NASA Contractor Report 165975

NASA-CR-165975
19820026253

Evaluation of Automated Decisionmaking Methodologies and Development of an Integrated Robotic System Simulation

Study Results

J. W. Lowrie, Dr. A. J. Fermelia, D. C. Haley,
K. D. Gremban, J. Van Baalen, and R. W. Walsh

Martin Marietta Aerospace
Denver Aerospace
P.O. Box 179
Denver, Colorado 80201

Contract NAS1-16759
September 1982

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

NASA CR-165975 September 1982

Study Results

EVALUATION OF AUTOMATED DECISIONMAKING METHODOLOGIES AND DEVELOPMENT OF AN INTEGRATED ROBOTIC SYSTEM SIMULATION

Prepared by:

James W. Lowrie
Dr. Alfred J. Fermelia
Dennis C. Haley
Keith D. Gremban
Jeff Van Baalen
Richard W. Walsh

This work was performed for NASA Langley Research Center under contract NAS1-16759.

MARTIN MARIETTA AEROSPACE
DENVER AEROSPACE
P.O. Box 179
Denver, Colorado 80201

FOREWORD

This document covers the work performed on contract NAS1-16759, Evaluation of Automated Decision-Making Methodologies and Development of Integrated Robotic System Simulation, for the Langley Research Center of the National Aeronautics and Space Administration. It was prepared by Martin Marietta Aerospace in accordance with the contract, Part II, Statement of Work.

The final report for this study consists of three volumes:

NASA CR-165975 - Study Results

NASA CR-165976 - Appendix A, Software Documentation

NASA CR-165977 - Appendix B, Derivation of Requirements Tool Dynamics
Appendix C, Derivation of Simulation Tool Dynamics
Appendix D, Derivation of Requirements Tool Control Law
Appendix E, Simulation Methodologies

Comments or requests for additional information should be directed to:

Jack Pennington
Mail No. 152D
Contracting Officer Representative
Langley Research Center
Hampton, VA 23665

or

James W. Lowrie
Mail No. 0570
Martin Marietta Aerospace
P.O. Box 179
Denver, CO 80201

CONTENTS

FIGURES

TABLES

1.0 INTRODUCTION

1.1 BACKGROUND

During the last decade, the need for automation technology in NASA advanced missions has become apparent. This trend has been driven by many factors including the following:

Cost of Ground Support - Current ground operations are reliant on large teams of specialists to perform functions such as fault detection, fault isolation, failure mode workaround, command processing, and tracking processing. It is becoming less feasible to maintain these large cadres of technical people during flight operations due to economic limitations. Furthermore, these individuals in a typical operation scenario are under-utilized until a failure occurs. With the maturing of decision-aiding techniques such as expert systems, it is becoming feasible to supplement individuals with aids that more effectively extract information from the ever-increasing volume of data. This, in turn, will enable the large cadres of individuals to be significantly reduced while increasing the ability of operators to make decisions effectively.

Man Support in Hazardous Environments - The cost of supporting man in the hostile space environment is necessarily much larger than that of supporting an unmanned system. Manned vehicles must incorporate costly life support systems which decrease vehicle payload capacity. The quality of components and level of redundancy for manned vehicles must be higher than those for unmanned missions. However, human problem solving capabilities are still required for many applications. For some near-term applications, it will be feasible to automate the control process in order to remove man from the vehicle. For more complex operations, problem solving abilities may be remotely incorporated through telepresence and remote man-in-the-loop control. As the state of the art in automated problem solving advances, the individual can be removed from the control loop and will become a supervisor over an automated system.

Non-Optimal Human Control - Laboratory studies have shown that if good models of the system dynamics and appropriate control laws can be developed and implemented in real time, an automated system provides more optimal control than does a man in the loop. This has been demonstrated during physical simulations of rendezvous and docking but applies as well to other areas such as manipulator control.

Pyschological Considerations - When placed in a highly-repetitive, mundane environment, humans have a tendency to become lackadaisical and make mistakes. Many of the tasks inherent in space operations and ground support have this quality. Decision aids in this environment would reduce the repetition and provide a means to quickly evaluate the volumes of data, provide a synopsis of the data, generate recommendations, and allow the human to use the inherent powers of reasoning more effectively.

Limited Strength - For many application scenarios such as payload retrieval, limited human strength becomes a negative factor. Furthermore, human dexterity is significantly reduced by the cumbersome life support

equipment required in space. With the state of the art in actuators and materials, it is possible to develop mechanisms that deliver more torque with faster response times than a human counterpart.

The current state of robotics is limited to relatively-simple, pre-programmed tasks with little or no machine intelligence and very restricted sensing of environments. For the more complex and dynamic environment associated with space applications, it is still necessary to incorporate man in the control loop. In order to achieve the ambitious goals of the space program in areas such as space station and long-life reserviceable spacecraft, it is essential to reduce direct human control of the robotic systems. This reduction can most naturally occur over a four-phase development process.

The first phase is to develop the required system with man in the loop to provide control and problem solving functions. The second phase of robotic system evolution is to extract the man from the primary control loop to assume a supervisory role. In this role, the operator will perform the functions of planning out a sequence of tasks to achieve a specific goal. The robotic system will perform the tasks of trajectory planning, obstacle avoidance, and joint control. In the third phase, the individual will be extracted one more level. In this phase, the operator will perform the function of establishing intermediate goals for the robotic system. The robotic system will perform the functions associated with breaking down the specific goals into individual tasks to be performed. The final phase of robotic evaluation is the development of a fully-autonomous robotic system.

1.2 CONTRACT OBJECTIVES

This contract supports an effort:

1) to bring within NASA, machine intelligence methodologies in automated decision making, and

2) to develop an integrated robotic system simulation as a testbed for applying and extending this technology.

Specifically, the purpose of this investigation is to provide a basis for the development of a robotics simulation (ROBSIM) computer program by:

1) Identifying and evaluating applicable artificial intelligence techniques;

2) Defining the framework of the simulation--the structure of the software, dynamic equations, algorithms for control and decision making, etc.;

3) Developing mathematical models of manipulator components;

4) Developing a graphics display.

Development of the complete simulation software system is not within the scope of this contract.

1.3 REPORT ORGANIZATION

This document covers the work performed on Task 1 through Task 7 of Contract NAS1-16759, "Evaluation of Automated Decision-Making Methodology and Development of Integrated Robotic System Simulation," for the Langley Research Center of the National Aeronautics and Space Administration. It was prepared by Martin Marietta Corporation in accordance with the contract, Part II, Statement of Work.

This document consists of six sections and five appendices. Section 1.0 provides an introduction. Section 2.0 contains the results of Task 1 and Task 2 activities regarding artificial intelligence techniques. Section 3.0 provides a description of the robotic simulation tool developed under this contract and covers work done under Task 3, Task 4, Task 5, and Task 7. Section 4.0 discusses robotic control techniques (Task 6). Section 5.0 presents a short discussion of future use of the robotic simulation in support of the Remote Orbital Servicing System (ROSS). Section 6.0 provides conclusions and recommendations for further work. Sections 1.0 through 6.0 make up Volume I of the Final Report.

Appendix A contains descriptions of the software routines developed under this contract and is contained in Volume II, Part I, of the Final Report. Appendices B through E are contained in Volume II, Part 2, of the Final Report. Appendix B contains the derivation of the dynamic equations for the force/torque requirements analysis tool. Appendix C contains the derivation of the simulation tool nonlinear and linear dynamics equations. Appendix D contains the derivation of the coordinated rate control algorithm. Appendix E contains a discussion of simulation methodologies.

It should be noted that the results of the work performed under Task 8 were previously submitted as a separate report, "Remote Orbital Servicing System (ROSS) Final Report," MCR-82-533, dated April, 1982.

2.0 DECISION-AIDING TECHNIQUES

Artificial intelligence (AI) is the branch of computer science concerned with developing programs that perform functions people consider intelligent. These include interacting with people in a manner natural to them, adapting to changing environments, driving automated systems, and assisting in the process of making complex decisions. The last of these has become especially important because of the computer's extraordinary ability to process large volumes of information. As the technological state of the art progresses, computers are being relied on more and more heavily to assist in decisionmaking. In these roles, computer systems must "understand" global situations, explore hypothetical alternatives, and adjust priorities as situations warrant.

There are several domains where it has become necessary for computers to play a more active role in decisionmaking. One such area is in tactical decision aids. In this environment, the U.S. and its NATO allies are faced with unfavorable force structure ratios compared to the Warsaw Pact countries. The U.S. depends heavily on superior tactical command, control, communication, and intelligence (C^3I) systems to counter the extensive quantitative lead of its potential adversaries. Decision aids in the C^3I environment must do more than just collect, process, and display data. They must begin to perform many decisionmaking functions because of extremely high data rates and the need for nearly instantaneous reaction in the C^3I environment.

Another area in which computers must become reliable decision makers is in applications of autonomy, such as automated weapons systems and deep-space probes. Future unmanned weapon systems cannot afford to rely on teleoperator control due to the many issues accompanying survivability. Therefore, it is important to understand the automation requirements of such systems. Automated weapons systems must navigate, recognize potential threats from a deceptive adversary, and recognize friendly situations. Making these kinds of decisions requires an intelligent decision maker. Deep-space probes must also be autonomous. As they travel further from the Earth, they must be increasingly self-sufficient because monitors on the Earth can no longer react to unknown situations in a time-critical manner due to the communication delay involved.

As the complexity of functions such as these escalates, traditional algorithmic approaches to software systems become inadequate. In some cases this results from inefficiency and in some from the difficulty of specifying an explicit algorithm, as for example in the case of threat assessment or the medical diagnosis of disease.

AI has evolved to fill this gap. Many of its techniques capture fuzzy nonalgorithmic reasoning processes and, more importantly, heuristic reasoning principles for decision making and problem solving that can be used to address situations not anticipated in the original design of a system. AI's action- or goal-oriented approaches allow a higher level of control, which greatly enhances efficiency and permits a more ready coordination of subtasks en route to the desired goal. AI's use

of natural languages such as English significantly eases user interaction and certain of its techniques can be used to reduce system development time.

One important AI approach that critically depends on decisionmaking is the expert system. An expert system is a computer program designed to perform an intelligent task normally done by a human expert--for example, to assess tactical threat situations, diagnose medical diseases, isolate faults during satellite or computer system failures, or control an automated system. Expert systems generally contain an inference mechanism that drives the actual decisionmaking process and hence coordinates the actions of the expert system and the processes the expert system directs. Because inference engines base their decisionmaking on their knowledge of the world, how that knowledge is represented is critical. Representation is, in fact, one of the prime factors in the speed, sophistication, adaptability, and complexity of the inference mechanism, which, in turn, crucially determine the power and range of the entire expert system.

Decisionmaking systems should ideally incorporate a computationally simple, yet powerful, technique. This is not always possible, of course, and speed (reaction time) is generally inversely proportional to complexity. Systems must thus be appropriately situated on the speed-complexity continuum weighing the tradeoff in accord with the system's application. Adaptability--the ability to deal with unanticipated situations--becomes increasingly necessary as the autonomy of an application increases. While speed, reaction time, and adaptability are measures of a system's behavior, sophistication is in part a measure of the robustness of a system's representation.

The degree of robustness often has a direct effect on system behavior. Two kinds of sophistication must be considered when evaluating a representation. The first is level of abstraction. For example, representing an electronic device as an organized set of functional circuits with functions describing input and output relationships is a higher level of abstraction than representing the device as a conglomeration of integrated circuits, resistors, etc. If the higher abstraction level is appropriate for analysis, being forced by a formalism to use the lower level is both slow and cumbersome. Often it is effective to be able to represent an application at multiple levels and to facilitate switching between levels in the course of problem solution. The second type of sophistication involves another form of multiple levels. In this, a mechanism is provided that supports a hierarchy of representations. The lowest level contains the application representation; the next level contains a representation of the first level's organization. This is often called a metarepresentation. Any number of levels of metarepresentation can be present in the hierarchy. This type of structure can be critical if a system must consider its actions before it performs them.

This chapter discusses a variety of decisionmaking techniques and tools that have been identified and evaluated by the AI group at Martin Marietta Denver Aerospace during the last year as part of the ROBSIM project. Emphasis was placed on identifying those techniques developed in

the field of artificial intelligence that could be applied to the aerospace environment.

2.1 DECISION TREE MANIPULATORS

The first of these techniques is the decision tree manipulator. A decision tree is a tree data structure frequently used in gaming situations where the nodes are possible world states and the arcs are operations that cause the world states to change. If, for example, the domain were the game of tic-tac-toe, a world state represented by a node would be the configuration of the board after a given number of moves. The operations represented as the arcs branching from such a node would be the possible moves available to the player at that point in the game. Associated with each node is an evaluation function that determines the relative desirability of the given world state. (These evaluation functions are generally derived empirically for a given application.) A system simulates possible actions by generating several levels of a tree in accord with its knowledge of the domain.

Thus, from the start node of a tic-tac-toe application, the system would generate nine nodes or world states corresponding to the nine possible opening moves. For the next or third level, it would generate eight nodes for each of the nine nodes at the second level, since the second player must choose among eight moves. This third level therefore has 72 nodes, corresponding to the 72 possible board configurations or world states that are possible at the end of two moves. After generating several such levels, the system would choose the best path or arc based on the result of the evaluation function. This method of tree search is highly dynamic--branches of the tree are not generated until it is time to search them.

Obviously, both the world model and the relative desirability of a state depend on the domain. Given that both of these problems are solved, decision trees sometimes provide an effective method of making decisions. They can be made to be highly sophisticated: a game tree, one type of decision tree, can represent all possible sequences moves and end states of a simple game. The tree, however, becomes massive. For example, the size of a full game tree for tic-tac-toe can be roughly estimated by noting, as described above, that the start node has 9 daughters, each of which has 8, etc. Thus the bottom of the tree has 9! or 362,880 nodes. The total number of nodes in the tree is 9!/1! + 9!/2! + ... + 9!/9! or 623,530. Of course, many paths in the tree terminate in end-states at a level shallower than 9 so the actual number will be somewhat less--but it will still be enormous and the tree would be computationally intractable.

The ability of decision trees to address complex applications is hindered by a lack of both types of sophistication discussed above. This contributes to the computational complexity of the decision tree manipulation algorithm and, consequently, a slow reaction time. A simple algorithmic approach to tree manipulators is clearly insufficient:

manipulators must include the ability to incorporate domain-specific information and heuristics as well as tree pruning heuristics such as alpha-beta pruning (Ref 1). The latter is used during the dynamic generation and search of the tree, as described above. It is a technique that determines when it is unnecessary to search entire subsections of a tree and therefore saves the system the expense of generating them.

The savings such techniques bring, however, will only partially mitigate the above problems with decision trees. In any case, decision trees tend to be unadaptable to changes in the environment that could not be foreseen--to, as it were, branches in the tree that could not be generated a priori. Finally, it is difficult to display decision trees and hence to understand their dynamic behavior. Therefore, special graphics routines must be developed to display trees and to move this display up and down through different subtrees.

2.2 PROBLEM SOLVERS

A second class of decisionmaking systems is the first-order predicate calculus theorem provers. In this deductive method, world knowledge is represented as a set of axioms and theorems. The system makes decisions by proving new theorems. To decide whether a particular goal; e.g., "Move box u to place p," were possible, it would represent the goal as a well-formed formula (wff) in the predicate calculus:

$(\exists u)$ [Box(u) AND AT(u,p)]. (Ref 2)

This is considered to be a theorem, and the system attempts to find a proof for it from the axioms and theorems already in the system.

The technique generally used to construct proofs is called resolution (Ref 3). The basic idea behind resolution is to add to the data base the negation of the conclusion of the theorem to be proved and then attempt to find a contradiction in the data base. This is a general technique and has serious drawbacks. For example, to perform resolution, all axioms and theorems must be in clause form. In clause form, formulas must appear as a set of clauses connected by "AND" operators, as in the above example. A clause is a formula with constants, variables, or predicates connected by "OR" operators.

While the translation of arbitrary well-formed formulas to clause form is relatively straightforward, it obscures the meaning of the original formula, making reconstruction and explanation difficult. Also, as the number of clauses in the data base grows, it becomes increasingly difficult to find the correct clauses to resolve in constructing a proof. The search technique used by these theorem provers for finding such clauses is guided by general and not particularly effective heuristics, such as "Only resolve an axiom with a theorem--don't resolve it with another axiom."

Several systems have been implemented based solely on theorem proving--including both problem-solving and question-answering programs (Ref 4), but they are highly inefficient. Again this inefficiency can be attributed, in part, to this technique's lack of either type of sophistication. Clearly, when search in these systems is not constrained, the combinatorial explosion renders them inappropriate for most applications. One method of limiting search is to incorporate domain-specific information. Unfortunately, theorem provers by themselves have no mechanism for doing this: they only manipulate symbols syntactically. They do not consider the semantics or meaning of an expression. In fact, a general theorem prover is often used as the primary deductive component of a larger problem-solving system. Domain-specific meanings can be included by, for example, integrating planning techniques such as the means-ends analysis of the General Problem Solver (Ref 5) and STRIPS (Ref 6). In fact, SHAKEY, one of the early AI experiments in robotic control, used just that combination (Ref 7).

Such problem-solving systems generally define a set of states the model of the world can be in and a set of operators that change the world model from one state to another. They begin their analysis at the goal state, working backwards to identify a sequence of actions (i.e., operators) that could get to the final state from the initial state.

Means-ends analysis determines the difference between the initial and goal states and picks the operator that would most reduce that difference. If this operator can be applied in the initial state, it is. If the result is the goal state, the search is over; otherwise the difference between this new state and the goal state is found, a new operator is chosen, and the process continues.

If the original operator can't be used in the initial state, the preconditions of that operator are set as a subgoal. The search process is used recursively in an attempt to find an operator that achieves the subgoal. If this succeeds, the original operator is then applicable and the search continues as above. If the subgoal is unattainable, another operator is found to reduce the difference between the initial and goal states and the process iterates.

Another primary mechanism for circumscribing search is backtracking. Here an operator is chosen that, if applied, would attain the goal. If it can be used in the initial state, it is, and the search is complete. If not, its preconditions are established as a subgoal and the process is used recursively in an attempt to attain the subgoal. This continues until either a sequence of operators is found that, when applied in the initial state, leads to the goal, or a state is reached such that no operator, when applied, would result in that state. In the former case, the search is finished; in the latter, the search "backs up" to the state just preceding the blocked state (i.e., to the most recent choice point), a different operator is chosen, and the search continues in this "depth-first" or "chronological" fashion.

Basic search strategies such as means-ends analysis or backtracking can be augmented in a number of ways. Backtracking, for example, can be

not simply to the most recent choice point, but to the one that has been identified as the reason (or one of several possible reasons) for failure. In addition to this "relevant" (or "dependency-directed" or "nonchronological") backtracking, means-ends analysis can be organized hierarchically so as to focus initially on only the most important aspects of a situation, leaving the lengthy elaboration of details until after a full, high-level plan has been formed. Numerous other techniques such as plan repair, constraint satisfaction, and goal regression can be incorporated to enhance the efficiency of the procedures that guide search in a problem-solving system (Ref 8). The reason for increased efficiency of some of these is partially due to "level of abstraction" sophistication. If appears, however, that no meta-representation sophistication is present in any of these techniques.

2.3 RULE-BASED SYSTEMS

Another decisionmaking mechanism developed within artificial intelligence is the rule-based system. A rule-based system consists of a long-term memory, a short-term memory, and a rule interpreter. Long-term memory contains rules, short-term memory contains information defining the current state of the world, and the interpreter systematically applies the rules to the world state. Rules are situation-action pairs: the left side of a rule contains a set of predicates in conjunctive normal form and the right side of a rule contains a sequence of actions. In conjunctive normal form, an expression is written as the conjunction of a set of disjunctions of literals, e.g.,

[R(x) OR T(y,z)] AND P(y) AND [-P(z) OR T(x,z)].

A rule resembles an if-then statement: If the left-hand side of a rule holds true, then the actions on the right-hand side are performed. A rule might say, for example,

If Satellite downlink is off nominal rate,

Then Advise "Re-tune bit synchronizer center frequency," and examine rules for satellite-redundant crystal oscillator.

The rule interpreter searches the predicates on the left-hand sides of rules in long-term memory for a match with some of the "state-of-the-world" predicates in short-term memory. It finds all the rules whose predicates all match, i.e., all the rules that accurately describe some aspect of the current world state. The interpreter uses a conflict resolution mechanism to determine which of these rules is most appropriate and then executes the sequence of actions on the right-hand side of that rule. These actions normally modify short-term memory--i.e., change the state of the world--and perform I/O operations. However, in some rule-based systems, the actions can also modify the rules themselves, giving these systems significant potential for adaptability.

As applications become complex, requiring more sophisticated distinctions of subtlety, the number of rules may become extremely large and searches of the rule base computationally expensive. To circumvent this problem, many rule-based systems allow for meta-rules, i.e., rules defining the way rules should be searched and applied. The rule base or long-term memory might be partitioned into sets of related rules such that any given set would be relevant only in particular situations. When such a situation is identified--i.e., when the predicates on the left-hand side of some meta-rule are all found to match or be true in the current world state--the meta-rule would direct the rule interpreter to confine its search to the appropriate set of rules. One use of this partitioning is to organize different levels of representational abstraction. Thus, both types of sophistication are provided for to varying degrees in different rule-based systems.

A prime feature of these partitioning systems is a high degree of modularity and extensibility: isolable partitions into which rules are grouped can readily be added to or transported between systems. Rule-based systems generally have numerous other unique features that can be advantageous for certain applications. These include significant sophistication and adaptability, ease of rule modifications, a natural parallel to the way human beings often make decisions, an ability to incorporate judgment into making decisions, a high degree of efficiency when rules are carefully organized, and the ability to trace the decisionmaking process and present it to the human user as part of an explanatory justification.

2.4 LOGIC PROGRAMMING LANGUAGES

In addition to specific techniques, several tools of varying generality have been developed. One such inferencing tool for computer decision-making is a Logic Programming Language (LPL) such as PROLOG (Ref 9) or LOGLISP (Ref 10). These languages support a technique of programming that is radically different from traditional programming languages. Computational specifications in an LPL consist of a set of declarative sentences that the system attempts to show are true. Thus, the control structure of a program is handled entirely by the LPL run-time support system. More specifically, the run-time system, in the process of executing a program, searches a knowledge base attempting to demonstrate the truth of statements in the program. Statements comprise both the program and the knowledge base. Statements in the knowledge base are referred to as assertions, and statements in the program are called conditionals. All statements have the same form:

```
a : -b,c,d
```

and can be read declaratively as "a is true if b and c and d are true" or procedurally as "to get a, successfully execute b, c, and d." An LPL acts, in effect, as a theorem prover on individual program statements. Unlike a theorem prover acting at a global level, however, an

LPL's resolution of a program statement can affect any number of decision-related consequences--from the simple branching of control to the execution of an action or even the alteration of the knowledge base or inferencing strategy. Because decisionmaking includes a complex form of knowledge base querying, an LPL is also ideal for knowledge base manipulation. Since facts in the knowledge base are stored in the knowledge base, LPLs allow for both "level of abstraction" and meta-representation sophistication. The latter, however, is not encouraged in the LPL style of programming.

