B.3 EXECUTION MONITORING

In a sometimes malevolent and only partially-modelled real world, a robot executing a plan may not actually achieve the desired effects of each action while still obeying any constraints imposed by the plan. There are a host of potential problems which may create a divergence at execution time from the expected effects of actions as described in the plan. It is the role of execution monitoring processes to determine whether the plan is executing nominally, and to track the state of the world during plan execution. A number of capabilities are important:

- **Verification and sensor planning.**
  The ability to determine how best to employ sensors during plan execution to verify that robot actions are achieving their intended effects, and to notice other events which may affect successful plan execution. Planning and scheduling, resource management, and sensor modelling techniques are aspects of this process.

- Expectation generation.
  In order to determine that the intended effects of a plan are actually occurring, qualitative and quantitative predictions about what information will be obtained on sensors during plan execution must be generated.

- **Situation assessment and sensor data fusion.**

In lieu of perfect models of the sensors and the world, the expectations about information on various sensors will be approximate at best. Techniques must be developed to make accurate judgements of whether expectations are satisfied or violated from partial data in predictions and obtained from sensors. Frequently, information from multiple sensors will need to be coordinated in this process.

- **Recognition of unexpected events.**
  The ability to recognize and characterize events or states of the world which are unanticipated. In the formulation of plans, the robot will not have the ability to predict all events which may affect the successful execution of the plan. In order to robustly accommodate potential conflicts at plan execution time, the robot must first be able to notice and describe unexpected situations.

B.4 DIAGNOSIS AND ERROR INTERPRETATION

When errors or unexpected events are detected during plan execution, diagnosis and other error interpretation processes must attempt to discover the source of the problem. In the absence of a precise diagnosis, error recovery or restart procedures will be coarse and may needlessly cause the duplication of work successfully completed or even the overall failure to achieve the required goals of the robot. On the other hand, a precise diagnosis may indicate a simple procedure to recover (e.g., regrasp a tool if it slips during usage) or the need to plan a more substantial recovery (e.g., when the satellite's access panel is not attached as described in the CAD database). The following capabilities are required:

- **Localizing system failures.**
  Identification of single and multiple point failures of the hardware of the robot or associated systems. This type of diagnosis is critical for determining the status of the robot and its ability to accomplish required tasks.

- **Knowledge-base criticism.**
  Occasionally, the knowledge which the robot uses to plan actions, monitor, or otherwise solve problems will be missing or in error. The capability to spot these gaps or errors will be essential, especially when machine learning becomes an important process.

- **Debugging lower-level controllers.**
  In some cases, lower-level controllers may operate incorrectly for a variety of reasons. The ability to make this decision (and possibly suggest a restart of the controller or other recovery action) is important.

- **Diagnostic test design.**
  The robot should have the ability to devise and utlitize diagnostic tests to discriminate among multiple competing hypotheses. Some tests may involve passive inspection using sensors (e.g., use vision to determine if an access panel fastener is missing). Others might involve more active procedures which could compound the error if applied without due consideration (e.g., to check if the effector is jammed, rotating it in a confined area could damage nearby satellite components).

- **Diagnostic techniques.**
  The capability to employ a variety of different styles of diagnosis as required by the problem at hand. Sometimes heuristic techniques will be applicable and serve to quickly localize a problem. In other cases, the ability to reason in depth about the structure and function of robot components will be important. The ability to use causal modelling techniques to project the effects of possible failures will be useful in other circumstances.

- **Explanation.**
  The ability to describe and justify diagnostic conclusions to the level of detail as required by a system operator (see **Human Interface** below) or other knowledge based system. For effective error recovery, a diagnosis must have enough precision to specify the requirements for error recovery (e.g., "The forearm is broken" lacks sufficient detail for recovery). In addition, the ability to justify the particular conclusion reached (especially when there are closely competing alternative hypotheses) will be important for human evaluation and risk assessment of possible recovery procedures.

B.5 ERROR RECOVERY PLANNING

After a diagnosis has been determined, it is the role of error recovery procedures to plan how to get the plan back on track. This may involve small modifications to the plan at hand, the replanning of significant procedures, or even the abandonment of goals deemed impossible to accomplish in the changed circumstances. Some capabilities which will be important are:

- **Selection of predefined procedures.**
  In some contingencies, recovery procedures will have been devised beforehand. The robot should have the ability to determine the applicability of these recovery procedures from a machine or human generated diagnosis.

- **Discrepancy analysis.**
  The ability to determine what the effects of a particular failure are on planned actions, scheduled events, or actions in progress. In major failures, much of the plan remaining to execute will become invalid. However, in other cases the effects of failures may be slight and the existing plan resumed after some simple local error recovery. Discrepancy analysis will be critical in not wasting the work already put into planning and plan execution.

- **Ad hoc planning and plan integration.**
  If predefined recovery procedures are inapplicable to the situation at hand, the robot must have the ability to specify the requirements of a novel procedure, invoke the appropriate planning process, and then incorporate the new procedure into the existing plan. Precision in the ability to *excise* the invalid components of the old plan and *splice in* the new procedure will be probably depend on **Discrepancy Analysis** as described above.

- **Goal maintenance.**
  The robot must be able to determine when a plan is irrecoverable and a goal must be abandoned or revised. In addition to the case where recovery is

impossible, in some cases recovery may be too time consuming, too risky for the robot or workpiece, or unsafe for astronauts working nearby (e.g., venting fuel from a clogged or kinked refueling hose).

- **Revising and acquiring knowledge.**
  If diagnosis processes have determined that some aspect of the robot's knowledge was in error, misapplied, or missing, a recovery procedure must determine how to correct the problem. This is one aspect of the **machine learning** problem, resolution of which will certainly be important in the achievement of full robot autonomy.

B.6 SIMULATION AND PREDICTION

Central to much of the robot's ability to solve problems is the capability to create and reason about the potential extended effects of alternative actions or events. In order to plan, the cumulative effects of alternative actions must be considered. In order to anticipate the long-term effects of potential failures, the immediate effects of the failure must be extended forward through time. Sometimes, in order to discover the cause of current anomalies, the ability to reason how the effects of hypothetical *previous* failures could propagate will be important.

- **Causal Simulation.**
  The ability to utilize causal models to envision the effects or causes of particular states or events which concern the robot.

B.7 REAL-TIME PROBLEM SOLVING AND CONTROL

Of considerable concern is the robot's ability to behave in real-time. Situations which are beyond the robot's ability to "freeze" will be common (e.g., moving obstacles) and high performance will be required. The following capabilities are important:

- **Meta-level control.**
  The ability to utilize knowledge about on-going problem solving to judge the relative importance, efficiency, reliability, and potential for success of alternative problem-solving strategies which are competing for real-time computing resources. The ability to make a quick and accurate judgement about where to focus problem-solving will be important in achieving high performance.

B.8 INTEGRATION

The Telerobot requires that a diverse set of software and hardware must smoothly integrated. From the point of view of the AI systems involved, two capabilities are critical:

- **Integration of multiple knowledge based systems.**
  Much of this paper describes capabilities which will be provided by knowledge based systems. The ability to integrate this eclectic group of systems into a single functional unit is critical in achieving the full scope of robot autonomy.

In the past, the technical problems associated with each of the areas described have generally been studied in relative isolation from one another. A fully autonomous system, however, must rely on multiple knowledge based systems which can interact, cooperate, or merely tolerate one another in a common computing environment.

- **Integration with non-ai systems.**
  The knowledge based systems, part of the robot's task planning and reasoning component, must be able to interact with the other non-AI systems which compose the bulk of the robot's control, sensory, and human interface abilities. Protocols for interaction with these systems must be established.

B.9 UNCERTAINTY MANAGEMENT

As the System Autonomy research plan states, uncertainty management is "...the ability to make sensible judgements and carry out reasonable actions when world knowledge is imprecise or incomplete, heuristics or models have built-in uncertainty, or actions have uncertain effects." Any system which behaves intelligently and robustly in the real world must account for the inherent lack of precision in knowledge and ability to control the world. The following capabilities will be important:

- **Identification of sources of uncertainty.** The robot must be able to locate and accommodate a wide variety of sources of uncertainty. For example, uncertainty can arise from partial or imprecise models, such as gaps in knowledge about the cause and effects of events or properties of objects (including the robot itself), lack of knowledge about the effects of active physical processes, or imprecision in modelling the intentions and plans of other agents.

B.10 HUMAN INTERFACE AND INTERACTION

Several human operator responsibilities must be supported by the human interface to the Artificial Intelligence components of the Telerobot:

- **Supervision.**
  The operator must have the ability to determine, select, and specify goals for the robot to achieve prior to and during plan execution. The operator must have the ability to preview and select among alternative plans suggested by the robot. At all times during plan execution, the operator must have the ability to suspend or redirect the activities of the robot.

- **Criticism.**
  The operator must have the ability to evaluate and criticize plans. This could include the ability to make modifications to plans (e.g., manipulator trajectories) which are otherwise acceptable to the robot.

- **Cooperation.**
  The operator must have the ability to be involved in both the problem-solving activities of the robot as well as the actual execution of plans themselves. Especially in the early stages of development, the robot will often lack critical

knowledge about how to locate or recover from specific errors which occur during plan execution. The human operator must have the ability to instruct the robot and/or carry out the operations alone. In later Telerobot development, the ability for the robot and human to coordinate their activities will be important

B.11 KNOWLEDGE ACQUISTION AND LEARNING

There is a difficult bottleneck in the creation of knowledge based systems, i.e., the identification, acquisition, representation, verification, and management of the knowledge which is required for problem solving. Knowledge-engineering has today developed into a skilled craft with numerous tools to aid a human developer of knowledge based systems. However, the volume and complexity of the knowledge required for the Telerobot (and other complicated systems) is sure to overwhelm the techniques which exist today. Ultimately, the robot should be able to perform much of the knowledge acquisition and maintenance problem autonomously, i.e., learn. Machine learning has the role of resolving uncertainty, correcting knowledge errors or gaps, and in generating new capabilities for the robot. The following capabilities leading to autonomy are important:

- **Use of CAD/CAM databases.**
  Effective diagnosis, assembly, inspection, and other tasks designated for the robot will require detailed knowledge about the structure and function of the objects it manipulates, such as satellites. Much of this information is expected to be available in computable form in CAD/CAM databases. Techniques must be developed for exploiting this information and transforming it into representations usable for knowledge based problem-solving.

- **Use of human documentation.**
  Many of the tasks which the Telerobot is expected to automate have been designed for humans. To a certain extent, human readable documentation exists on these tasks or the systems to be manipulated. Techniques for utilizing this knowledge resource should be developed, including natural language parsing.

- **Management of massive knowledge bases.**
  There are numerous problems associated with managing the volume of knowledge which will be required by the robot. There will probably be multiple representations of the same information which must be consistent. At any given stage of problem-solving, only a subset of knowledge is required. This must be quickly and efficiently provided. The knowledge base management problem appears to subsume many traditional database management problems.

- **Learning by experience.**
  When the robot makes a mistake during plan execution, and identifies the problem as imprecision in its knowledge, it shouldn't make the same mistake a second time. In some cases, the knowledge is correct, but simply inapplicable to the current situation in which the error occured. In this case, the robot must be able to recognize in future situations when the same mistake could occur, recall the correction it devised previously, and implement the correction in the current situation.

- **Learning by discovery.**
  The robot should have the ability to fortuitously notice or bring about situations which are instructive.

- **Learning by teaching.**
  The robot should have the ability to acquired knowledge from direct interaction with humans, either through factual presentation, reasoning from examples, or other methods of instruction.

C. TECHNICAL AREAS FOR RESEARCH

This section presents some of the technical areas for Artificial Intelligence research which should be supported by the Telerobot project over the next several years to achieve some of the capabilities noted above. NASA probably does not need to sponsor work to achieve all of these capabilities; many required capabilities cross application boundries and the necessary research is currently funded by other government agencies. NASA does need, however, to focus research on those areas which are critical for the near-term establishment and success of the Telerobot (circa 1993). The following areas are relevant:

- **Planning and scheduling.**
  Near term: Research on conditional, contingency, and least commitment planning. Research on incorporation of physical, spatial, and temporal constraints and associated planning processes with task activity planning. Research on resource management. Management of uncertainty in planning. *Longer term:* Research on multi-agent cooperation, including: Plan recognition, communication of plans and intentions between agents, command and information request communications, real-time compensation for other agent action discrepancies, supervisory versus distributed control issues.

- **Spatial planning and reasoning.**
  Near term: Spatial model representations and databases which are useful for problem-solving tasks and easily integrable with knowledge based systems. Qualitative causal modelling of physical processes. Reasoning about spatial and physical constraints on task planning processes. Manipulator trajectory planning and collision avoidance. *Longer term:* 3-D robot mobility and navigation.

- **Execution Monitoring.**
  Near term: Sensor planning issues, including: Plan action analysis to support sensor allocation, sensor resource allocation and real-time management, formal languages for specification of sensor plans, issues in active versus passive sensing and plan interactions, selective versus exhaustive monitoring issues, grain-size of monitoring, expectation generation. Monitoring issues such as: symbolic classification of sensor data, partial matching. *Longer term:* Issues in multi-sensor and temporal data fusion, uncertainty management, partial matching, noticing unexpected events.

- **Diagnosis.**
  Near term: Identification of source of plan failures, including system failures, knowledge failures, or unexpected world states or effects of actions. Utilization

of multiple kinds of diagnostic knowledge, including heuristic, first principles, procedural. *Longer term:* Classification of novel failures, localizing multiple failures, assumption changes, troubleshooting knowledge failures.

- **Error recovery.**
  Near term: Analysis of discrepencies on exisiting plan, plan representation and operators for combining plan fragments such as recovery plans. Integration with planning processes. *Longer term:* Determining discrepency effects on planning knowledge (a learning problem), including: recognizing that knowledge is at fault, determining which knowledge, explaining why it is wrong, and implementing corrective action.

- **Simulation and prediction.**
  Near term: Causal modelling for the Telerobot domain, issues in modelling temporal constraints and physical processes. *Longer term:* Dealing with bad or ambiguous causal models.

- **Real-time problem-solving.**
  Near term: Issues in goal, planning, and execution management, including: goal valuation and priority, goal viability, recognition and management of dynamic resources required to achieve goals, interaction with human goals. *Longer term:* Issues in parallel or distributed processing of knowledge based systems. Real-time problem-solving software architectures.

- **Integration.**
  Near term: Issues in integration of multiple knowledge based systems, including: Message and request communication, shared knowledge, executive versus distributed control. Issues in integration with non-ai systems, including superordinate and subordinate roles. *Longer term:* Integration architectures, including blackboards, distributed computing systems.

- **Human interface.**
  Near term: Knowledge based system command languages, graphical and iconic display of plans, interface for supervised plan execution. *Longer term:* Natural language understanding and generation, mixed initiative dialogs, generation of explanations of knowledge based system behavior, communication of shared knowledge.

D. METHODOLOGY

The Telerobot project involves research and development in Artificial Intelligence and other areas which spans the range from highly risky and innovative basic research through applied research and engineering to application engineering of operational systems. It is essential that a flow of information, techniques, skills, and even personnel be maintained across this span of research and development. A recent perspective article in *Science* by Dr. A. M. Clogston of AT&T Bell Laboratories may be instructive:

> "...The research-development interface is a difficult enough barrier to surmount, even within a highly integrated R&D laboratory, and it is

more difficult to import research, even with the best will on both sides. The best way to import university research into an R&D laboratory is through active in-house basic and applied research groups, a channel that would be closed if the laboratory relied too much on external research."
– A. M. Clogston, "Applied Research: Key to Innovation", *Science*, **235**, 4784, (1987)

Our methodological goal should be to combine a set of tightly focussed external research contracts with substantive in-house basic and applied research directed at:

- Importing this technology and technology sponsored by other agencies into the Telerobot project.
- Filling in the "gaps" in basic and applied research areas which are important for Telerobot.
- Performing the applied research necessary to move conceptual breakthroughs into operational demonstrations of robot capability.

