SEVENTH SEMI-ANNUAL REPORT

ON

RESEARCH ON

CONTROL OF FREE-FLYING SPACE ROBOT MANIPULATOR SYSTEMS

Submitted to

Dr. Henry Lum, Jr., Chief, Information Sciences Division
Ames Research Center, Moffett Field, CA 94035

by

The Stanford University Aerospace Robotics Laboratory
Department of Aeronautics and Astronautics
Stanford University, Stanford, CA 94305

Research Performed Under NASA Contract NCC 2-333
During the period March 1988 through August 1988

Professor Robert H. Cannon Jr.
Principal Investigator

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This document reports on the work done under NASA Cooperative Agreement NCC 2-333 during the period March 1988 through August 1988. The research was carried out by a team of six Ph.D. candidate students from the Stanford University Aerospace Robotics Laboratory under the direction of Professor Robert H. Cannon, Jr. The goal of this research is to develop and test new control techniques for self-contained, autonomous free-flying space robots. Free-flying space robots are envisioned as a key element of any successful long term presence in space. These robots must be capable of performing the assembly, maintenance, and inspection, and repair tasks that currently require astronaut extra-vehicular activity (EVA). Use of robots will provide economic savings as well as improved astronaut safety by reducing and in many cased eliminating the need for human EVA.

The focus of our work is to develop and carry out a set of research projects using laboratory models of satellite robots. These devices use air-cushion-vehicle (ACV) technology to simulate in two dimensions the drag-free, zero-g conditions of space. Using two large granite surface plates (6' by 12' and 9' by 12') which serve as the platforms for these experiments we are able to reduce gravity-induced accelerations to under $10^{-5}g$ with a corresponding drag-to-weight ratio of about 10^{-4}—a very good approximation to the actual conditions in space.

Our current work is divided into six major projects or research areas: Cooperative Manipulation on a Fixed Base, Cooperative Manipulation on a Free-Floating Base, Global Navigation and Control of a Free-Floating Robot, Multiple-Vehicle Cooperation, an alternative transport mode called LEAP (Locomotion Enhancement via Arm Push-Off), and Adaptive Control of LEAP.

Fixed-base cooperative manipulation work represents our initial entry into multiple arm cooperation and high-level control with a sophisticated user interface. This experiment is now fully on-line and has already produced several significant new results.

The floating-base cooperative manipulation project strives to transfer some of the new technologies developed in the fixed-base work onto a floating base. This experiment will be using our second-generation space robot model which is still under construction. Experiments include grasping and manipulating a floating object using a floating base.

The global control and navigation experiment seeks to demonstrate simultaneous control

of the robot manipulators and the robot base position so that tasks can be accomplished while the base is undergoing a controlled motion.

Multiple-vehicle cooperation project is a new activity that was started during this report period. The goal of this project is to demonstrate multiple free-floating robots working in teams to carry out tasks too difficult or complex for a single robot to perform. A third vehicle, similar to the one being built for the LEAP project, is also currently under construction, and it will bring the fleet of air cushion vehicles to three.

The LEAP activity was started about a year ago with the goal of providing a viable alternative to expendable gas thrusters for vehicle propulsion wherein the robot uses its manipulators to throw itself from place to place. This work will be carried out with a slightly revised version of second generation space robot which is currently under construction.

Adaptive LEAP project is a new activity that was started during the last report period. Because the successful execution of the LEAP technique requires an accurate model of the robot and payload mass properties, it was deemed an attractive testbed for adaptive control technology. Initial studies are underway to evaluate various adaptive control algorithms.

The chapters that follow give detailed progress and status reports on a project-by-project basis.

Chapter 2

Fixed-Base Cooperative Manipulation Experiment

Stan Schneider

2.1 Introduction

To accelerate our development of multi-armed, free-flying satellite manipulators, we have developed a fixed-base cooperative manipulation facility. Although the manipulator arms are fixed to a rigid base, they manipulate free-flying objects. This facility allows us to experiment quickly with cooperative algorithms, expediting our study of space-based manipulation and assembly. This section describes the progress made to date in our research on cooperative manipulation.

2.1.1 Progress Summary

During this report period, much of the theory developed over the last three years was implemented. In particular, the object impedance control, state table programming, real-time vision, and graphical user interface modules were installed and demonstrated.

Our initial research effort into cooperative manipulation is now complete. The vision system is capable of identifying, tracking, and capturing a moving object. We have demonstrated high-performance cooperative control, both in "free" motion, and when the manipulated object is in contact with its environment. Automatic capture, docking (connector insertion), withdrawal, and throwing functions are supported by the strategic command module. The mouse-based graphical user interface allows an operator to direct the activities of the system at the conceptual level. The operator commands only object motions; the arm actions required to effect these motions need not be specified. This design allows simple specification of many tasks; in particular, simple assembly operations can be easily accomplished. Each of these functions has been fully demonstrated.

More specifically, the major accomplishments of the period March, 1988 through August, 1988 were:

- Implemented and demonstrated dual arm cooperative object impedance control. This was shown to be an effective dynamic control methodology for cooperative manipulation.
- Implemented the real-time vision system point tracking algorithm.
- Developed algorithms for identification and tracking of moving objects. Demonstrated vision-guided dual-arm intercept and capture.
- Implemented and demonstrated a novel robot programming methodology—state table programming. This forms the heart of the strategic control module.
- Designed and installed an interactive graphical user interface to provide conceptual level system command.
- Demonstrated the operational cooperative system under user control.

Not all of our proposed approaches were successful; the client-server real-time software structuring concept [6] was finally abandoned as impractical. A simpler—and more successful—software structuring approach was implemented; it is outlined below.

2.1.2 Background: Research Goals

Space construction requires the manipulation of large, delicate objects. Single manipulator arms are incapable of quickly maneuvering these objects without exerting large local torques. Multiple cooperating arms do not suffer from this limitation. Unfortunately, cooperative robotic manipulation technology is not yet well understood. The goal of this project is to study the problem of cooperative manipulation in a weightless environment, and to demonstrate experimentally a cooperative robotic assembly.

Four aspects are to be studied in detail:

- The dynamic control of multiple arm manipulation systems
- The utilization of video "vision" data for real-time control
- Real-time software structuring for cooperative robotic systems
- User interfacing: the acquisition and utilization of strategic commands

2.2 Facility Development

2.2.1 Mechanical Hardware

The fixed-base cooperation facility consists of a pair of two-link manipulators, affixed to the side of a "small" granite table (4 feet × 8 feet). Each arm is of the popular SCARA configuration—basically anthropomorphic, with vertical-axis, revolute "shoulder" and "elbow" joints. The arms are capable of motion in the plane of the table, and can interact

with objects floating on air-cushions on the granite surface. This facility is essentially complete—no major additions or modifications occurred during the report period.

2.2.2 Computer System

Our real-time computer system combines a proven UNIX development environment with high-performance real-time processing hardware. Motorola 68020/68881 single board processors running the pSOS real-time kernel provide inexpensive real-time processing power. VME bus shared-memory communications permit efficient multiprocessor operation. The real-time processors are linked, via the VME bus, to our Sun/3 engineering workstations. Thus, we benefit from Sun's superb programming environment, while providing the capacity for relatively cheap, unlimited processing expansion. This system is also essentially complete. Only evolutionary modifications to the various software modules were performed during the report period.

2.2.3 Calibration

During the last period, automated sensor calibration programs were developed for: joint angular positions, joint pseudo-velocities, endpoint forces, and motor torque outputs. During this period, the vision system was installed as a major new sensor system. This posed two new calibration problems: calibration of the vision system itself, and sensor fusion with the endpoint information provided by the joint angle sensors. Since these are both logically vision system issues, they are discussed in the vision system section below.

2.3 Multiprocessor Real-time Software Environment

During this report period, the client-server real-time structuring methodology described in the original proposal was abandoned. A simpler—and more successful—software structuring approach was implemented. This section discusses the reasons for this action. This section also presents an overview of the software structure as implemented in the "final" operational system.