2.5 REPRESENTATION LANGUAGE LANGUAGES

An extremely useful tool for decisionmaking is a representation language language (RLL) such as the Modifiable Representation System (MRS) (Ref 11). An RLL is actually a union of the representation formalisms of a knowledge representation system and the inference mechanisms of a decisionmaking system. The knowledge representation aspect of RLLs gives them sensitivity to the kind of information being manipulated. Different kinds of knowledge are best represented in different ways. Certain procedural information, for example, might best be captured in a production rule format. Declarative knowledge about the attributes of particular items, e.g., connectivity relations among components of an integrated circuit, might most effectively be stored on the property list of the individual components or in an associative network. Associating attribute information so closely with the items themselves might increase efficiency of computations involving such information. Information about sets of objects, on the other hand, might be represented in bit vectors for optimum storage and perhaps computational efficiency.

An RLL has the ability to use different representational modes for different kinds of knowledge--all within the same program. This can be done, if desired, at a level independent of any user program. Programs can thus treat different kinds of knowledge in a uniform way. The RLL's inference mechanisms determine what kind of knowledge is being manipulated, accessed, or stored, and on that basis decide which means of representation is most appropriate for the particular instance. Consequently, the structure of the knowledge base--or even the knowledge base in toto--can readily be changed without requiring modification of any user program.

Sensitivity to the external environment is also an important metric for decisionmaking systems. Some RLLs have a unique approach to external sensitivity. Because these systems are designed to allow different types of knowledge to be represented differently, when they are asked to store and retrieve knowledge, they must first determine how to do so. This approach is generalized in such a way that before an RLL performs an operation, it determines how this operation should be carried out. Thus, the system inspects its knowledge base, performs inference, or examines the external environment to determine how to do an operation. The last of these affords RLLs an extraordinary sensitivity to

the external environment, assuming it has access to the appropriate sensors. As a simple example, suppose an RLL is asked to do X and the RLL "knows" two ways to accomplish X. The first is memory efficient and computationally expensive. The second is memory expensive and computationally efficient. The RLL can determine the amount of free memory and decide accordingly which method to use. Constraints in the external environment can affect an RLL's behavior in the same fashion.

Besides integrating representation and decisionmaking in a single system, an RLL goes beyond other AI tools in another way: it provides a meta-representation level in which its own structure is expressed. The meta-representation exploits the same formalism as the first level representation. RLLs not only provide meta-representation sophistication but encourages its use. They also provide for "level of abstraction" sophistication in a straightforward fashion.

An RLL can thus manipulate a representation of itself in the same way it manipulates user-defined representations. This allows RLLs to reason about themselves and even to modify their organization and inference strategies. To elaborate an earlier example, an RLL might have, in its representation of itself, explicit knowledge about the methods of computation and storage it could use and the differential memory and computational efficiencies of those methods. It might monitor the general system load to identify the long-term use trend, and on the basis of this use assessment, the RLL could decide in general what operational and representational methods would be best. It could use that decision in future situations and could even restructure its current knowledge base and its existing computational procedures accordingly. This capacity for learning is a prime feature of RLLs. It gives them enormous potential for sophistication and adaptability.

2.6 DEVELOPMENT TOOLS

Finally, several specific tools have been created to assist development of rule-based expert systems, a widespread form of AI decisionmaking tool. ROSIE (Ref 12), AGE (Ref 13), and EMYCIN (Ref 14) present relatively "user-friendly" frameworks for expert system development including facilities of varying elaboration for such things as documenting and responding to user requests for justification or explanation. OPS5 (Ref 15) is a substantially more limited tool with certain structural problems (such as not allowing the evaluation of functions on the left-hand side of rules) and few or none of the frills for such things as explanation. AMORD (Ref 16), a rule-based system for problem solving, is one of the only to treat seriously the problems of how adding new information or rules affects the system's consistency.

Each of these development tools was designed for different purposes and hence addresses different issues and problems. Of all of them, ROSIE is perhaps the most flexible and powerful, permitting the user to modify many major components of the system to suit his needs--for example, to design his own control structure or rule interpreter but not, it

should be noted, to alter the way rules and knowledge are represented. Much of its flexibility is due to the authors' attempts to include a limited form of meta-representation sophistication and a very general "level of abstraction" sophistication. The former is realized by collocating the rule set and the knowledge base. This allows rules to operate on other rules as though they were data.

Yet even ROSIE is limited, allowing only rule-based techniques. A mature expert system for robotic control must consist of many components--it might, for example, have a rule-based reasoner to establish what needs to be done, a planner or problem solver to determine how to do it, a component for extracing information from the human expert to construct the rule base, a component to maintain the consistency of the rule base and the knowledge base, as well as a natural language interface to communicate with the human users and developers.

This is not to say, of course, that these specific rule-based development tools could not be used to create systems to perform a wide variety of tasks--after all, the simplest Turing machine is sufficient for any computable task. The problems, however, are along the lines of efficiency (both computational and representational), flexibility, modifiability, and ease of use. A given application may require any number of control structures, inferencing and representational techniques, and modes of user interaction.

2.7 THE EXPERT SYSTEM-EXPERT SYSTEM

A truly useful development tool would aid in the creation of all these facets. Such a tool would be an expert system for building expert systems--as it were, an "expert system-expert system" or $(ES)^2$. The Artificial Intelligence group at Martin Marietta has been investigating $(ES)^2$ for some time. A primary constituent of this tool will be the interface component that will help establish the optimal configuration of natural language (e.g., English), graphics, and special-purpose restricted languages (e.g., for limited data base query or command and control) for system development as well as user interaction. The natural language interface is an especially important aspect of this: it allows a user to communicate in near-English with a computer program--to give commands and requests, ask questions, and receive justifications and explanations. This ability is essential for the widespread use of expert systems.

Given that there is no sufficently-encompassing theory of grammar and language, current natural language interfaces are developed as specially designed programs for particular applications. Thus, in addition to establishing the configuration for interaction, the interface component of $(ES)^2$ will be expert at building natural language interfaces. It will combine its linguistic knowledge with considerations of not only the intended use and users of a system but also the domain-specific constraints and the contents of the knowledge base. It will have a similar capacity for developing graphics aids for the interface and for

incorporating special-purpose restricted languages to handle those interactions for which a highly-limited set of options is possible.

One of the most important aspects of $(ES)^2$ involves the transfer of expertise from human experts to a computer knowledge base. This has traditionally been a multistep, iterative process in which the AI researcher extracts knowledge from an expert, encodes it in inferencing rules, and tests the expert system with those rules. He then takes the output back to the expert who determines which parts of the output are incorrect and is then coerced by the researcher to identify and correct inaccurate rules. The researcher returns to his program with the new set of rules, obtains new output, and the lengthy process iterates.

The knowledge transfer component (KTC) of $(ES)^2$ will help automate and simplify this convergence to an accurate rule base. Knowing the domain and general intent of the application, it will, in English, interactively obtain information from the expert. This information will be examined for inconsistencies; if any are found, the expert will be consulted for emendations. The knowledge transfer component will integrate the information into its evolving model of the expert system under development. This will be used to determine the logical completeness of the information: again, the human expert will be consulted when incompleteness is uncovered. The model will also provide a basis for the KTC to present scenarios for which the expert describes the appropriate response or output.

The KTC may also incorporate the traditional iterative process, translating the knowledge into a form for inferencing (e.g., into standard situation-action rules), running the developing expert system, and presenting the output and its inaccuracies to the human expert. Once the expert has identified the inappropriate or incorrect portions of the output, the KTC can examine the reasoning of the expert system to pinpoint the rules at fault. The process then iterates as the KTC returns to the expert and requests corrections to those rules, i.e., to those parts of the information supplied by the expert earlier. Using such techniques as consistency, completeness, and iteration in a user-friendly, English-based interactive environment, will enable the automated transfer of knowledge from the human expert to the knowledge base of an expert system.

Another constituent of our "expert system-expert system" will handle structuring and coordinating the knowledge base and inference engine. As noted above, different representational schemes are appropriate for different applications and different inferencing techniques and computational procedures prove more effective under different circumstances. This component of $(ES)^2$ will choose among the varied combinations of representation, inference and control mechanisms, determining the configuration most suited for the specific application, or for specific aspects of an application. It will also embody techniques to adapt and extend existing knowledge bases at any point in their development and use.

In a similar way, different aspects of a problem might be solved more effectively via different problem-solving techniques and different control mechanisms. In the example mentioned earlier, determining what to

do might be best decided by a rule-based system while deciding exactly how to accomplish it might be handled most effectively by a planner. Thus, another component of $(ES)^2$ will be responsible for decomposing the problem tasks and determining the most effective means of handling each aspect.

The systems developed by $(ES)^2$ will flexibly support any number and any combination of representational, computational, inferencing, and problem-solving techniques. An excellent vehicle for doing this is a representation language language (RLL). An RLL's capacity for choosing among these varied techniques, depending on the nature of the information or task, would allow the user to treat not only all data but also all tasks in a uniform manner, without concern for how the data will be manipulated or the task carried out.

$(ES)^2$ has been the long-range goal of the Martin Marietta Artificial Intelligence group since its inception. A crucial factor in creating and maintaining the flexibility of $(ES)^2$ is the identification and development of new technologies. Towards this end, the AI group has established a replete tool base of AI technology, which it is extending by actively investigating and evaluating research being conducted by other members of the AI community. The group is further extending the tool base by pursuing its own primary and applied research toward the ultimate goal of creating and using $(ES)^2$, the expert system for building expert systems.

2.8 REFERENCES

1. For a discussion of alpha-beta pruning, refer to Nils Nilsson: Principles of Artificial Intelligence. Tioga Publishing Company, Palo Alto, CA, 1980, pp 121-126.

2. This example is taken from Nils Nilsson and Richard Fikes: "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving." Technical Note 43, SRI Project 8259, SRI International, Menlo Park, CA, 1970.

3. C. Green: "Theorem Proving by Resolution as a Basis for Question Answering Systems." Machine Intelligence 4, B. Meltzer and D. Michie (Eds.), American Elsevier Publishing Co., Inc., New York, 1969, pp 183-205.

4. C. Green: "Application of Theorem Proving to Problem Solving." Proc Intl Joint Conf on Artificial Intelligence, Washington, DC, May 1969; and C. Green: "Theorem Proving by Resolution as a Basis for Question Answering Systems." Machine Intelligence 4, B. Meltzer and D. Michie (Eds.), American Elsevier Publishing Co., Inc., New York, 1969, pp 183-205.

5. G. Ernst and A. Newell, GPS: "A Case Study in Generality and Problem Solving." ACM Monograph Series, Academic Press, 1969.

6. Nils Nilsson and Richard Fikes, op. cit.

7. Ibid for discussion of STRIPS, the driver for SHAKEY. For further discussion of SHAKEY per se, see Richard Fikes, Peter Hart, and Nils Nilsson: "Learning and Executing Generalized Robot Plans." Artificial Intelligence, 3, 1972, pp 251-288; and Richard Fikes, Peter Hart, and Nils Nilsson: "Some New Directions in Robot Problem Solving." Machine Intelligence 7, B. Meltzer and D. Michie, Edinburgh University Press, Edinburgh, 1972, pp 405-430.

8. For an overview of these and other tactics, see Earl Sacerdoti: "Problem Solving Tactics." Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, August, 1979, pp 1077-1085.

9 For a practical introduction to PROLOG, see W. Clocksin and C. Mellish: Programming in PROLOG. Springer-Verlag, New York, 1981.

10. J. Robinson and E. Sibert: "Logic Programming is LISP." Rome Air Defense Center-TR-80-379, Vol 1, 1981.

11. Michael Genesereth: "The Architecture of a Multiple Representation System." Memo HPP-81-6, Stanford Heuristic Programming Project, Stanford University, Stanford, CA, 1981.

12. D. A. Waterman, et al, "Design of a Rule-Oriented System for Implementing Expertise." Rand Publication N-1158-1-ARPA, The Rand Corporation, Santa Monica, CA, 1979.

13. Penny Nii and Nelleke Aiello: "AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs." Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979, pp 645-655.

14. W. Van Melle, et al: "The Mycin Manual." Report No. STAN-CS-91-885, Stanford University, Stanford, CA 1981.

15. Charles Forgy: OPS5 Manual. Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1980.

16. Johan de Kleer, et al: "AMORD: A Deductive Procedure System." MIT-AI-Memo 435, MIT, Cambridge, MA, 1978.

3.0 ROBOTIC SIMULATION

The development of robotic technology and operational systems in the near future will require the aid of a number of computer aided tools. These tools are necessary to pursue research in key technology areas, demonstrate concept feasibility, aid in system design, and provide an operations analysis capability. One of the major goals of this contract was to define the overall structure of a robotic simulation tool and to develop a framework for that tool.

The Robotics Simulation (ROBSIM) Program has been designed to provide a wide range of computer capabilities in the areas of robotic system design and analysis. Under this phase of development, implementation has been in the form of building a framework for the overall ROBSIM program and a framework for the simulation tool within the ROBSIM program. These frameworks provide a base upon which to build and integrate robotics capabilities in a structured and coordinated manner. Addition of capabilities and models is made easy by the modular framework developed. This design allows a collection of capabilities to be built up as needed and will in time result in a powerful set of design and analysis tools. The ROBSIM program is expected to be a continually-evolving and expanding set of capabilities.

The overall ROBSIM program structure is composed of three major functions controlled by a program executive as shown in Figure 3-1. The three major ROBSIM functions are:

1) System Definition

2) Analysis Tools

3) Post Processing

Each of the major functions is designed in a modular fashion to allow for easy future expansion.

The System Definition function handles user input of the robotics system geometry and mass properties as well as environmental parameters and geometry. A disk file is created to be used as input to the Analysis Tools and Post Processing functions. The Analysis Tools function handles the computational requirements of the ROBSIM program. The Post Processing function allows for more detailed study of the results of the Analysis Tools function execution(s). Each of these areas is discussed more fully in the following sections.

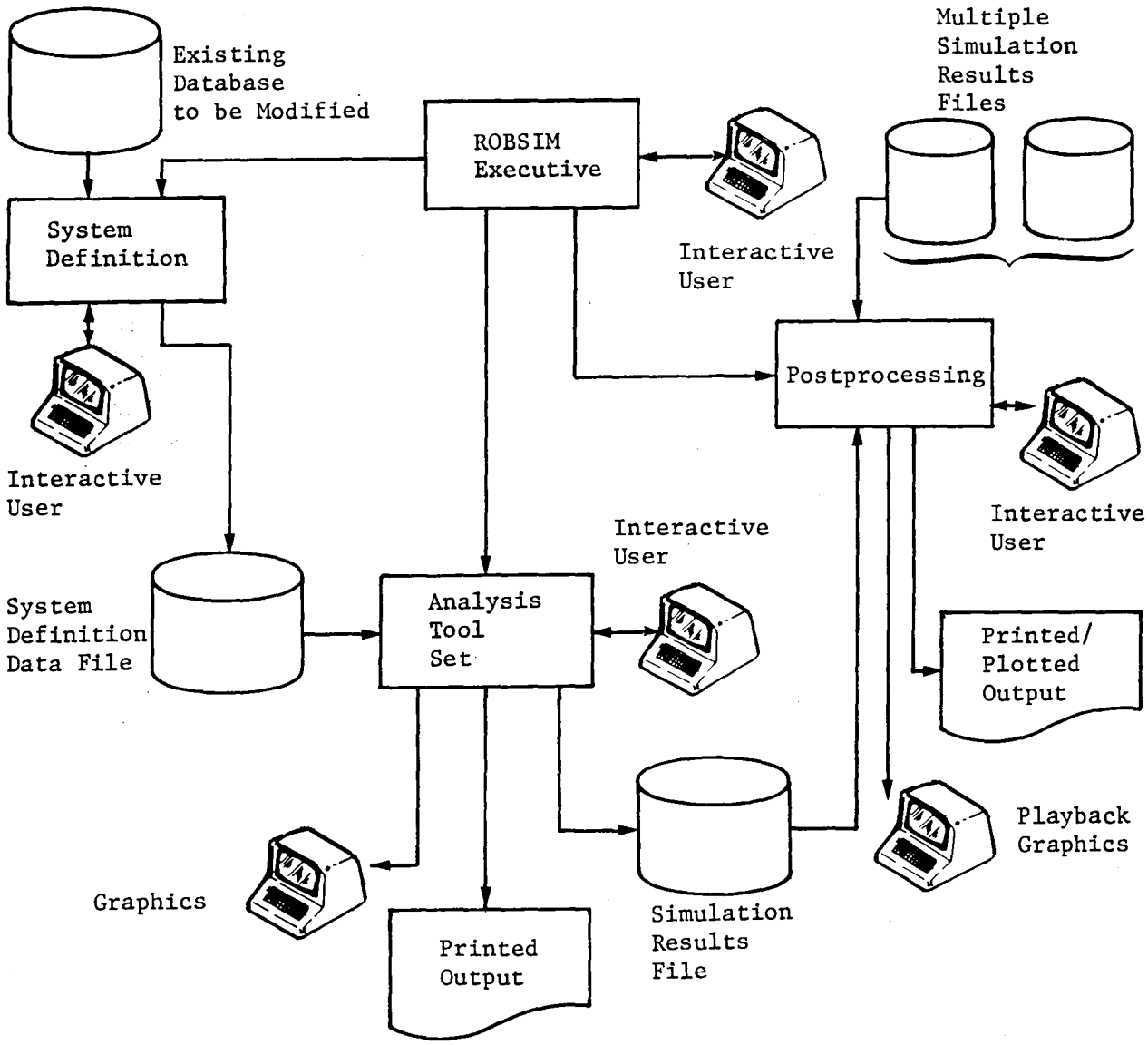


Figure 3-1 ROBSIM Top-Level Structure

3.1 SYSTEM DEFINITION

The system definition portion of the program operates in an interactive mode and allows the user to define a robotic configuration for analysis. This definition includes specification of the manipulator geometry and mass properties, environmental parameters, and graphics representations for the manipulator and the environment. The types of joint configurations that may be specified include hinge (Fig. 3-2), swivel (Fig. 3-3), and sliding (Fig. 3-4). With these three basic joint types, it is possible to specify a large variety of manipulator arm configurations.

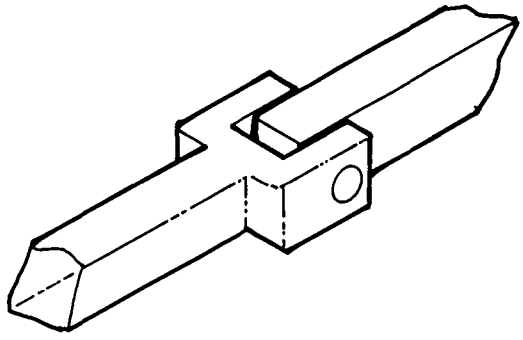


Figure 3-2 Typical Hinge Joint

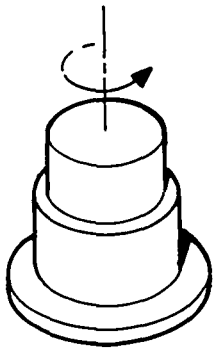


Figure 3-3 Typical Swivel Joint

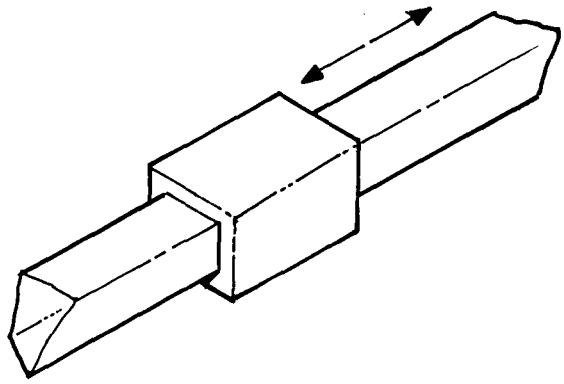


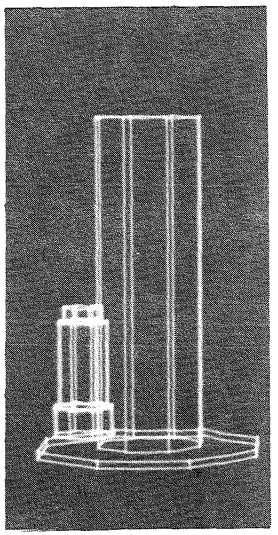
Figure 3-4 Typical Sliding Joint

Along with the type of joint, the operator must specify the orientation of the joint with respect to the individual manipulator links. The links are defined by their physical dimensions and mass properties. The mass properties include the total mass, the location of the center of gravity (cg), and the inertia matrix. In the current implementation of ROBSIM, joint mass properties are not separated from the link mass properties but may be accounted for by considering the joint to be a part of its associated link. For example, if joint 2 of a manipulator had a mass of 4 kg and link 2 a mass of 10 kg, then the joint mass could be accounted for by inputting a link mass of 14 kg. The cg location would be the weighted average of the joint and link cgs, and the inertia matrix would be the sum of the joint and link inertia matrices taken from the same coordinate system (see Appendix B for a discussion of transforming inertia matrices).

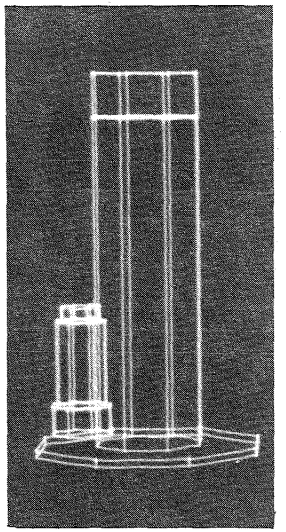
A graphics display package was designed and is connected to the system definition module so that the operator can observe the system as it is being defined. This allows the operator to gain a physical understanding of the system and helps to identify potential problem areas early in the development process. If the operator is not satisfied with the defined system, any of the input parameters may be interactively modified. Figures 3-5a through 3-5e illustrate the graphics display during the definition of a PUMA 600 manipulator arm.