To facilitate the importing of university technology, individual graduate students and professors conducting research sponsored by, or of interest to, the Telerobot project will be invited to visit the laboratory and work with JPL personnel for short periods of time. This should enhance the education of JPL personnel in emerging AI technology as well as provide the necessary project visibility and feedback to basic research efforts.

E. CONCLUSIONS

T~~his paper has presented~~ a view of the capabilities and areas of artificial intelligence research which are required for autonomous space telerobotics extending through the year 2000. In the coming years, JPL will be conducting directed research to achieve these capabilities, as well as drawing heavily on collaborative efforts conducted with other research laboratories.

F. ACKNOWLEDGEMENT

The research described in this paper was done at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

S35-61
33117
100
188117
CH188054

Geometric Reasoning

R.F. Woodbury and I.J. Oppenheim
Carnegie-Mellon University
Pittsburgh, PA 15213

1. Introduction

~~1.1. Cognitive robotics~~ and space telerobotization

Cognitive robot systems are ones in which sensing and representation occur, from which task plans and tactics are determined. Such a robot system accomplishes a task after being told "what" to do, but determines for itself "how" to do it. Cognition is *required* when the work environment is uncontrolled, when contingencies are prevalent, or when task complexity is large; it is *useful* in any robotic mission. A number of distinguishing features can be associated with cognitive robotics, and one to be emphasized here is the role of artificial intelligence in knowledge representation and in planning. While space *telerobotics* may elude *some* of the problems driving cognitive robotics, it shares many of the same demands, and it can be assumed that capabilities developed for cognitive robotics can be employed advantageously for telerobotics in general.

The top level problem is *task planning*, and it is appropriate to introduce a hierarchical view of control. Presented with certain mission objectives the system must generate plans (typically) at the strategic, tactical, and reflexive levels. The structure by which knowledge is used to construct and update these plans endows the system with its cognitive attributes, and with the ability to deal with contingencies, changes, unknowns, and so on. T~~his paper is not on the topic of AI task planning *per se*, but rather on~~ issues of representation and reasoning which are absolutely fundamental to robot manipulation, decisions based upon *geometry*, are discussed here, not AI task planning per se.

1.2. Representation and Manipulation of Geometric Information

The title to this section is the essential statement of our research interest; it can be paraphrased as *geometric reasoning*. Consider the following steps in problem solving for a robotic manipulation problem:

- Identification of the environment and objects within it.
- Identification of physical relationships between objects.
- Generation of plans to accomplish robotic task objectives; decisions at different levels of abstraction.
- Execution of any manipulator motion.

Complex cognition on geometric objects is reflected in every step. The research question before us is to permit computer-based reasoning on geometry, utilizing constructs that human designers and users of robotic systems can employ. A particular purpose is the provision of an environment in which a knowledge based system (KBS) can be implemented which presents geometric knowledge employing the powerful concepts that humans use, and which does so uniformly with other knowledge in the KBS.

Geometric reasoning spans issues in the representation, and intelligent manipulation of models which describe geometric objects. It is of general applicability over the entire range of disciplines which address real-world physical objects. While differences in the information requirements of various disciplines exist, the commonalities with respect to geometric information are marked. This research takes as its point of departure these commonalities and seeks an architecture for geometric reasoning. The methodology of the research has two related aspects. The first is a search for concepts around which such an architecture may be organized. The second is the development of a prototype system, both as a medium for exploration and as a demonstration of concept.

The group of computer languages known as knowledge based expert system languages are of greatest interest, as it is in these languages that much of the current work in real-world intelligent computation is being expressed. Of particular interest are object oriented languages, for two reasons: 1) they provide convenient means for the description of and computation on inherently hierarchical information, 2) techniques are rapidly developing to integrate the other main forms of expert system programming, rules and logic programming, into the object oriented paradigm.

1.3. Outline of Paper

Section 2 discusses the *Background* to work in this area: geometric reasoning itself (where it has been identified as such), treatment of constraints, certain origins within artificial intelligence proper, geometric modelling, computational geometry, and graphics. Section 3 contains our proposed *Architecture* for geometric reasoning: the requirements, the concepts brought to our design, the system description, and a brief statement of the present implementation. Section 4 contains the *Conclusions and Recommendations*, including an outline of further approaches to geometric reasoning by some colleagues at Carnegie-Mellon University.

2. Background

Developments in several disciplines provide starting places and points of departure:

Geometric Reasoning itself has a significant, but modest and scattered, literature. Kuipers [Kuipers 77] created a a theoretical model of human spatial cognition using concepts of representation developed by Lynch [Lynch 60]. These concepts are related together in a network, through which propagation is used to perform inference. An implicit hierarchical organization is provided by using containment relations on spatial entities. Brooks [Brooks 81] developed a constraint based geometric reasoner as part of of the ACRONYM vision system. The reasoner maintains a class hierarchy (Brooks calls it a restriction graph) of geometric representations based upon generalized cylinders. The parameters which determine a generalized cylinder are restricted through the use of algebraic constraints. Geometric reasoning is accomplished by algebraic and numerical methods to determine bounds on satisfiability of constraint sets and extremum of objective functions. McDermott described metric spatial inference, a technique to make inferences about the relative locations of objects based on simple descriptions. This work was extended by Davis [Davis 81]. Constraints on the relative locations of coordinate systems are used to define *fuzzy maps* which capture the constraint information in a form amenable to computation of queries. Davis [Davis 86] greatly expanded on this earlier work to develop a theory of cognitive mapping which particularly addressed issues of representation, retrieval and assimilation. The seminal contribution of this work is the formalization of a means of building a map incrementally and of performing inferences on incomplete maps. Akiner [Akiner 85] built a geometric reasoner based on resolution theorem proving in the domain of right rectangular orthogonally oriented cuboids. Geometric objects are expressed as assertions in Prolog and reasoning is accomplished by application of the backtracking search in Prolog using a set of rules about geometry. Dixon [Libardi 86] [Nielson 86] has used the concept of features to build various specialized representations for design problems. Features are groupings of boundary elements of a solid that provide convenient units of concept to application programs. Wing and Arbab [Wing 85] outlined a deductive geometric reasoning system. The main contribution of their proposal is recognition of the need for a clear language for geometric reasoning.

Constraints are the basis for certain types of computation which seek to maintain some characteristics of an object constant under modification of the object. Constraints are intimately related to dimensioning, tolerancing and variational geometry. The literature on constraints is large and growing. Sutherland [Sutherland 63] created SKETCHPAD, recognized as the seminal work in the area. The contributions of SKETCHPAD include propagation[1], numerical solution through relaxation and a method (pins) for linking constrained objects. Steele and Sussman [Sussman 80] present a language for the representation of almost hierarchical constraint networks, a method of explanation of constraint calculations and a simple solution technique called *local propagation*. They also discuss algebraic transformations of constraint networks. Steele, in his dissertation [Steele 80], deeply examines the implementation of constraint systems. Borning [Borning 79] developed a programming language dominated by constraints. His contributions included a strong use of the object oriented programming paradigm, a local and fast propagation algorithm and the use of planning for propagation. Light and Gossard [Light 82], demonstrate *variational geometry*, an instance of the relaxation algorithm used to dynamically manipulate parametric primitives. Gosling [Gosling 83] contributed several new techniques for constraint satisfaction. These include: the use of breadth-first search to plan propagation, the use of automatic transformation of constraint networks and a fast graph isomorphism algorithm to make frequent use of transformation feasible.

Artificial Intelligence and related fields provide key concepts of search, hierarchical knowledge representation, inheritance, theorem proving and deduction. Specific results in robot task planning [Fikes 71] [Fahlman 74], geologic map interpretation [Simmons 82] and real-world simulation [Klahr 82] provide useful precedents. STRIPS [Fikes 71] demonstrated robot task planning in a circumscribed domain. In contrast, Fahlman [Fahlman 74] developed a planning system which was highly dependent on a powerful underlying model. The resulting simplification of the actual planner demonstrates the effectiveness of isolating geometric issues. Simmons [Simmons 82] used a diagramming scheme separate from the rest of his program to compute adjacencies, positions and orientation of parts of geologic formations. Klahr [Klahr 82] discussed various approaches to geometric relationships in the development of an object-oriented battle simulator. His conclusions concerning the use of so-called *auxiliary objects* strongly support an independent treatment of geometric information.

Geometric Modelling grew out of early applications of computers to design and manufacture [Requicha 80] [Eastman 79]. Geometric Modelling is concerned with the representation and manipulation of subsets of three-dimensional Euclidean space and with the construction of computer systems to support these tasks. Several significant and distinct methods of representation are derived from Geometric Modelling. These methods are: pure primitive instancing, spatial occupancy enumeration, tree decomposition, sweep representation, cell decomposition, constructive solid geometry and boundary representation. Of these, tree decomposition, constructive solid geometry and boundary representation are of most significance to this work. In recent years a great number of results have emerged in this field.

Computational Geometry is, according to Preparata and Shamos [Preparata 85] an attempt to *"reshape - whenever necessary - the*

[1]Called the one pass method by Sutherland.

classical discipline (of geometry) into its computational incarnation." This field is concerned with the design and analysis of problems in the construction and computation of properties of geometric objects. Preparata classifies problems in computational geometry into five categories, each describing problems in one of the following areas: convex hull, intersection, geometric search, proximity and geometric optimization. From this field may be garnered good algorithms for these problems.

Computer Graphics has spawned a wide variety of techniques for the creation of images on computer output devices and for visually based human-machine interaction. The fundamentals of this field, especially rendering algorithms and interaction techniques have relevance to the present research and are of interest for the services they can provide in creating a usable geometric tool.

3. An Architecture for Geometric Reasoning

3.1. Requirements

An architecture for geometric reasoning must meet several requirements. It must:

- Represent a wide DOMAIN OF GEOMETRIC ENTITIES. A geometric entity is a subset of three dimensional Euclidean space (R^3). Combinations of restrictions based upon finite describability, compactness and regularity can be used to specify a large and useful set of classes of geometric entities. A myriad of algorithms exist to represent and manipulate members of these classes.
- Represent a wide DOMAIN OF SPATIAL RELATIONSHIPS. An object in space is located with reference to some other object or to some coordinate frame. Such a reference establishes a relation between the located object and its referent. Relationships are a key component of modelling systems as most modelled objects are not homogeneous but consist of many interrelated parts. Many types of relationships exist; of particular interest here kinematic, locational and topological relationships.
- Support a wide variety of useful QUERIES on geometric data. Queries take as input a description of subsets of R^3 and produce as results another subset of R^3, a vector, a scalar or some textual description. The results of queries may be considered as partial models of the original subset of R^3.
- Support the general CLASSIFICATION of geometric objects. Geometric entities may be classified by the results of any query on those entities. A classification scheme examines a representation of geometric information and determines if it denotes an object which belongs to any of the classes defined in the scheme.
- Provide a general means to GENERATE families of objects. An incomplete model of a geometric object defines a class of objects, C, all of which are partially described by the incomplete model, but which differ in characteristics not described by the partial model[2]. A generation algorithm produces members of the class C when given as input an incomplete geometric model. If the generation algorithm can produce all such members, it is said to be exhaustive.
- Provide a wide variety of MODIFICATION TECHNIQUES on geometric data. A modification of geometric data is an operator which changes some aspect of the data while leaving other portions unaffected. Both object definition and object altering operators are modification techniques.
- Be directly ACCESSIBLE from a knowledge based system. A geometric reasoner is intended to be used as a utility for the representation and manipulation of geometric information within a knowledge based system. It must be accessible from such a system in a clear, well-defined manner.
- Be EXTENSIBLE to include new algorithms for geometric computation, without structural change to the system. Both the number of applications which use geometry and the number of algorithms on geometric representations are growing explosively. A geometric reasoner must be able to gracefully accept the addition of new algorithms and representations if it is to continue to be useful.

3.2. Concepts

The requirements above pose constraints on the performance of a system, not upon the means of achieving such performance. We have developed a system architecture within which these requirements may be achieved. This architecture is comprised of three component concepts: classes of spatial sets, geometric abstractions and features. The following describes each of these concepts individually and then relates the concepts together to form the overall architecture.

3.2.1. Classes of spatial sets

Spatial sets may be organized into classes. A class is distinguished by certain properties that must be true for all of its members. For example, the class of regular polyhedra consists of those polyhedra which have identical vertices and dihedral angles and faces which are congruent, convex and regular. Classes are recursive; they may contain subclasses. A subclass is governed by all of the true properties of its superclasses. The class concept is familiar to many, particularly those who have studied object-oriented programming. It addresses three of the requirements outlined above.

[2]An example is an adjacency matrix as a partial representation of a layout of rectangles.

1. It facilitates the definition and specification of a wide domain of spatial sets.
2. It provides a mechanism for the automatic classification of spatial sets.
3. It establishes a clear framework for extensibility.

A wide domain of spatial sets is naturally supported by the class concept. Structuring geometry into classes allows the transparent definition of useful representations and algorithms for different objects. Each class of objects, for example polygons, may have a specialized data structure or structures and a set of algorithms for computing such properties as displays, areas, convex hulls and bounding boxes. A class may partially define some its properties. For example, the location of an object in space may be unrestricted by a class, or may be restricted to a subset of space. This ability to partially define properties supports the meaningful creation of instances of a class. An instance displays all of the properties defined for its class, but holds specific values for properties that have been only partially defined by the parent class.

The automated classification of spatial sets may be supported by algorithms which operate on members of a class and graduate those members down the subclass tree if they meet the conditions specified by a subclass. This classification can be extremely efficient, in that its complexity is proportional to only the depth of the class tree, not to the number of classes defined.

Extensibility is addressed by the class concept as new classes may be defined by specification of additional properties on an existing class or set of classes. All of the properties of the old classes are maintained in the new class.

3.2.2. Geometric Abstractions

Geometric abstractions are representations of only a portion of the properties of a spatial set. A geometric abstraction may have its own representation and set of algorithms for the computation of properties represented by that abstraction. For example, the vertex-edge graph of a polyhedron can be simply represented in a variety of ways and is sufficient for wireframe display. Geometric abstractions are organized into a hierarchy in which an abstraction is a sub-abstraction of another in the hierarchy if it can be computed from the higher level abstraction. Figure 3-1 shows an abstraction hierarchy for some simple abstractions on an assembly of geometric objects.

The concept of geometric abstractions contributes to six of the eight requirements for geometric reasoning:

1. It provides a means to represent entities within the modelling domain prior to having complete knowledge of their configuration.
2. It provides a uniform representation for relationships between sets of objects.
3. It unifies queries on objects with representations of objects.
4. It provides a framework for the formulation of generation problems.
5. It provides a framework for the maintenance of consistency of geometric information under modification.
6. It provides transparent access to appropriate levels of representation.