2.3.1 Real-time software environment

Each real-time processor runs the pSOS kernel. The pSOS kernel is a small, fast, priority-driven multi-tasking kernel. The features used most heavily by our software structure are the multi-tasking scheduler, the inter-process message facility, and the event-signal facility. The interested reader should consult [35] for details.

2.3.2 Client-Server Structuring

According to the client-server paradigm, the real-time software is divided into small, independently executing modules, each with a well-defined function in the controller data flow path. The modules communicate their data to other modules via message passing. For example, one processor (the server) might be assigned to read and process the analog inputs. This module is responsible for pre-processing the incoming data, and maintaining its integrity. Client processes—possibly running on a remote CPU—request information of the server when they require it. The pre-processed data is then sent via a message to the client process. A significant advantage of this scheme is that changes to the analog (server) environment are well isolated from the client code—the system is highly modular.

As noted in the previous report, the biggest disadvantage with this structure is the loading on the server—the processor responsible for reading and pre-processing the analog inputs. The fixed-base facility has a three-processor computer system. The processing load divides naturally as one processor to calculate each arm's dynamics, etc., and one processor for "system" level tasks, such as vision, object dynamics, etc. With the vision system and the object impedance controller active, the "system" processor is already heavily loaded. Using it also to pre-process sensor data is simply not reasonable.

A second disadvantage is the communication overhead incurred. As noted in the last report, we were able to reduce this to an acceptable level. However, the shared-memory structure outlined below proved far superior regardless. The authors feel that client-server structuring is still attractive for some situations: when independent processing power is available to dedicate to sensor processing, sensors require significant pre-processing, small delays in data presentation are not significant, and the extra overhead is tolerable. With the exception of the vision system, none of these conditions exists for the processing load of this system. It was judged better to adapt the vision system to a simpler structure, rather than diminish the performance of the entire system.

2.3.3 Shared-Memory Structuring

Instead of the client-server data structuring, a much simpler global shared-memory structure was adopted. Under this paradigm, the controller data flow paths are always executed by a single stream of execution, on a single processor. Thus, there is no data communication overhead within the loop. Data communications to external modules must still occur; these are accomplished via simple global data structures. An interprocessor lock gate is provided with each structure. Each communicating module is responsible for obtaining the lock before accessing the data.

Note that command or temporal information flow is not efficient with shared data structures. This type of communication is handled via communication "channels," described below.

This scheme has several disadvantages. First, it obscures the responsibility for maintaining the integrity of the data. This did not prove to be a significant problem for our system. A more secure system (although not nearly as secure as the client-server system) would result from installing active (i.e. callable subroutine) modules in each processor to mediate the data access. This would result in some additional overhead; it should not be significant.

Second, the shared-memory scheme is less modular. Changes to any part of the global data structure affect every module that accesses any part of the structure. Worse, any module that "breaks the rules" and, for instance, gets the lock and holds it, can bring the whole system a halt. Of course, both these effects could be ameliorated by installing active mediation modules.

This scheme does, however, have one redeeming quality; it's fast. If each accessing module keeps its access time small, communication overhead is practically eliminated. It also fits the processing load of the fixed-base facility nicely—the "arm" processors are relatively lightly loaded compared to the "system"processor, and can easily handle the additional load of reading and processing the analog inputs.

2.3.4 Inter-processor Communications

While simple shared-memory structures are sufficient for data communication, other types of information must also be transmitted. For instance, temporal information ("the data in structure xxx is ready now"), and commands ("switch to independent arm operation mode" or "catch that object") are difficult to convey with global data structures. To handle this type of information flow, we have also installed an inter-processor (or inter-process) communications facility, called "channels". A channel is simply a byte-stream two-way data flow—analogous to Unix "sockets" or "pipes"—with optional interrupt notification.

Channels can be created by any process (on any processor) in the system, and attached (opened) by any other process. Two mechanisms are available for a receiving process to obtain data from a channel. First, if the data is temporally urgent, the receiving process can indicate that it will monitor a pSOS kernel "event", and check the channel immediately when the event (and its associated interrupt) occurs. The sending process can then issue a special command to the channel to cause the receiver to be activated. Second, the

receiver may simply poll the channel at its leisure to determine if any new data is present. This paradigm allows the receiver to control the rate of data flow. Combinations of these techniques are explicitly allowed; the receiver may, for instance, wait for an event, but only for a fixed period of time. The sender thus has the option of sending messages in the background or indicating that an urgent need has arisen at any time.

2.4 Software structure

This section outlines the system software structure. The reader should refer to Figure 2.1 throughout this discussion.

The cooperating arms application software consists of four major modules. The modules are: dynamic control, vision, strategic control, and user interface. Execution of these modules is spread over three real-time 68020 processors, and the Sun graphical workstation.

The dynamic control module is responsible for reading the various sensors and calculating the actuator torques required to produce the desired system behavior. This module is further divided into three sub-modules as shown in Figure 2.1: the object impedance controller, and the two arm dynamic controllers.

The vision module is responsible for interpreting the incoming video pixel data, and disseminating the object and arm endpoint positions.

The strategic control module is responsible for the overall command of the system. It fields high-level requests from the user interface module, and translates them into sequences of primitives that the dynamic control module can implement. It also monitors the various conditions and activities of the system and directs appropriate response actions.

Finally, the user interface collect conceptual-level commands from the operator, and communicate them to the strategic controller.

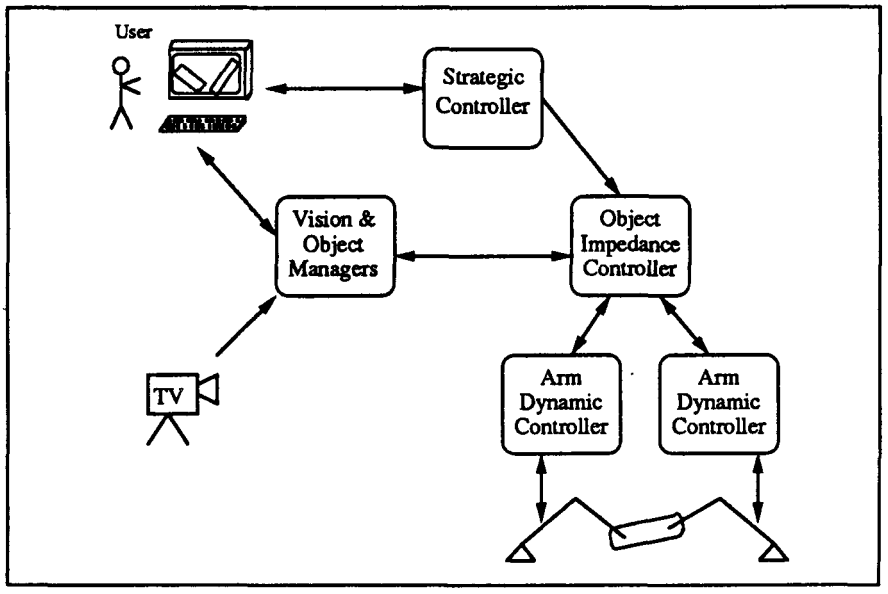


Figure 2.1: System Structure

2.4.1 Structure Chart Format

More detailed analysis of each module's structure is presented with the module descriptions below. The design is multi-process in nature; concurrency is employed to help modularize the code and permit asynchronous operation. To aid the reader in understanding the inter-process relationships, software structure charts will be presented in each section.

The structure charts utilize a common shape-encoding, presented in Figure 2.2. Rounded corner boxes denote any general "module", any set of related subroutines. Module routines may be accessed by many processes asynchronously. Square boxes with rounded tops denote independently executing processes. Rounded tops with the letters "ISP" indicate execution flow within interrupt service procedures. Named square boxes indicate inter-processor shared memory structures.