The level of detail by which the manipulator is represented can vary as a function of the phase of design. For example, during initial concept definition, the detailed physical dimension of the links is unknown, and approximations must be used. The system definition program contains an option whereby any link can be defined by simple cylindrical elements. The diameter and length of the cylindrical elements are defined by the operator. Figure 3-6a illustrates a simplified graphic display of the Martin Marietta SMA manipulator shown in Figure 3-6b.

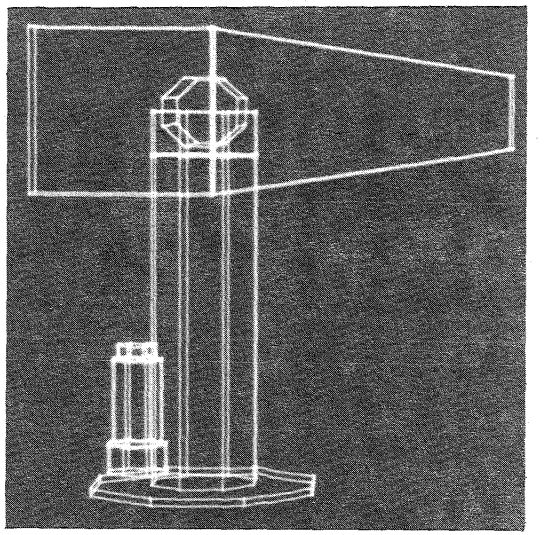
The output of the system definition package is a file that contains a description of the robotic system that can be used as input by the various analysis tools. This file may be archived so that a number of different analyses may be performed on the same manipulator configuration.



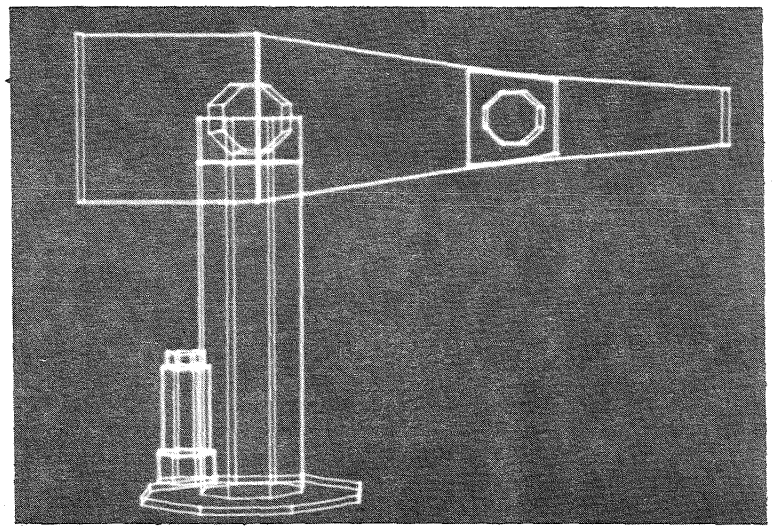
a Base



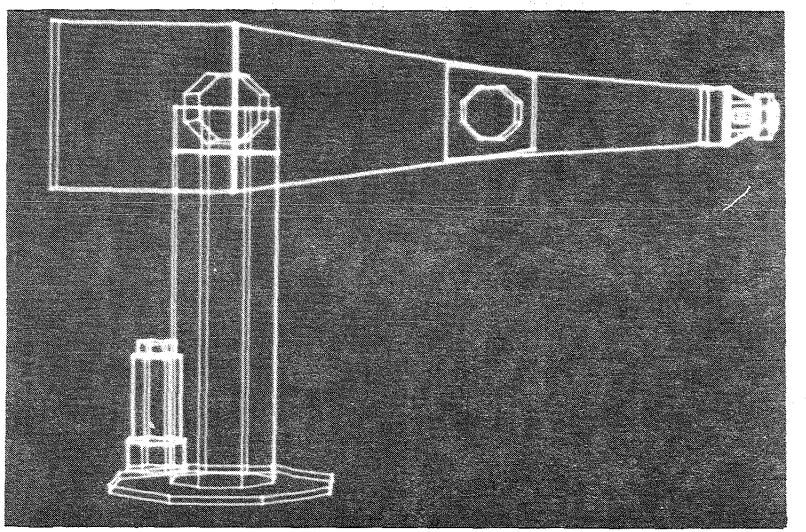
b Shoulder Yaw



c Shoulder Pitch

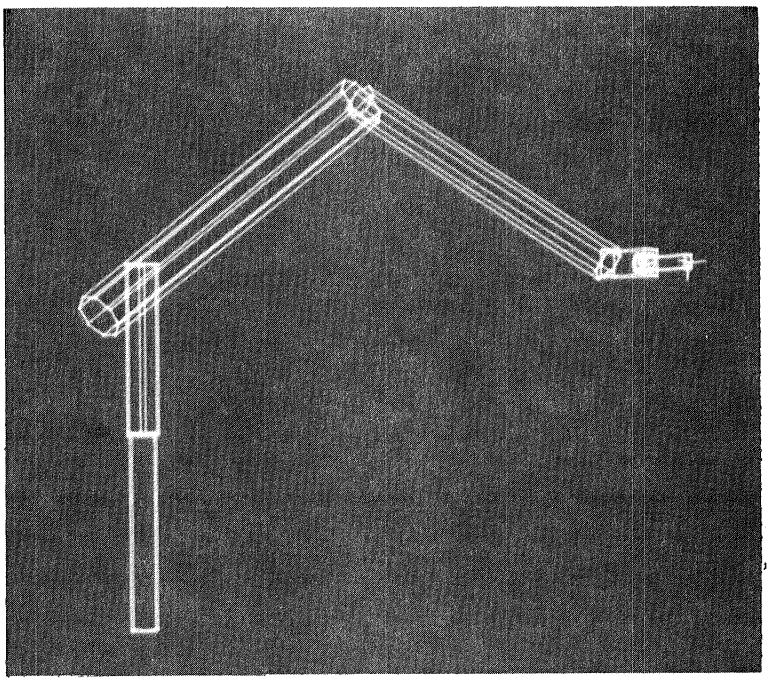


d Forearm

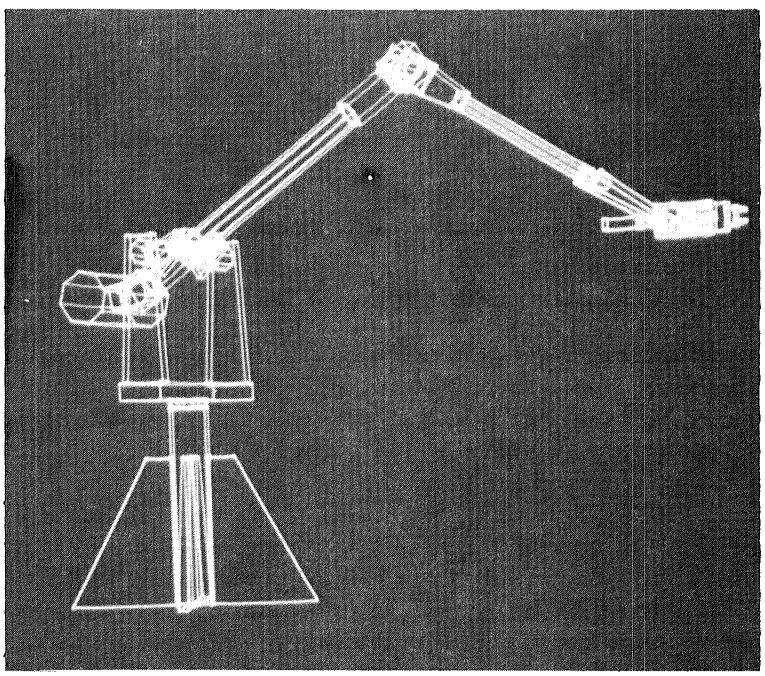


e Complete Manipulator

Figure 3-5 Graphics Representation of PUMA - 600 Manipulator



a Simple Representation



b Detailed Representation

Figure 3-6
Simplified and Detailed Graphics Representations

3.2 ANALYSIS TOOL SET

The analysis tool set has been separated (Fig. 3-7) into a library of tools that may be used for a variety of studies on the manipulator arm defined during system definition. Among the tools that will be in the library are a force/torque requirements analysis tool, a time-domain dynamic simulation, a frequency-domain simulation, and an operating-envelope analysis tool.

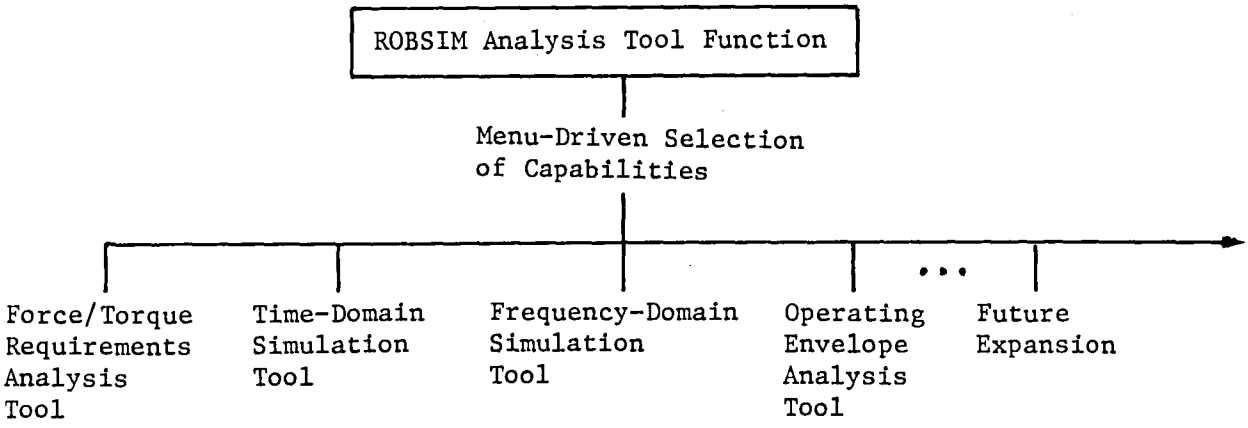


Figure 3-7 ROBSIM Analysis Tool Structure

The operating-envelope analysis tool will allow an operator to observe the operating envelope of the manipulator through the graphics display. This will allow the operator to efficiently determine whether the manipulator configuration will be able to perform the necessary maneuvers in a restricted environment. For example, for spacecraft refurbishment, a manipulator arm must be able to reach through an access panel, grab a replaceable component, and extract the component without colliding with any part of the spacecraft. Joint configuration is an important parameter in providing the necessary agility for a specific mission. With the envelope-analysis tool, it will be possible to observe the entire operating envelope for individual joints and combinations of joints up to the end effector.

The frequency-domain simulation of a robotic system will enable an analysis of the stability of the arm about specific operating positions and will help in developing the control system. This tool will also enable a careful evaluation of singularity positions and their effects on the controllability of the system. The output from this tool will include frequency-domain plots of the arm.

During this phase of the contract, the major portion of the force/torque requirements-analysis tool and the framework for the time-domain simulation were developed. These two tools are described in more detail in the following sections.

3.2.1 Force/Torque Requirements-Analysis Tool

The force/torque requirements-analysis tool is intended to allow a designer to quickly identify what the operational force and torque requirements for a specific system design will be. After identifying an arm configuration, the system will be tasked to perform a maneuver. The requirements tool will then calculate the necessary forces and torques required to produce the maneuver and the resultant forces and torques acting on the joints and links. A mechanical designer may efficiently use this information to specify motor capabilities and material requirements for the joints and links, and to identify the effect of the motion on the control of the hosting body (i.e., spacecraft). A graphics package has been provided so that the operator may observe the motion of the arm during the analysis. All data is written to disk file to allow later post processing to obtain plots of any of the forces and torques that have occurred. Figure 3-8 illustrates the functional flow of the force/torque requirements-analysis tool.

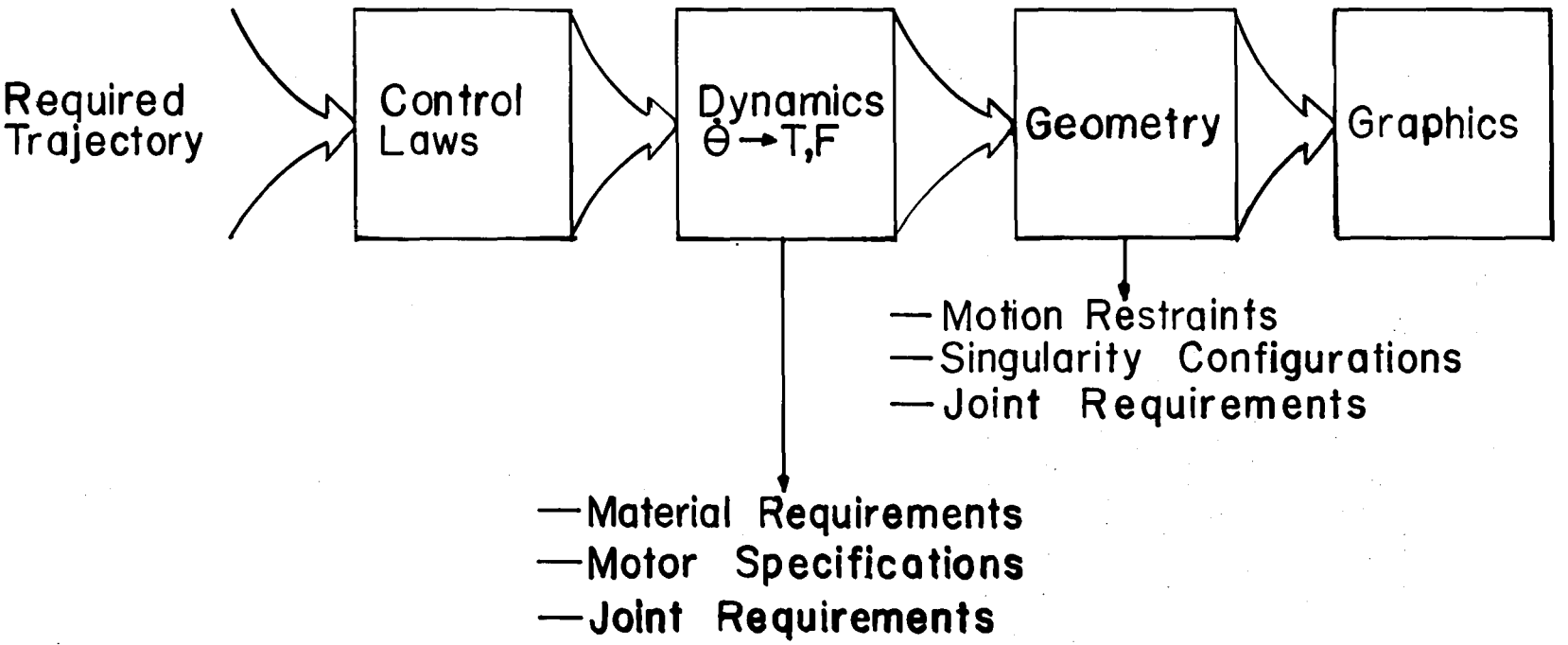


Figure 3-8 Functional Flow of Requirements Analysis Tool

In the current implementation of ROBSIM, a sequence of rate profiles describes the manipulator trajectory. These rate profiles are second-order polynomial functions that describe either individual joint rates or end-effector rates, depending on user selection. Start and stop times for the profiles are also user options.

If the user requests individual joint rate control of the model, then for an N-joint arm, the user must input N rate profiles. The rate profiles in this case describe the rate of displacement of each joint from reference positions. If the user requests end-effector rate control, then six rate profiles must be input. Each of the profiles determines one component of the complete end-effector angular and linear velocity vectors. Six profiles are necessary--three for the linear velocities along the x, y, and z axes, and three for the angular velocities around the x, y, and z axes. From the end-effector profiles, the rates and displacements of each joint at any given time can be calculated. The equation relating joint rates to end-effector velocity is

$$\begin{bmatrix} v \\ \cdots \\ w \end{bmatrix} = J \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \vdots \\ \Omega_N \end{bmatrix}$$

where

v = Linear velocity vector of the end effector;
w = Angular velocity vector of the end effector;
Ω_i = Rate of displacement of joint i;
J = 6 x N Jacobian matrix.

Given J and the desired end-effector velocities, the required joint rates can be found by inverting J:

$$\begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \vdots \\ \Omega_N \end{bmatrix} = J^{-1} \begin{bmatrix} v \\ \cdots \\ w \end{bmatrix}$$

Appendix D describes the derivation of the end-effector rate control equations and discusses a technique that can be used to find an approximate solution for that case in which J is singular.

Any motion can be described as a sequence of short, simple motions. Thus, any manipulator motion can be specified by inputting a sequence of rate profiles. ROBSIM currently permits the user to define a sequence of up to 20 different rate profiles.

Equations 3-1 to 3-6 can be used to calculate the forces and torques acting on a system in motion. The derivation of these equations is contained in Appendix B.

$$\underline{w}_i = \underline{w}_{i-1} + \underline{\Omega}_i \tag{3-1}$$

$$\underline{s}_i = \underline{s}_{i-1} + \tilde{\underline{w}}_{i-1}\underline{h}_{i-1,i} + \dot{\underline{h}}_{i-1,i}\underline{\mu}_{h_{i-1,i}} \tag{3-2}$$

$$\underline{\alpha}_i = \underline{\alpha}_{i-1} + \dot{\underline{\Omega}}_i\underline{\mu}_{\Omega_i} + \tilde{\underline{w}}_{i-1}\underline{\Omega}_i \tag{3-3}$$

$$\underline{a}_i = \underline{a}_{i-1} + \tilde{\underline{\alpha}}_{i-1}\underline{h}_{i-1,i} + 2\tilde{\underline{w}}_{i-1}(\dot{\underline{h}}_{i-1,i}\underline{\mu}_{h_{i-1,i}}) + \tilde{\underline{w}}_{i-1}(\tilde{\underline{w}}_{i-1}\underline{h}_{i-1,i}) + \ddot{\underline{h}}_{i-1,i}\underline{\mu}_{h_{i-1,i}} \tag{3-4}$$

$$\underline{f}_i = \underline{f}_{i+1} + m_i[\underline{a}_i + \tilde{\underline{\alpha}}_i\underline{h}_{i,i} - \underline{g}] \tag{3-5}$$

$$\underline{t}_i = \underline{t}_{i+1} + \tilde{\underline{h}}_{i,i+1}\underline{f}_{i+1} + I_i\underline{\alpha}_i + \tilde{\underline{h}}_{i,i}m_i[\underline{a}_i + \tilde{\underline{\alpha}}_i\underline{h}_{i,i} - \underline{g}] + \tilde{\underline{w}}_i I_i\underline{w}_i \tag{3-6}$$

where

$\overline{X}$ - a coordinate system located at joint i;

$\underline{s}_i$ - the linear velocity of $\overline{X}_i$ with respect to an inertial coordinate system;

$\underline{w}_i$ - the angular velocity of $\overline{X}_i$ with respect to an inertial coordinate system;

$\underline{\Omega}_i$ - the angular velocity of $\overline{X}_i$ with respect to $\overline{X}_{i-1}$;

$\underline{\alpha}_i = \dot{\underline{w}}_i$ - the angular acceleration of $\overline{X}_i$ with respect to an inertial coordinate system;

$\underline{a}_i = \dot{\underline{s}}_i$ - the linear acceleration of $\overline{X}_i$ with respect to an inertial coordinate system;

$\underline{h}_{ij}$ - { $i \neq j$, $\underline{h}_{ij}$ is the vector from $\overline{X}_i$ to $\overline{X}_j$; $i=j$, $\underline{h}_{ii}$ is the vector from $\overline{X}_i$ to the cg of link i;

I_i - the inertia matrix of link i with respect to the cg of link i;

m - the mass of link i;

$\underline{f}_i$ - the reaction force acting at $\overline{X}_i$;

$\underline{t}_i$ - the reaction torque acting at $\overline{X}_i$;

$\underline{g}$ - gravitational acceleration;

The use of a tilde "~" over a vector has been used to denote the 3x3 skew-symmetric matrix of a vector cross product. That is, if

$$\underline{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad \text{and} \quad \underline{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

then

$$\tilde{\underline{a}}\,\underline{b} = \underline{a} \times \underline{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

The inputs to the equations are the quantities that describe the motion of manipulator relative joint displacements, velocities, and accelerations. The equations are used to recursively calculate the motion of each joint with respect to an inertial coordinate system and then the forces and torques acting at each joint. The inputs to the dynamics routine are obtained by averaging the values of joint displacement, velocity, and acceleration provided by the control routine over a single time step. Any error due to averaging is minimal if time steps are small.

The dynamics routine has the capability of outputing either parameter plots or printed output. If plots are requested, data representing the dynamic reactions at each joint at each time step are stored in a plot file. The user may then select the parameters to be plotted. Figures 3-9 and 3-10 are examples of the types of plots that can be requested. Note that the parameters can be expressed in different coordinate systems, and several parameters may be included in the same plot.