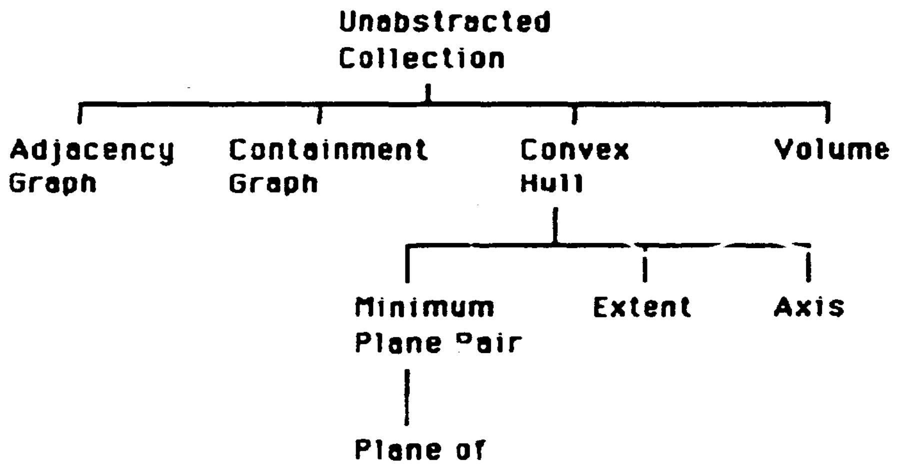


Figure 3-1: An abstraction hierachy for collections of polyhedra

The domain of representation discussed in Section 3.1 is a specification of the classes of spatial sets that should be modelled by a geometric reasoner. It does not give guidance for creating the representations which support modelling. Geometric abstractions provide a means to build multiple representations, and to link those representations together into a hierarchy. Multiple representations support the incomplete definition of geometric entities as each of the representations can be considered a partial model, which captures some properties but leaves others undefined. Linking these multiple representations together into a hierarchy provides a framework for inference upon partial knowledge. Figure 3-2 demonstrates how an abstraction hierarchy supports this inference. If all that is known about a particular geometric entity is its convex hull, then several different abstractions (minimum plane pair, plane-of, extent and axis) may still be computed. Computation of additional information demands that information be added to the reasoner in some fashion.

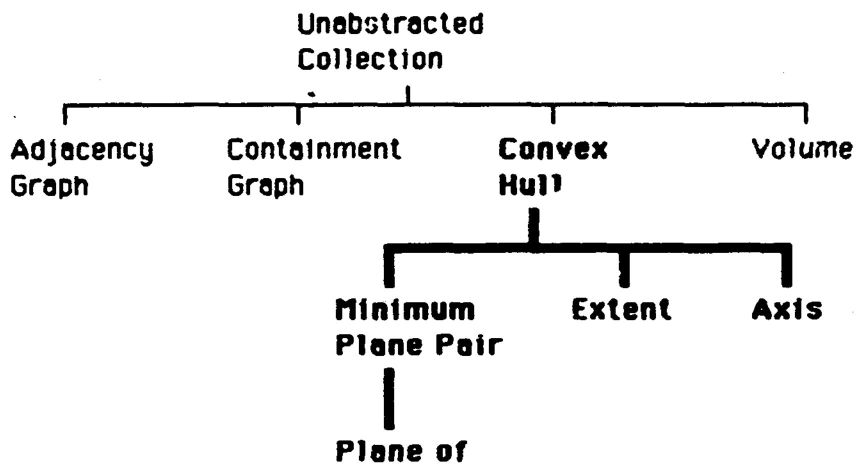


Figure 3-2: Use of an abstraction hierarchy for inference on incomplete information

Information about relationships between objects may be captured in a graph representation which is independent of the relationship modelled. The only difference between different relationships is the algorithms which are used to examine the graph data structure[3]. For example, a graph which captures information about adjacencies can be searched transitively to determine connected sub-components. A graph which captures proximity information cannot be so searched as proximity is not a transitive relation.

Geometric abstractions unify notions of queries and partial representations. A query on a geometric entity produces an abstraction or partial representation of that entity. This abstraction may be geometric object in its own right, or may be a graph representation or a vector or scalar quantity. In any case, for every query on a geometric entity there exists an entry into the abstraction hierarchy for that object.

An abstraction hierarchy can be interpreted in two directions. If *downward computation* is defined as the direction from complete models to less complete models, and *upward computation* is the reverse, then downward computations correspond to queries on models and upwards computations correspond to generation algorithms. For each query there can be formulated a problem in generation, which may or may not have a reasonable computational solution. In almost all cases the algorithm will be combinatoric in complexity. For example, generation of rectangular layouts from an orthogonal graph representation produces on the order of 10^6 possible layouts when only 10 rectangles are considered. This dual interpretation of abstraction hierarchies links together queries and generation problems.

The abstraction hierarchy provides a means for maintenance of consistency under modifications. A change to one of the representations in an abstraction hierarchy may impact information contained in other representations. A conservative strategy is to delete all information dependent upon the changed information and to recompute the deleted information as it is needed. The abstraction hierarchy can be used to incrementally improve upon that conservative strategy by including specific modification algorithms to selectively update nodes in the hierarchy. For example, under isometry transformations, the convex hull of an object will be changed by exactly the exactly the same transformation that applies to an unabstracted collection.

When a query is put to a model stored using an abstraction hierarchy, the hierarchy is searched for a representation from which that query can be answered. If the representation is not found, it is computed. If the required representation cannot be computed due to insufficient information being stored in the hierarchy, then that fact is returned as the response to the query. All of this can occur in a manner completely transparent to the application which is using the reasoner.

3.2.3. Features

Features[4] capture the nearly hierarchical decompositions of composite objects that humans impose upon these objects. This decomposition is essential to the process of understanding, manipulating and creating geometric forms, yet it has little to do with geometry *per se*. It is rather a recursive naming mechanism, that supports the aggregation of geometric entities into complex composites. This naming mechanism should be as flexible and general as possible, to support many different approaches to decomposition. Virtually the only restriction that is reasonable to place on this mechanism is that it not be self-referential, that is, that no

[3]These could be called inference algorithms without doing violence to common usage.

[4]The origin of the term "features" arises from its general usage in computer-aided design as a means of specification of portions of a model that are of interest. Recently specific use of the term has occurred in the context of mechanical engineering design and machining [Libardi 86] [Nielson 86]. Its meaning here does not contradict these usages.

feature can contain itself as one of its sub-features. The concept of features addresses four of the global system requirements:

1. It adds the notion of an assembly of objects into the domain of representation.
2. It provides a framework for the computation of relationships between objects.
3. It provides a framework for the computation of queries on objects.
4. It is a basic interface mechanism, allowing uniform access to a system.

The fundamental domain of representation for a geometric reasoning system is subsets of R^3. Features provide a mechanism to structure combinations of these sets in any way required by an application. The single limitation on features, that of non-self-referentiality, imposes very little on the scheme. Features may be used to represent strictly hierarchical structures such as the instance tree of modelling systems [Woodbury 86] and high performance graphics packages [Brown 85] as well as structures which approach a general data access scheme.

By providing a means to group together geometric entities of interest, features provide an mechanism for passing arguments to functions which compute relationships between entities. For example, polyhedra which represent a robot arm may be grouped together under a single feature and submitted to an algorithm which computes the allowable movements of the assembly from the constraints imposed by the joint geometries. If each of the links of the robot arm is represented by a number of polyhedra, these may be grouped into a sub-feature and considered as one unit for purposes of the allowable movement algorithm. The information computed by the algorithm is a property of the feature as a whole, not of the individual spatial sets which compose the feature.

In a similar fashion, features may be used for the computation of queries on objects. For example, the convex hull of a feature which is composed of a number of spatial sets will in general be different from any of the convex hulls of the component sets. Queries on features compute properties which pertain to the entire assembly, not necessarily to its constituent parts.

Features are a basic access mechanism to a geometric reasoner. They provide a means to reference geometric objects in terms of the semantics of the application at hand, rather than in terms of data structures which represent geometric sets. A user need know little more than the basic structure of features and the protocol of operations which apply to them to effectively build and access complex geometric models.

3.3. The Concepts Together: An Architecture

The three concepts, classes of spatial sets, abstractions and features mutually interact. Out of these interactions can be developed an overall system architecture.

The two concepts of classes of spatial sets and features are related by a single rule: a feature may be declared as a member of a particular class of spatial set. This rule has two implications:

1. It preserves the uniformity of features as the main access scheme to the geometric reasoning system. The specific geometric models which may be part of the definition of a feature are hidden from the user of the reasoner. All that is visible is the feature hierarchy.
2. It further restricts the allowable contents of a features. A feature which is also a spatial set may contain sub-features, but these sub-features are constrained to be elements from the boundary of the spatial set[5]. For example, a window regulator[6] for an automobile may be described as having several parts, one of which is the backplate. If the backplate is declared to belong to the class of polyhedra, then all sub-features of the backplate must be a part of the boundary of the polyhedron which describes the backplate.

The interaction between features and abstractions can be expressed by a single rule: Features are the fundamental units of abstraction. This rule has two implications:

1. Only features may be abstracted. An abstraction hierarchy grows whenever queries are performed upon an entity. Since features are the only entities upon which queries are defined, they are also the only entities which support abstractions.
2. Features store abstraction information at the appropriate level. An abstraction generated on a particular feature is stored in the abstraction hierarchy associate with that feature. Any abstractions generated in the process of computation are automatically stored at the appropriate level in the feature hierarchy. For example, consider Figure 3-3. Representations which abstract the window regulator as a whole are stored with the feature labelled *window regulator*. Representations which abstract only the backplate of the window regulator are stored with the feature labelled *backplate*.

Classes of spatial sets and abstractions are simply related. The rule in this case is: every abstraction is represented by either a spatial set, a graph, a vector or a scalar. This rule has a single implication:

1. It makes all of the power of the underlying operations on spatial sets available to the abstraction mechanisms in the geometric reasoner. For example, in computing the bounding box of an assembly of objects, the reasoner could proceed

[5]This restriction on allowable contents creates a close approximation to most implementations of part-whole hierarchies.

[6]A window regulator is the device, inside the door of an automobile, which moves the glass of the window in response to turning the window crank (or to pushing the activator switch in electrical models.)

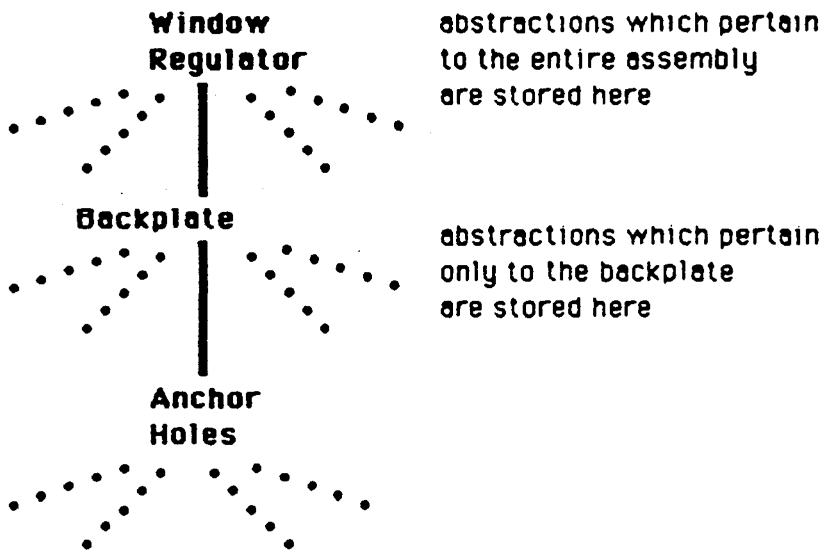

Window Regulator
abstractions which pertain to the entire assembly are stored here
Backplate
abstractions which pertain only to the backplate are stored here
Anchor Holes

Figure 3-3: Storing abstractions at the appropriate level

from either the complete assembly or could first compute the convex hull of the assembly. In either case, by virtue of both representations being defined in terms of polyhedra, the same algorithm would apply.

When the three concepts are considered simultaneously an overall system architecture emerges. Figure 3-4 diagrams this organization In this figure, the boxes refer to each of the concepts and the lines are inheritance relations between concepts. The top level box, labelled *object* denotes the unity of all of the concepts as data objects in an implementation.

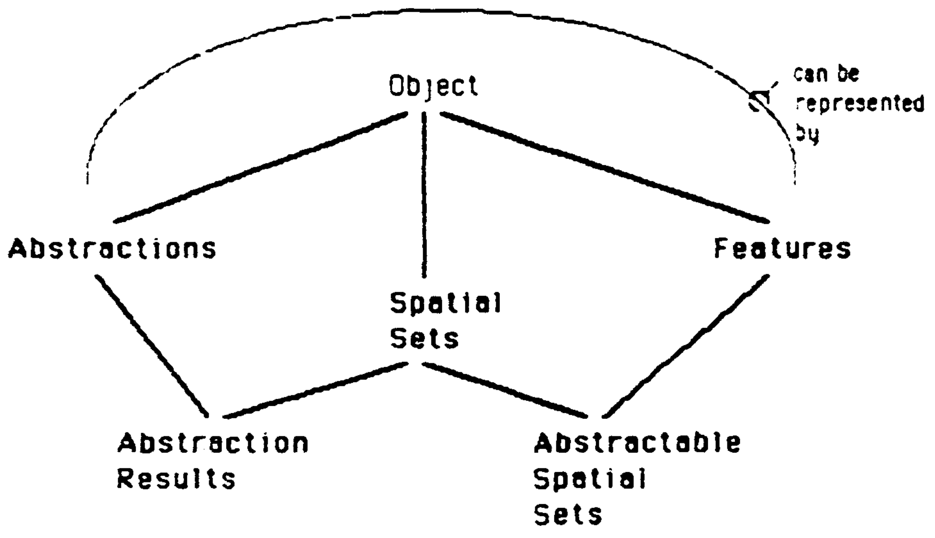


Figure 3-4: The overall system architecture

3.4. Implementation

This implementation of this architecture is affected by the choice of programming environment. We chose object oriented programming for this exploration as its concepts of polymorphism and inheritance map naturally onto the hierarchical structuring of geometric information that is proposed here. The particular programming environment used, CommonLoops, has two special properties, multiple inheritance and multi-methods, that affect the overall system architecture:

1. Multiple inheritance permits facile merging of the system concepts. The class *abstractable spatial sets* is formed by inheriting from the classes *features* and *spatial sets* and by defining a small number of methods specific to the new class.
2. Multi-methods permit the specialization of methods on the basis of class membership of more than one of the method

arguments. This is in distinction to so-called *classic message passing* in which only the class of the first argument is significant. Multi-methods simplify the creation of uniform protocols over operations which involve more than one object.

We have implemented a simple two-dimensional geometric reasoning system based upon the three concepts, classes of spatial sets, geometric abstractions and features. With the lessons learned from this exercise we are currently implementing a much larger scale demonstration in three dimensions, using an existing solids modelling system as the basis for geometric computation.

4. Conclusions and Recommendations

The *architecture for geometric reasoning* contained in section 3 is offered as a promising approach to representation and manipulation of geometric information for knowledge based systems. Its implementation is underway, and it will be evaluated for problems of robot manipulation. It will also be applied to problems of mechanical parts description for manufacturing or assembly.

Additional approaches to geometric reasoning, offered by colleagues of the authors, are partly outlined here. One approach is an investigation of natural language function in conveying geometric information; this work has its origins both in cognitive psychology and in architectural research into spatial cognition [Goel 86]. Another approach, convincingly demonstrated in two-dimensional space, is the representation and abstraction of loosely packed arrangements of rectangles in automated layout research [Flemming 86]. A third apprach is reflected in language design for geometric representation and manipulation [Wing 85]. A fourth approach is found in studies of automated assembly and disassembly of objects. A fifth approach is exemplified by the design of domain models for robot manipulation within complex facilities such as nuclear power plants, together with the design of blackboards for robot control in comparably rich environments [Keirouz 86]

5. Acknowledgements

The main research approach presented herein (section 3) is supported by the Electric Power Research Institute under contract RP2515-1, "Robotics in Construction and Maintenance." The authors are pleased to acknowledge the support, interest, and guidance of EPRI and Dr. Floyd Gelhaus. We are especially grateful for permission to discuss this approach with the research community prior to its conclusion and delivery to EPRI later this year, and point out that the results have not yet been reviewed or checked for accuracy.