Example meanings of arrows connecting the boxes are also indicated in Figure 2.2. The arrows may have different meanings in different contexts [1]. For instance, an arrow into a module box indicates a subroutine call, a transaction representing flow of both information and execution control. An arrow out of a process box indicates a routine call-out. Arrows into process boxes indicate asynchronous information transfers: messages being sent to the process or the signaling of an event. Arrows into data structures indicate writes into the structure, arrows out of the structure indicate data reads.

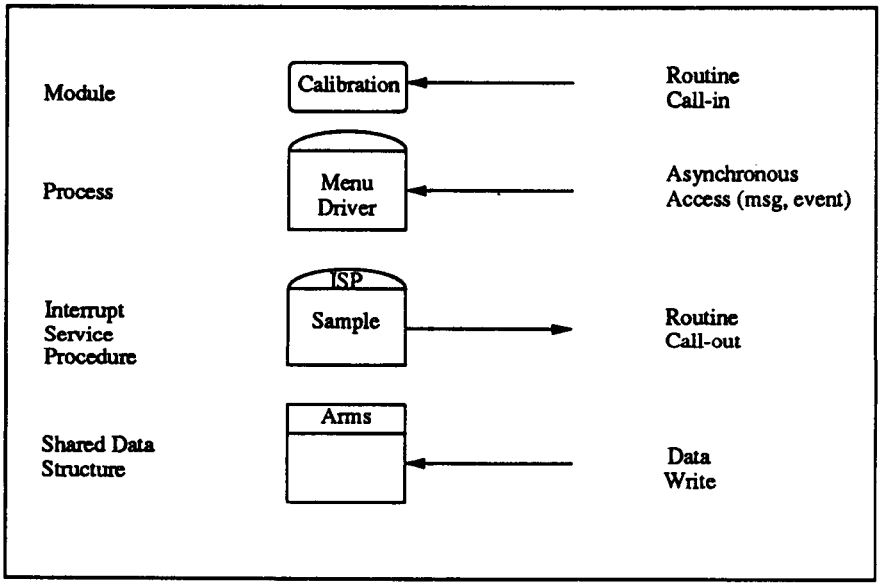


Figure 2.2: Structure Chart Key

[1] This is unfortunate, but multiple arrow types make the figures very complex.

2.5 Strategic Control

The strategic control module was also implemented during this report period[2]. The strategic control module is based on a finite state machine programming technique. It provides the user interface with a set of simple automated commands, allowing it to perform many simple tasks.

2.5.1 Strategic Control as a Module in the Larger System

The purpose of the strategic controller is to provide an interface between the conceptual commands provided by the user interface[3] and the dynamic control module. An effective strategic control methodology has several important features: it encourages modular programming design, it allows simple specification of the actions required to complete the desired tasks, and it facilitates intuitive task programming. A strategic control module for a cooperative system must also be able to synchronize the motions of the manipulators. In addition, conceptual requests should be communicated as simply as possible, and the results of those requests should be reported promptly.

To be effective, the strategic controller must have access to a large variety of system data and activities. It must monitor the motions of the various objects in the workspace, control the activities of the manipulators, and interact with the conceptual interface to the user. In short, although strategic control clearly fits into the command hierarchy between dynamic control and conceptual command, from a data-flow perspective it spans all the levels.

2.5.2 Traditional Robotic Programming Techniques

Traditional robot programming takes one of three forms: teach programming, specialized robot programming languages, and explicit use of general-purpose languages. This section examines some of the merits and problems associated with each method.

Teach programming Teach programming requires the programmer to move the robot through the desired motions. The system then "plays back" the sequence of motions. Teaching is only effective for position control tasks. It is impossible to specify synchronization with other moving obstacles or manipulators. Several attempts to improve the means of teaching have been made, including one with cooperating robots [15]. These all still suffer from the basic weakness: it is very difficult to specify complex motions, time dependencies, and interacting conditions.

Special robot programming languages Several robot-specific programming languages are in use. Examples are VAL [33] and IBM's AML [38]. These languages allow the programmer to specify the manipulator's motion as a series of instructions. Motions that require waiting for a condition must be programmed as a loop, such as:

[2]The underlying theory was developed over the last few years.

[3]or an artificially intelligent command module

```
repeat {
    Move in the -X direction 0.01 meters;
} until ( Force X exceeds 2 Newtons )
```

This is a considerable improvement in the types of tasks that can be programmed; force conditions are also specifiable. Note that force control is *not* available; the best that can be done is to check for force levels that exceed given values.

Multiple conditions pose a more significant problem. The only recourse with standard programming techniques is to test for all conditions in the loop, as in:

```
repeat {
    Move in the -X direction 0.01 meters;
} until ( (Force X exceeds 2 Newtons) or (X < 0.2 meters) )
```

This quickly becomes unwieldy. If, for example, a sequence of moves must be completed in a limited period of time, each move in the sequence must incorporate the time-out test in its execution loop. As the tests become more sophisticated—and the sequences of moves more complex—this results in very confusing, hard to maintain software. An even more vexing problem is created if the condition test takes a long time to execute. The programmer is left with two bad choices: execute the test every loop and slow the motion considerably, or execute it every n^{th} loop (another messy addition to the loop code), and produce uneven motion.

IBM's AML language does allow the setting of overall error condition tests. If the error condition arises, a special code segment is executed to handle the condition. Unfortunately, normal occurrences of multiple pending conditions must be dealt with as above; there is no facility for changing the flow of execution asynchronously.

The LM robot programming system [23] implements a considerably more powerful set of constructs to support concurrent execution. A facility is provided to start a motion while the "main" execution flow continues. Synchronization primitives permit changing a motion in progress, or waiting for completion. In addition, LM defines a "guarded command" structure to address the multiple pending conditions problem. A parallel-execution facility[4] is also provided. The intent is to provide the ability to specify multiple programs for multiple robots.

While it is clear that LM provides a large set of parallel programming facilities, it does not provide the programmer with a simple structure to assist in managing the concurrency in the system. An integrated system involving multiple manipulators, real-time vision, and interactive operator control is a complex, event-driven environment. With a fundamentally sequential underlying programming paradigm, the burden of managing these asynchronous events is left to the programmer. For instance, "guarded commands" allow changing of the course of a trajectory. However, to change the flow of execution of the "main" program when an outstanding pending condition occurs requires sprinkling conditional tests throughout

[4]This implements a "cobegin { <blocklist> } coend;" structure, similar to Concurrent Pascal.

the "main" execution flow.

In addition, while the multiple-parallel-block execution structure is appropriate for coordinating multiple robots, it is considerably less attractive for execution of a tightly-coupled parallel control algorithm—such as a cooperative dynamic controller. Switching to a cooperative control mode upon the occurrence of some event is particularly awkward.

In summary, synchronization of multiple cooperating manipulators handling a single object is very difficult with traditional robot programming languages. Although a few attempts have been made [43], they consider only coordinated position control policies, and thus are of little practical use.

General-purpose language programming There have also been several implementations of motion control libraries written to facilitate programming in general-purpose languages [14]. These libraries usually provide useful data structure definitions—such as "transform"—and motion primitives to allow specification of motion sequences.

Although some concurrency issues are addressed, these libraries do not directly aid the programmer in utilizing the concurrency in the system. All still rely on a basically sequential user-code execution model. In RCCL [14], for example, the robot program consists of one main "user" process, and one background "control-task" process. The "user" process executes a sequential series of instructions, and communicates them to the "control-task" process, executing at the control loop sample rate. The only synchronization provided is via a "waitfor" primitive; no support for independently executing *user-written* processes is provided.

As a result, these implementations do not alleviate the multiple pending conditions problem. None provide support for multiple loops, nor multiple sample rates.

In addition, no support is provided for direct interfacing with other large real-time modules, such as sophisticated dynamic controllers, or vision systems[5]. These systems are naturally concurrent in nature; complex asynchronous interactions must be managed. A sequential execution stream model can not, for instance, deal effectively with multiple-arm synchronization, especially if the arms are running different programs on different processors.

2.5.3 The state table programming technique

This section presents an alternative programming methodology, referred to as state table programming. It is a form of explicit programming in a general-purpose language, but provides a structure the previous attempts lack.