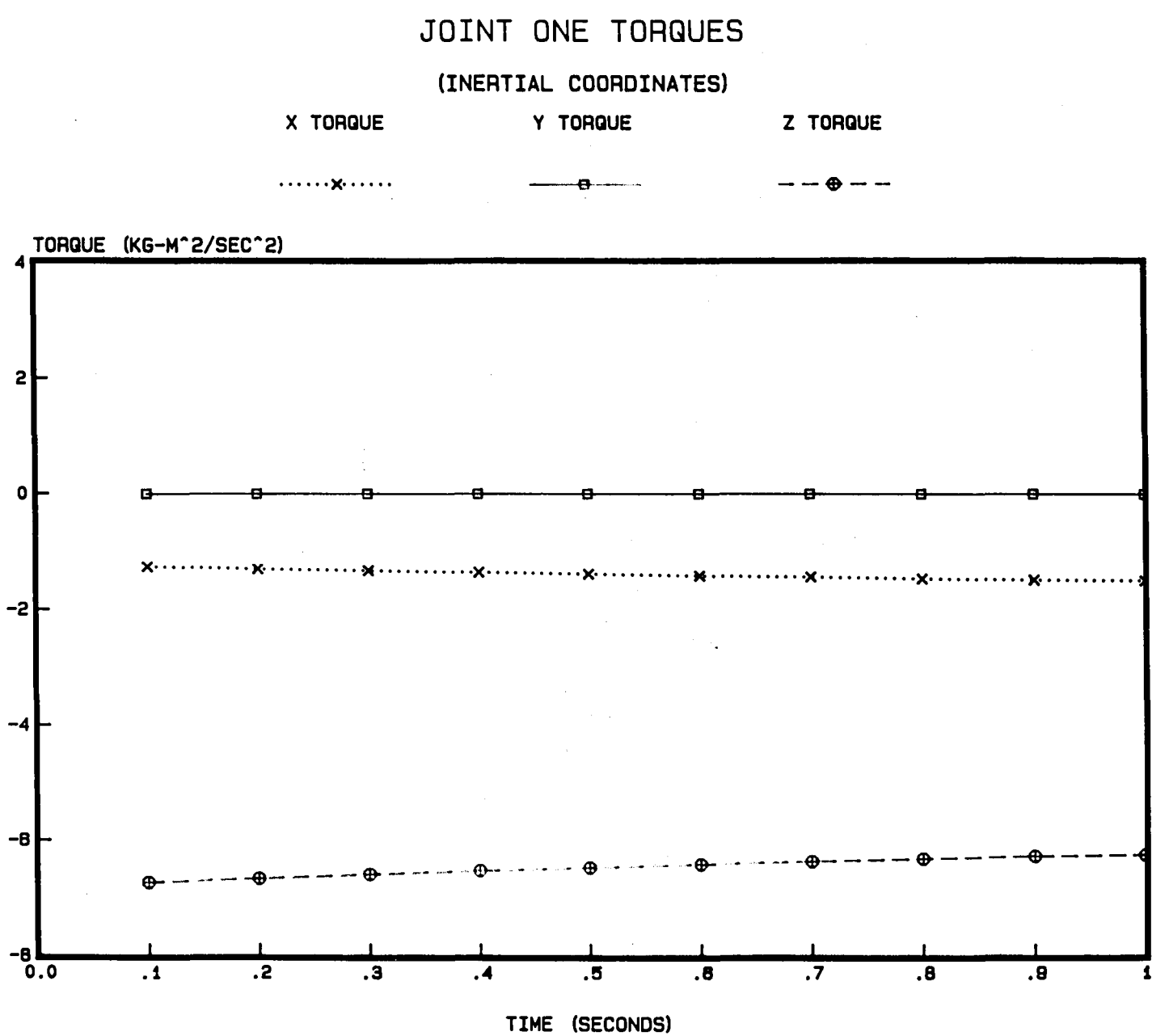


Figure 3-9 Joint Torque Plots in Inertial Coordinate System

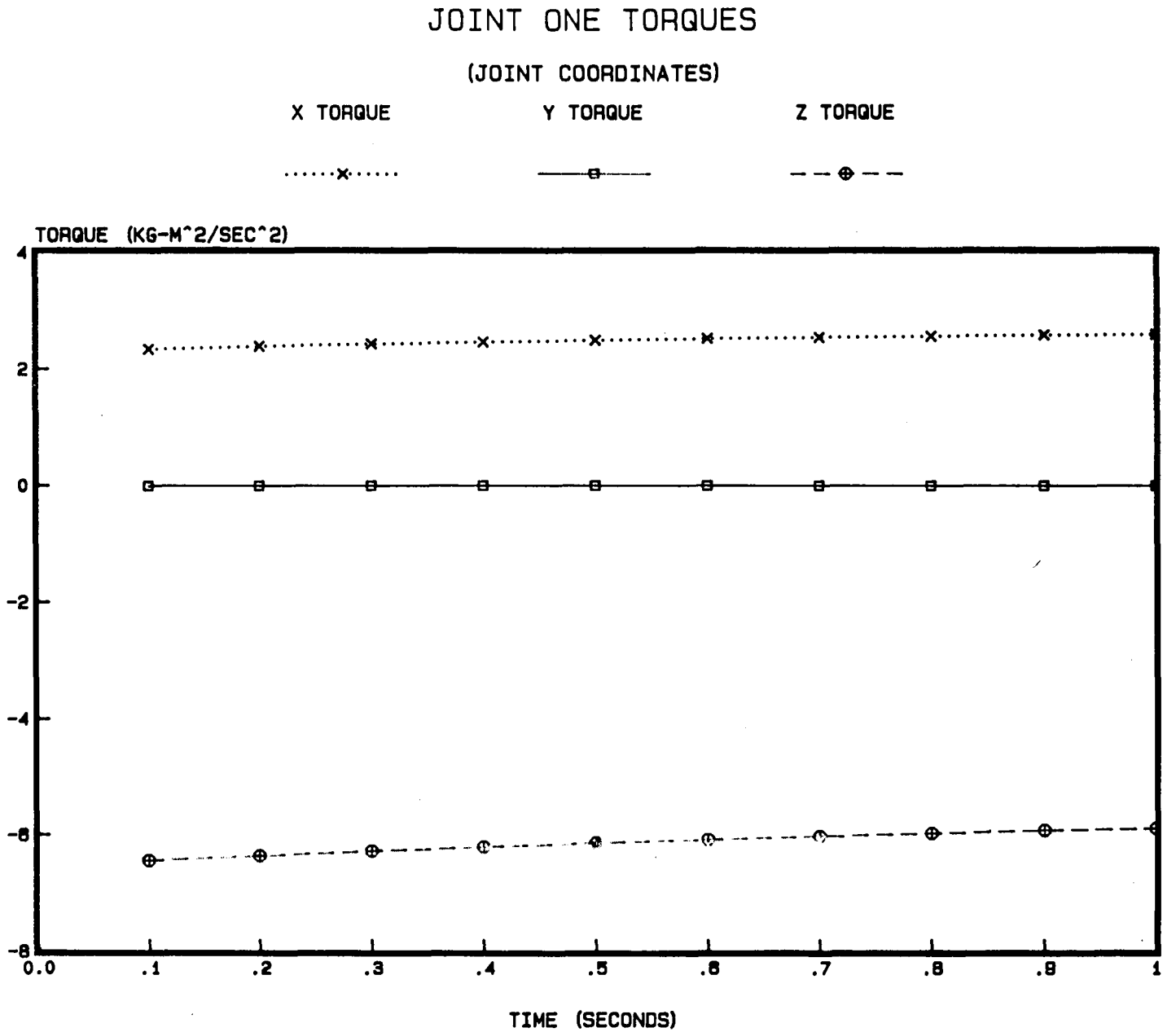


Figure 3-10 Joint Torque Plots in Joint Coordinate System

If printed output of the dynamic analysis is requested, the data describing the system initial state are immediately written to a print file. The initial state data for dynamics include a description of joint configuration, component dimensions and mass properties, and the control trajectory. At each time step specified, the average values of joint displacement, velocity, and acceleration are written to the print file, as are the data describing the forces and torques acting at each joint. Figure 3-11 is an example of the initial state output from the dynamics routine. These data were also used to generate the plots shown in Figures 3-9 and 3-10. Figure 3-12 is an example of the output at each time step during dynamic analysis.

Because force and torque are vectors, each quantity consists of three scalars. Each scalar represents the component of force (or torque) along a coordinate axis. Thus, six scalars are output at each time step for the dynamic reactions at each joint. If the joint has n degrees of freedom ($n \leq 6$), then n of the scalars represent actuator forces (or torques) that must be applied to drive the system. The remaining 6-n scalars represent the dynamic reactions that support the motion; these terms define the structural requirements at each joint.

The force/torque requirements-analysis tool also contains a graphics interface that can be used to aid in a kinematic analysis of the manipulator. The graphics display depicts the position of the arm during a predefined motion. This enables an operator to analyze the effect of joint constraints on the overall motion of the arm and to quickly identify singularity points in the arm configuration.

The graphics package also allows the local environment around the manipulator to be represented by simple geometric solids. Figures 3-13 and 3-14 illustrate the Martin Marietta SOS arm in two different environments. Figure 3-14 depicts the arm in a large space system assembly function, which recreates a physical simulation previously performed with the actual arm.

Kinematic analysis of a design concept can be performed by moving the model within the workspace and by using the output capabilities of ROBSIM. Questions concerning manipulator reach and dexterity can be answered and the model arm modified until the kinematic requirements can be met.

```
                                DYNAMICS TEST

NUMBER OF JOINTS IN MANIPULATOR = 1

SIMULATION STARTS AT      T = 0.00
SIMULATION STOPS AT       T =    3.00000
SIMULATION STEP SIZE IS DT =     0.50000


JOINT/LINK INITIAL VALUES:

      JOINT 1 IS A HINGE JOINT

           LINK DIMENSIONS (X,Y,Z):          60.00000      0.00000      0.00000   INCHES

                                              1.52400      0.00000      0.00000   METERS

           LOCATION OF LINK C.O.G. (X,Y,Z):  30.00000      0.00000      0.00000   INCHES

                                              0.76200      0.00000      0.00000   METERS

           MASS OF LINK =                    10.00000

           THE LINK INERTIA MATRIX IS:    20.00000       0.00000       0.00000
                                           0.00000   12010.00000       0.00000
                                           0.00000       0.00000   12010.00000
```

Figure 3-11 Sample Dynamics Output, Initial Condition

```
JOINT/LINK 1

 ANGULAR DISPLACEMENT =     0.52360 RADIANS
                           30.00000 DEGREES

 INERTIAL LOCATION (X,Y,Z) =   0.00000        0.00000        0.00000      INCHES

                               0.00000        0.00000        0.00000      METERS

 CURRENT TRANS. MATRIX =       0.00000        1.00000        0.00000
                              -0.86603        0.00000       -0.50000
                              -0.50000        0.00000        0.86603
```

	JOINT COORDINATES			INERTIAL COORDINATES		
	X	Y	Z	X	Y	Z
LINK VECTOR (INCHES)	60.00000	0.00000	0.00000	0.00000	-51.96152	-30.00000
(METERS)	1.52400	0.00000	0.00000	0.00000	-1.31982	-0.76200
REL. ANG. VEL. (R/SEC)	0.00000	0.52360	0.00000	0.52360	0.00000	0.00000
REL. ANG. ACC. (R/SEC↑2)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
TOTAL ANG. VEL. (R/SEC)				0.52360	0.00000	0.00000
TOTAL ANG. ACC. (R/SEC↑2)				0.00000	0.00000	0.00000
TOTAL LIN. ACC. (I/SEC↑2)				0.00000	0.00000	0.00000
(M/SEC↑2)				0.00000	0.00000	0.00000
REACTION FORCE (KG-I/SEC↑2)	-82.24670	0.00000	0.00000	0.00000	71.22774	41.12336
(KG-M/SEC↑2)	-2.08907	0.00000	0.00000	0.00000	1.80918	1.04453
REACTION TORQUE (KG-I↑2/SEC↑2)	0.00000	-0.00012	0.00000	-0.00012	0.00000	0.00000
(KG-M↑2/SEC↑2)	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
INERTIA MATRIX	20.00000	0.00000	0.00000	12010.00000	-0.00039	-0.00023
	0.00000	12010.00000	0.00000	-0.00039	3017.50000	-5191.82227
	0.00000	0.00000	12010.00000	-0.00023	-5191.82227	9012.50000
* * *	* * *	*	*	* * *	*	*

```
 MANIP. TOOL LOCATION (INERTIAL COORDINATES):                          0.00000      -51.96152      -30.00000
```

```
      *       *       *       *       *       *       *       *       *       *       *       *
      *       *       *       *       *       *       *       *       *       *       *       *
```

Figure 3-12 Sample Dynamics Output

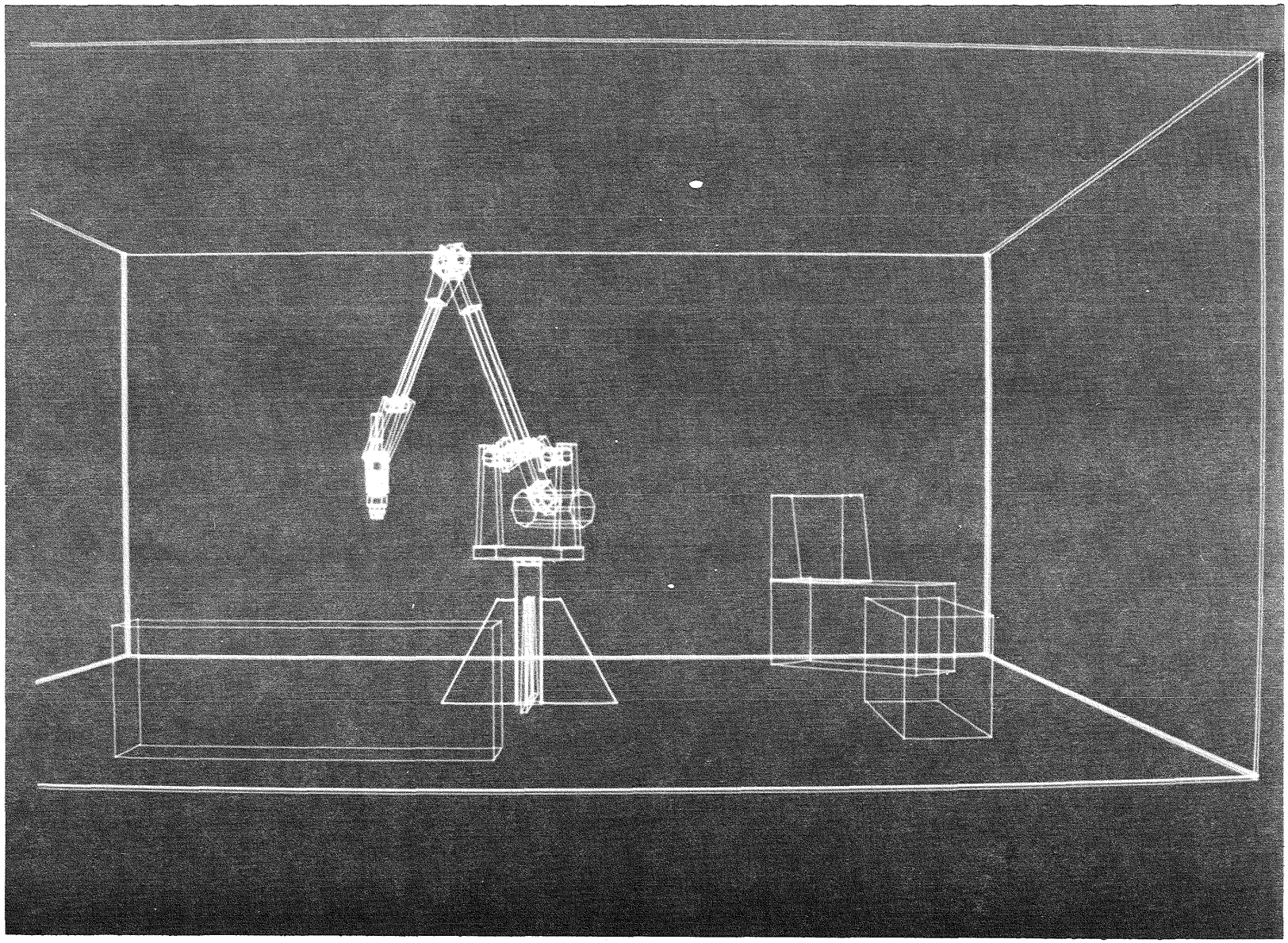


Figure 3-13 The Martin Marietta SOS Manipulator Depicted in a Laboratory Environment

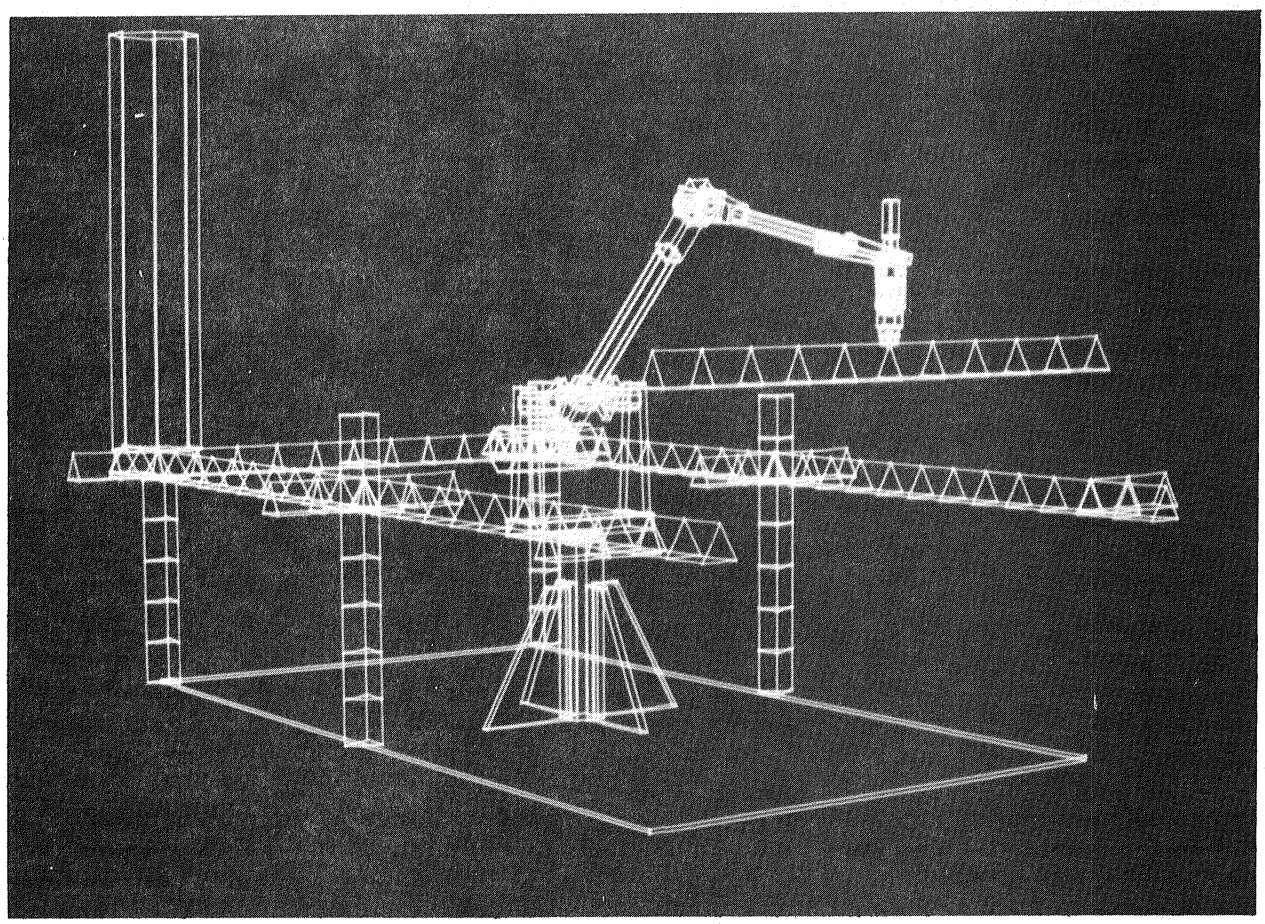


Figure 3-14

Martin Marietta SOS Manipulator in Space Structure Assembly Environment

3.2.2 Time-Domain Simulation Tool

The simulation tool is intended to ultimately provide a realistic computer simulation of a manipulator composed of actual or proposed components. Using models of manipulator components, such as amplifiers, motors, power trains, etc, along with a complete system dynamics formulation, the simulation tool will provide a realistic testbed for control technique studies, incorporation of artificial intelligence technology within manipulator capabilities, and general study of manipulator system and component performance. Within the current development phase, the simulation tool structure has been defined and implementation begun. The simulation tool has been designed using closed-loop methods and state variable formulation. Appendix E is a discussion of closed-loop versus open-loop methods as well as a justification for using the state-variable approach. Figure 3-15 presents a block diagram of the closed-loop method. The work performed within the current development phase has been restricted to developing a single joint model using the state-variable formulation. Placed within the context of Figure 3-15, all current work has been within the a priori block. The adaptive methods will be addressed in future expansion of the simulation tool capabilities.

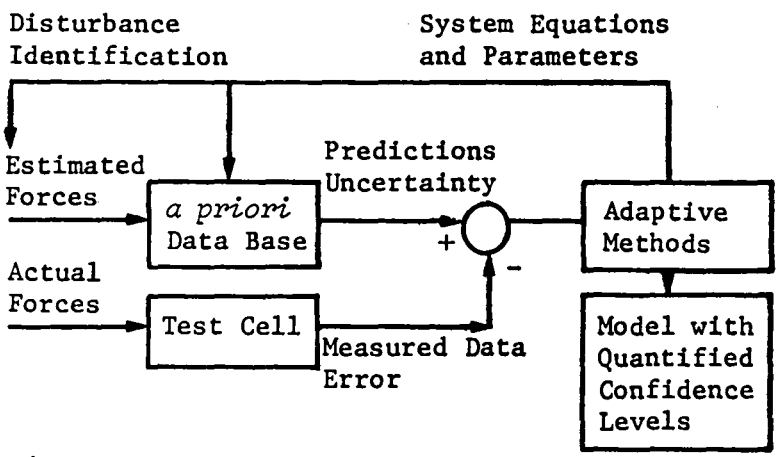


Figure 3-15 Closed-loop Simulation Tool Structure

It should be noted that the work performed under the current contract used a linear state variable approach to simulation modeling. The software structure of modular components with well-defined interfaces is not limited to a linear state variable application. A non-linear state variable approach (or any modeling approach) can be used. The linear state variable approach taken allowed easy modeling of the single joint system and easy application of the Kalman filter to that system.