References

[Akiner 85] V.T. Akiner. Topology - 1: A Reasoning System for Objects and Space Modelling Via Knowledge Engineering. In *Design Engineering Division Conference on Mechanical Vibration and Noise*. American Society of Mechanical Engineers, Cincinnati, Ohio, September, 1985.

[Borning 79] Alan Borning. *ThingLab- A Constraint Oriented Simulation Laboratory*. Technical Report, Palo Alto Research Center, July, 1979.

[Brooks 81] Rodney Allen Brooks. *Symbolic Reasoning Among 3-D Models and 2-D Images*. PhD thesis, Stanford University, June, 1981.

[Brown 85] Maxine Brown. *Understanding PHIGS*. Template, The Software Division of Megatek Corp., 9645 Scranton Rd., San Diego, Ca., 92121, 1985.

[Davis 81] Ernest Davis. *Organizing Spatial Knowledge*. Technical Report 193, Yale University, Computer Science Department, January, 1981.

[Davis 86] Ernest Davis. *Research Notes in Artificial Intelligence: Representing and Acquiring Geographic Knowledge*. Pitman Publishing Ltd., 128 Long Acre, London, England, WC2E 9AN, 1986.

[Eastman 79] Charles M. Eastman and Kevin Weiler. *Geometric Modeling Using the Euler Operators*. Technical Report 78, Institute of Physical Planning, Carnegie-Mellon Univ., February, 1979.

[Fahlman 74] Scott Elliot Fahlman. A Planning System for Robot Construction Tasks. *Artificial Intelligence* 5(1):1-49, 1974.

[Fikes 71] R.E. Fikes and N.J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2:189-208, 1971.

[Flemming 86] Ulrich Flemming, Michael D. Rychener, Robert F Coyne, Timothy J. Glavin. *A Generative Expert System for the Design of Building Layouts: Version 1*. Technical Report, Center for Arts and Technology, Carnegie-Mellon University, June, 1986.

[Goel 86] Vinod Goel. Assigning Locative Prepositions to the Spatial Relations Implicit in 3-D Geometrical Models of Objects and Scenes. Master's thesis, Faculty of Environmental Studies, York University, Toronto, Ontario, Canada, July, 1986.

[Gosling 83] James Gosling. *Algebraic Constraints*. PhD thesis, Computer Science Department, Carnegie-Mellon University, May, 1983.

[Keirouz 86] Walid T. Keirouz, Daniel R. Rehak, Irving J. Oppenheim. Object-Oriented Domain Modeling of Constructed Facilities for Robotic Applications. In *Applications of Artificial Intelligence in Engineering Problems, 1st International Conference*, pages 141-150. Springer-Verlag, Southampton University, U.K., April, 1986.

[Klahr 82] Philip Klahr, David McArthur and Sanjai Narain. Swirl: An Object-Oriented Air Battle Simulator. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 331-334. American Association for Artificial Intelligence, 1982.

[Kuipers 77] Benjamin Jack Kuipers. *Representing Knowledge of Large-Scale Space*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, July, 1977.

[Libardi 86] Eugene C. Libardi, John R, Dixon, Melvin K. Simmons. Designing with Features: Design and Analysis of Extrusions as an Example. In *Spring National Design Engineering Conference*. American Society of Mechanical Engineers, Chic Illinois, March 24-27, 1986.

[Light 82] R. Light, D. Gossard. Modification of geometric models through variational geometry. *CAD - Computer Aided Design* 14(4):209-214, July, 1982.

[Lynch 60] Kevin Lynch. *The Image of the City*. MIT Press, Cambridge, Mass., 1960.

[Nielson 86] Eric H. Neilson, John R. Dixon, Melvin K. Simmons. *How Shall We Represent the Geometry of Designed Objects?*. Technical Report, Department of Mechanical Engineering, University of Massachusetts, 1986.

[Preparata 85] Franco P. Preparata and Michael Ian Shamos. *Texts and Monographs in Computer Science: Computational Geometry - An Introduction*. Springer-Verlag, New York, N.Y., 1985.

[Requicha 80] Aristides A.G. Requicha. Representation for Rigid Solids: Theory, Methods and Systems. *Computing Surveys* 12(4):437-464, December, 1980.

[Simmons 82] Reid F. Simmons. Spatial and Temporal Reasoning in Geologic Map Interpretation. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 152-154. American Association for Artificial Intelligence, Carnegie-Mellon University, Pittsburgh, Pa., August, 1982.

[Steele 80] G.L. Steele Jr. *The Definition and Implementation of a Computer Programming Language Based on Constraints*. PhD thesis, Massachusetts Institute of Technology, August, 1980.

[Sussman 80] G. Sussman and G. Steele. CONSTRAINTS - a language for expressing almost hierarchical descriptions. *Artificial Intelligence* 14(1):1-40, August, 1980.

[Sutherland 63] I.E. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System.* Technical Report 296, MIT Lincoln Lab., 1963.

[Wing 85] Jeannette M. Wing, Farhad Arbab. *Geometric Reasoning: A New Paradigm for Processing Geometric Information.* Technical Report cmu-cs-85-144, Computer Science Department, Carnegie-Mellon University, 1985.

[Woodbury 86] Robert Woodbury. VEGA: A Geometric Modelling System. In *IBM Advanced Educational Projects Conference.* IBM, San Diego, Ca., April, 1986.

536-61
78
[illegible]

A Qualitative Approach for Recovering Relative Depths in Dynamic Scenes

S.M. Haynes and R. Jain
University of Michigan
Ann Arbor, MI 48109

MX [illegible]

Abstract

This approach to dynamic scene analysis is a qualitative one. It computes relative depths using very general rules. The depths calculated are qualitative in the sense that the only information obtained is which object is in front of which others. The motion is qualitative in the sense that the only required motion data is whether objects are moving toward or away from the camera. Reasoning, which takes into account the temporal character of the data and the scene, is qualitative. This approach to dynamic scene analysis can tolerate imprecise data because in dynamic scenes the data are redundant.

Keywords: qualitative vision, dynamic scene analysis, relative surface depths.

1 Motivations for qualitative vision

For many reasons computer vision has proven a difficult task, far more difficult than was originally suspected. The complexity of the real world is sampled spatially and temporally and projected onto a time ordered sequence of frames. The description of objects, relationships and events among those objects is a signal to symbol transformation which requires the top-down use of knowledge (i.e. an interface to a memory). Because the projection process "loses" a dimension, interpretation must be able to tolerate ambiguous data. Noise compounds the ambiguity of the frame sequence. Roberts' work in this field, [22], attempted to compensate for noise using several heuristics for line detection and a top-down model-fitting approach. An approach to solving this problem of model-fitting under noise is given by Brooks [8]. In this work, he has the problem of unknown transforms between model and image; to solve this problem he uses a constraint manipulation package to limit the matches between image and model and hypothesize other matches consistent with the constraints. The transforms are initially under-specified, then increasingly constrained.

Two approaches have proven to be particularly restrictive. The first is the focus on single frame analysis. Early researchers felt that it was necessary to first process one frame, and only then examine the subsequent frame. This is a sterile approach because it avoids all temporally changing scenes (i.e., things like pictures and maps), including most scenes of interest. The second approach which has disappointed is the careful computation of numerical features in a data driven manner. Examples are 3-D positions of feature points obtained via structure-from-motion or of surface normals from optical flow, shape from shading, or texture, motion parameters: v_x, v_y, v_z, and optimization of objective functions (for citations describing these various approaches see: [1], [20], [24], [27], [32]). Most rely on an inverse transformation, from two dimensions to three, and that, combined with the noise inherent in sensor and the sampling and digitizing processes, means that algorithms providing quantitative solution will be inherently very sensitive to noise.

Along with the growing interest in dynamic scenes, the realization that improving accuracy in a highly restricted set of features does not particularly help the interpretation process some vision researchers [25] are being drawn to the qualitative approaches being used for common sense reasoning, naive physics and circuit analysis [6], [12]. The reasons for this are that qualitative approaches show that it is possible to obtain useful results when solving problems with uncertain, approximate or only signs of parameters. The representations of the problem domains are an attempt to capture the fundamental nature of the system, while avoiding the complexity of dynamic equations. Currently, much of the work done in qualitative physics involves determining appropriate states and symbols and an understanding of the nature of state change. Another important component is a simulation process, which allows one to get a grasp on causality. The qualitative approaches thus far attempted in AI have generally included things like signs of derivatives [9] [16] or transitions [10]. Computer vision is also a testbed where considerable intelligence is required. Further, the data in computer vision are always noisy, frequently redundant, and often misleading.

Another approach for handling the noise for bottom-up processes, having support from biological vision systems, is to use a variety of window sizes or a collection of band-passed images. Larger sized operators average over a greater area, and thus for reasonably well behaved noise, the noise has less effect on the result. Unfortunately, the larger the window, the more likely it is computing a single result over two or more different pixel source populations. Some researchers have proposed using a set of different window sizes, large ones for large scale and perhaps low contrast changes and smaller ones for more local changes [17], the scale space is a continuous version of this [32]. It is not clear how to combine information among these many channels, partly because the channels are being used for two different things: detecting (or measuring) at different scales, and using larger channels to reduce noise effects at the lower channels.

Event detection in its most general sense locates the interface between qualitatively different sources of pixel population. The idea of event detection is not to smooth over noise, and thus over different pixel populations by using a magically chosen window size, but instead to detect where the pixel population changes and avoid any integration across that boundary. Using this paradigm are [5], [11], and [13]; also [24], using a finite element approach, can fracture the surface at appropriate places.

This noise issue has especially frustrated dynamic scene researchers because it has been shown mathematically that all 3-D information (to a scale factor) is available in the optical flow field. Attempts to get the information have been fruitless because even the best obtainable flow fields are too badly corrupted. Thompson, et al., [26] take the approach that if precise values are not computable, then compute the qualitative information: which segment is the occluder and which is the occluded. Jain [14] has also obtained this information for different sorts of scenes. Both use only a crude, though computable, approximation to optical flow. The first uses an approximation to the flow field called a disparity field requiring good feature detection and correspondence algorithms. The second uses a more qualitative approximation, computing the time history of pixel changes. But in any case it is clear that useful results are possible, even from the noisy data available, using and computing qualitative attributes rather than precise, brittle ones.

2 Using models in computer vision

There are a number of reasons for building or using a model in computer vision.

1. A model provides a simplified representation. For example, the motion of a point may be specified with initial state and state transition equations, thus it is not necessary to store, for each point in time, the position of a point.

2. At some, perhaps low, level, the data can be said to be understood when there is a model which fits them. For example, at a very low level when the data fit, e.g., a straight line, the line is a model for the data, and as a line is understood, so is the underlying phenomenon giving rise to the data. This is like number 1 above. At a higher level, for the model which is a line $y = mx + b$, one can say e.g., m is velocity and b is starting position. At the highest level, if the model is a frame with slots, then the general practice is to use *a priori* default values for slots which are not filled from the data. Thus, a limited data set which cause a particular frame to be instantiated, triggers a top-down use of the model in which more is understood than can be derived in an immediate sense from the data.

3. If the data can be said to fit some model, then, because in general, each data point need not be kept around, the interpretation processes can be made more tolerant to noise. In the representation, data can be allowed to lie within a range of values, as signal + noise, where the noise has some understood or assumed statistical properties.

4. Missing or ambiguous data can be handled by assuming the data exist according to the model, but are not measurable for some reason. This property of models is heavily relied upon by computer vision researchers because occlusion is rampant.

5. When a model is available, it can play the role of a kind of short term memory where the integration of data, especially of errorful data can be incorporated. This property was used in [2] in their very early work on dynamic scene analysis, in a primitive fashion.

There are a number of representation techniques for models. Perhaps the most popular is to use frames having slots [18]. A labelled frame will have a number of labelled slots which can be filled with numbers, attributes (i.e. symbols), variables or links to other frames. The procedures which manipulate these slots, for example how the slot with label velocity relates to dynamical equations which can predict future positions are also part of the modelling process. The dynamical equations are modelling the motion. These sorts of calculations are not as easily represented in frames, since this sort of knowledge is more naturally given as procedures. Interpretation involves instantiating the model from among a set of competitors that best match the data. An event indicates where models, or perhaps only parameters of the models, change. Confidence is a number expressing some kind of probability that the model is correct or that the data measurement is correct, when that information is available.

3 Purpose and use of chronologies

The very earliest works in dynamic scene analysis required the representation of velocities and positions over time. It is not enough to give a simple initial state, because most motions are not describable with simple dynamic equations (consider hierarchical or non-rigid motions), and because they do not incorporate changes in motion descriptions, and because other interesting temporal characteristics are not included in a natural fashion. The early works did not keep chronologies. Instead, in [2] for example, they kept a model of the scene for one time instant only, and used that to predict the model for the next frame. This implicitly incorporates the initial state description. Robot planning frequently requires the description of several actions over an *extended period of time*. These are generally inspired from the approach of describing the state of the world and robot at each time instant (for an advanced use of this see [7]).

Tsotsos [28] made extensive use of chronologies which were essentially time-ordered positions of points to choose among hypotheses for high level motion descriptions (e.g. expand, sway). His system chose the best hypothesis by examining the time-course of confidences of the possible schemas. This example exemplifies a major use of chronologies: to disambiguate local motions into more global, longer term motion descriptions. Other uses are to be able to predict future positions and circumstances, to identify interesting motions, and to localize events in the motions. In addition to obtaining long term motion descriptions, a history of events or of motions, or of relationships between object parts, is useful on its own, or for deriving other, even higher level descriptions. That is, one may be able to describe *oscillatory motion* as such, rather than as a repeating sequence of position and velocity.

Chronologies are not really models, however, because they neither provide a simplified representation for the data, nor do they provide understanding. They provide a description. Chronologies also provide a representation in which noise tolerance, occlusion and integration of data in a temporal fashion can be supported, especially under the control of temporally dependent operation.

4 Local Temporal Inferencing

4.1 Introduction

When values can be tied to a number line, they are quantitative. Permitting bounds on values, that is, restricting them to an interval on the number line, one can still do numerical operations on them [3]. Naive physics researchers use the qualitative (symbolic) descriptors: increasing or decreasing. These values are obtained by considering the sign of derivatives, also an interval. If the sign is positive, the variable values are increasing. We also use the intervals $(-\infty, 0^-)$, $(0^-, 0^+)$, $(0^+, +\infty)$ as qualitative values. Another sort of qualitative value is a *relative* statement. For example x is faster than y, or a is closer than b. This sort of relation constrains the value of x with respect to y (and vice versa), but does not tie the value to the number line. Hasse diagrams are a graphical representation describing such relative statements when the relation provides a partial ordering. The qualitative example involving partial order is different from the notions of state and of symbol. It is a comparison. The ordering qualitative example is also more robust to noise – though not because the error tolerance is greater.

There are other qualitative relations between attributes which are interval in their nature, i.e., which have begin and end points. Vilain [30] and Allen [4] have developed an interval-based temporal reasoning and labelling system. Their works, and those of others, are applicable to domains like story understanding, where there tend to be fixed endpoints to the temporal intervals. Vere [29] has developed a system which will generate parallel plans for achieving goals within time constraints.