The state table programming technique provides a coherent structure to guide the programmer in producing code that is easily interfaced to other real-time system modules. It directly exploits the facile multiple-process generation and communications provided by modern real-time kernels. In fact, management of multiple asynchronous processes is central to the structure of the system; the programmer is actively encouraged to divide the

[5]There are several integrated vision positioning and inspection interfaces [42, 24], but they do not address the *real-time* vision issues.

problem into small, independently executing programs. In addition, it is based on a very intuitive task description technique.

The state table programming technique is not a replacement for a library of robot control routines. Trajectory generation utilities and world modeling utilities are still required. The technique merely provides a frame-work that weaves the multiple streams of execution in the system into an easy-to-use structure.

State transition graphs State transition graphs provide a simple, intuitive means of visualizing the sequence of actions required to effect a task. A "state" is defined operationally: the system is always in some state, a new state is entered whenever the set of conditions (stimuli) the system can process changes. A state is usually characterized by the system performing a single operation, and waiting for some indication of its completion[6]. State transitions are indicated by arrows labeled with the event that causes the transition. When an event occurs, a transition routine is executed. The result of that routine determines the next state entered.

For example, Figure 2.3 presents a simplified series of steps required to catch a moving object. The system starts out in the "Idle" state. The receipt of the "Acquire" stimulus, presumably sent from the conceptual level, causes the system to enter one of two states: "Waiting" if the object is out of reach, or "Reaching", if not. While the system is in the "Reaching" state, the arms are executing an intercept trajectory. When the trajectory completes ("TrajComplete" event) , the arms track the object until the gripper endpoints match the targeted grip ports precisely. At this point, the system enters the "Gripping" state while the grippers engage. Finally, the catch is complete, and the "Manipulating" state is active until a "Release" command is received.

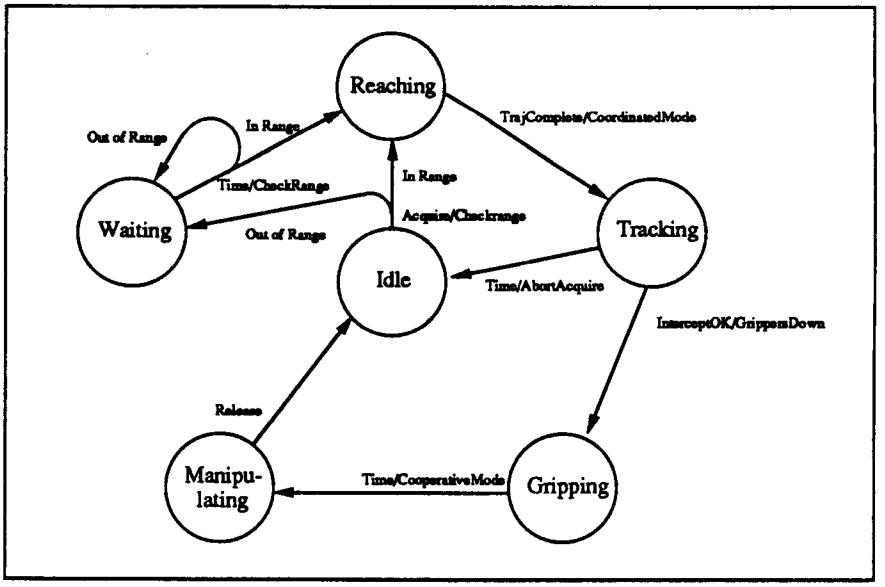


Figure 2.3: State Transition Graph - Catch Task

[6]The example below should help clarify this concept.

State	Stimulus	Transition Routine	Next States	
Idle	Acquire	CheckRange	Reaching	Waiting
Waiting	Time	CheckRange	Reaching	Waiting
Reaching	TrajComplete	CoordinatedMode	Tracking	
Tracking	InterceptOK	GrippersDown	Gripping	
	Time	AbortAcquire	Idle	
Gripping	Time	CooperativeMode	Manipulating	
Manipulating	Release	GrippersUp	Idle	

Table 2.1: State Transition Table

This is a rather simple example. Much more complex series of actions—with multiple branches—are easily representable.

The state table programming technique The state table programming method provides a simple means to implement the series of actions indicated in the state transition graph. A state table is constructed directly from the information in the state transition graph. Associated with each state are the events (stimuli) understood in that state. Each stimulus has a transition routine, and a list of "next states" that will be entered based on the value returned by the transition routine.

The state table entries corresponding to the states of Figure 2.3 are presented in table 2.1. The "Idle" state in this example understands only the "Acquire" stimulus. When "Acquire" is received, the routine "CheckRange" will execute. If "CheckRange" finds the object to be "in range", then "Reaching" will be the next state, otherwise, the next state will be "Waiting". Note that the same transition routine, "CheckRange", can be used in several different contexts. Its job is simple: check on the location of the object, and report the outcome. Note also that multiple pending conditions are handled very naturally, for example, the "Tracking" state waits for either "InterceptOK" or "Time".

Stimuli may be generated by any of the modules of the system, but most originate from one of two sources: command stimuli from the user interface, and condition-event stimuli from independently executing "helpers". The "Acquire" and "Release" stimuli are examples of the former. Several examples of the latter also occur in table 2.1. The "Time" stimulus is created by a simple process that simply sleeps for a specified time before sending its message. The "InterceptOK" stimulus is sent by a process that executes in the background, and simply checks to see if both arms match the gripper port positions. Other helpers (not shown in the table) perform "safety checks"; for example, the "OutOfRange" stimulus is sent by a process that checks every so often to insure that the object being acquired is not suddenly moved out of range.

Unexpected stimuli may be handled in either of two manners. First, if no stimulus matches the incoming stimulus, a default error handling routine is executed. Second, the special stimulus "AnyStimulus" may appear as the last entry in any state's table entry. All stimuli match this entry. The transition routine associated with "AnyStimulus" may then process the unexpected stimulus as required.

Advantages of state table programming State table programming provides an attractive basis for a strategic control module.

First, it strongly encourages modular program design. Each of the state transition routines defined above perform a single, well-defined function. Multiple pending conditions are easy to handle; in fact, they are encouraged by the stimulus list paradigm. The condition testing "helper" routines are also encouraged to be independent modules, with a single purpose. Note that even complicated tests may be performed at a slower rate as background tasks, since they can execute asynchronously. Exception handling is also simplified, as exception procedures can easily be specified on a state-by-state basis.

Second, the state table method provides an intuitive programming environment. State transition graphs are a very natural means of describing a task. Since they also provide a quick, visual overview of the entire program, they are easy to understand and modify. Translating the graphs to the state table format is also simple; the table format makes changes simple as well.

Third, synchronization of multiple manipulator actions, or of any concurrent events, is natural; *all* events are treated as asynchronous, and are handled within the framework of the implementation. Both polled conditions and transient events can be handled easily. Conditions that are more easily tested periodically can be polled by a simple "helper" process that sends a stimulus when the synchronizing event occurs. Conditions that are more clearly modeled as discrete events can be handled by defining synchronization states. An example of both types of synchronization is depicted in Figure 2.4. In both cases, the state table method provides the programmer with a natural means of specifying the relationship.