Implementing simulation tool capabilities within the current development phase has consisted of designing, coding, and testing a computer model of a single manipulator joint using the state-variable formulation. Figure 3-16 shows the basic joint model design. The computer model design is modular with well-defined interfaces between

blocks to allow flexibility in defining the configuration of components. Each block within the model uses the state-variable formulation. Figure 3-17 shows a typical component block diagram illustrating the state-variable approach. The basic equations in the state variable formulation can be written as:

$$\underline{x}(k+1) = \phi(k+1,k)\underline{x}(k) + \theta(k+1,k)\underline{u}(k) + \underline{w}(k) \quad (3-7)$$

$$\underline{y}(k+1) = C(k+1)\underline{x}(k+1) \quad (3-8)$$

$$\underline{z}(k+1) = H(k+1)\underline{y}(k+1) + \underline{v}(k) \quad (3-9)$$

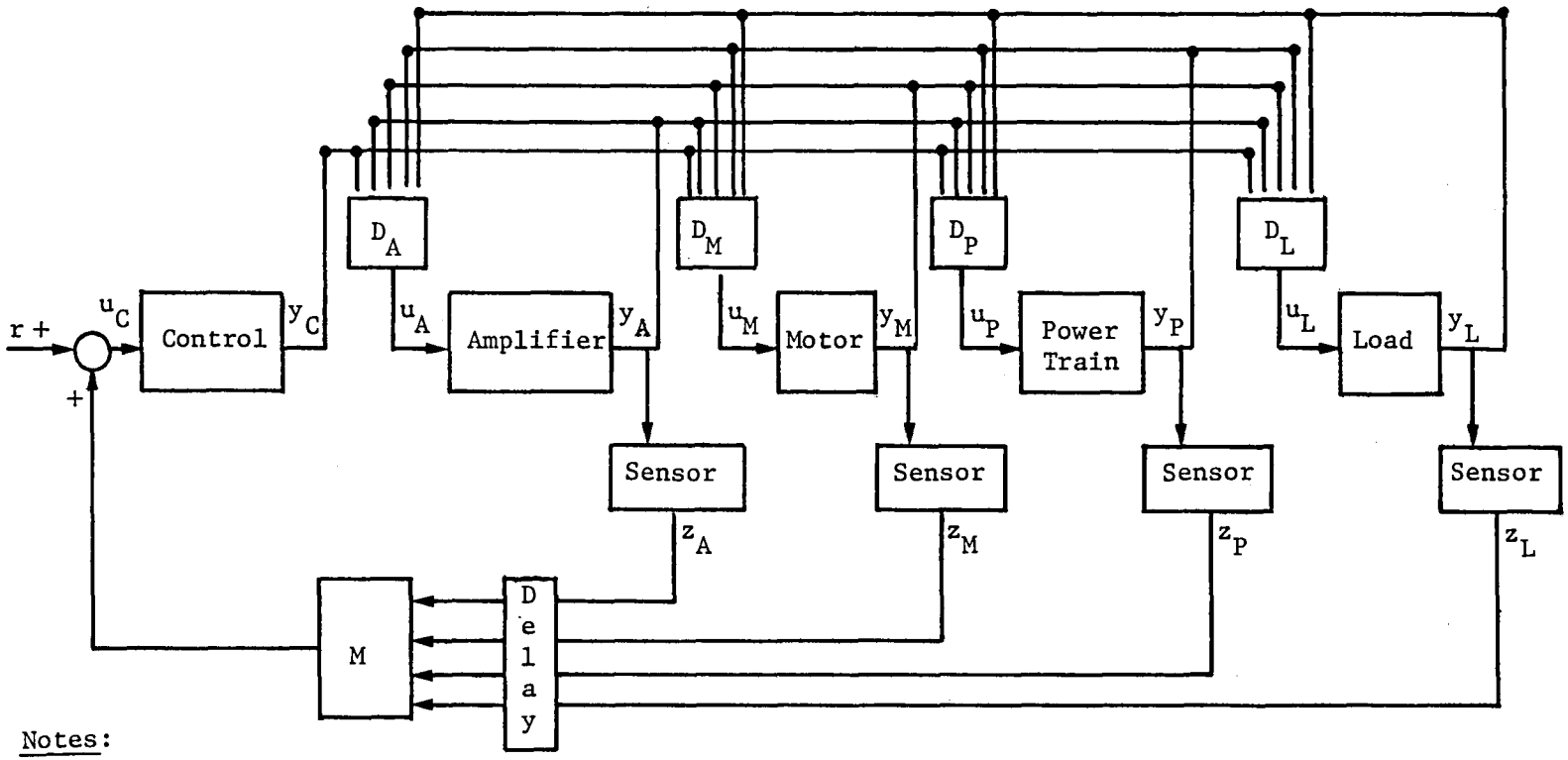
where

u = control array
x = state variable array
y = observable array
z = sensor output array
w = process noise array
v = sensor noise array
ϕ = dynamics matrix
θ = control matrix
C = observability matrix
H = measurement matrix

The equations that define the operation of the component must be placed in state variable form, i.e., Equation [3-7]. This process defines the control array, u, and the state-variable array, x, as well as the required ϕ and θ matrices. The observability matrix, C, must be defined in a manner that will convert the state variable array, x, to the actual component output array, y. Sensor modeling is handled by defining an appropriate H matrix to convert the component observable output array, y, to the sensor output array, z, as shown in Equation [3-9]. Figure 3-16 shows that the control array, u, is constructed from system observable data through the prespecified component D matrix. The equation representing this procedure is

$$\underline{u}(k) = D\underline{y}(k) \quad (3-10)$$

Figure 3-16 also shows that sensor data are combined and modified appropriately by the M matrix to compare sensor data with the prespecified reference signal, r. The error resulting from this comparison is the input to the control box. The current computer model handles the computations for each component exactly, as depicted in Figure 3-16 using Equations [3-7] through [3-10].



Notes:

1. All blocks are in state-variable formulation.

2. Configuration is flexible, i.e., it has unlimited block linkage.

Figure 3-16 Joint Model

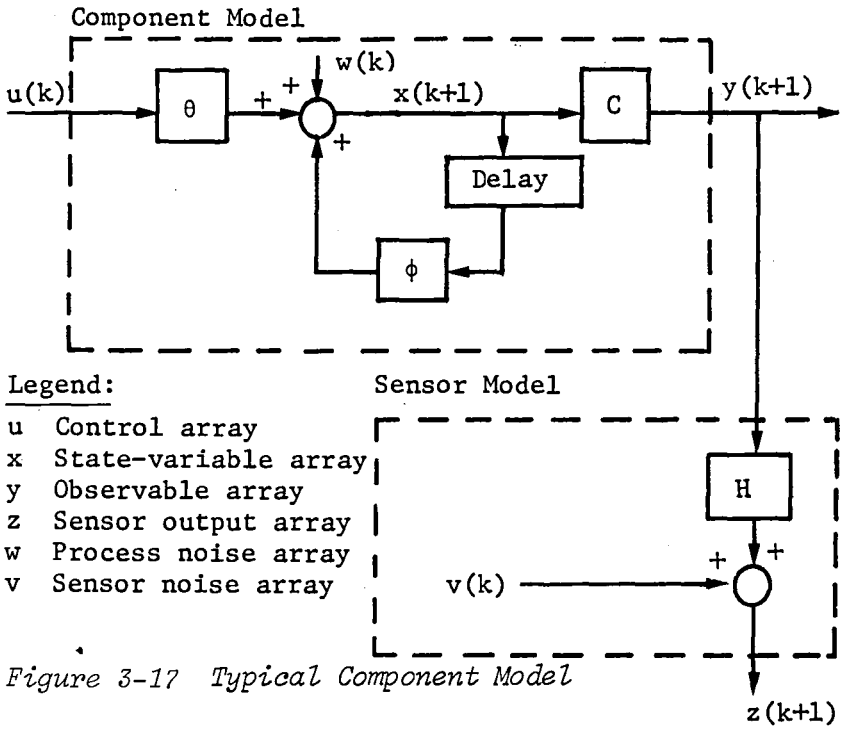


Figure 3-17 Typical Component Model

Also, a Kalman filter for the entire joint model has been implemented. The Kalman filter formulation requires the development of the system of equations representing the entire joint model. Using Equations [3-7] through [3-10], the set of equations required to represent the entire joint model can be written as

$$
\begin{aligned}
x_C(k+1) &= \phi_C(k+1,k)x_C(k) + \theta_C(k+1,k)[r(k) + M\underline{z}(k)] + w_C(k) \\
x_A(k+1) &= \phi_A(k+1,k)x_A(k) + \theta_A(k+1,k)u_A(k) + w_A(k) \\
x_M(k+1) &= \phi_M(k+1,k)x_M(k) + \theta_M(k+1,k)u_M(k) + w_A(k) \\
x_P(k+1) &= \phi_P(k+1,k)x_P(k) + \theta_P(k+1,k)u_P(k) + w_P(k) \\
x_L(k+1) &= \phi_L(k+1,k)x_L(k) + \theta_L(k+1,k)u_L(k) + w_L(k)
\end{aligned}
\tag{3-11}
$$

$$
\begin{aligned}
y_C(k+1) &= C_C(k+1)x_C(k+1) \\
y_A(k+1) &= C_A(k+1)x_A(k+1) \\
y_M(k+1) &= C_M(k+1)x_M(k+1) \\
y_P(k+1) &= C_P(k+1)x_P(k+1) \\
y_L(k+1) &= C_L(k+1)x_L(k+1)
\end{aligned}
\tag{3-12}
$$

$$
\begin{aligned}
z_A(k+1) &= H_A(k+1)y_A(k+1) + v_A(k) \\
z_M(k+1) &= H_M(k+1)y_M(k+1) + v_M(k) \\
z_P(k+1) &= H_P(k+1)y_P(k+1) + v_P(k) \\
z_L(k+1) &= H_L(k+1)y_L(k+1) + v_L(k)
\end{aligned}
\tag{3-13}
$$

$$
\begin{aligned}
u_A(k) &= D_A\,\underline{y}(k) \\
u_M(k) &= D_M\,\underline{y}(k) \\
u_P(k) &= D_P\,\underline{y}(k) \\
u_L(k) &= D_L\,\underline{y}(k)
\end{aligned}
\tag{3-14}
$$

In matrix form, Equations [3-11] through [3-14] can be written

$$\begin{bmatrix} x_C \\ x_A \\ x_M \\ x_P \\ x_L \end{bmatrix} = \begin{bmatrix} \phi_C & & & & \\ & \phi_A & & 0 & \\ & & \phi_M & & \\ & 0 & & \phi_P & \\ & & & & \phi_L \end{bmatrix} \begin{bmatrix} x_C \\ x_A \\ x_M \\ x_P \\ x_L \end{bmatrix}_k + \begin{bmatrix} \theta_C & & & & \\ & \theta_A & & 0 & \\ & & \theta_M & & \\ & 0 & & \theta_P & \\ & & & & \theta_L \end{bmatrix} \begin{bmatrix} (r+Mz) \\ u_A \\ u_M \\ u_L \\ u_P \end{bmatrix} + \begin{bmatrix} w_C \\ w_A \\ w_M \\ w_P \\ w_L \end{bmatrix} \tag{3-15}$$

$$\begin{bmatrix} y_C \\ y_A \\ y_M \\ y_P \\ y_L \end{bmatrix} = \begin{bmatrix} C_C & & & & \\ & C_A & & 0 & \\ & & C_M & & \\ & 0 & & C_P & \\ & & & & C_L \end{bmatrix} \begin{bmatrix} x_C \\ x_A \\ x_M \\ x_P \\ x_L \end{bmatrix} \tag{3-16}$$

$$\begin{bmatrix} z_A \\ z_M \\ z_P \\ z_L \end{bmatrix} = \left[\begin{array}{c|ccc} & H_A & & 0 \\ 0 & & H_M & \\ & & & H_P \\ & 0 & & \end{array}\right] \begin{bmatrix} y_C \\ y_A \\ y_M \\ y_P \\ y_L \end{bmatrix} + \begin{bmatrix} v_A \\ v_M \\ v_P \\ v_L \end{bmatrix} \tag{3-17}$$

$$\begin{bmatrix} u_A \\ u_M \\ u_P \\ u_L \end{bmatrix} = \left[\begin{array}{ccc} & D_A & \\ \hline & D_M & \\ \hline & D_P & \\ \hline & D_L & \end{array}\right] \begin{bmatrix} y_C \\ y_A \\ y_M \\ y_P \\ y_L \end{bmatrix} \tag{3-18}$$

Equation [3-15] can be rewritten as

$$\begin{bmatrix} x_C \\ x_A \\ x_M \\ x_P \\ x_L \end{bmatrix}_{k+1} = \begin{bmatrix} \phi_C & & & & \\ & \phi_A & & & \\ & & \phi_M & & \\ & & & \phi_P & \\ & & & & \phi_L \end{bmatrix} \begin{bmatrix} x_C \\ x_A \\ x_M \\ x_P \\ x_L \end{bmatrix}_k + \left[\begin{array}{ccccc} & & 0 & & \\ \hline \theta_A & & & & \\ & & \theta_M & & \\ & & & \theta_P & \\ & & & & \theta_L \end{array}\right] \begin{bmatrix} u_A \\ u_M \\ u_P \\ u_L \end{bmatrix} + \left[\begin{array}{cc} \theta_C & \theta_C^M \\ \hline & \\ 0 & 0 \\ & \end{array}\right] \begin{bmatrix} r \\ z \end{bmatrix} + \begin{bmatrix} w_C \\ w_A \\ w_M \\ w_P \\ w_L \end{bmatrix} \tag{3-19}$$

Examining only the second term in the right-hand side of Equation [3-19] and using Equations [3-16] and [3-18], the following equation can be written

$$\begin{bmatrix} \theta_A & & & & 0 \\ & \theta_M & & & \\ & & \theta_P & & \\ 0 & & & & \theta_L \end{bmatrix} \begin{bmatrix} u_A \\ u_M \\ u_P \\ u_L \end{bmatrix} = \begin{bmatrix} \theta_A & & & & 0 \\ & \theta_M & & & \\ & & \theta_P & & \\ 0 & & & & \theta_L \end{bmatrix} \begin{bmatrix} D_A & & & \\ & D_M & & \\ & & D_P & \\ & & & D_L \end{bmatrix} \begin{bmatrix} C_C & & & & \\ & C_A & & & \\ & & C_M & & \\ & & & C_P & \\ & & & & C_L \end{bmatrix} \begin{bmatrix} x_C \\ x_A \\ x_M \\ x_P \\ x_L \end{bmatrix} \quad (3\text{-}20)$$

Equation [3-19] can now be written as

$$\underline{x}(k+1) = \Phi(k+1,k)\underline{x}(k) + \Theta(k+1,k)\underline{\alpha}(k) + w(k) \quad (3\text{-}21)$$

where

$$\underline{x} = [x_C \; x_A \; x_M \; x_P \; x_L]^T$$

$$\underline{\alpha} = [r \; \underline{z}]^T$$

$$W = [w_C \; w_A \; w_M \; w_P \; w_L]^T$$

$$\Phi = \begin{bmatrix} \phi_C & & & & \\ & \phi_A & & 0 & \\ & & \phi_M & & \\ & & & \phi_P & \\ 0 & & & & \phi_L \end{bmatrix} + \begin{bmatrix} \theta_A & & & & 0 \\ & \theta_M & & 0 & \\ & & \theta_P & & \\ 0 & & & & \theta_L \end{bmatrix} \begin{bmatrix} D_A & & & \\ & D_M & & \\ & & D_P & \\ & & & D_L \end{bmatrix} \begin{bmatrix} C_C & & & & \\ & C_A & & 0 & \\ & & C_M & & \\ & & & C_P & \\ 0 & & & & C_L \end{bmatrix}$$

$$\Theta = \begin{bmatrix} \theta_C & \theta_C M \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The system sensor equation can be written using Equations [3-16 and [3-17].

$$\underline{z}(k+1) = H(k+1)\underline{x}(k+1) + \underline{v}(k) \tag{3-22}$$

where

$$\underline{z} = [z_A \; z_M \; z_P \; z_L]^T$$

$$\underline{x} = [x_C \; x_A \; x_M \; x_P \; x_L]^T$$

$$\underline{v} = [v_A \; v_M \; v_P \; v_L]^T$$

$$H = \left[\begin{array}{c|cccc} & H_A & & & \\ & & H_M & & 0 \\ 0 & & & H_P & \\ & & 0 & & H_L \end{array}\right] \left[\begin{array}{c|cccc} C_C & & & 0 & \\ \hline & C_A & & & \\ & & C_M & & 0 \\ 0 & & & C_P & \\ & & 0 & & C_L \end{array}\right]$$

The Kalman filter equation for the system can now be written as

$$\begin{aligned} \hat{\underline{x}}(k+1/k+1) &= \Phi(k+1,k)\hat{\underline{x}}(k/k) + \Theta(k+1,k)\underline{\alpha}(k) \\ &\quad + G(k+1)[\underline{z}(k+1) - H(k+1)\Phi(k+1,k)\hat{\underline{x}}(k/k)] \end{aligned} \tag{3-23}$$

where $\Phi, \Theta, H, \underline{z}$, and $\underline{\alpha}$ are defined for the system Equations [3-21] and [3-22] and

$\hat{\underline{x}}$ = the Kalman filter best estimate of the system states and the filter G matrix is defined by the following equations:

$$P(k+1/k) = \Phi(k+1,k)P(k/k)\Phi^T(k+1,k) + Q(k)$$

$$P(k+1/k+1) = [P(k+1/k)^{-1} + H(k+1)^T R(k+1)^{-1} H(k+1)]^{-1}$$

$$G(k+1) = P(k+1/k+1)\, H(k+1)^T R(k+1)^{-1}$$

where

$$R(k+1) = \begin{bmatrix} \sigma^2_{v_A} & & & 0 \\ & \sigma^2_{v_M} & & \\ & & \sigma^2_{v_P} & \\ 0 & & & \sigma^2_{v_L} \end{bmatrix}$$

$$Q(k) = \begin{bmatrix} \sigma^2_{w_C} & & & & 0 \\ & \sigma^2_{w_A} & & & \\ & & \sigma^2_{w_M} & & \\ & & & \sigma^2_{w_P} & \\ 0 & & & & \sigma^2_{w_L} \end{bmatrix}$$

$$P(k/k) = \begin{bmatrix} \sigma^2_{x_C} & & & & 0 \\ & \sigma^2_{x_A} & & & \\ & & \sigma^2_{x_M} & & \\ & & & \sigma^2_{x_P} & \\ 0 & & & & \alpha^2_{x_P} \end{bmatrix}$$

(initially set to large values)

Figure 3-18 shows the system-level block diagram for the joint model and Kalman filter as defined by Equations [3-21], [3-22], and [3-23]. The implementation of the Kalman filter in the computer model includes the switching logic shown in Figure 3-18. This allows the user to select either the joint model sensor estimate array, $\underline{z}$, or the Kalman filter sensor estimate array, $\underline{\hat{z}}$, in the feedback loop.

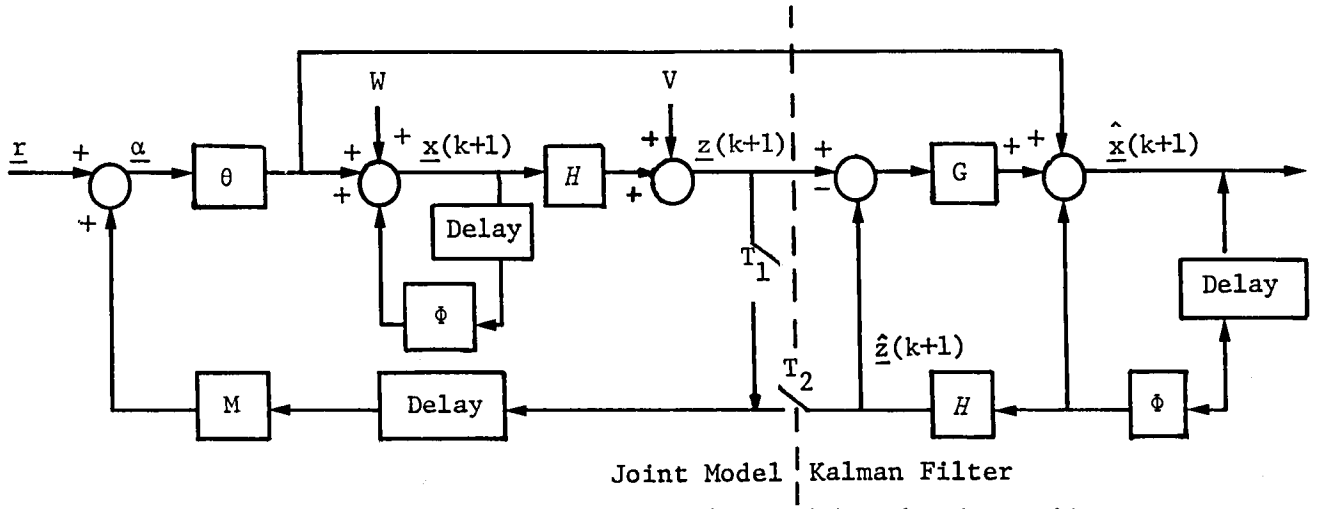


Figure 3-18 System Level Block Diagram of Joint Model and Kalman Filter

The next step in developing the simulation tool will be the expansion of the single-joint model to a two-joint system. Problems associated with multiple-joint systems will be identified and addressed at the two-joint level before attempting to expand the model to handle an N-joint system. The most apparent problem associated with expanding from a single-joint model to a multiple-joint model is the need to consider the dynamics (or load block) problem on a system basis, instead of for individual joints. In simplified form, the multiple joint-model is shown in Figure 3-19. The dynamics equations required for the two-joint case have been developed within the state variable formulation. Appendix C presents the two-joint dynamics.

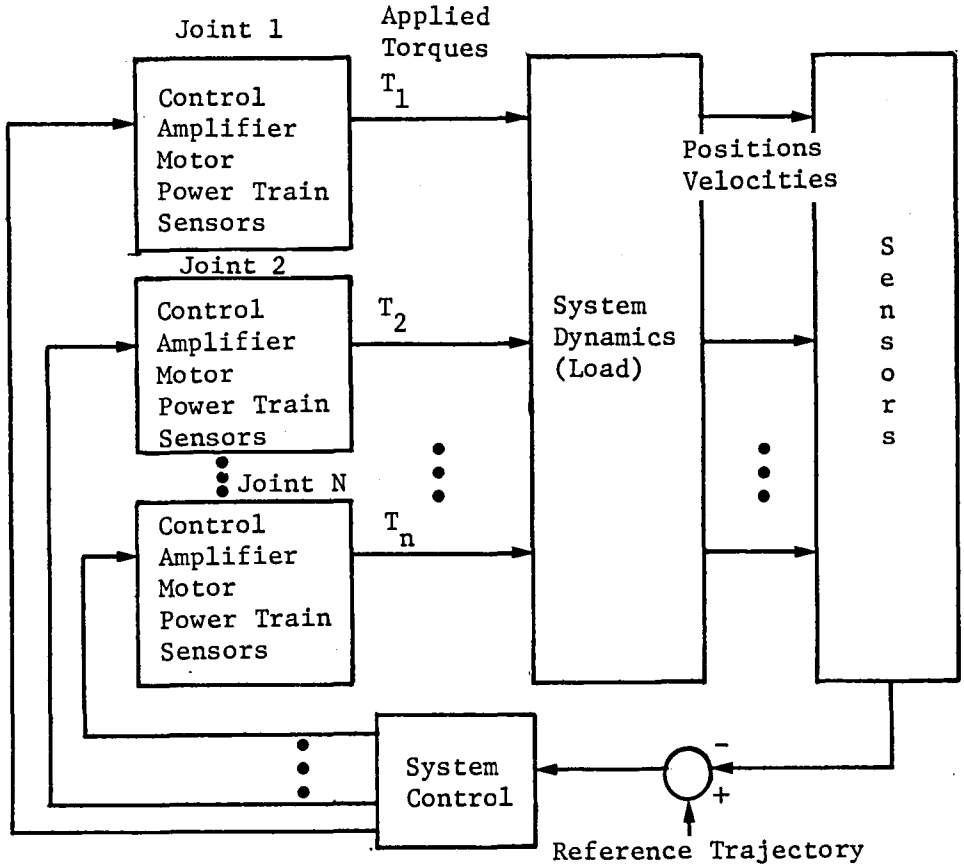


Figure 3-19 N Joint Model Structure

3.2.3 Single Joint Simulation Example

Implementing the single-joint model and Kalman filter provided a computer capability using the state variable approach. The computer model allows demonstration of the basic concepts of component modeling as well as some of the state-variable formulation concepts that will be required to implement the adaptive methods in future simulation tool enhancements.