4.2 Constraints on domain and general description

For the work reported here we wish to describe the relations among objects, without recourse to object or scene models, over extended frame sequences. In particular, we wish the program to provide the relative depths among surfaces, when computable, and histories of surface to surface relations.

The data for this work are time-ordered lists of occluder-occluded pairs and directions of motion in depth of surfaces (toward or away from camera). [14] and [26] have shown methods whereby occluder-occluded relations may be obtained. No further data are required.

Suppose that all objects are stationary (i.e., as in static scene analysis). The data provided are triples of the sort: A occludes B. In terms of depth, z, this means, for surfaces with changes in depth that are negligible with respect to inter-surface depths, that $z(A) < z(B)$. We use the notation $A < B$. Occlusion data thus places a partial ordering on the depths of surfaces; and for static scenes, transitivity suffices to provide all computable depth constraints between surface patches. This partial ordering does not change over time. Thus, for example, given the data set: $A < B$; $B < C$; $B < D$, transitivity gives us that: $A < C$, $A < D$, and that there is *no* ordering in depth between C and D. Inconsistencies in data and in deductions are trivially detected, though not trivially resolved.

When objects are permitted to move in a plane parallel to the image plane the rule for combining depth constraints is again transitivity. If there is no change in depths of objects then the relative depths will not change.

If objects are allowed to move in depth, then the depth order obtained by a local occlusion analysis can no longer be used as a sorting criterion. Transitivity does not hold into the future when depths change over time. An approach to this problem is to project depth constraints between two objects into the future, and then use those derived constraints in transitive relations at the time of interest.

4.3 Velocity rules

There are four physically derived rules which give the projection into the future for the depth orderings. Motion in depth is v. If $Sign(v_{t,t+\Delta t}) < 0$ then motion is *toward* the observer on the temporal interval $(t, t+\Delta t)$. For $Sign(v_{t,t+\Delta t}) > 0$ motion is *away* from the observer on the same interval. $Sign(v) = 0$ means there is no significant motion in depth. The four rules are:

- *rule1* $A_t < B_t$ and $v_{t,t+\Delta t}(A) = 0$ and $v_{t,t+\Delta t}(B) = 0 \Longrightarrow$ $A_{t+\Delta t} < B_{t+\Delta t}$
- *rule2* : $A_t < B_t$ and $v_{t,t+\Delta t}(A) < 0$ and $v_{t,t+\Delta t}(B) = 0 \Longrightarrow$ $A_{t+\Delta t} < B_{t+\Delta t}$
- *rule3* $A_t < B_t$ and $v_{t,t+\Delta t}(A) = 0$ and $v_{t,t+\Delta t}(B) > 0 \Longrightarrow$ $A_{t+\Delta t} < B_{t+\Delta t}$
- *rule4* $A_t < B_t$ and $v_{t,t+\Delta t}(A) < 0$ and $v_{t,t+\Delta t}(B) > 0 \Longrightarrow$ $A_{t+\Delta t} < B_{t+\Delta t}$

These rules are all expressed in
$A_t < B_t$ and $v_{t,t+\Delta t}(A) <= 0$ and $v_{t,t+\Delta t}(B) >= 0 \Longrightarrow$
$A_{t,t+\Delta t} < B_{t,t+\Delta t}$
They are referred to in later text as *velocity rules*. For other motions of A and B, that is, for $A_t < B_t$ where $v(A) > 0$ or $v(B) < 0$ there is no relative depth information between A and B at time $t + \Delta t$. Inconsistencies are detectable in this scheme when conflicting relations are derived (cycles are detected).

4.4 Temporally local inferencing on qualitative relations

The data are occluder-occludee pairs, and the direction of motion in depth (toward or away from observer). From these data, one can derive infront (and behind) relations for the present time using transitivity of depth ordering, and for the future using the velocity rules. The derivations for the future hold under the assumption that there is no change in the direction of depth velocity (magnitude is unimportant). However, we prefer not to have to deal with such an unstructured, open-ended future. Since the data are arriving at this system at each time step, we make the inferences into the future for one time step only. Thus, at time t_0, we have a set of depth relations $\Psi(t_0)$. From these relations we use the velocity rules to derive, for the next time t_1, a set of relations $\Psi^*(t_1)$. This set of relations, ignoring time and labels, will be a subset of the relations at t_0. Recall velocity rules are of the sort $A < B$ at t_0 plus some constraints on velocity of A and $B \Rightarrow A < B$ at t_1. There is no point in applying the transitivity rule at this point, it will not add any new arcs. Incorporating the data at time t_1 will add some new relations. This will give rise to the set of relations $\Psi^{**}(t_1)$. Now one applies transitivity at this point to obtain the set $\Psi(t_1)$, and the set of relations is ready to project into the future one time step again. See figure 1 for a layout of the order of operations on the relations.

Thus, this system incrementally incorporates the data as it becomes available. It makes no attempt to predict further into the future than to the next time step. A time step is defined as when the next datum is available. It has no memory beyond one time step. It is *local*, temporally speaking. More global temporal knowledge is kept elsewhere in the system – specifically, in the object histories.

In the system there may be several relations between a given pair of segments. That is, each relation has two segments and a label. For the segment pair, A and B, we may have, e.g.

order	label
$A < B$	*rule1*
$A < B$	*data*

As long as the data are consistent and correct, these inferences will iteratively build a consistent partial order on the segments which is as complete as is possible for these rules at the current time. What happens when a datum is incorrect? In that case we will have an *inconsistent* set of relations. This inconsistency is signalled by a *cycle* in the graph. For example, suppose we have the relation: $A < B$: *rule1* for the graph in $\Psi^*(t)$. We then read the datum $B < A$. The graph then contains the cycle $A \rightleftharpoons B$.

Because we have labels on relations, we know what gave rise to the inconsistency. For the above example we know that, because there is a cycle, the datum $B < A$ is wrong, or the *rule1* applications was wrong or both. Conceivably, we could trace the cause of the inconsistency back further into the past. For the above example, if the *rule1* application at time $t-1$ was wrong then either the $v(A)$ was wrong, $v(B)$ was wrong, the relation $A < B$ at time $t-1$ was wrong, or any subset of these three was wrong. For this one inconsistency involving only two objects and two relations we have already fingered as possible culprits four attributes or relations going back only one time step. Indeed, if we kept only a slightly more complete audit trail the relation $A < B$ at time $t-1$ could be further tracked down. This gives rise to even more possibilities of the source of inconsistency even more remotely in time.

We are not doing this for a number of reasons. The most important of these is than in a dynamic scene understanding system, one does not have the resources to spend a lot of time and energy resolving past conflict: data are continually arriving, and it is better to have the current (and future) interpretations be correct than those of the past. Secondly, many culprits are fingered for each inconsistency. This is a lot of overhead and cannot be resolved or reduced from the information currently available. For a third reason, resolution is possible only in the future when more data is available – there is no resolution possible in the past (where the inconsistency arose). Fourth is that we rely on the fact that there are a lot of data. Even though some are wrong, most will be right; we do not want to devote much effort to inconsistency resolution because we may expect that future data will set things right. There is one important consequence of this for the implementation: we do not keep an extensive audit trail. We label each arc with only the rule that most recently derived it.

So we do not make any attempt to undo any bad effects from possible bad data in the past. The present data are a different matter. We do, however, want the correct data to eventually outweigh any incorrect inferences. We have a number of options on how to go about doing this. Essentially there are two questions:

1. how to propagate, to the next time step a relation which has a contradiction

2. how to incorporate a contradicting pair of relations into object histories

We deal with this difficulty by taking the position that one must trust the current data at the current time. Any inferences from that data, especially into the future may be suspect, but the data themselves are assumed to be correct for the time now. We could take the position that any relation which contradicts *data* will be deleted immediately, and is not allowed to propagate into the future. But we still have the problem of contradictory relations which do not have *data* on either of the labels. For example, see figure 2 in which we show the derivation of an inconsistency.

We are drawn to the use of a certainty factor for each relation in order to accomodate the possibility of occasionally invalid data. The certainty of *data* relations will be highest. As inferences are derived, the certainty factor of those relations will decrease. There are a number of technical difficulties involved in dealing with certainty factors and getting them to be rigorously correct. We avoid these by relying on the fact of large amounts of mostly right data. To rigorously derive a calculus of certainty factors, it is necessary to have a sufficiently deep understanding of the nature of the domain, especially of the nature of the data, noise, and sometimes even *a priori* probability values of the data, as well as assumptions of independence, or and knowledge of correlations. In the spirit of qualitative processing, we wish to avoid making such restrictive assumptions until necessary. In this system we are attempting a qualitative approach in which we know there is noise, though we don't know its precise precise properties. Because of the fortunate choice of using dynamic scenes as data, however, we can use the fact that the data will be mostly redundant. Thus even though rigorous derivations for certainties have been done [23] and may be applicable we are not currently investigating that direction. We just want certainty factors to decrease with time and with transitive "distance" from *data*. There are a number of choices on how to combine certainty factors when making inferences. We are currently experimenting with this. The problem of which of two contradictory relations to propagate we deal with heuristically: relations which are contradictory are not permitted to activate the transitive rule. All other relations may activate both transitive and velocity rules. We put this restriction on transitive-derived contradictory rules for computational reasons only. Many relations are derived using transitivity, and when one of the links is suspect, all links derived from it are suspect. Inferences whose certainties are decreasing to zero are deleted after a fixed number of time steps.

5 Chronologies

5.1 Representational issue—indexing

In building a chronology of depth-ordered relations among surface patches for use, either as a descriptive device, or as an intermediate data structure for further processing, there are two ways of indexing. The first is to organise the relations temporally. In dynamic scene analysis, unlike story understanding, the data are arriving in a time-ordered fashion, e.g. in frame i, the is some set $\Psi(i)$ of relations, at frame $i+1$ some other set $\Psi(i+1)$. The chronology of relations has the same appearance as the data with the addition of derived relations. In this case it is easy to see what is happening at a particular time instant, because time is the index into list of relations, e.g.

time	*relations*
1	$(< AB)\,(< AD)$
2	$(< AB)\,(< AD)$
3	$(< AB)\,(< AD)$
4	$(< AB)\,(< BC)\,(< AC)$
...	...

We see in one indexing step which relations exist at t=3. Given the way the velocity rules are formulated, it is also easier to make predictions into the next time step. For example, if the motions in depth of A, B, C are negligible, then at time 4 we can make the prediction that at time 5, the following relations will hold: $(< AB), (< BC), (< AC)$.

The second indexing method is to organise by relation. That is, one surface is the primary index, the second surface the secondary index. For example:

1°	2°	*temporal intervals*
A	B	(1, now)
	C	(4, now)
	D	(1, 3)
B	C	(4, now)
...	...	...

If relations persist, or are repetitious, then it saves on space to index by surfaces. This representation trades off relationship storage for temporal storage space advantageously when relations are long-lived or recur frequently. To determine at a particular time instant which relations are active requires inspection of a lot of data. The time course of relations is easier to access. Event marking makes deriving an interval-based description easy. And this method of indexing is better for dealing with noise removal, occlusion and integration over time.

5.2 History and world model of depths

Despite the fact that dynamic vision has a lot of data available, thanks to its *rampant redundancy* [31], it is both more efficient as well as satisfying to keep histories of relations which are indexed by surface. The fact remains, however, that in order to make derivations (or predictions) using the velocity rules and to be able to make a computationally fast statement about the relative depths at time now, we keep the current list of relations, though redundant with histories. That is, for now we have a "temporally indexed" set of relations. For now as well as all the past we have object-indexed relations. This means that if relative depths for any past time is desired, though the information is calculable (indeed, was calculated, then discarded), from the histories, it is not immediate. Rather, the system will have to step through the histories, essentially re-creating the world for the desired time. There are certain similarities with *envisioning* [9]. For now we can get an ordered relative depth map by doing a straight-forward topological sort [15].

6 Experiments

The local temporal inferencing system was implemented as described in a previous section. We made a few adjustments, for pruning purpose, to the inferencing procedure as follows.

- Relations which were contradictory, e.g. $A < B$ and $B < A$, were not permitted to participate in any transitivity inferences. This is because the only result from applying transitivity on contradictory relations is many more contradictory relations. Contradictory relations are allowed to propagate into the future with the velocity rules.

- Only relations derived from transitivity which had a larger or equal confidence factors than other relations already present (between the same nodes) were posted. For example, suppose we have the relation $A < B$ with confidence .70 present, then we derive $A < B$ with confidence .35 from transitivity. That new relation is ignored.

- Data relations are taken with confidence 1.0 (indicated as $cf = 1.0$), and other relations already present between the same nodes as the data relation were deleted. That is, if we have $A < B$ because of a *rule1* application with $cf = .9$, then we read the datum $A < B$, the previous *rule1* edge is deleted, only the data edge remains.

We did not perform theoretically rigorous derivations of confidence factors. This is because the exact rules for combining confidences is not important in our scheme. Confidences propagated with velocity rules have a "time-decay" built in. That is, if $A < B(t_0); cf : x$, with appropriate velocities, then we derive $A < B(t_1); cf : x'$, where $x' < x$. Confidence factors are combined for transitivity rules by taking the *min* of all the relations involved, then applying a decay factor (called a "spatial decay") to the resulting number.

We present the results of an experiment in the series of figures 3 - 7. The input is echoed in the "input-list" window. The "active-relations" window contains a list of edges with the label (reason) and confidence. In figure 3 the data is only the three v motions of the objects A, B, and C. In figure 7, the data are $A < C$ and $B < A$, the transitive closure is performed which derives $B < C$. Data relations have confidences of 1.0. Transitive relations, i.e., tc are decayed. In figure 7, the same three relations remain, because the velocity rule rule1 infer them. The confidences have all decreased. Notice the confidence for the relation $B < C$, the relation originally derived through transitivity, is less than that of the other two relations originally derived from data. In figure 7, we have in the data $D < A$; note the new relation has confidence 1.0. In addition, we have derived through transitivity the new relation $D < C$. In this figure notice that $B < C$ actually has two edges. One is a velocity rule edge with confidence 0.6, derived from previous $B < C$ edge. The other is a transitivity edge derived from the edges you see present, $B < A; cf : 0.8$ and $A < C; cf : 0.8$. This did not happen in figure 7 at time 8 because of the nature of the spatial and temporal decay factors. For this experiment, the spatial decay is larger than the temporal decay. Figure 7 has the contradictory relation $A < D$ just read in. In figure 7, only the maximum relation of the contradictory relations is printed out. The old $A < D$ because of *rule1* is not drawn, though it will be propagated into the future. There are other ways of choosing a set of relations without cycles. For example one may add up the confidences on $A < B$ relations and on $A > B$ relations, then choose the maximum of the two.

7 Conclusion, consequences and next steps

In this paper we have described a dynamic scene analysis system which uses qualitative information, available with current computer vision abilities, to calculate relative depths between surfaces. The qualitative information required are motion toward or away from observer and occluder-occludee ordering. The system derives further relations from the data. Errors and inconsistencies are tolerated by requiring confidence factors to decay on each inference step. We have also described in this paper our approach to representing histories of such qualitative values.

This research has opened a number of questions. Among the more important is the problem of using such qualitative calculations as control for other computer vision processes, or as intial estimations for those iterative algorithms requiring them. We see this qualitative assessment as capable of providing a focus of attention when resources are limited and for making real-time dynamic scene analysis possible. The integration of qualitative procedures with numerical ones is an interesting problem.