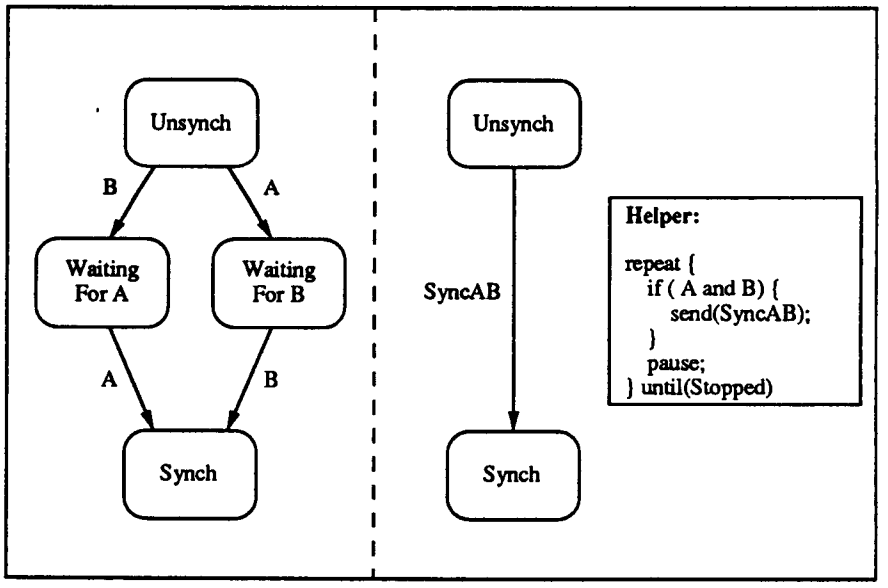


Figure 2.4: Synchronization Type Comparison

Finally, the stimulus event model directly supports a high-level request/response type interface. This has proven to be a very attractive paradigm for conceptual-level command

interaction. The user interface section discusses this in detail.

Disadvantages of state table programming State table programming does have costs, mostly because it naturally produces multiple-process programs. Understanding and debugging a multiple-thread-of-execution program requires some exposure to operating system and concurrent programming principles. This is unfortunate, but in some sense unavoidable; a complex robotic system is a naturally concurrent system. Application of the hard-learned lessons of Computer Science is entirely appropriate.

Also, events from various sources arise asynchronously, and must be woven into a synchronous stream of state transitions. Fortunately, modern message-passing real-time kernels provide exactly this service; if the stimuli are sent as messages to a state machine driver process, the message queuing facility performs the synchronization.

2.5.4 Implementation

This section describes the strategic controller implementation for our cooperating manipulator system.

Software Structure The strategic controller structure is presented as Figure 2.5. The heart of the system is the Finite State Machine (FSM) driver. The FSM driver is an independently executing process. It acts as a central command post, receiving messages from many sources in the system and taking the appropriate action. The user interface daemon sends command stimuli from the user interface. The arms daemon may report conditions detected by the arm dynamic controllers. The various helpers exist solely to report specific conditions when they occur [7]. Two helpers are shown: the time helper simply reports time-out events, and the range helper checks for unexpected object positions.

Each message contains a stimulus code; the FSM driver uses that code to reference the state table, and select a state transition routine to execute. The state transition routines perform the actual work: they change controller modes, start and stop helper processes, and interact with the trajectory generation module.

State transition graph Figure 2.6 depicts an overview of the state transition graph of the cooperative arms system. There are three states shown: Idle, Manipulating, and Inserted. These three are the only states that the system can remain in for an indefinite amount of time[8]. "Idle" corresponds to the arms at rest under independent control. When the arms are cooperatively moving an object in free space, the system is in the "Manipulating" state. Finally, "Inserted" state corresponds to the arms holding the object with a connector inserted into some other object, either just after completing the connection, or just prior to disconnection.

Seven major tasks are supported. They are:

[7] Helpers are transient; they are only active when needed.

[8] Under normal operation.

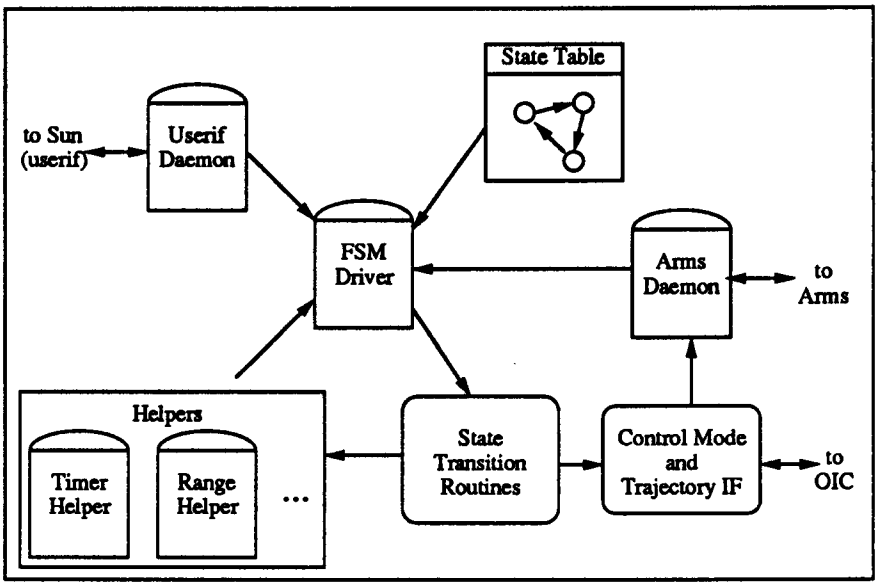


Figure 2.5: Strategic Controller Software Structure

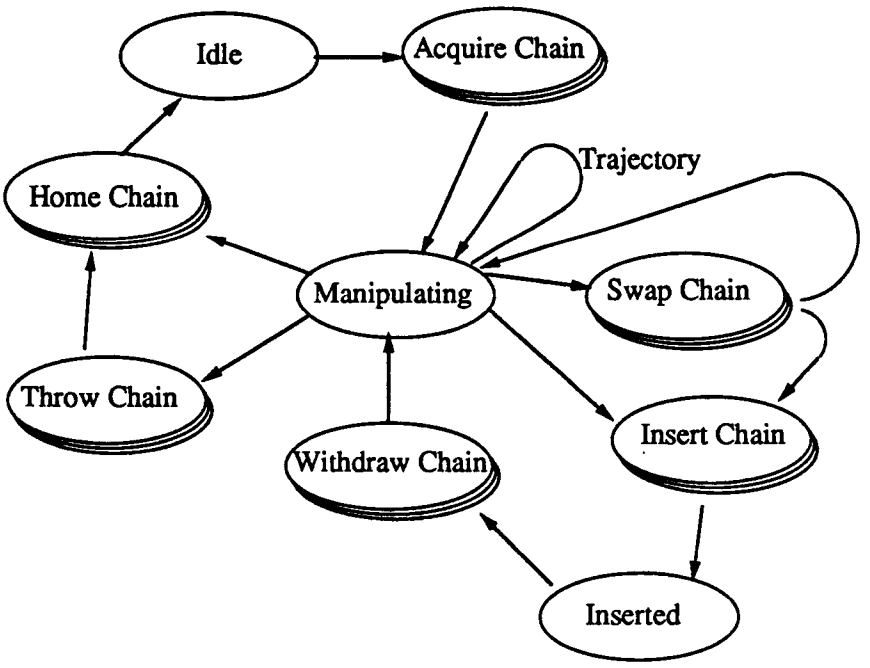


Figure 2.6: Overall State Transition Graph for Experimental System

- *Acquire* performs the object capture
- *Swap* exchanges the arm grip positions
- *Insert* performs a connector insertion
- *Withdraw* unlatches the connector
- *Throw* releases the object on a controlled trajectory
- *Home* releases the object and returns the arms to their idle positions
- *Trajectory* supports point-to-point cooperative motions.

All these except "Trajectory" require a "chain" of state transitions; they are multi-step operations.

2.5.5 Future Additions

The addition of callable state sub-chains would add considerable power to the state programming implementation presented here. The implementation above actually uses several sub-chain "calls"; for instance, the insert chain calls the "swap" sub-chain if the grip must be re-adjusted. These are implemented in the current version as simple "test and branch" code segments. A more general facility would greatly facilitate sub-chain usage.

2.6 User Interface

During the last report period, functionality of the graphical user interface was expanded considerably. The user is now able to direct many system activities at a conceptual level. For unexpected or unusual tasks, manual manipulation modes are still fully supported.

2.6.1 The User Interface as a Module in the Larger System

The purpose of the user interface is to gather conceptual commands from the user, and communicate them to the strategic control module.