The discrete, state-space model that was developed in the previous section is illustrated in this section. A relatively simple system consisting of an amplifier, armature-controlled dc motor, and an inertial load is used as an example. The block diagram for this system is shown in Figure 3-20.

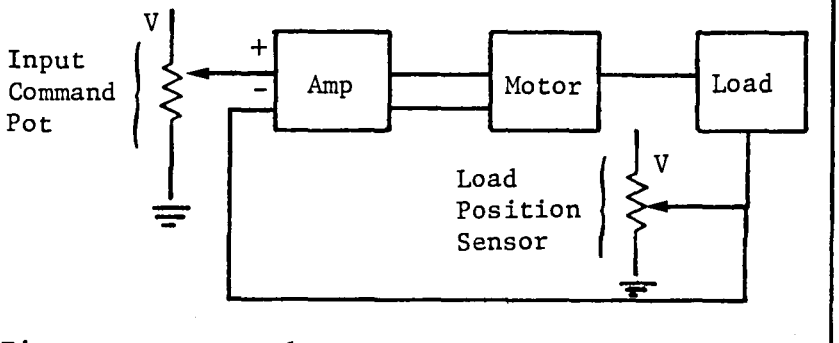


Figure 3-20 Example Servo System

Note that this is an "analog" control system; all individual blocks and the feedback signal are continuous. Continuous state equations are easily discreted by considering the nature of the state transition equation. This determines how state variables transition from one time point to the next. The discrete state representation is formulated by determining a suitably small time interval, T, and assuming that the inputs, u, can be considered to be constant over this interval. Because T will be chosen to represent a sampling frequency above the Nyquist rate, the assumption on the inputs is valid.

The discretizing process is illustrated using the continuous system

$$\dot{x} = Fx + Gu \tag{3-24}$$

This has the continuous time solution,

$$x(t) = e^{F(t-t_o)}x(t_o) + \int_{t_o}^{t} e^{F(t-\tau)}Gu(\tau)d\tau \tag{3-25}$$

Now, the assumption $t - t_o = T$ leads to the discrete system

$$x(n+1) = \Phi x(n) + \Gamma u(n)$$

$$\Phi = e^{FT} \qquad (3\text{-}26)$$

$$\Gamma = \int_o^T e^{Fn} G dn$$

The models for each block are described below. Each model is "discretized" in the state-variable format described earlier. (The component values and appropriate constants that are used in the example have been chosen arbitrarily; they do not correspond to a real system.

Amplifier - The amplifier has been modeled as a first-order system as shown below.

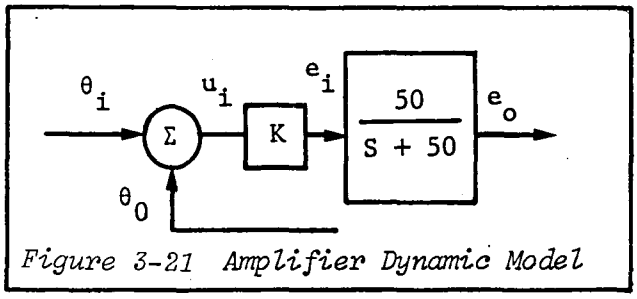


Figure 3-21 Amplifier Dynamic Model

The voltage input to the amplifier is "e_i" and consists of the difference between the reference signal (θ_i) and the position feedback signal (θ_0). The amplifier output voltage is "e_o".

The differential equation describing the amplifier is

$$\frac{de_o(t)}{dt} + 50Ke_o(t) = 50Ke_i(t) \qquad (3\text{-}27)$$

The amplifier state equations are derived using Equation [3-27]. The state variable is chosen as e_o (t). Therefore,

$$\dot{x}_A(t) = -50\, x_A(t) + 50K\, u_A(t) \qquad (3\text{-}28)$$

Note, however, that u_A (t) is a function of two other variables: $\theta_i(t)$, the reference input; and $\theta_0(t)$, the load output displacement.

However, the load position is an observable variable for the load subsystem, Y_{1L}. This means that the amplifier is actually a two-input, single-output system. This is shown below in equation [3-29]:

$$\dot{x}_A(t) = -50\, x_A(t) + [50K - 50K][\theta_i(t) + Y_{1L}(t)] \tag{3-29}$$

With

$$\theta_i(t) = u_{1A}(t) \text{ and } Y_{1L}(t) = u_{2A}(t)$$

Equation [3-29] takes the form of (3-7) with no noise.

It should be noted that for all blocks in the example, the observability matrices, C, and the measurement matrices, H, are all set to the identity matrix, I.

DC Motor - A system diagram for the dc motor is shown below:

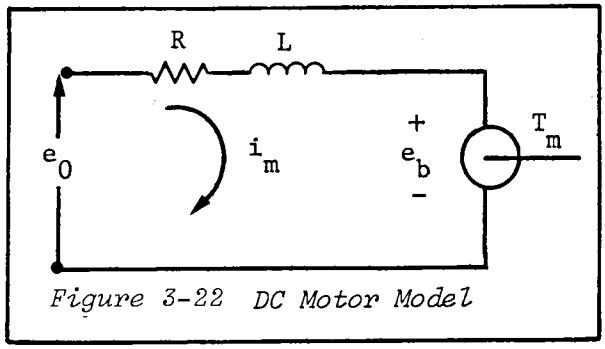


Figure 3-22 DC Motor Model

The voltage equation for the motor is

$$e_o(t) = Ri_m(t) + L\frac{di_m(t)}{dt} + e_b(t) \tag{3-30}$$

The back-EMF voltage (e_b) is a function of the armature angular velocity and the motor back-EMF constant (K_B). If the armature angular velocity is $\dot{\theta}_A$, then the voltage equation is (in Laplace operator notation):

$$E_o(s) = (R + Ls)I_m(s) + K_B s(\theta_A(s)). \tag{3-31}$$

The output torque of the motor, (T_m) is related to the motor current, (i_m) by the torque constant, K_T:

$$T_m(t) = K_T\, i_m(t) \tag{3-32}$$

The motor state equations are derived from equation [3-30]. The motor current, i_m, is an appropriate choice for a state variable.

$$\dot{x}_m(t) = -\frac{R}{L} x_m(t) - \frac{e_b(t)}{L} + \frac{e_o(t)}{L} \tag{3-33}$$

Note that e_o (t) is actually Y_a (t) (the observable variable for the amplifier), and e_b (t), as discussed earlier, is a function of other system variables.

Load - The load dynamic system is shown below:

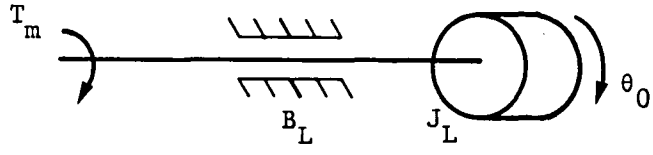


Figure 3-23 Load Dynamic System

The load differential equation is

$$T_m(t) = J_L \frac{d^2(\theta_o(t))}{dt^2} + B_L \frac{d\,\theta_o(t)}{dt} \tag{3-34}$$

In Laplace notation

$$T_m(s) = (J_L s^2 + B_L s)\theta_o(s) \tag{3-35}$$

The basic load equation is [3-34]. Because this is a second-order DE, two state variables are required. These are chosen as

$$x_{1L} = \theta_o(t)$$

$$x_{2L} = \dot{\theta}_o(t)$$

The state description for this system is

$$\begin{bmatrix} \dot{x}_{1L} \\ \dot{x}_{2L} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{B_L}{J_L} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J_L} \end{bmatrix} T_m(t) \tag{3-36}$$

Digital Control Implementation - So far, the discussion has described the simulation of a continuous feedback system. However, any control system can be simulated. Because control variables are handled in a separate part of the program, control inputs can be altered at any integer multiple of the update rate of the continuous system. Because the control subroutine is separate, any control algorithm is feasible.

Numerical Example - The values shown in Table 3-1 were used to develop a closed-loop transfer function.

L = .2	B = 10
R = 1	K_b = 1
J = 1	K_T = 1

Table 3-1 Values Used for Closed-Loop Transfer Function

The resulting closed-loop transfer function is

$$\frac{\theta_o(s)}{\theta_i(s)} = \frac{250K}{s^4 + 65s^3 + 800s^2 + 250s + 250K} \tag{3-37}$$

With a gain of K = 20, the characteristic equation becomes

$$s^4 + 65s^3 + 800s^2 + 250s + 5000 = 0 \tag{3-38}$$

The values in Table 3-1 were used in the discrete state-space model. The time interval update used was 0.05 seconds. A plot of the computer model response for a gain of K = 20 is shown in Figure 3-24. This gain value was used to compare the performance of the discrete state model to the continuous system performance. Damped natural frequency and percent overshoot were used as the comparison factors. The response of the discrete state model for several different values of gain is shown in Figure 3-25.

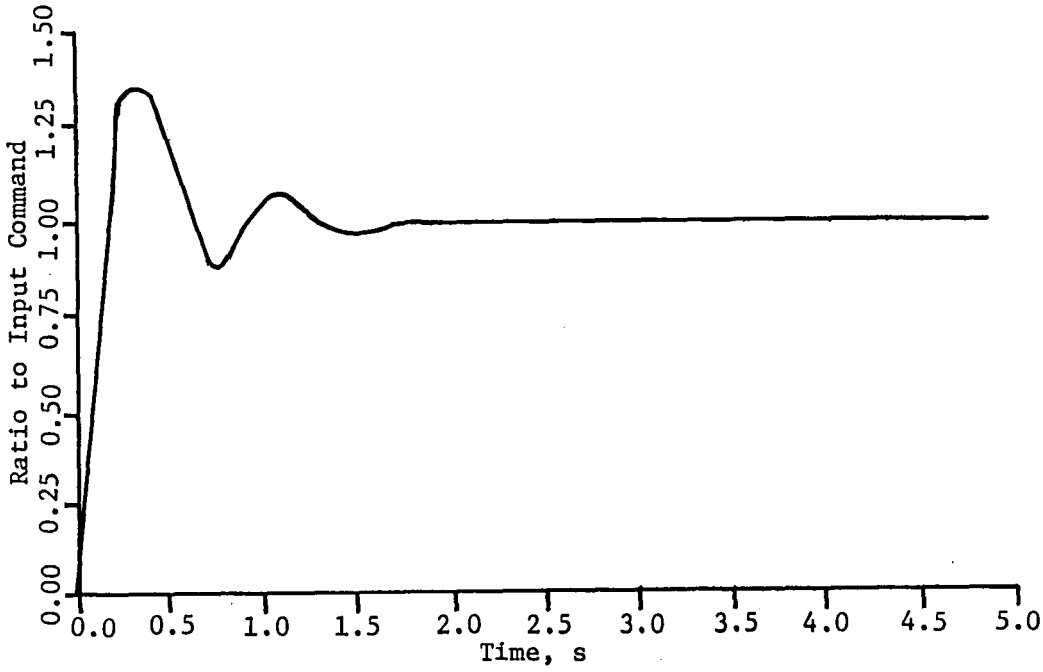


Figure 3-24 Joint Model Response for Gain = 20

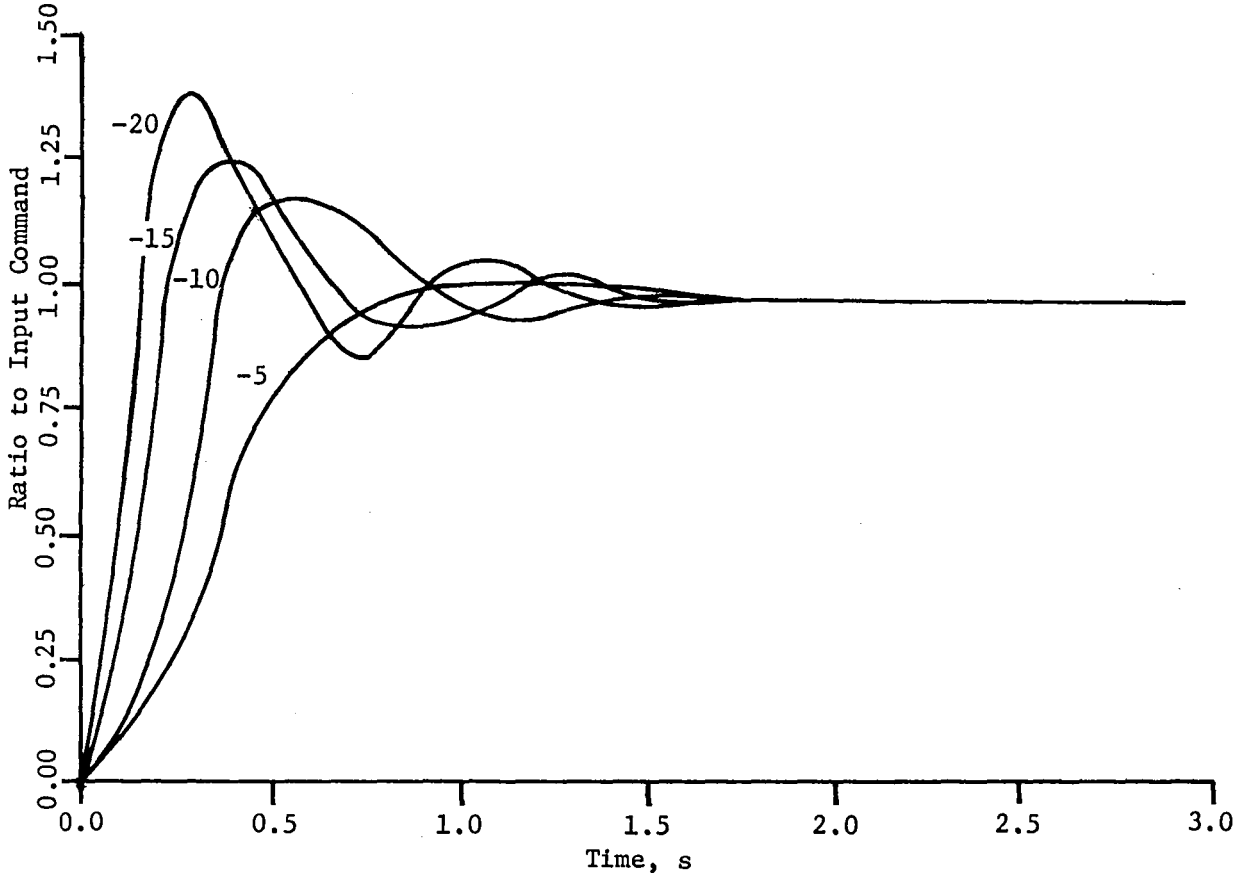


Figure 3-25 Joint Model Response for Gain Values of 5, 10, 15, and 20

Results shown in these figures were obtained from the computer simulation and verified by hand calculation. This simple example was to verify the computer model performance and the state-variable implementation.

Estimation Example - In Appendix E, a thesis was presented which implied that modeling of a particular system was impossible if the classification set was not attainable. In addition, a problem was postulated that required solving the estimation problem to solve the control problem. This example will serve to clarify these notions.

Consider the example used previously. For this example, it can be seen that input to the control box requires feedback from the load box sensor. To dramatically illustrate the need for estimation, it was assumed that moderate process noise is propagated and that the measurement noise on the load cell is extremely high. This corresponds to a condition characteristic of a broken sensor or an extremely poor device.

A constant value of 10.0 was the assumed reference input. The objective of this demonstration is to show it is impossible to achieve an output equal to 10 under these noisy conditions without performing the estimation function. Figure 3-26 shows a time trace of the error resulting from the reference signal minus the sensor feedback. As shown, this resulting error makes it impossible to regulate the output of the load to approach the desired reference signal. However, by using the Kalman filter, the noisy feedback signal created by the poor sensor can be cleaned, producing a modified feedback signal, $\hat{2}$. This signal can be used to provide the feedback required to compare to the reference input. Implementation of this scheme results in Figure 3-27. Comparing this figure with Figure 3-26 clearly demonstrates the improved performance that can be achieved by solving the estimation problem.

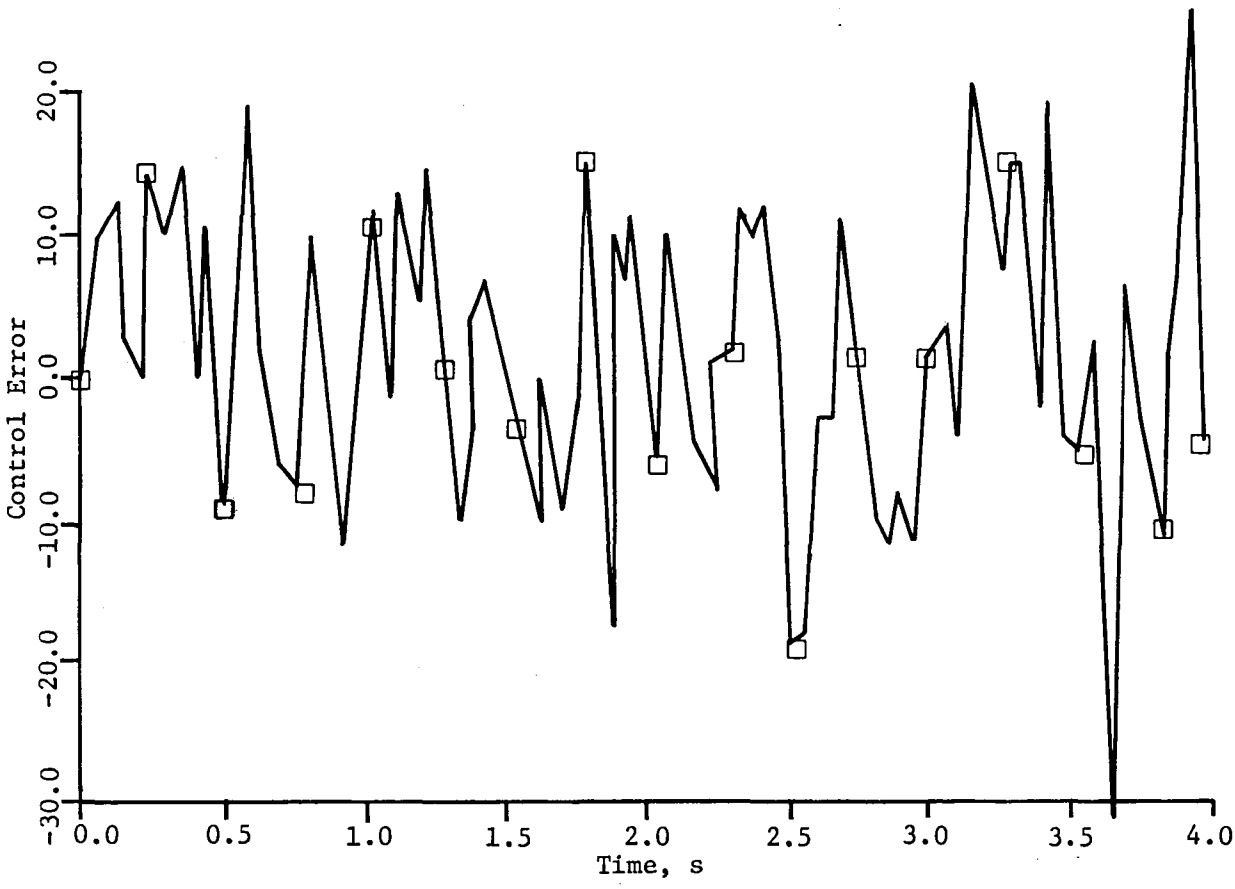


*Figure 3-26 Control Error **without** Kalman Filter Feedback*

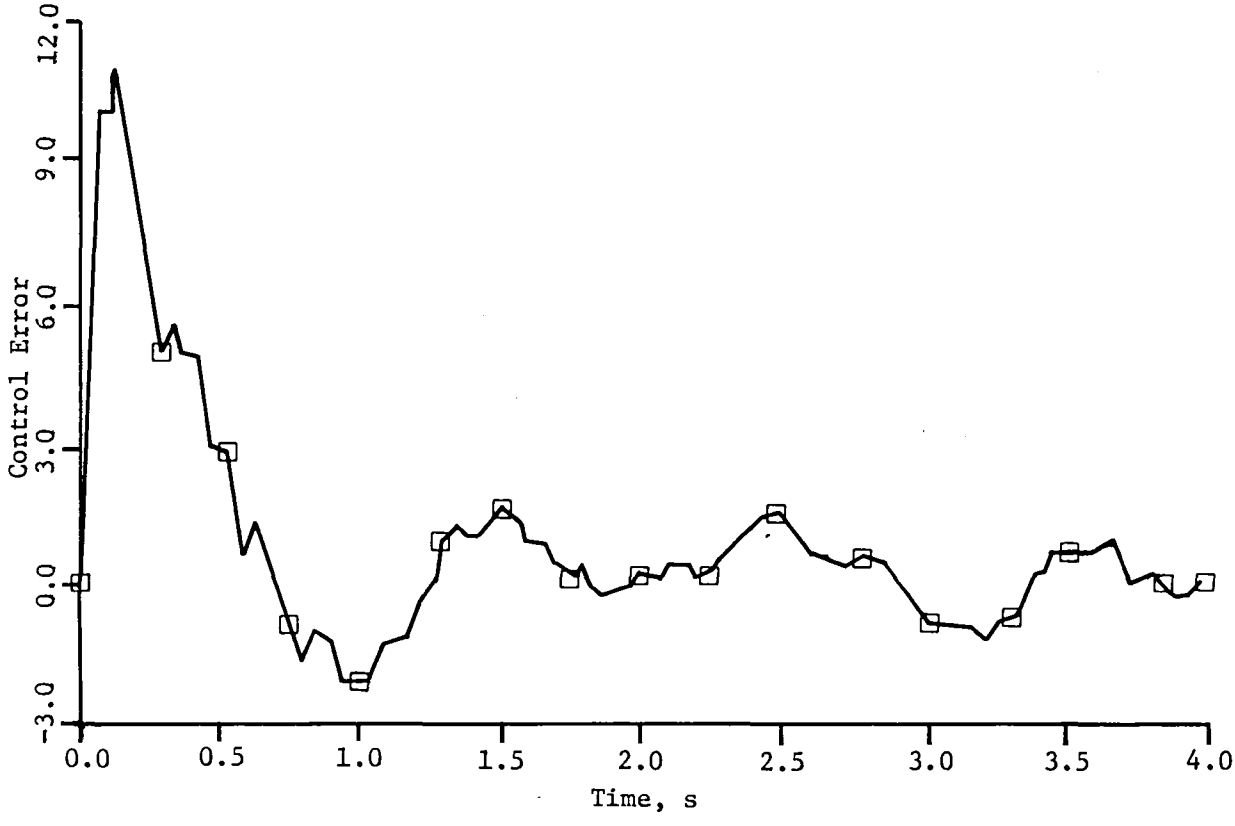


Figure 3-27 Control Error Using Kalman Filter Feedback

3.3 POST PROCESSING

The postprocessing package is intended to provide the ability to perform more detailed study on the results of analysis tools executions. Creation of plot files and other data files can be requested during the execution of any of the analysis tools. These files are then available for input to the various postprocessing capabilities. Implemented postprocessing capabilities include the ability to replay the system motion graphics from the requirements analysis tool and the ability to produce parameter plots for requested data from either the requirements or simulation tools. Eventually, this package will be expanded so that a number of statistical tools will be available. Current implementation allows postprocessing on the results of a single analysis tool execution. Future enhancements will provide for the study of the results of multiple analysis tool executions to allow parametric studies.