References

[1] Adiv, G. "Determining three-dimensional motion and structure from optical flow generated by several moving objects," *IEEE Trans on Patt. Anal. Machine Intell., PAMI-7*, 1985, 384-401.

[2] Aggarwal, J.K. and R.O. Duda, "Computer analysis of moving polygonal images," *IEEE Trans on Computers, C-24*, #10, October 1975, 966-76.

[3] Alefeld, G. and J. Herzberger, *Introduction to interval computations*, Academic Press: New York, 1983.

[4] Allen, James F., "Maintaining knowledge about temporal intervals", *Communications of the ACM, 26*, #11, November 1983, 832-843.

[5] Besl, Paul and Ramesh Jain, "Segmentation through symbolic surface descriptions", *Proceedings* CVPR, 1986.

[6] Bobrow, Daniel G. (Editor), *Qualitative reasoning about physical systems*, MIT Press: Cambridge, MA, 1985.

[7] Borchardt, G.C., "Event calculus," *Proceedings*: 9th Internat. J. Conf. on Art. Intell., 1985, 524-27.

[8] Brooks, R.A., "Model-based three-dimensional interpretations of two-dimensional images", *IEEE Transactions on Patt Anal and Machine Intell, PAMI-5* #2, March 1983, 140-150.

[9] de Kleer, J. and J.S. Brown, "A qualitative physics based on confluences," in [6].

[10] Forbus, K., "Qualitative reasoning about space and motion," in D. Gentner and A. Stevens (Eds), *Mental Models*, Erlbaum: Hillsdale, NJ, 1983.

[11] Grimson, W.Eric L. and Theo Pavlidis, "Discontinuity detection for visual surface reconstruction" *Computer Vision, Graphics, and Image Processing, 30*, 1985, 316-330.

[12] Hayes, P.J., "The naive physics manifesto" in D. Michie (Ed), *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press: Edinburgh, 1979.

[13] Haynes, S.M. and R. Jain "Event detection and correspondence" *Optical Engineering, 25*, #3, March 1986, 387-393.

[14] Jain, Ramesh, "Dynamic scene analysis using pixel-based processes", *Computer*, August 1981, 12-18.

[15] Knuth, D.E., *The Art of Computer Programming, III*, Addison-Wesley: Reading, MA, 1973.

[16] Kuipers, B., "Commonsense reasoning about causality: deriving behavior from structure," in [6]

[17] Marr, David and Ellen Hildreth "Theory of edge detection", *Proc Royal Soc B, 207*, 1980, 187-217.

[18] Minsky, M., A framework for representating knowledge, MIT AI Lab Memo #306, Cambridge, MA 1974.

[19] O'Rourke, J. and N. Badler, "Model-based image analysis of human motion using constraint propagation" *IEEE Transactions on Patt Anal and Machine Intell, PAMI-2*, #6, November 1980, 522-536.

[20] Prazdny, K., "Egomotion and relative depth map from optical flow," *Biological Cybernetics, 36*, 1980, 87-102.

[21] Roach, J.W. and J.K.Aggarwal, "Computer tracking of objects moving in space", *IEEE Transactions on Patt Anal and Machine Intell, PAMI-2* #2, April 1979, 127-135.

[22] Roberts, L.G. "Machine perception of three-dimensional solids", in *Optical and Electro-optical Information Processing*, Eds: J.T. Tippett, et.al., 1965, 159-197.

[23] Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press: Princeton, NJ, 1976.

[24] Terzopolous, Demetri, "Image analysis using multigrid relaxation methods", *IEEE Trans on Patt Anal and Machine Intell, PAMI-8*, #2, March 1986, 129-139.

[25] Thompson, W.B., "Inexact vision," *Proceedings* IEEE Workshop on Motion: Representation and analysis, May 1986, 15-22.

[26] Thompson, W.B., K.M. Mutch, and V.A. Berzins "Dynamic occlusion analysis in optical flow fields", *IEEE Transactions on Patt Anal and Machine Intell, PAMI-7*, #4, July 1985, 374-383.

[27] Tsai R.Y. and T.S. Huang, "Uniqueness and estimation of three-dimensional motion parameters of rigid objects with curved surfaces," *IEEE Trans on Patt Anal and Machine Intell, PAMI-6*, 1984, 13-27.

[28] Tsotsos, John, "A framework for visual motion understanding", Technical Report CSRG-114, University of Toronto, June 1980.

[29] Vere, Steven A., "Planning in time: windows and durations for activities and goals", *IEEE Trans on Patt Anal and Machine Intell, PAMI-5*, #3, May 1983, 246-267.

[30] Vilain, Marc B., "A system for reasoning about time", *Proceedings:* National Conference on Artificial Intelligence, AAAI, August 1982, 197-201.

[31] Terry Weymouth, Personal communication.

[32] Witkin, Andrew P. "Scale-space filtering", *Proceedings* 8th Internat J Conf on Art Intell, 1983, 1019-22.

$$\begin{tikzcd}
 & {} \arrow[d, "data"] & & & \\
\Psi_{t-1} \arrow[r, "vel\ rules"] & \Psi_t \arrow[r] \arrow[loop below, "transitivity"] & \Psi_{t+1} & &
\end{tikzcd}$$

$$\Psi(t-1) \xRightarrow{\text{vel rules}} \Psi^*(t)$$
$$\Psi^*(t) \xRightarrow{\text{data}} \Psi^{**}(t)$$
$$\Psi^{**}(t) \xRightarrow{\text{transitivity}} \Psi(t)$$

Figure 1: The Ψ system

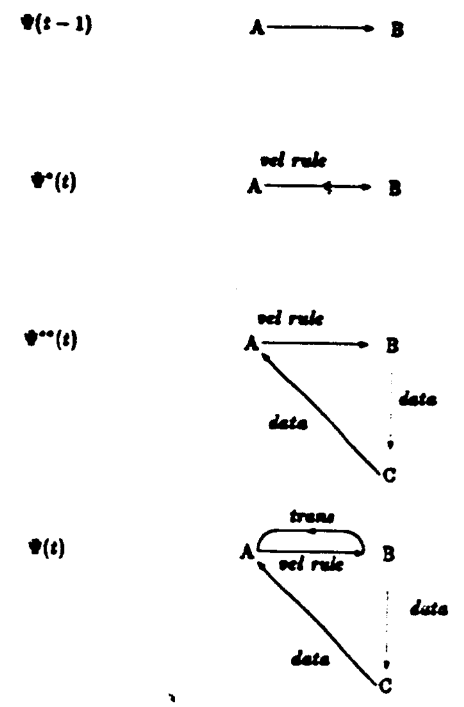
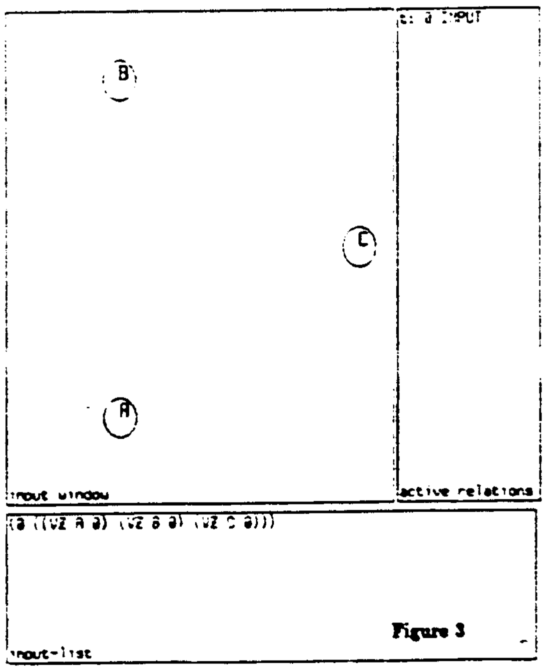


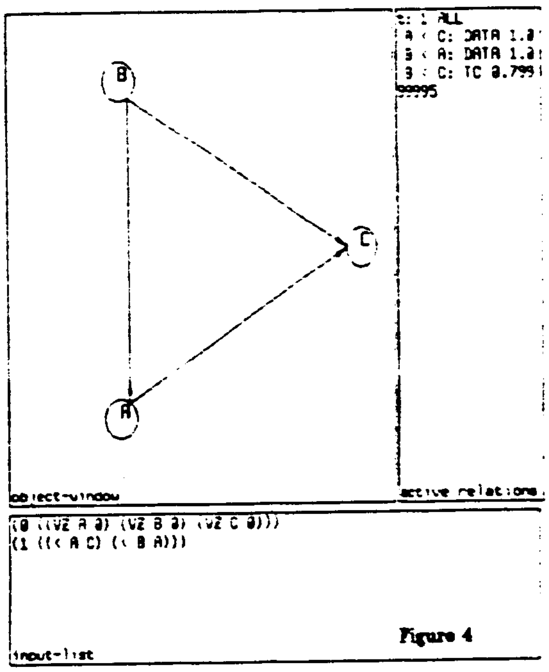
Figure 2: Contradictory relations not having *data* label



```
t: 0 INPUT
```

```
(0 ((VZ A 0) (VZ B 0) (VZ C 0)))
```

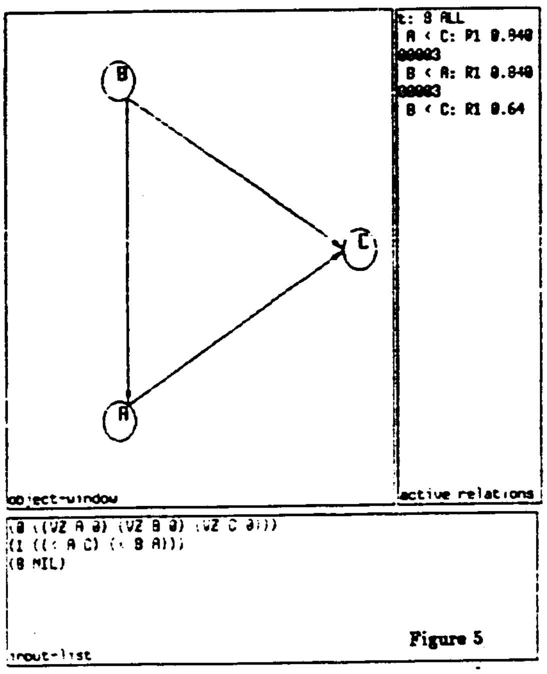
Figure 3



```
t: 1 ALL
 A < C: DATA 1.0
 B < A: DATA 1.0
 B < C: TC 0.799
99995
```

```
(0 ((VZ A 0) (VZ B 0) (VZ C 0)))
(1 ((< A C) (< B A)))
```

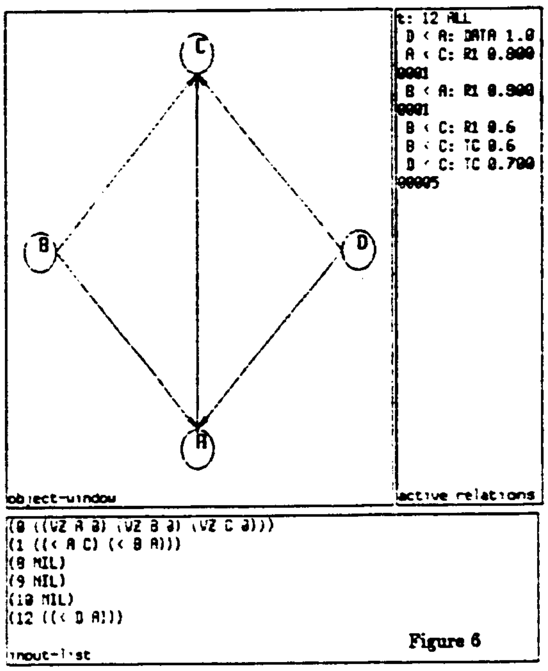
Figure 4



```
t: 8 ALL
 A < C: P1 0.940
00003
 B < A: R1 0.840
00003
 B < C: R1 0.64
```

```
(0 ((VZ A 0) (VZ B 0) (VZ C 0)))
(1 ((< A C) (< B A)))
(8 NIL)
```

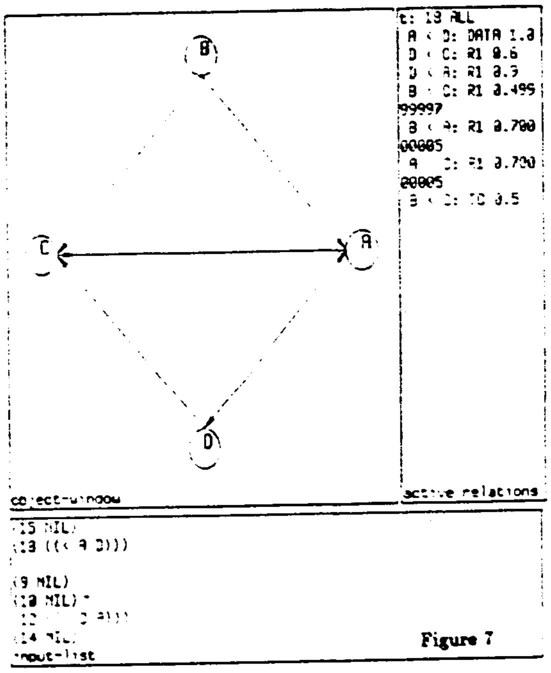
Figure 5



```
t: 12 ALL
 D < A: DATA 1.0
 A < C: R1 0.800
0001
 B < A: R1 0.900
0001
 B < C: R1 0.6
 B < C: TC 0.6
 D < C: TC 0.700
00005
```

```
(0 ((VZ A 0) (VZ B 0) (VZ C 0)))
(1 ((< A C) (< B A)))
(8 NIL)
(9 NIL)
(10 NIL)
(12 ((< D A)))
```

Figure 6

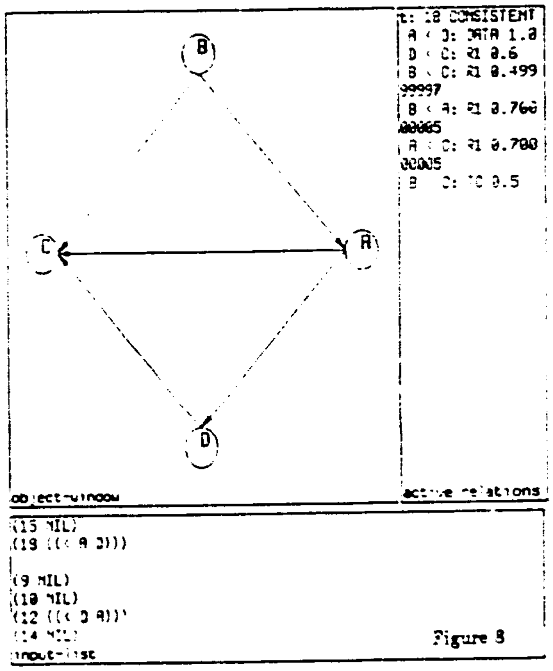


```
t: 13 ALL
 A < D: DATA 1.0
 D < C: R1 0.6
 D < A: R1 0.9
 B < C: R1 0.499
99997
 B < A: R1 0.700
00005
 A   C: R1 0.700
00005
 B < C: TC 0.5
```

```
(15 NIL)
(13 ((< A D)))

(9 NIL)
(10 NIL)
 12 [illegible] D A)))
(14 NIL)
```

Figure 7



```
t: 18 CONSISTENT
 A < D: DATA 1.0
 D < C: R1 0.6
 B < C: R1 0.499
99997
 B < A: R1 0.760
00005
 A < C: R1 0.700
00005
 B   C: TC 0.5
```

```
(15 NIL)
(19 ((< A D)))

(9 NIL)
(10 NIL)
(12 ((< D A)))
(14 NIL)
```

Figure 8

S37-63
7P
118119

A Framework for Qualitative Reasoning About Solid Objects

E. Davis
New York University
New York, NY 10012

N 00522[illegible]

1. Abstract

Predicting the behavior of a qualitatively described system of solid objects requires a combination of geometrical, temporal, and physical reasoning. Methods based upon formulating and solving differential equations are not adequate for robust prediction, since the behavior of a system over extended time may be much simpler than its behavior over local time. ~~This paper discusses~~ a first-order logic, in which one can state simple physical problems and derive their solution deductively, without recourse to solving the differential equations. This logic is substantially more expressive and powerful than any previous AI representational system in this domain.

is discussed

2. Introduction

To operate effectively in an uncontrolled environment, an autonomous robot will have to reason about, understand, and predict external physical events. In many circumstances, however, it will be necessary to reason about physical events on the basis of partial information; the objects involved may not be wholly perceived, or the complete physical specifications may be too complex to use, or the robot may need to reason about hypothetical or generic situations. In such cases, the robot will have to reason qualitatively, inferring general characteristics from incomplete knowledge. Human common sense is often very good at speedy prediction of physical events in qualitative terms; conventional computational schemes are typically very poor at it.