The most important function of the user interface is to present a clear and intuitive means for the user to specify his wishes. Simple or often repeated motions should be automated, or at least reduced to a few short steps. The progress of the system in executing those commands should also be displayed in a simple, clear manner.

The user interface and the strategic controller are quite inter-dependent; design philosophies adopted in one directly effect the effectiveness of the other. For instance, the set of commands supported by the strategic controller should be designed to provide a powerful set of primitives to accomplish most of what needs to be done. The user interface then need only provide a simple means of specifying those command sequences.

2.6.2 Tele-operation versus Autonomous Control

Tele-operation and "autonomous" operation are the two main approaches to robotic manipulation direction in the literature. In this section, we compare and contrast these approaches, and discuss their merits.

Tele-operation Tele-operation is defined here as control with the operator "in the loop". The operator is responsible for directly controlling the system activities at all times. Tele-operation has two main advantages. First, since the system is under direct user control, it is strategically capable of performing any operation that it can physically execute. This makes tele-operation very flexible. All the reasoning abilities of the human operator are immediately available. Second, the sensor and sensor processing systems can usually be quite simple. The human user is amazingly adept at gleaning detailed system information from whatever sensors are easily available.

Unfortunately, tele-operation also has several disadvantages. Since the user is in direct control of each operation, it is difficult to get assistance from the computer. Even simple, repetitive operations must be done manually. Precision work is difficult, or not possible. Operators must usually train extensively, and then devote their complete attention to the system operation. Finally, since the operator has complete system control, it is difficult to build safe-guards into the system operation. Operator error is difficult to guard against.

Autonomous operation Autonomy is difficult to define. Perhaps the best definition is by the commands it recognizes; a system is more autonomous if it can accept higher-level conceptual commands with little supervision. It is less autonomous if it must be

constantly directed. Thus, we can characterize a system's degree of autonomy by the types of commands it requires, and how often it requires new user input.

The advantages of higher degrees of autonomy are obvious: the user is relieved of the mundane decisions, and can concentrate on more complex strategic issues. Non-expert users are able to utilize at least the simplest operational capabilities of the system. Also, since much of the system capability is specified before its actual use, the experience and care of the system designer can be used to program many actions. Thus, precise or dangerous operations can be directly assisted by the computer control system.

Disadvantages of high degrees of autonomy are also clear. Foremost, complex autonomous actions require a level of artificial intelligence (AI) proficiency that is not currently available. Alternatively, some autonomous actions can be pre-programmed. This is quite effective for simple tasks, but rapidly becomes overwhelming when the tasks become complex. It is also impossible to pre-conceive of all possible combinations of tasks.

Our approach We have tried, in this research, to take a centrist approach. The simple, repetitive tasks required to effect an assembly—acquiring a part, fastening a connector, etc.—are pre-programmed, and therefore quite autonomous. More complex motions can then be specified as sequences of these actions. In recognition of the need for unanticipatable actions, a manual operation mode is also provided. This combination should allow the completion of most assembly tasks.

Note that no attempt has been made at artificially intelligent assistance. This is intentional; we are attempting to create an environment where the user is tapped only for his reasoning and analysis abilities. This allows the study of the division of labor between pre-programmed primitives and reasoned actions. It is our contention (and hope) that this will also provide insight into a natural interface for AI interaction in the future, as a (perhaps gradual) replacement for the user's reasoning capacities.

2.6.3 Interface to the Real-time System

The user interface must communicate with the vision system and the strategic controller. However, these modules execute on different processors. The graphical user interface executes on the Sun workstation, while the others reside on one of the real-time processors.

A special real-time process, the "user daemon", exists solely to arbitrate communication requests between the real-time modules and the user interface. The user daemon communicates with the user interface over a "channel" (see section 2.3.4) via the VME bus shared memory. Although in reality all communications pass through the user daemon, this transaction is transparent to the underlying modules. The remainder of this section will describe the communications as if they were direct.

2.6.3.1 Interface to the vision system

The user interface interaction with the vision system is very simple. Whenever the user interface is "idle", it sends a request to the vision system for a screen update. The vision system then simply sends a list of the status of all the objects currently being tracked. These

positions are used to update the user screen display. Thus, the user is always presented with a current representation of the manipulator's environment.

2.6.3.2 Interface to the Strategic Controller

This section presents the communications protocol between the user interface and strategic controller.

The request / response model The nature of the protocol is a request/response pattern; the user interface requests an action, and the strategic controller returns a status response when the action completes. This very simple (and obvious) communication paradigm allows considerable flexibility.

The design of the strategic control module directly supports this model. User requests are simply passed directly to the finite state machine driver in the same manner as any other system stimulus. Asynchronous requests do not pose a special case, since all stimuli are assumed to be asynchronous. Typically, each stimulus causes one "state chain" (see section 2.5.4) to execute. The responses to the user interface are the responsibility of the state transition routines that exit the chain. Since there are usually only a few of these—one for a successful exit, and one for each type of error exit—the status reporting code is very simple.

An example: acquire A typical example of a request cycle is presented by the "Acquire" task. To initiate an acquire operation, the user simply "clicks on" one of the objects displayed on his screen. This action sends the name of the object, along with an acquire stimulus, to the strategic controller. Consequentially, the acquire task chain is executed. When the acquire operation is complete, an "object acquired" status message is returned. If the stimulus is not recognized by the current system state, or an error occurs during the operation, an "acquire failed" status is returned.

2.6.4 Modes of Operation

The user interface supports two modes of operation: automatic mode and manual mode. The user can select either mode by clicking on a button displayed on the screen. The modes effect only the manipulation functions of the system; the acquire and release functions are identical.

Automatic mode The purpose of automatic mode is to facilitate the most common operations. When automatic mode is active, two "views" of the object being manipulated are displayed. The actual position of the object is displayed, as usual, by a solid-lined figure. In addition, a "desired" position of the object is drawn with dashed lines. The user can move the broken-line (ghost) object by clicking on it and dragging it around the screen. A typical portion of a user display is shown in Figure 2.7.

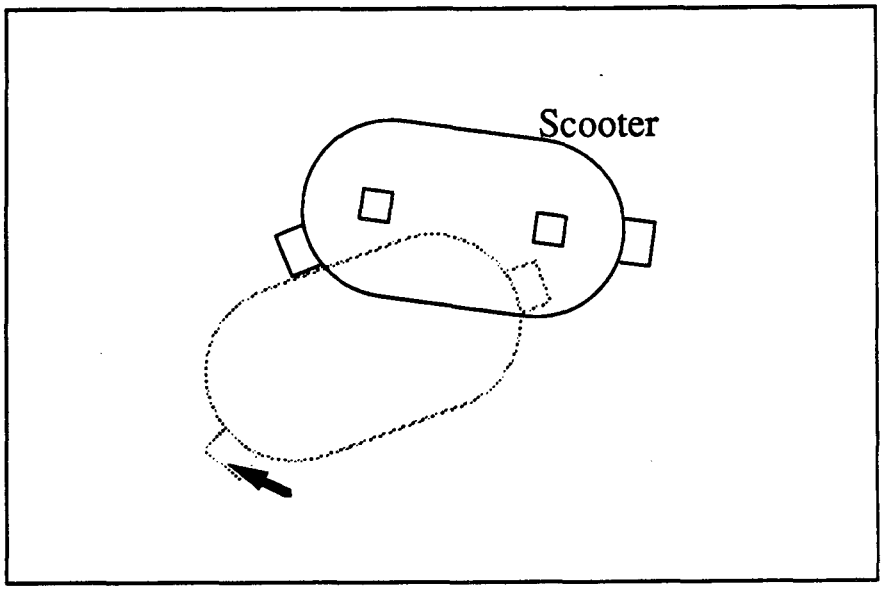


Figure 2.7: Automatic Mode Display with Previewing "Ghost"

At all times, the ghost object represents the state that the system will attempt to produce if the mouse button is released. It is thus a one-move preview of the new state of the system.