4.0 MANIPULATOR CONTROL TECHNIQUES

4.1 INTRODUCTION

This section addresses the problem of manipulator system control. Task 6 of the Statement of Work proposed definition of a set of control algorithms to drive models of the Unimate 600 and Martin Marietta SOS manipulators. Two factors limit the feasibility of modeling these specific manipulators. First, complete physical and mass properties data was not available for either arm. Secondly, the simulation capabilities of the ROBSIM program are not sophisticated enough at this stage of development to allow simulation of these arms to any realistic level. Consideration of specific manipulators at this time is not required in order to proceed toward the goal of this contract, which is to develop a general simulation capability.

What has been accomplished is an extensive literature survey on the topic of manipulator control and a general study of control techniques. Subsection 4.6 contains a large list of current references which are referred to throughout the following discussions. Implementation and testing of various control techniques will be carried out in the next phase of ROBSIM development. What follows is a general discussion of current manipulator control philosophy.

4.2 PROBLEM DESCRIPTION

The manipulator control problem is a composite of several individual problems. The primary problems of interest are introduced by considering a generic manipulator system as shown in Figure 4-1.

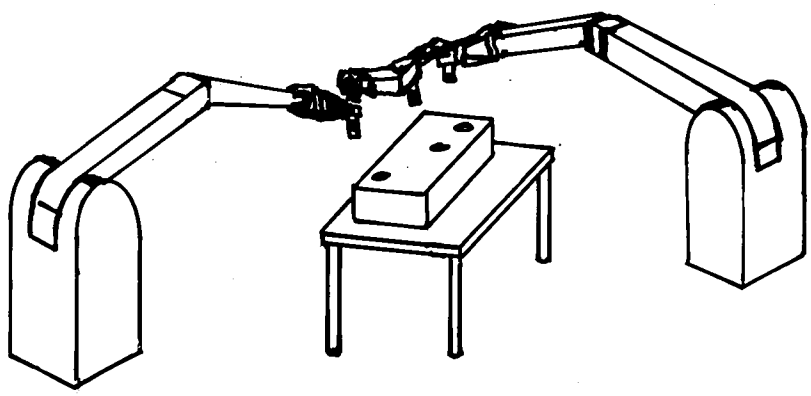


Figure 4-1 Generic Manipulator System

In this system two manipulators are shown performing a coordinated task. This immediately identified one problem of interest: coordinated arm control.

There are three main problems associated with the control of a single manipulator. The first problem has to do with kinematics, essentially the geometry of the manipulator. The second problem deals with the dynamics of the manipulator. The last problem addresses manipulator performance requirements.

4.2.1 Kinematics

The kinematics problem arises from the fact that a manipulator task is generally specified in some local coordinate frame. The most natural frame for describing a manipulator, however, is in terms of joint angles. This results in a need for a means of transforming from local to manipulator coordinates and vice-versa. This has been addressed in the literature quite extensively. Deriving the basic transformation equations and the necessary inverse relationships is not difficult. The problem lies in achieving computational efficiency. This problem has recently been addressed in [3] and [4].

4.2.2 Dynamics

The central focus of the manipulator control problem is the inherently nonlinear nature of the associated dynamic equations. These can be written as, [5];

$$\underline{A}(\underline{q})\ddot{\underline{q}} + \underline{H}\dot{\underline{q}} + \underline{f}(\dot{\underline{q}},\underline{q}) + \underline{P}(\underline{q}) = \tau \quad [4\text{-}1]$$

where

$\underline{A}(\underline{q})$ = 6x6 inertia matrix

$\underline{H}$ = 6x6 diagonal viscous friction matrix

$\underline{f}(\dot{\underline{q}},\underline{q})$ = 6x1 vector incorporating coriolis and centrifugal terms

$\underline{P}(\underline{q})$ = 6x1 vector defining gravity terms

$\underline{\tau}$ = 6x1 vector of input generalized forces

$\underline{q}$ = 6x1 vector of joint positions

The nonlinearities represent a severe problem from the standpoint of control system design in that no unified methodology exists for controlling a nonlinear system. This has resulted in many control approaches that incorporate a dynamic simulation model, [5]-[8], into the controller. This can be viewed as a form of "model-reference" control. The widespread need for simulation of manipulator dynamics has resulted in a continued interest in achieving simplified dynamic models [9]-[11].

Another important area of interest relating to manipulator dynamics is the modeling of distributed parameter effects (vibrational modes). The need for this type of research is recent in origin since manipulators have traditionally been "over-designed". The current trend is towards lighter weight arms with less inertia. This trend arises both from performance and economic requirements. Another motivation for studying flexible arm dynamics comes from the forecasted use of very large arms in space for the construction of large space structures. A summary of current work in this area is contained in [13], [14].

4.2.3 Performance Requirements

Any control system design centers on a performance requirement for the closed loop system. In the case of a manipulator system, however, there might be several different sets of requirements that govern performance over a period of time. To perform a typical task, e.g., inserting a peg in a hole, the manipulator control system must first move the peg to the vicinity of the hole. This amounts to following a displacement-velocity (and possibly acceleration) profile for both position and attitude of the end-effector. To actually insert the peg, however, sensed force and torque information must be included into the controller to account for misalignments, friction, etc. This is generally referred to as "compliance" based control. Task-oriented control has been discussed extensively in the literature [16] - [29].

4.3 APPROACHES TO SERVO CONTROL

As pointed out in the previous section the manipulator presents a unique challenge for control system design. This stems from the fact that all classical control design methods (both continuous and digital) assume that the plant to be controlled is a linear time-invariant system. An attempt to use unaltered linear approaches to the manipulator problem results in system performance that is dependent on manipulator configuration, joint velocities, etc. This has led to a series of approaches that attempt to incorporate knowledge of the nonlinear dynamic system into the control structure.

4.3.1 Inverse Controllers

The idea of the "inverse controller" has permeated the majority of control schemes that have been proposed in the literature [5] - [8], [16], [20]. The basic idea is that for a required trajectory of the manipulator, Equation [4-1] can be solved in reverse; i.e., given the desired joint values, τ can be explicitly solved for. Because it is assumed that there is not a direct match between the model and the actual manipulator, additional feedback gain terms are necessary to account for differences in desired and actual system states. In general these gain terms operate on position and velocity errors.

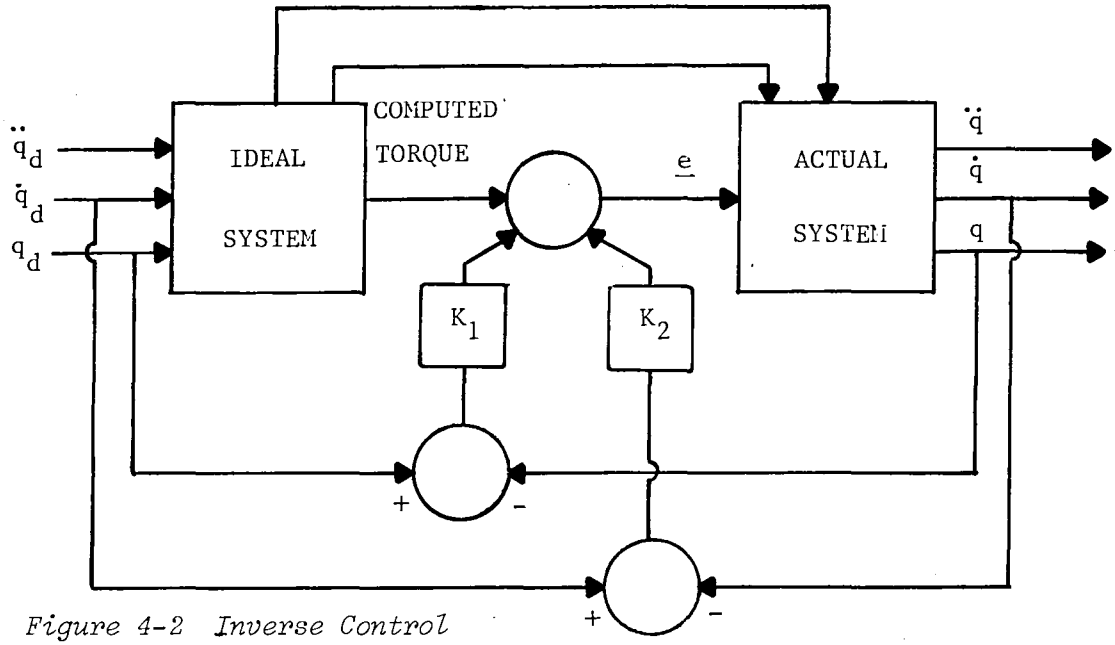


Figure 4-2 Inverse Control

The asymptotic stability of this method is demonstrated by examining the equations governing the error term, ($\underline{e}$), shown in Figure 4-2. Defining e as:

[4-2] $$e = q_d - q$$

The error differential equation becomes:

[4-3] $$\ddot{e} + k_1\dot{e} + k_2 e = 0$$

Since this is an ordinary differential equation, the path of e is totally defined by initial conditions, e(o), and the values of k_1 and k_2.

This is a totally viable scheme if the exact system dynamics are known a priori. The problem is that, if a situation occurs in which the load is unknown, then Equation [4-3] is no longer valid.

Another practical drawback to this method is the need for carrying along a complete dynamic simulation of the ideal system. Because of the computational complexity involved, simplifications in the dynamics are generally made [9]. This also results in error.

4.3.2 Stability-Based Controllers

Other approaches to manipulator control that differ significantly from those described above have viewed the problem from the standpoint of modern control theory [26] - [27], [30]. These methods describe controller stability from the standpoint of Liapunov functions. This is a natural approach, considering the structure of the underlying dynamics.

As in the case of the inverse controllers discussed previously, this approach is feasible if it is assumed that the dynamics governing the system are explicitly known. In the case of [30], for example, the system Hamiltonian must be known. When knowledge of the system dynamics is imprecise the control law generated is only an approximation to that desired.

Other researchers have attempted to apply features from modern control theory to the manipulator problem. In [31] - [32] and [14], linear quadratic optimal control theory has been applied. In [33], results from multivariable control theory have been applied to the problem of "decoupling" the system. This implies either state variable feedback or a transfer matrix that turns a coupled multivariable system into a series of single-input, single-output systems.

4.3.3 Adaptive Control

The different approaches to manipulator control that have been discussed all rely on one common assumption: complete knowledge of the manipulator dynamic state. While the general form of the equations is well known the coefficients, specifically the inertia matrix, will be time-varying in the general case. One approach to overcoming the impact of the time-variations on manipulator performance is to make the controller adaptive. While the term adaptive has been used loosely throughout various disciplines the discussion here will center on a specific definition; the so-called "Parameter Adaptive" controllers.

A parameter adaptive controller adjusts coefficients of a variable "filter." A general adaptive controller is shown in Figure 4-3.

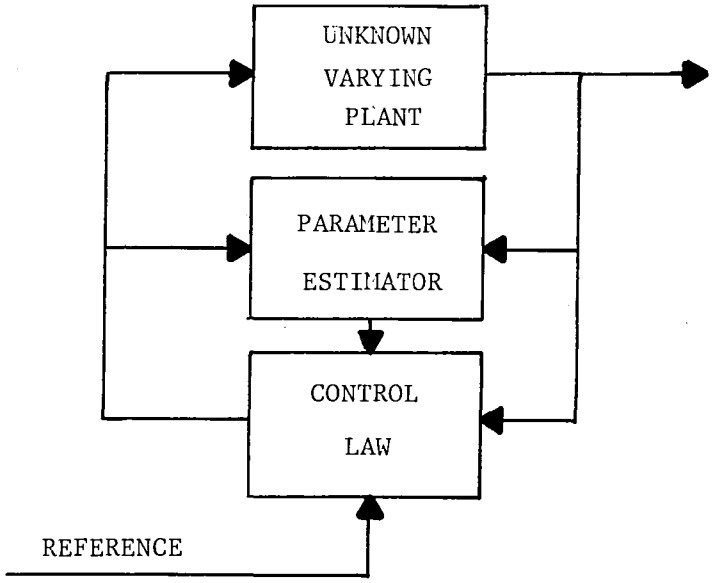


Figure 4-3 General Adaptive Controller

As shown in the figure, it is assumed that the plant to be controlled is either initially unknown or time-varying. The parameter adaptive controller consists of two primary blocks: the parameter estimation block and the control law block. The functions of each of these is discussed in more detail below.

The parameter estimation block determines using sensor data, a set of coefficients that adequately describe the system to be controlled. This definition is purposefully vague because of the wide variety of parameter estimation techniques that exist. A typical parameter estimation block is shown in Figure 4-4.

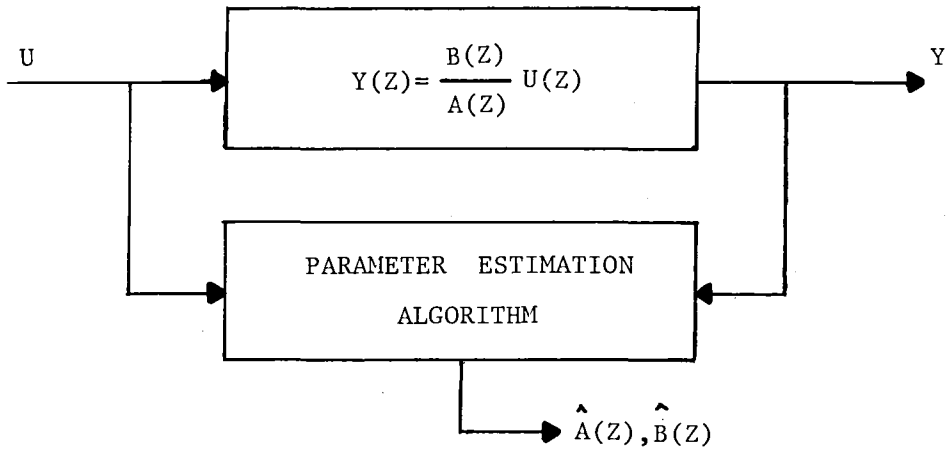


Figure 4-4 General Parameter Estimation

The parameter estimation algorithm operates on input and output data to generate an estimate of the unknown parameters - $\hat{a}(z)$ and $\hat{b}(z)$. Techniques for parameter estimation have been widely studied both for control and signal processing applications. Algorithms are available for deterministic and stochastic systems.

The output of the parameter estimation block is then used in control law formulation. There are many ways of doing this just as there are many methods for designing non-adaptive controllers. The most general control form is model-reference control. In this case the controller functions to make the closed-loop system match a desired transfer function (or matrix). A special case of this type of control is "one-step ahead optimal," or, minimum-variance control. This is discussed in more detail below.

4.4 Predictor Model Control and Equation Error Parameter Estimation

The adaptive controller formulation presented here was first introduced in a 1978 Technical Report [34], and then in the IEEE Transactions on Automatic Control in 1980, [35]. The 1980 paper is widely recognized as being the first complete proof of global stability for any adaptive controller.

The adaptive controller has two functional blocks - a control law block that calculates the inputs to be applied to plant actuators, and a parameter estimation block that provides a specific set of parameters that are used in control law formulation. (The structure is similar to Figure 4-3.)

The control law that is being used is referred to as a "one-step-ahead" or "deadbeat" algorithm.

When the plant to be controlled can be modeled as:

$$Y(k+1) = a_1Y(k)+\ldots+a_nY(k-n+1)+b_0U(k)+\ldots+b_mU(k-m) \quad [4\text{-}4]$$

Then a predictor type control law can be formulated. This assumes that the reference signal is known one step in advance, i.e., Y*(k+1) is known at time k. (This leads, of course, to a tracking delay of one sampling instant.) The actual control law is derived by setting Y(k+1) = Y*(k+1) in Equation [4-4] and then solving for U(k).

$$U(k) = \frac{1}{b_0}\left[Y^*(k+1)-a_1Y(k)-\ldots-a_nY(k-n+1)-b_1U(k-1)-\ldots-b_mU(k-m)\right] \quad [4\text{-}5]$$

With the a's and b's known, the U(k) calculated will result in Y(k+1) = Y*(k+1). This same formulation can be used when there is a system delay of more than one, i.e., the output at k+1 is a function of inputs applied two or more sampling instants in the past. It is shown in [36] that the same basic approach can be used to execute "model-reference" control. (The formulation in [4-4] and [4-5] leads to what has been called "inverse control", i.e., the transfer function of the closed loop plant is a simple delay.) This approach to control law formulation is general and flexible and does not unnecessarily constrain the designer.

The general concept of parameter estimation was introduced in the previous section. A specific parameter estimation algorithm, recursive least squares, will be discussed here.

Recursive least squares falls into the family of the so-called "equation error" parameter estimation schemes. These are discussed extensively in [36] and [37]. The name "equation error" is derived from the basic structure of the estimation algorithm, (see Figure 4-5).

Most parameter estimation algorithms have the form:

[4-6] $\hat{\Theta}(k+1) = \hat{\Theta}(k) + \Gamma e(k)$

In an "equation error" formulation,

[4-7] $e(k) = Y(k) - \Phi^T\hat{\Theta}(k)$

where

$$\Phi^T = [Y(k-1),Y(k-2),\ldots,Y(k-n+1),U(k-1),\ldots,U(k-m-1)]$$

$$\hat{\Theta} = [a_1,a_2,\ldots,a_n,b_0,b_1,\ldots,b_m]$$

Equation [4-7] is essentially the same as Equation [4-4], except that parameter estimates are used instead of the actual parameters. For the true parameters, e(k) = 0. Hence the name "equation error", i.e., how well the estimated parameters satisfy Equation [4-4]. For $e(k) \neq 0$, the parameter estimator will continue to update the estimate.

Figure 4-5 Equation Error Parameter Estimation

The " Γ " term in Equation [4-6] is what differentiates parameter estimation schemes. In the case of recursive least squares Γ is chosen as shown below.

[4-8]

$$\Gamma = \frac{a(k)P(k-d-1)\phi(k-d)}{1+a(k)\phi(k-d)^TP(k-d-1)\phi(k-d)}$$

$$P(k-d) = P(k-d-1) - \frac{a(k)P(k-d-1)\phi(k-d)\phi(k-d)^TP(k-d-1)}{1+a(k)\phi(k-d)^TP(k-d-1)\phi(k-d)}$$

(The "d" delay is necessary to retain causality). Simpler schemes are available but they have very poor stochastic properties.

Using the controller and parameter estimation algorithm described above, the adaptive controller functions as shown in Figure 4-6.

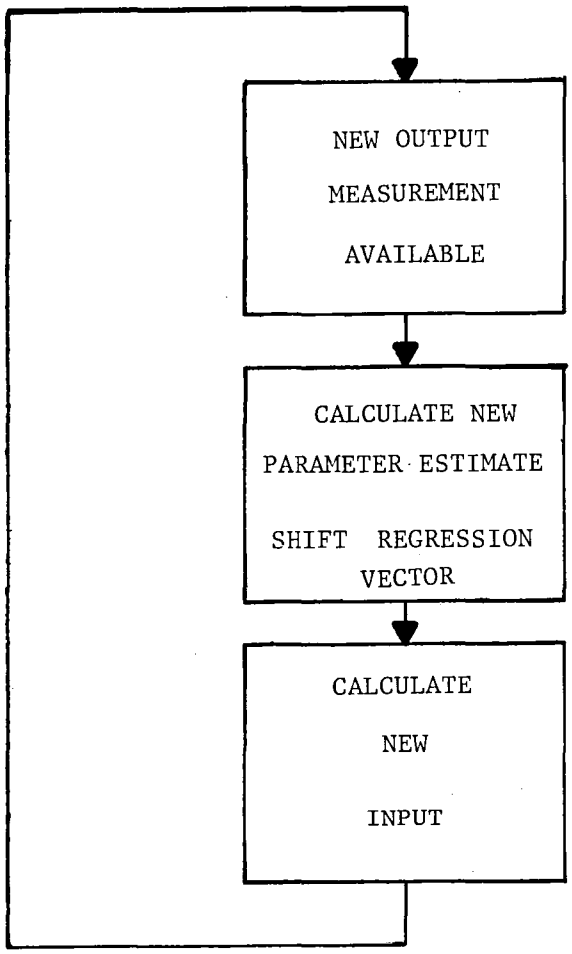


Figure 4-6 Adaptive Control Flow Diagram

4.5 CONCLUSION

This report has discusses some of the primary approaches that have been used in manipulator control. Adaptive control appears to be extremely promising in regard to projected performance since it is one method that compensates for parameter drift and load changes.

4.6 REFERENCES

[1] An Overview of the Basic Research Needed to Advance The State of Knowledge in Robotics. J. R. Birk, R. B. Kelley, IEEE Transactions SMC, Vol SMC-11, No. 8, August 1981.

[2] Issues in Advanced Automation for Manipulator Control. K. Bejczy, 1976, JACC.

[3] Kinematic Control Equations for Simple Manipulators. R. P. Paul, B. Schmano, G. E. Mayer, IEEE Trans., Vol SMC-11, No. 6, June 1981.