Understanding solid objects is particularly important in physical reasoning, and human beings are particularly adept at thinking about solid objects. Our objective is to build an AI program that can reason qualitatively about solid objects and that can derive correct predictions about their behavior in cases where these predictions are intuitively obvious. This is harder than one might first guess, owing to the many complex ways in which the geometry of the objects affects their behavior.

As a first step toward building such a program, we have analyzed the kinds of knowledge needed to support such reasoning, and we have defined a formal language L in which this formal language can be expressed. We have shown that interesting problems can be solved qualitatively by inference from plausible axioms expressed in L. The language L is more expressive and supports richer inferences than any previous representation scheme in this domain. We give the full details of L and its applications in [1]; here, we give only a sketch.

In concentrating on the representation and formulation of knowledge, and postponing questions of algorithms or control structure, we follow Hayes [2]. However, we depart from Hayes' research program in some respects. We do not attempt to model "naive" physics; rather, we have made free use of Newtonian mechanics, including concepts that have no commonsense analogue, such as total mechanical energy. Also, our proofs are lengthy, violating Hayes' dictum that obvious facts should have short proofs.

The mathematics used here is not "qualitative" in the restricted sense of representing quantities purely in terms of order relationships and constants [3]. Such a representation is too weak to support the inferences needed in this domain.

We have chosen two kinds of problems as foci for our analysis; predicting what happens when a die is dropped inside a funnel (Figure 1) and what happens when a block is dropped onto a table (Figure 2).

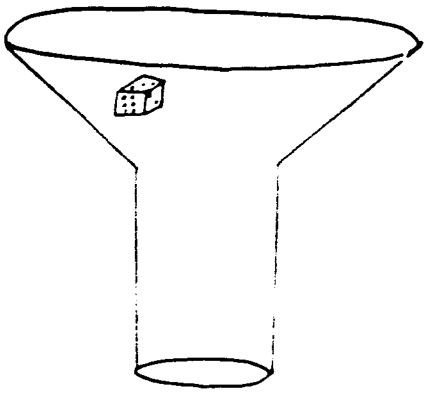


Figure 1: A die is released inside a funnel

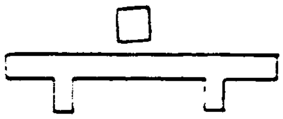


Figure 2: A block is released onto a table.

Different forms of these problems involve a rich, interconnected body of geometric and physical knowledge for their solution. This paper will focus primarily on the "die in the funnel" example.

3. Background

Several previous AI projects have studied the qualitative physics of solid objects. For example, Fahlman's BUILD program [4] determined the stability of a tower of polyhedral blocks. De Kleer's NEWTON [5] predicted the behavior of a point mass sliding on a

PAGE 368 INTENTIONALLY BLANK

constraint. Bundy's MECHO [6] used force analysis and conservation laws to make physical predictions in situations of specialized format. Forbus' FROB [7] predicted the behavior of a point mass flying among constraints. Funt's WHISPER [8] predicted the behavior of a collection of objects by simulating it in a retina-like image. Novak's ISAAC [9] identified English-language programs of fixed form, and applied special case equations to them. Shoham [10] analyzed the local mobility of an object within constraints.

All these programs provided valuable insights. They were, however, limited in geometrical expressivity and in the range of physics understood. Of these systems, only BUILD dealt with three dimensions; and only MECHO dealt with the motion of extended objects. Only a few kinds of physical interactions were considered.

Another limitation of these programs was more subtle, but more fundamental; they were based almost entirely on extrapolating differential behavior. To make a prediction, the program first determined how each state of the system will tend to change, and then extrapolated these changes to predict a continual trend of change up to the point that the structure of the system changes. This extrapolation could be done qualitatively, as in FROB and NEWTON, or symbolically, as in MECHO, or using *point-by-point simulation*, as in WHISPER, or by numerical integration, as proposed by McDermott and Bernecky (personal communication).

For example, FROB [7] predicts the behavior of a bouncing ball in a well by dividing physical space into 4 regions (the interior of the well, the bottom, and the two sides), and dividing the velocity space of the ball into nine (motionless, up, down, left, right, and the four quadrants.) (Figure 3) There are thus 27 possible states of the system. (4 × 9 - 9 impossible states). The laws of physics are then used to determine which transitions between states are allowed, and thus a transition graph of states is developed. FROB predicts that the system follows a path in this transition graph, ending in a stable state of rest.

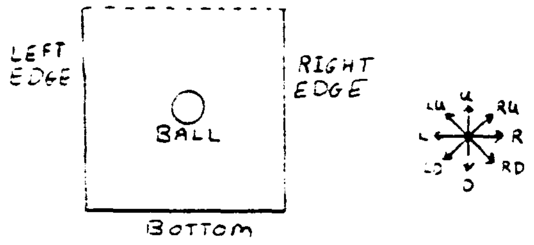


Figure 3: Discretized space and velocity in FROB

However extrapolation is done, simulation is inadequate for robust prediction. In this kind of analysis, each different set of boundary conditions is a different system state. Each such state must be separately detected, categorized, and analyzed, and the system's progress through these states must be recorded. Often, however, such a categorization is difficult and pointless. Consider the problem in figure 1; a small die is released inside a large steep funnel. Many states are possible: the die may be in free-fall; it may be colliding or in continuous contact with the top or bottom part of the funnel funnel on any of eight vertices, twelve sides, or six faces; it may be spinning, sliding, or rolling, up, down, or around the funnel. But the prediction that the die comes out the funnel does not require the enumeration of the states and the paths through them.

There are two further arguments. First, the sequence of states traversed depends delicately on the exact shapes, sizes, and physical properties of the die and the funnel, while the conclusion that the die comes out the bottom is robust with respect to small variations in these parameters. Therefore, if the problem is specified with some small degree of imprecision, simulation will either be impossible, or involve some monstrously branching tree of possibilities. But in qualitative reasoning the prediction that the die comes out the bottom should be almost as easy with imprecise data as with precise. Second, the complexity of simulation goes up rapidly with the number of interacting objects. In figure 4, for example, with one die inside another dropped inside a funnel, the set of system states is the cross product of the possible interactions of the two dice with the possible interactions of the outer die and the funnel. Nonetheless, the prediction that the two dice will come out the bottom is intuitively almost as easy as with only one die.

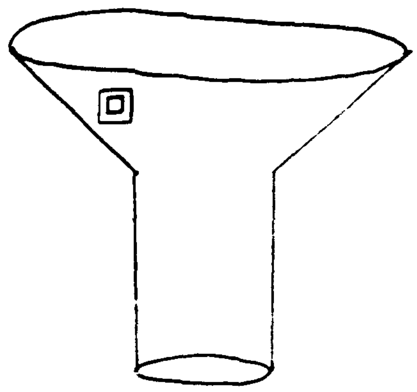


Figure 4: One die inside another released inside a funnel

In short, formulating and solving differential equations is an inadequate technique in this domain, since the behavior of these physical systems over extended time is often easier to characterize than their behavior over local time. A powerful physical reasoning system must be able to infer the general quality of a course of events from broad characterizations of the physical properties of the objects involved, without calculating each subevent.

The programs cited do use some techniques besides simulation. MECHO and NEWTON use energy conservation to prune possible system behaviors. Any state with more mechanical energy than the starting state can be ruled out as a possibility in all future states; for example, the die cannot come out the top of the funnel. FROB predicts that the system ends in a stable state. We believe that effective qualitative reasoning requires more inferences like these, and less use of simulation.

A natural knowledge engineering approach would use rules that state the desired prediction, such as "A small object released inside a steep, large-mouthed funnel will fall out the bottom." But rules of this kind are inadequate, and have rightly been rejected by previous researchers. Any single such rule covers only a small class of problems; covering large classes of problems requires many separate disconnected rules. In particular, a rule like the one suggested above applies only when the die and the funnel are the only objects involved. As soon as another object enters, the rule gives no guidance. That is, such rules are not compositional across objects. Even without other objects, if we allow wide ranges in the shape of the die and the funnel, the conclusion will sometimes apply and sometimes not. Since there is no simple, general rule for when the die comes out the bottom, a different rule must be stated for each special geometric case.

Maintaining a knowledge base with many special case rules is not effective. First, the knowledge base will have to be large and inefficient. Second, if a new case is not precisely covered in pre-canned categories, the system cannot even begin to deal with it. Third, this approach is aesthetically distasteful. A well-designed system should use similarities among different cases of a die falling through a funnel, and similarities between this problem and similar problems, such as a die shot through a tube, or a die dropped into a box. The analyses of these cases ought to have more in common than the use of rules which are syntactically similar. Finally, it seems plausible that an integrated system of rules will support learning better than a tabulation of special cases.

4. Examples and Analysis

We propose that the "die in the funnel" can be analyzed as follows: (i) Due to the topology of the funnel, if the die goes from inside it to outside it, the die must either exit the top or exit the bottom. (ii) Since the die is dropped from rest inside the funnel, it cannot have the energy to exit the top of the funnel. (iii) There is no stable resting point for the die inside the funnel, since it is smaller than the funnel's mouth, and the funnel's sides are steep. (iv) The die cannot stay forever moving within the funnel, for its kinetic energy will eventually be dissipated. Therefore, the die must exit the bottom of the funnel. We claim that in most cases where common sense predicts that the die will come out the bottom, it will be possible to carry out such an analysis, and to support the substeps by inferences from general rules. Different problems will vary in the justifications of the substeps.

Related problems will share parts of the analysis. For instance, in predicting that a die in a small-necked funnel will come to rest at the top of the neck, we may use the identical arguments (i) that the die must either exit the top, exit the bottom, or stay inside; (ii) that it cannot exit the top; and (iv) that it cannot stay inside in a perpetual state of motion. The argument (iii) that it cannot rest stably inside the funnel must be modified to an argument that it can only rest stably at the top of the neck of the funnel; and the additional argument must be made that it cannot exit the bottom of the funnel, since the orifice is too small.

This analysis avoids both problems discussed in section 2. We can avoid analysing, or even determining, the states of motion of the die inside the funnel; all we need to determine is that the die cannot rest stably inside. Different categories of problems are analysed in similar but not identical ways from general principles.

In the rest of this section, we look at variations of this example, and show how this analysis can be applied.

We begin with a simple case (figure 5). The die is a uniform sphere. The funnel is the surface of revolution about a vertical axis of a planar figure with a convex inner side. The radius of the die is less than the radius of revolution of the funnel. The steps of the argument are easily filled in. (i) The top and bottom of the funnel are the only orifices of free space connecting the inside of the funnel with its outside. Therefore, if the die is to go from inside to outside, it must go through the top or the bottom. (ii) Since the die is spherical, its center of mass is in its interior. Since the top of the funnel is horizontal, and directed upward, if the die were to exit it, each point in the interior of the die would be above the top of the funnel at some time. In particular, the center of mass would be above the top at some time. But the die started out from rest below the top of the funnel, and there is no source of additional energy for the die. Therefore, the die cannot come out the top. (iii) By a geometrical argument, the die can only abut the inside of the funnel in a single point. A uniform sphere can be stably supported at a single point only if the supporting surface is horizontal there. The inner surface of the funnel is nowhere horizontal. Hence there is no resting place for the die inside the funnel.

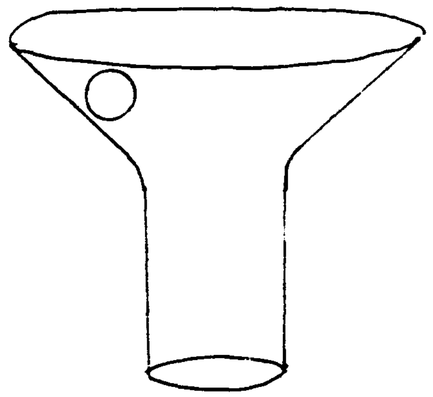


Figure 5: A spherical die inside a radially symmetric funnel

4.1. Out the top, out the bottom, or stay inside

We now consider how this argument can be generalized and modified. (Further modifications are discussed in [1].) Part (i), that the die must either exit the top, exit the bottom, or stay inside does not require that the funnel be a solid of revolution; it requires only that the funnel be a tube with only two orifices. We can weaken the condition further, and require only that all orifices other than the top or the bottom be too small to let the die through. Determining whether a die can go through a hole is an easy geometric calculation for various special cases.

4.2. Not out the top

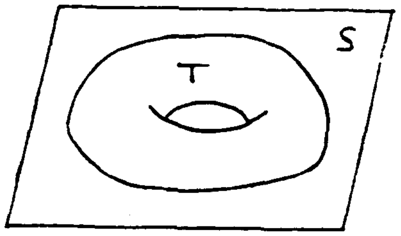
Part (ii), the argument from energy conservation that the die cannot come out the top, depends on the die being convex and on the center of mass of the die starting out below any part of the top. Convexity is only used to establish that the center of mass of the object is in its interior. If this can be done otherwise — for example, by exact calculation, or by establishing that the object shape is a small perturbation of a convex shape, — that is sufficient.

A still weaker sufficient condition is that the *ringed filling in* of the die is convex. The ringed filling in of a three-dimensional shape S is defined as follows: Consider any planar cross section of S . Let C be any simple closed curve that lies entirely in this cross section. Let p be a point in the plane in the inside of C. Then p is in the ringed filling in of S. (Figure 6)

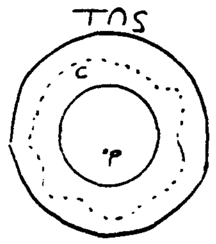
Let S be the shape of some object O, and let R be the ringed filling in of S. Assume R is convex. Clearly, R is equal to the convex hull of S, so R contains the center of mass of O. Let C be a closed curve lying in S and in a plane containing the center of mass of O. If C goes through a planar surface, then so does every point inside C. Thus, if O exits the top of the funnel, then the center of mass of O must likewise, and the proof goes through. Thus we can establish step (ii) for such shapes as a torus, a wiffle ball, or a cratered convex shape.

4.3. No resting point inside

Part (iii), the argument that the die cannot rest inside the funnel, depended in our first example on the strong



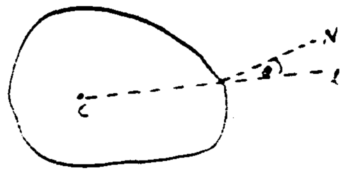
Torus T is cut by plane S



Cross section of T by S
C is a curve in the cross section.
p is a point inside C.