The user interface also helps the operator maneuver the object. For instance, if a male connector on the ghost object is moved near to a female connector on another object, the ghost object "snaps" to the connected position. To perform a connection operation, the user can simply point to any connector on the object, and drag it to the matching connector in the workspace. The display will show the ghost object in the final connected position. When the move is confirmed, the system performs the sequence of actions required to make the connection, and reports the status back to the user. This allows quick, simple assembly operations.

Manual mode It is not possible to anticipate all possible actions a user may want to perform. To allow completion of actions not anticipated, a manual "tele-operated" mode is available. In manual mode, no ghost object is displayed. Instead, manual mode allows direct access to the object impedance controller. When an object is being manipulated, the zero point of the impedance "virtual spring" is displayed. The user can cause motion to occur by simply dragging the zero point around the screen. An example screen display is presented as figure 2.8.

Previewing (ghost display) is disabled during manual operation; each operator motion is transmitted immediately to the real-time system. The object will thus follow the user's mouse commands in real-time.

Facilities are also provided to move the object's effective center of mass and change the impedance gains[9]. This permits the user to exercise direct control of the system dynamic

[9]See section 2.7 for details.

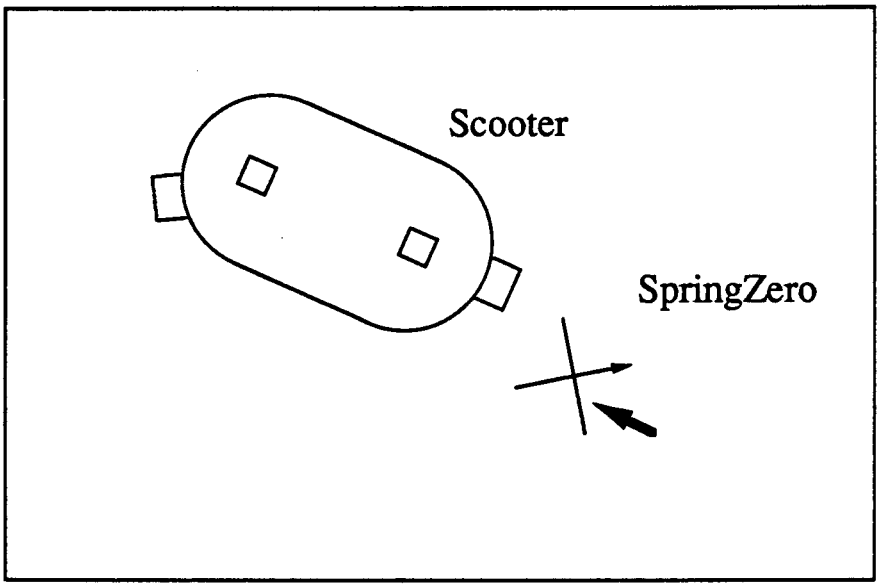


Figure 2.8: Manual Mode Display with Virtual "Spring"

response.

2.6.5 A brief operational description

This section presents a brief description of the user interface operation, and steps through an example application.

A typical screen display A typical screen display is presented as Figure 2.9. The screen is divided into three sections. The large lower section depicts the manipulator workspace in iconic form; the activities of the system are visually displayed here. The upper left window displays the system status. The first line in this window gives a short verbal description of the systems activity. A system control panel forms the upper right section.

In the example shown, there are three objects in the vision system's field of view. Scooter is the floating air-cushion object. Scooter has two gripper ports, and two male connectors. Multibase and Dock are both stationary objects with female connectors. The arms are currently holding Scooter, as evidenced by the presence of the Scooter ghost image. The user has just dragged the ghost's right connector over to Multibase's rightmost connector. The status line indicates that the insertion of one of Scooter's connectors into one of Multibase's connectors is in progress.

An example task: install a part An example of a typical task sequence is depicted in Figures 2.10 through 2.11. For the sake of this example, suppose that "Multibase" is affixed to a mobile robot, and represents a series of attach points for holding miscellaneous items. "Scooter" is a part that is to be installed into a remote module, represented by "Dock".

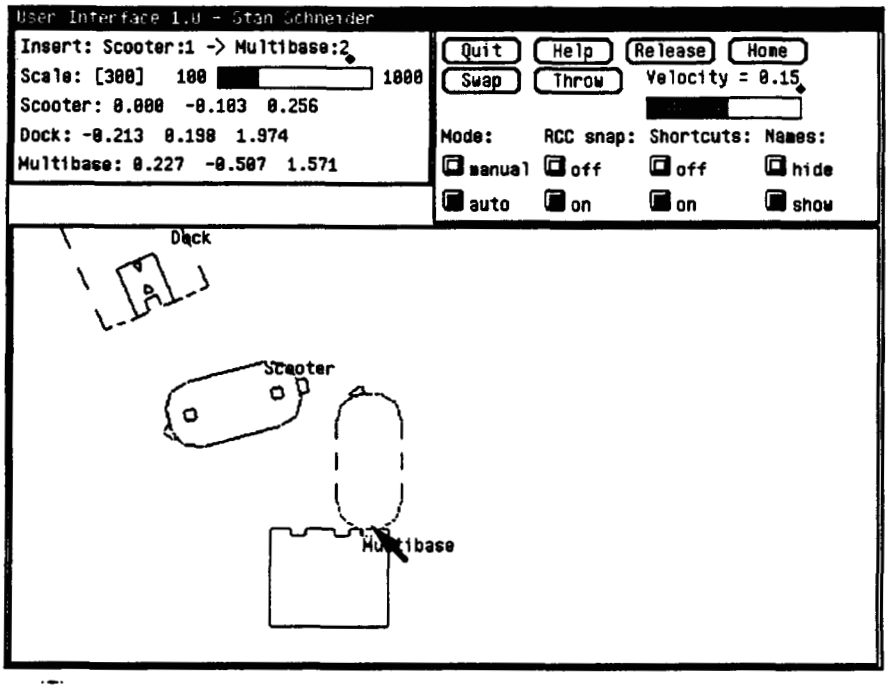


Figure 2.9: Example User Interface Screen Display

In Figure 2.10, the part is approaching the robot system[10]. The operator has just indicated Scooter is to be grasped, thus the "Acquiring Scooter" status message in the top left corner. In Figure 2.11, the operator indicates that Scooter should be affixed to the robot's base. Next, the robot is directed to navigate to the vicinity of Dock (navigational control is not discussed here). When Dock is in view, the operator indicates that the new part (Scooter) is to be installed. Figure 2.12 shows the first stage of that action. Finally, in Figure 2.13, the part has been released, and the installation is complete.

This entire procedure was accomplished with only four simple mouse motions. (Click on the approaching Scooter, connect Scooter to Multibase, connect Scooter to Dock, release.) Most of the assembly details—such as how to attach and detach connectors, how fast to approach the docking connector, etc.—are completely automated.

[10]Or, equivalently, the robot is approaching the part.