[4] Differential Kinematic Control Equations for Simple Manipulators. R. P. Paul, B. Shimano, G. E. Mayer, IEEE Trans., Vol. SMC-11,
No. 6, June 1981.

[5] Controller for a Mechanical Manipulator. J. Y. S. Luh, M. W. Walker, AC Theory and Applications, Vol 8, No. 1, January 1980.

[6] On-Line Computational Scheme for Mechanical Manipulators. Y. Y. S. Luh, M. W. Walker, R. P. Paul, Trans. ASME, JDSMC, June 1980.

[7] Resolved Motion Force Control of Robot Manipulator. C. H. Wu, R. P. Paul, IEEE Sys, Man. Cyber, Vol. SMC-12, No. 5, 1982.

[8] Resolved-Acceleration Control of Mechanical Manipulators. J. Y. S. Luh, M. W. Walker, P. R. Paul, IEEE Trans. Auto. Control, Vol. AC-25, No. 3, June 1980.

[9] Simplified Robot Arm Dynamics for Control. A. K. Bejczy, P. R. Paul, 1981,

[10] Development of Equations of Motion of Single-Arm Robots. R. L. Huston, F. A. Kelly, IEEE Sys, Man. Cyber, Vol. SMC-12, No. 3, 1982.

[11] Efficient Dynamic Computer Simulation of Robotic Mechanisms. M. W. Walker, D. E. Orin, 1981, JACC.

[12] The Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators. W. M. Silver, 1981, JACC.

[13] Modeling of Flexibility Effects in Robot Arms. F. Kelly, R. L. Huston, 1981, JACC.

[14] Optimal Control of Non-Rigid Robot Linkages. D. R. Falkenburg,19. P. Judd, Oakland University, Rochester, MI.

[15] Feedback Control of Two Beam, Two Joint Systems with Distributed Flexibility. W. J. Book, O. Maizza-Neto, D. E. Whitney, Trans. ASME, J. DSMC, December 1975.

[16] Force Feedback Control of Manipulator Fine Motions. D. E. Whitney, Trans. ASME, JDSMC, June 1977.

[17] Compliance and Control. R. Paul, B. Shimano, 1976, JACC.

[18] Computer-Controlled Assembly. J. L. Nevins, D. E. Whitney, Scientific America, 1978.

[19] Compliance and Force Control for Computer-Controlled Manipulators. M. T. Mason, IEEE Trans., Vol. SMC-11, No. 6, June 1981.

[20] Joint Torque Control by a Direct Feedback for Industrial Robots. J. Y. S. Luh, W. D. Fisher, R. Paul, 1981, CDC.

[21] Force Feedback Control. E. G. Gerelle, 8th Int'l Symp. on Industrial Robots, Stuttgart, W. Germany, 1978.

[22] Asynchronous Interactive Control Systems. M. I. Vuskovic, E. Heer, Trans. ASME, JDSMC, Vol. 104, March 1982.

[23] Issues in Advanced Automation for Manipulator Control. A. K. Bejczy, 1976, JACC.

[24] Joint Coordination for Manipulator Tracking. T. Turner, 1981, CDC.

[25] Computer Control of Multijointed Finger System for Precise Object Handling. T. Okada, IEEE Sys. Man. Cyber, Vol SMC - 12 November 3, 1982.

[26] Simulation and Control Synthesis of Manipulators in Assembling Technical Parts. D. Stokic, M. Vukobratovic, Trans. ASME, JDSMC, Vol. 101, December 1979.

[27] One Engineering Concept of Dynamic Control of Manipulators. M. Vukobratovic, D. Stokic, Trans. ASME, JDSMC, June 1981.

[28] Hybrid Position/Force Control of Manipulators. M. H. Raibert, J. J. Craig, Trans. ASME, JDSMC, June 1981.

[29] Quasi-Static Assembly of Compliantly-Supported Rigid Parts. D. E. Whitney, Trans. ASME, JDSMC, March 1982.

[30] A New Feedback Method for Dynamic Control of Manipulators. J. Takegaki, S. Arimoto, Trans. ASME, JDSMC, June 1981.

[31] Minimum Time, Sequential Axis Operation of a Cylindrical, Two-Axis Manipulator. P. M. Lynch, 1981, JACC.

[32] Microprocessor Implementation of Optimal Control for a Robotic Manipulator System. W. E. Snyder, W. A. Griver, 1979, JACC.

[33] Dynamic Decoupling of a Remote Manipulator System. J. S. C. Yuan, IEEE Trans., Vol AC-23, No. 4, August 1978.

[34] Discrete Time Multivariable Adaptive Control. G.C. Goodwin, P.J. Ramadge, P.E. Caines, Harvard Univ. Tech. Report, November 1978.

[35] Discrete-Time Multivariable Adaptive Control. G. C. Goodwin, P. J. Ramadge, P. E. Caines, IEEE Trans. on Automatic Control, Vol. AC-25, No. 3, June 1980.

[36] Adaptive Filtering, Prediction, and Control. G. C. Goodwin, K. S. Sin, Manuscript TBD, Dept. of Commerce and Electrical Engineering, University of Newcastle, New Sourth Wales, Australia.

[37] Discrete Techniques of Parameter Estimation. J.M. Mendel, Marcel-Dekker, Inc., New York, 1973.

[38] The Application of Model-Referenced Adaptive Control to Robotic Manipulators. S. Dubowsky, D.T. DesForges, 1979, JACC.

[39] Control of Robotic Manipulator With Adaptive Controller. A.J. Koivo, T.H. Guo, 1981, CDC.

5.0 ROBSIM SUPPORT FOR ROSS

5.1 INTRODUCTION

One of the objectives of the Robotic Simulation Tool is to provide support for the design, development, and deployment of telepresence systems for in-space satellite servicing and maintenance.

The design of such a system was the subject of Task 8 of the current contract, Remote Orbital Servicing System (ROSS). The results of Task 8 were documented in a previous report, MCR-82-533.

Specific technology issues must be addressed. These issues will be related to two segments of the system. These two segments are: 1) the ground segment and 2) the space segment. Elements associated with the ground segment are: a) the human operator, b) hardware/software supporting the man at the control station, c) data processing, and d) mission assessment. Space segment elements will be: e) the work site and f) data processing.

Technology issues which require resolution are best defined with respect to the elements described above. Table 5-1 defines these issues.

Many of the issues given above can be resolved by creating a simulation tool representing the teleoperator and robotic system. To answer these concerns, the simulation must be capable of the following functions:

1) For specific designs, it will allow system evaluation, timeline and task planning, operator training, and backup support during mission operations.

2) Act as a design tool for evaluating various levels of automation, direct, and supervisory control.

3) Allow evaluation at the system level, advances in subsystem technology.

4) Allow parametric studies of subsystems and components supplying guidance as to high leverage areas requiring research at the technology base level.

5) Allow error analysis and investigation of error recovery and backup mode operations.

6) Allow investigations of advanced control techniques including trajectory optimization.

7) Allow investigations of the applications of artificial intelligence (AI) technology in automated decision making.

8) Allow studies of man/machine interface with remote systems to develop an educated telepresence and to enhance man's capability to accomplish remote operations by increasing his supervisory capabilities for complex systems.

Table 5-1 Technology Issues

a) Human Operator	o Acceptance of Technology o Number of Operators Required o Single Operator Time-Constrained Capabilities o Human Endurance o System Response Time o Perceptual Limits o Information Assimilation Rate & Capacity o Cognitive Limits
b) Control Station	o Delay o HW/SW Architecture o Method of Stereo Vision Display o Visual Enhancement o Display Mechanisms o Command Mechanisms o Integration of Display and Command Mechanisms o Communication of Task Semantics
c) Data Processing	o Degree of Autonomy o Servo Loop Stability o Delay o Information Bandwidth o Coordination of Multiple Effectors o Collision Avoidance o Topography Estimation o Robust Control o Mistake Monitoring
d) Mission Assessment	o Task Plan Definition o Definition of Performance Measure o Experimental Verification and Compliance
e) Work Site	o Lighting o Detector Configuration o Space Qualifications o Auto-focus and Auto-point o Number and Configuration of End Effectors

5.2 APPROACH

In concert with the ROSS program development, development of the simulation tool will take place under three phases. The output of the Phase I level of effort will be the baseline hardware and software required to support a ground station design for the ROSS. Phase II simulation will support actual flight hardware and can be used to validate software models used in the simulation to represent actual hardware. Information obtained in Phases I and II will be incorporated to develop a high fidelity simulation that will support the deployment of the Remote Orbital Servicing System (ROSS).

Activities to be performed under ROSS Phase I are:

1. Define initial configuration of ROSS ground station, procure, construct, and install equipment.

2. Develop software models of components and subsystems of ROSS.

3. Conduct lab tests of available components and subsystems and validate software models.

4. Integrate developed models of ROSS into robotics simulation.

5. Initiate evaluation of ROSS.

Completion of these activities will result in initial ROSS evaluations.

Phase II ROSS activities providing guidance and evaluation of ROSS during phase C/D and in evaluation of advanced subsystems are:

1. Continue ROSS evaluations with modifications as defined from phase C/D and lab tests.

2. Develop models of components and subsystems resulting from base technology programs. Install in robotic simulation, validate, and conduct parametric analysis at systems level to define benefits and required modifications or improvements.

3. Modification of ground control station to allow reconfiguration to conduct basic tests of man/machine interface.

4. Final ROSS configuration modification, validation, and operator training.

5. Research in enhancement to ROSS concept for future missions.

The ultimate goal of the simulation will be accomplished at the completion of the Phase III level of effort. By completing this phase, a simulation will exist that has the following properties: 1) validated simulation which can be used to evaluate the teleoperator and robotic system and 2) provide a research tool to be used to design, develop, and deploy man/machine systems used in conjunction with remote manipulators. Activities to be performed under this phase are summarized below:

1. Incorporate vehicle dynamics for capture, docking, and stabilization.

2. ROSS training and mission support.

3. Final validation of ROSS using flight experiment results.

4. Perform research on next generation servicer and space construction using remote systems.

When properly designed, the simulation can be used to: 1) aid in requirements analysis and conceptual design, 2) hardware and software design verification, 3) personnel training, and 4) mission support and assessment. In performing these activities, the simulation can be used to define and allocate man/machine functions. The operational mode of each task function will be designated as being either manual, semi-automatic, or automatic.

Associated with the five activities which support man-in-the-loop simulations, certain simulation utility tasks will be required. Varying degrees of fidelity will be required to accomplish these tasks. Table 5-2 presents these relationships.

From heuristic arguments, it can be shown that a simulation provides the foundation for the ground control station configuration (GCSC). Examination of Figure 5-1 shows the GCSC evolution from a digital simulation. As shown in this figure, a digital simulation of the teleoperator in conjunction with the manipulator dynamics will lead to the software requirements necessary to support the ROSS mission. In addition to providing a high fidelity model of the entire system, the digital simulation will also identify the critical parameters, give an analytical definition of the system requirements, provide a mechanism for allocating error budgets, give an upper bound on system performance, and aid in the design of flight experiments.

Having determined the optimal performance of the ROSS system with the digital simulation, components of the system can be developed. For example, the manipulator dynamics would be replaced by the actual manipulator itself. Also, analog components such as a hand controller would be augmented with digital components ultimately creating a hybrid simulation. The output of such a hybrid configuration would be the hardware and software which would serve to baseline the GCSC.

Payoffs other than the HW/SW requirements of the hybrid simulation are: 1) verification of the digital models, 2) provide a measure of system component compliance to specifications, 3) provide a measure of the ability to control a mechanism from a remote observation post, and 4) provide an aid to the design of experiments to be conducted.

Table 5-2 Man-in-the-loop Simulation Activity

ACTIVITY	SIMULATION UTILITY TASKS	SIMULATION FIDELITY PHASES
Requirements Analysis and Conceptual Design	o Conceptual Development o Allocation of Functions Between Man and Machine o Feasibility Demonstration	o Functional Simulation o System Modeling o Subsystem Modeling o Component-Level Modeling
Hardware and Software Design and Development	o Analysis of System Performance o Development of Controls and Displays o Development of Flight Experiments	
Hardware and Software Design Verification	o Functional Verification o Subsystem Compliance o Mission Timelining o Development of Procedures and Checklists	o Breadboard Hardware and Software o Prototype Hardware and Software
Personnel Training	o Personnel Selection o Part Task Training o Full Up Training	o Flight-Type Hardware and Software
Mission Support and Assessment	o Real-Time Mission Support o Post Flight Analysis	

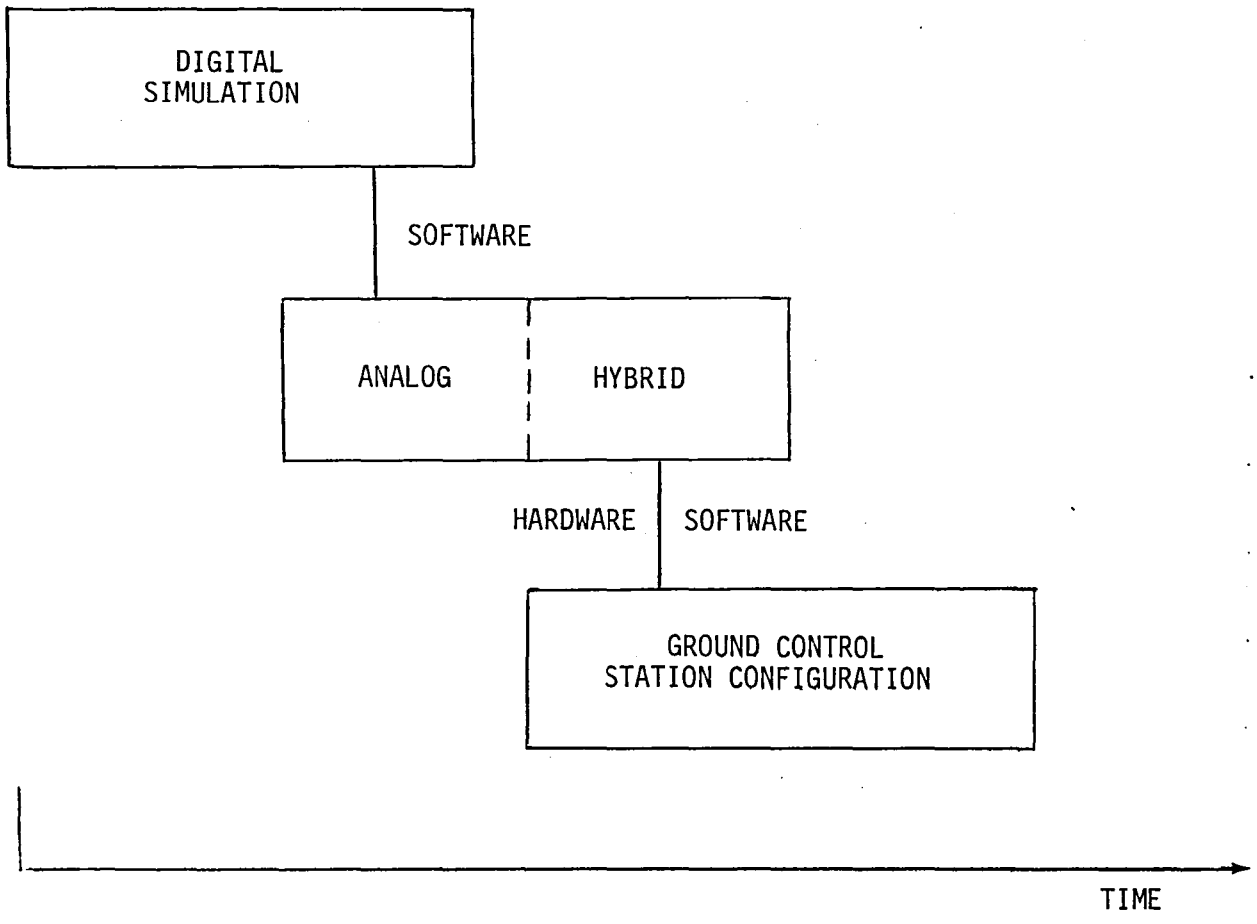


Figure 5-1 Simulation: A Foundation for the GCSC

6.0 CONCLUSIONS AND RECOMMENDATIONS

The current phase of development (Phase I) has provided an evaluation of artificial intelligence techniques, a computer program framework, some basic capabilities for the ROBSIM program, and study into manipulator control techniques. Future work (Phase II and III) will include several study areas as well as enhancement and expansion of computer capabilities.

Short-range (Phase II) expansion of the ROBSIM capabilities will be conducted throughout the remainder of FY1982. Four major areas are to be considered:

1) Investigate trajectory planning modules (collision avoidance, minimum energy, other constraints);

2) Develop video simulation and image processing modules;

3) Define interactive initialization program modules;

4) Identify benefits and limitations of control laws, including both classical and modern control techniques.

Several enhancements and expansions are planned for computer capabilities. The system definition capability will be expanded to provide more detailed system component input. A library of parts will be started to store data describing frequently used components. Graphics capabilities will also be expanded. The force/torque requirements analysis tool will be expanded to provide for external forces and torques, rate and position limit checks, and multiple arm capability. The simulation tool will be expanded from a single-joint model to a two-link model. Complete dynamics for the two-link case will be added. The two-link case will then be used for control technique studies. The postprocessing capabilities will be expanded to handle input from multiple analysis tool executions.

Long-range (Phase III) expansion of the ROBSIM capabilities (FY1983 and beyond) will be directed primarily toward supporting teleoperator and robotic systems capable of remote space operations, i.e., the Remote Orbital Servicing System (ROSS) concept. In FY1983, it is proposed to incorporate software modules for the total simulation of the ROSS concept into the simulation framework. The simulation framework will be expanded in the areas of actuator, sensor, and mobility system modeling; the ability to simulate multiple arm configurations; and a parts library. Mathematical models of the components and subsystems of ROSS will be developed and incorporated into the ROBSIM framework. The ROSS subsystems to be modeled will include the vehicle dynamics, the manipulator system, and the flight sensors. This activity will require expanding the ROBSIM framework to include multiple arm configurations and modeling of the host vehicle. For the specific ROSS configuration, it is anticipated that the models for the manipulator motors, the sensors, and the end effector are sufficiently different from existing models to warrant developing new software modules. However, the mass properties, physical dimensions, and joint configurations can be

directly incorporated into the existing dynamics models. Also included under the FY1983 activity, laboratory experiments should be conducted to validate software modules and to define hardware requirements for the ground control station for manned control of the ROSS.

The trajectory planning technique defined under the Phase II activity should be implemented and incorporated into the ROBSIM framework. The trajectory planning module will perform obstacle avoidance, optimal trajectory planning, and singularity avoidance. The man-machine interface, coupled with software-simulated remote systems, would then be in place to evaluate teleoperator and robotic systems capable of remote space operations.

Various approaches to autonomous task planning should also be studied. Task planning is one level of sophistication above trajectory planning where the system solves the subtle problems associated with task scheduling to achieve a specific mission goal. Techniques to be considered include directed graph theory, rule-based systems, and theorem provers.

Continued development during Phase II and Phase III may be carried out on virtually any modern digital computing machine of the super-mini and higher class. However, with the exception of the newest and fastest machines, the necessary processing will not be accomplished in real time by a single machine. Specialized machines with multiple processor architectures will be necessary for the accomplishment of this goal. Areas such as mechanical configuration optimization, accuracy analysis, control requirements and stability studies, and work environment studies may require simulation but not necessarily in real time.

If real-time solutions of the simultaneous simulation equations are determined not to be necessary, the continued development on the VAX 11/780 is quite appropriate. However, it should be pointed out that there are several significant downstream requirements pointing to a need for a real-time simulation associated with a hardware development program such as ROSS. Some of these are:

1) Realistic Graphics - When the simulation system is coupled with a graphic display of the computer solutions in real time, the results are much more realistic.

2) Surrogate Hardware - Properly-simulated hardware may be used to replace hardware subsystems that have failed or are not yet available. The remainder of the hardware may be used in the system as intended. System problems may be isolated by substituting software modules for suspected failing hardware modules. System integration may be approached more gradually by bringing hardware on line one module at a time or as the hardware is available.

3) Real-Time Test Bed for Algorithm Development - Control and estimation algorithms may be evaluated and optimized using the true or emulated control computer without the use of scaling within the algorithms and no potential damage to the existing manipulator assembly.

4) Real-Time Research Test Bed for Artificial Intelligence (AI) and Man/Machine Interface (MMI) - Safe, real-time response to decisions and trajectory definitions by the AI as dictated by sensor output and command input through the MMI is very important to these research areas.

It should be emphasized that in order to achieve the goal of an operational robotic simulation, a large number of complex areas must be considered. Many of these areas are already provided for in the ROBSIM framework; others must be added as required. Table 6-1 provides a list of candidate areas for consideration. Areas such as sensor configuration and sensor data processing may present significant challenges in simulation. Other areas such as control, intelligence, and man/machine interface are expected to be research oriented and therefore somewhat fluid. All areas should be represented, even if only by simplified modules, in order to approach full-scale simulation.

Table 6-1 Simulation Considerations

A. Manipulator Configuration
1. Joint Dynamics
2. Joint Sensors
3. Joint Actuators
4. Manipulator Geometry
5. Inner Loop Control
6. End Effector
7. End Effector Control
B. Sensor Configuration
C. Mobility Configuration
1. Mobility Sensors
2. Mobility Control
D. Sensor Data Processing
E. Control Philosophy
F. Intelligence
G. Man/Machine Interface

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-165975		

4. Title and Subtitle	5. Report Date
EVALUATION OF AUTOMATED DECISIONMAKING METHODOLOGIES AND DEVELOPMENT OF AN INTEGRATED ROBOTIC SYSTEM SIMULATION-STUDY RESULTS	September 1982
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
J. W. Lowrie, Dr. A. Fermelia, D. C. Haley, K. D. Gremban, J. Van Baalen, R. W. Walsh	MCR-82-581 Vol. I
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Martin Marietta Aerospace Denver Division P.O. Box 179 Denver, CO 80201	NAS1-16759
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Jack Pennington
Final Report

16. Abstract

A study was performed to evaluate a variety of artificial intelligence techniques which could be used with regard to NASA space applications and robotics. The techniques studied were decision tree manipulators, problem solvers, rule-based systems, logic programming languages, representation language languages, and expert systems. Another major goal of the study was to define the overall structure of a robotic simulation tool and develop a framework for that tool. Nonlinear and linearized dynamics equations were formulated for n-link manipulator configurations. A framework for the robotic simulation was established which uses validated manipulator component models connected according to a user-defined configuration.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
robotics, artificial intelligence, simulation, manipulator control, ROSS	Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	91	

For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document