Figure 6

assertions that the die was a uniform sphere and that it could contact the funnel only in a single point. We can easily generalize to nearly uniform, nearly spherical dice. The following formula holds: let θ be the slope of the support; let μ be the coefficient of friction; let ϕ be the maximum angle between the line from the center of mass to a point on the surface and the normal to the surface at that point (Figure 7). The ball can stand still if $\mu \geq \tan(\theta)$ and $\phi > \theta$. Similarly, if one die is a spherical shell containing another die, they rest stably only if the joint center of mass of the two dice is located directly above the contact point of the outer die with its support, and the inner die rests stably inside the outer die.



N is the normal to the surface.
c is the center of mass.
l is a line through c.
θ is the angle between N and l.

Figure 7: Distorted sphere

If the die can contact the funnel in several points with different surface normals, the analysis becomes harder. The wider the range of the horizontal component of the surface normals at contact points, the steeper the slope must be, for the normal forces at the various contact points will tend to act against each other, and thus generate larger friction forces. The following rule holds: Let *A* be in contact with *B*. Let θ be the minimum slope of the surface of *B* at a contact point. Consider the horizontal components of the surface normals of *B* at the contact points, and assume that there is some direction which lies within some small angle ϕ of all these horizontal components. Let the coefficient of friction be μ. If $\mu < \cos\phi \tan\theta$, then *A* will slide down *B*.

Combining all the different ways in which the results (i), (ii), and (iii) may be established, and all the ways in which their geometrical preconditions may be satisfied, gives a rich, interconnected body of results, all with the conclusion, "The die falls out the bottom of the funnel."

5. The Block on the Table

The behavior of the block on the table can be analyzed using a similar argument. After the block is released, it will fall to the table, tipple over a bit, and then move along the table in some combination of sliding, bouncing, and rolling. (Figure 8). It can be estimated how long it will take for the friction involved in sliding and the inelasticity involved in bouncing to consume all the energy gained in the fall and the tipple, and how far the block can travel during that time. A similar estimation can be made for rolling, as long as the object rolls sufficiently poorly. If the surface of the table is uniform, and if these motions will not bring the block off the edge of the table, then it can be predicted that the block will attain a stable state of rest within the estimated time, and within the estimated distance of the point of release.

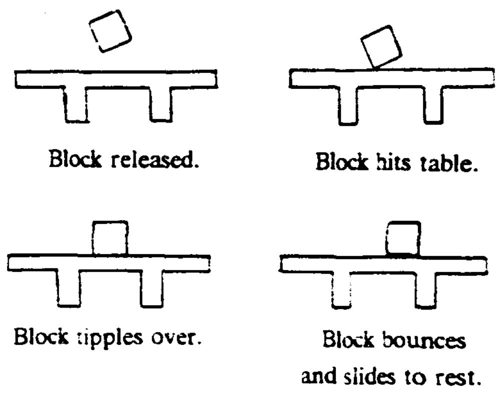


Figure 8: A block settling on a table

6. The Underlying Knowledge

6.1. Geometry

The arguments in section 3 used several different kinds of geometric knowledge, including:

- The ability to name and describe particular point sets that are connected to objects and are useful for physical reasoning, such as the *top* and *bottom* of a funnel. These are called "pseudo-objects" in our system; the problem of constructing them effectively is the same as the problem of constructing metric diagrams in FROB [8].
- Topological predicates. For example, the funnel forms topologically a box with two orifices, and the die starts out inside the box. [11]
- The use of a property quantified over all irregularities of a certain kind in an object. For instance, the funnel has no holes large enough to let the die through other than the top and bottom.
- Special shapes, such as spheres and surfaces of rotation.
- Inequalities on metric dimensions. For example, the radius of the die is less than the radius of the funnel. ,

- The bounding of the range of the surface normal over a part of an object's surface. For example, we wish to say that the slope of the funnel is everywhere positive in its inner surface.
- Convexity and related properties.

Any adequate geometric language will be strong enough that these, or most of these, can be either expressed directly or inferred.

6.2. Temporal Logic

Our temporal logic follows McDermott's [12]. A *scene* is an instantaneous snapshot of the universe. In our domain, a scene specifies the positions and velocities of all objects. A *chronicle* is a function from the time line to scenes. Chronicles include all continuous motion of objects through space, not just those that are physically possible.

The velocity of an object at an instant is defined to be the limit of its velocity from preceding time. Thus, we can speak of the velocity of an object at the instant of a collision.

The "frame" or "persistence" problem of determining what remains true over time [12, 13] does not arise. There are two classes of predicates in the domain. The first class includes predicates that depend on position and velocity of objects. These are not assumed to remain constant over any interval unless proven to do so. The second class includes structural predicates, depending only on the shapes and material properties of the objects. These are always constant over the problem, and so are defined atemporally. (The closed world assumption is made explicit through the predicate "isolated($OO.C$)", which asserts that, during C, no mobile object in the set of objects OO ever comes into contact with any object outside OO.)

6.3. Physics

The world consists of a finite set of solid objects moving in space through time. Objects are rigid and indestructible. The interior of objects may not overlap. Objects have two internal properties besides their shape: a distribution of mass, and a coefficient of elasticity, which determines how the object behaves in a collision. Any pair of objects have a coefficient of friction, which determines the frictive forces between the objects.

Objects are subject to four kinds of forces: a uniform downward gravitational force; normal forces, which act to prevent objects from overlapping; friction; and a weak drag force, which dissipates kinetic energy. Certain objects are *fixed*; they do not move, whatever the forces.

Necessary physical deductions include the following:

- Determining whether a set of objects can attain a stable scene while certain geometric conditions hold.
- Finding constraints on the location of the center of mass of an object or a set of objects.
- Resolving a set of forces, and determining motion under those forces.
- Predicting that the existing structure of contacts between objects will change.
- Predicting a collision.
- Predicting the result of a collision.
- Determining whether a chronicle violates a conservation law.
- Characterizing the paths that an object can take without coming to overlap other objects.

7. Ontology

The ontology for our language requires a number of sorts of individuals.

Quantities. Instants of time, quantities of mass, quantities of energy. These are modelled as real numbers.

Points and vectors. These are modelled as elements of R^3.

Point sets. Subsets of R^3.

Vector fields. These are functions from some point sets to the space of vectors. For example, the surface normals to an object in a fixed position, directed outward, form a vector field.

Rigid mappings. Mappings from R^3 to R^3 which preserve distance and handedness. These specify a change in position.

General velocities. The derivative of a rigid mapping. A general velocity is the composition of a linear velocity and an angular velocity about a specified axis.

Objects. These are primitive entities. The shape of an object is the point set that it occupies in some particular standard position. This is assumed to be connected, closed, and normal.

Scenes. A scene is a snapshot of the world. Formally, it is a function which maps an object to a pair of a rigid mapping, giving the position of the object, and a general velocity. The place of an object in a scene is image of the object shape under the mapping associated with the object in the scene.

Pseudo-objects. These are point sets that "move around" with objects, like the hole of a doughnut, the opening of a bottle, or the center of mass of any object. Formally, a pseudo-object is a pair of a source object and a point set, designating the point set occupied by the pseudo-object when the object is in standard position. The place of a pseudo-object in a scene is the image of its shape under the mapping associated with its source object in the scene.

Chronicles. A chronicle is a function from an interval of time to scenes.

All chronicles are subject to the following constraints:

i. All scenes in the range of the chronicle have the same objects in their domain.
ii. Objects move continuously in space.
iii. Object velocities are continuous from previous times.
iv. The velocity of an object is the derivative of its position.

Chronicles do not have to be physically possible. We use the predicate "phys-poss(C)" to distinguish chronicles that obey the laws of physics.

8. Axioms for Physical Reasoning

Based on the above ontology, we have developed a first-order language L and a set of axioms adequate to solve the first "die in the funnel" example. The complete analysis is rather lengthy; the language uses about ninety non-logical terms, not including the standard arithmetic operators, and the analysis involves about 140 axioms. Most (over two thirds) of the terms and axioms are purely geometrical; the rest relate to motion and to physics. We give below three sample axioms, and the complete statement of the "die in the funnel" example, as illustrations.

Geometric Axiom: Smoothness and the value of the surface normal are local properties of the boundary. Specifically, if two bodies share part of their boundary, then, at any interior point of the overlap, one is smooth iff the other is smooth, and their surface normals are either parallel or anti-parallel.

```
[ body(XX1) ∧ body(XX2) ∧
  XXA ⊆ boundary(XX1) ∩ boundary(XX2) ∧
  X ∈ interior(XXA) ∧ smooth(XX1,X) ] ⇒
[ smooth(XX2,X) ∧
  [ surf-norm(XX1,X) = surf-norm(XX2,X) ∨
    surf-norm(XX1,X) = −surf-norm(XX2,X) ] ]
```

Axiom of Motion: If an object O has zero velocity in every scene of a chronicle C, then it stays in the same place throughout C.

```
[ ∀S S ∈ scenes(C) ⇒ velocity(O,S) = 0 ] ⇒
[ ∀S1,S2 S1 ∈ scenes(C) ∧ S2 ∈ scenes(C) ⇒
    mapping(S1,O) = mapping(S2,O) ] ]
```

Physics axiom: The energy of an isolated set of objects OO never increases in a physically possible chronicle C.

```
[ phys-poss(C) ∧ isolated(OO,C) ∧ T1<T2 ] ⇒
energy(OO,scene(C,T1)) ≥ energy(OO,scene(C,T2))
```

Problem statement: Consider a spherical die, and a radially symmetric funnel. Assume that the inner radius of the funnel is greater than the radius of the die; and that the inner side of a radial cross section of the funnel is convex. If the die is released inside the funnel, and the funnel is held fixed far from the ground, then the die will eventually fall out the bottom of the funnel.

Constants of the example:

odie — the die
ofunnel — the funnel
c — the chronicle
xx-pfunnel — the planar form from which the funnel is generated
xx-center-line — the axis of the funnel
xcenter — a point on the axis of the funnel

Assumptions:

sphere(shape(odie)).
(The die is a sphere.)

mobile(odie).
(The die is not fixed.)

shape(ofunnel) =
solid-of-revolution(xx-pfunnel,xx-center-line).
(The funnel is the solid of revolution of xx-pfunnel around xx-center-line.)

planar(xx-pfunnel ∪ xx-center-line).
(xx-pfunnel is a radial cross section of the funnel.)

convex-side(inner-dside(xx-pfunnel,xx-centerline),
        xx-pfunnel).
(The inner boundary of xx-pfunnel with respect to xx-center-line is convex.)

distance(xx-pfunnel,xx-centerline) > radius(odie) > 0.
(The radius of the funnel is greater than the radius of the die.)

xx-centerline = make-line(xcenter,vup).
(The axis of the funnel is vertical.)

standard-position(ofunnel,startscene(c)).
(The funnel is oriented in standard position.)

fixed(ofunnel).
(The funnel is fixed.)

isolated({odie,ofunnel,oground},c).
(The die is isolated from everything but the funnel and the ground.*)

XF ∈ shape(ofunnel) ∧ XG ∈ shape(oground) ⇒
height(XF) − height(XG) > diameter(odie)
(The funnel is more than the diameter of the die above the ground.)

infinite(c).
(The chronicle is eternal.)

phys-poss(c).
(The chronicle is physically possible.)

motionless(odie,startscene(c)).
(The die starts from rest.)

place(odie,startscene(c)) ⊂
tube-inside(shape(ofunnel), s-tube-top(shape(ofunnel)),
        s-tube-bot(shape(ofunnel)))

(The die starts from inside the funnel.)

Prove:

exits(odie,
    pseudo-object(ofunnel,s-tube-bot(ofunnel,vup)),
    c).
(The die exits the bottom of the funnel.)

9. Conclusions

The strengths and limitations of this theory are evident. On the positive side: Using pure first-order logic, we give a formal analysis of a class of problems beyond the scope of any previous AI theory. Our analysis suggests that a qualitative physics for solid objects should include the following features, among others:

- A rich geometrical theory, including topological, metric, and differential descriptors, and special shapes.
- An account of the behavior of physical systems over extended intervals of time. Such an account should incorporate constraints placed by one object on another; conservation laws, especially conservation of energy; the principle that a physical system tends towards a stable resting point; and an account of the net effects of collisions over extended time periods.
- The ability to determine the existence of a stable configuration of objects within qualitatively described geometrical constraints.
- The ability to calculate, exactly or qualitatively, important physical parameters such as the center of mass. [14]
- The ability to bound the effect of small perturbations.

On the negative side: We have not shown that this type of analysis is extensible to cover all, or most, qualitative reasoning in this domain. We have not shown that such an extension would be, in the long run, any more parsimonious than simply enumerating special cases, as in the rule-based method rejected in section 2. We have not shown that any effective computational methods can be developed on the basis of this theory. We cannot give a final resolution to these problems until we have implemented a working system, and determined the range of problems that it is adequate to address.

We plan to begin implementation by developing an adequate geometric representation and inference system. Ultimately, we want to implement a physical reasoning system with all the features mentioned above.

* We need the ground, because otherwise the hypotheses are inconsistent with the axioms. The axioms assert that an infinite chronicle must come to an end in a steady state. Since we show that there is not steady state for the die in the funnel, we must provide it with the ground to rest on.

10. Acknowledgements

Discussions with Sanjaya Addanki, Alex Botta, Pat Hayes, Leo Joskowicz, and Drew McDermott have contributed immensely to the development of this theory. The research has been supported by NSF grants DCR-8402309 and DCR-8603758.

11. References

[1] E. Davis, "A Logical Framework for Solid Object Physics," NYU Tech. Report #245, 1986

[2] P. Hayes, "The Naive Physics Manifesto", 1978; revised as "The Second Naive Physics Manifesto", in J. Hobbs and R. Moore, *Formal Theories of the Commonsense World*, ABLEX, 1985

[3] D. Bobrow, ed. *Qualitative Reasoning about Physical Systems*, MIT Press, 1985

[4] S. Fahlman, "A Planning System for Robot Construction Tasks," *Artificial Intelligence Journal*, vol. 5, 1974, pp. 1-49

[5] J. de Kleer, "Qualitative and Quantitative Knowledge in Classical Mechanics," MIT AI Lab Tech. Report #352, 1975

[6] A. Bundy, "Will it Reach the Top? Prediction in the Mechanics World," *Artificial Intelligence Journal*, Vol. 10, pp. 129-146

[7] K. Forbus, "A Study of Qualitative and Geometric Knowledge in Reasoning about Motion," MIT AI Lab Tech. Report #615, 1979

[8] B. Funt, "Problem Solving with Diagrammatic Representations," *Artificial Intelligence Journal*, vol. 13, 1980, p. 201-230.

[9] G. Novak, "Representations of Knowledge in a Program for Solving Physics Problems," *IJCAI 5*, 1977, pp. 286-291

[10] Y. Shoham, "Naive Kinematics: One Aspect of Shape," *IJCAI 9*, 1985, pp. 436-442

[11] E. Davis, "Shape and Function of Solid Objects: Some Examples," NYU Tech. Report #137, 1984

[12] D.V. McDermott, "A Temporal Logic for Reasoning about Plans and Processes," *Cognitive Science*, 1982, pp. 101-155

[13] J. McCarthy and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence*, B. Meltzer and D. Michies (eds.), Edinburgh University Press, 1969

[14] A. Bundy and L. Byrd, "Using the Method of Fibres in Mecho to Calculate Radii of Gyration," in D. Gentner and A. Stevens, *Mental Models*, Erlbaum Associates, 1983