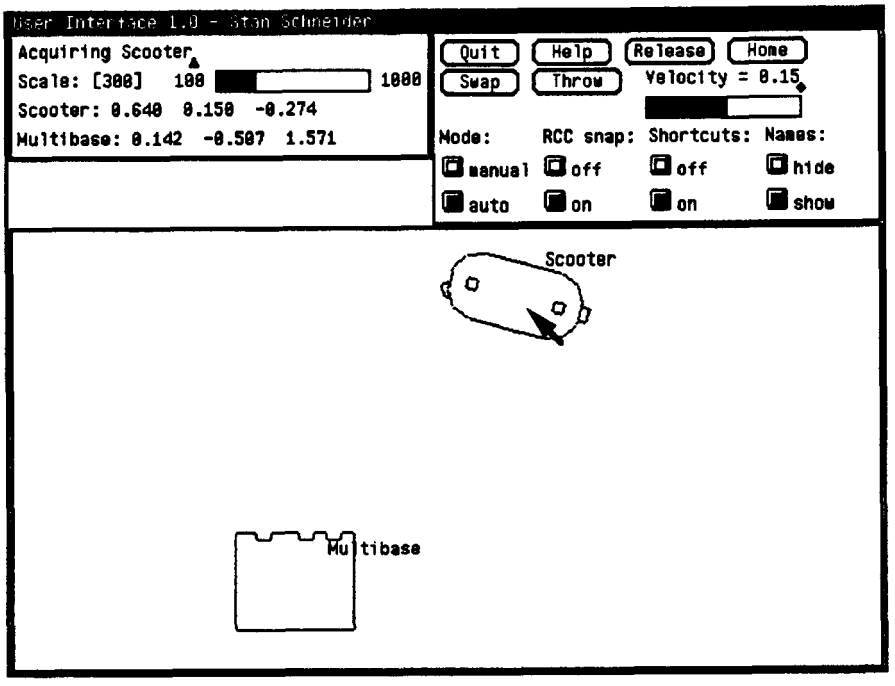


Figure 2.10: Installation Demonstration Example: 1

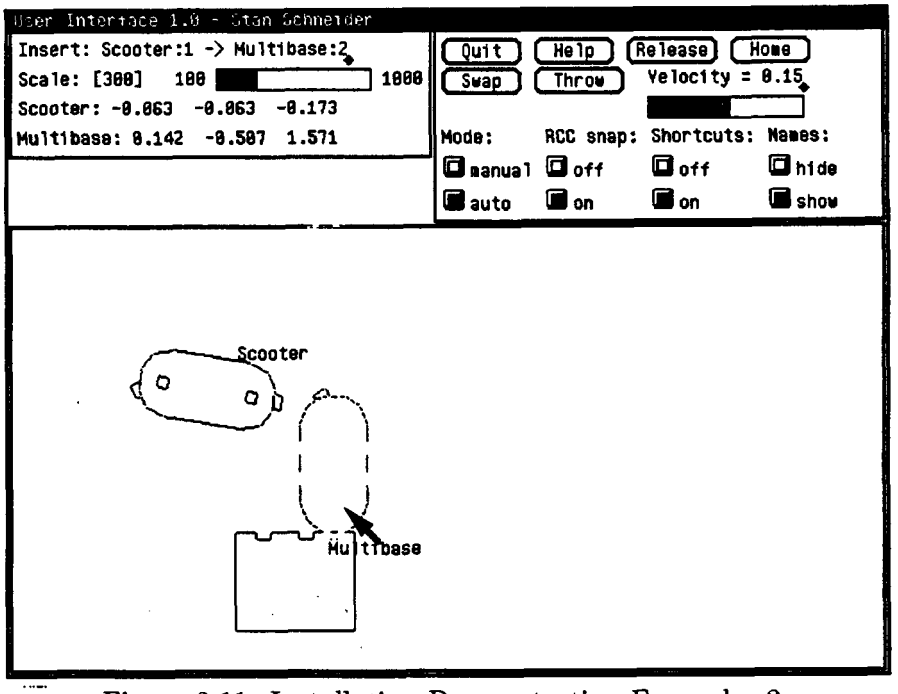


Figure 2.11: Installation Demonstration Example: 2

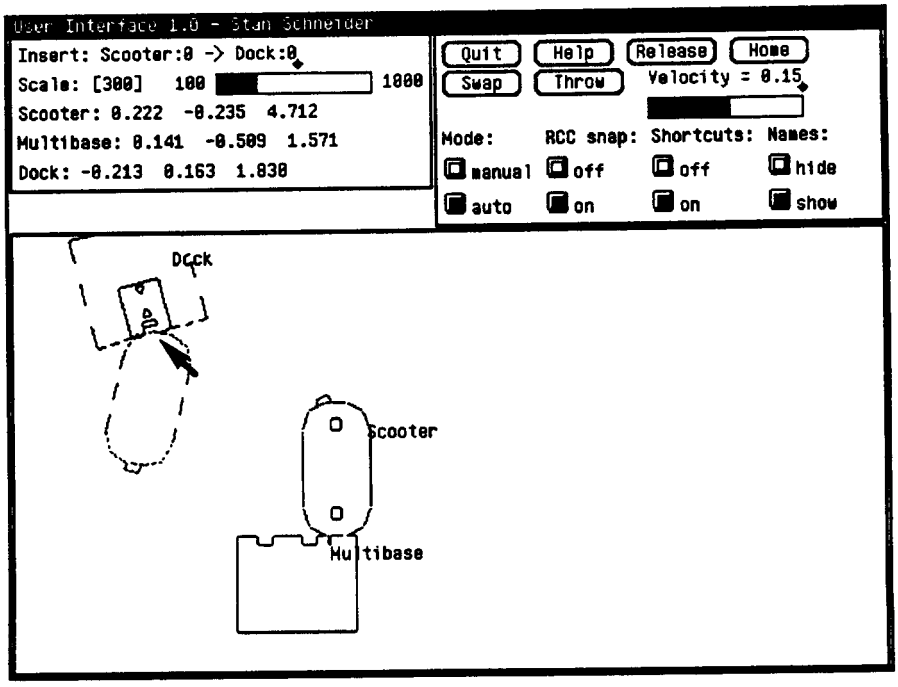


Figure 2.12: Installation Demonstration Example: 3

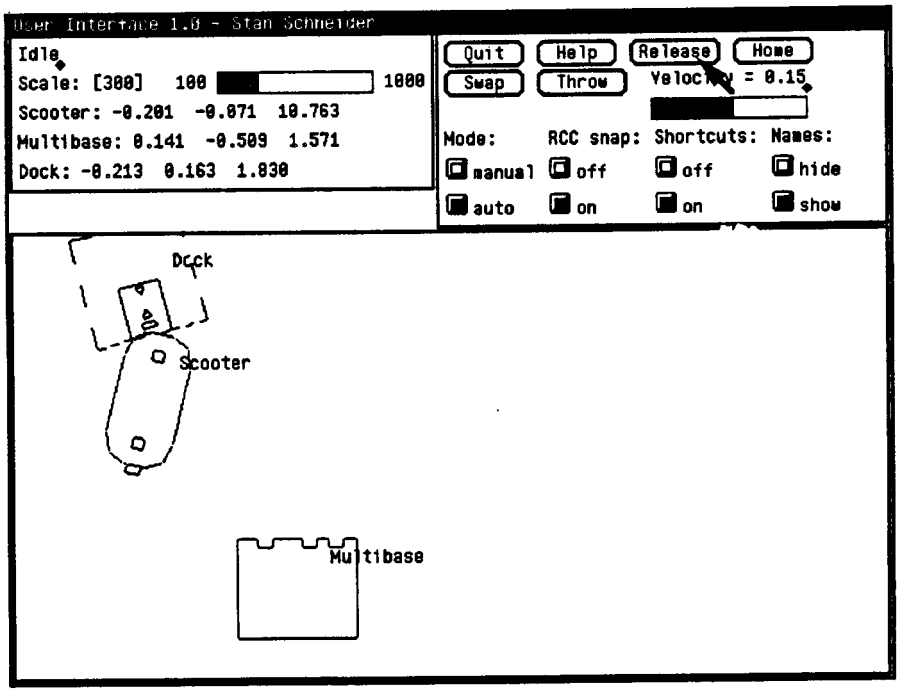


Figure 2.13: Installation Demonstration Example: 4

2.7 Dynamic Control

Many manipulative tasks are performed more easily with cooperative manipulators. For example, single manipulator arms are incapable of manipulating large objects without exerting large local torques. Precise manipulation of extended objects is difficult without utilizing the large mechanical advantage offered by separating the grasp points. Multiple cooperating arms do not suffer from these limitations.

Unfortunately, multiply-armed robotic systems are considerably more complex. The dynamic equations of motion of the closed-chain system are considerably less tractable. Strategic control of the system interactions is much more difficult; a consistent set of desired motions must be specified. As of yet, no satisfactory method of precise dynamic control coupled with a simple strategic command interface has been developed and experimentally demonstrated.

This section outlines a strategy for the control of a cooperative robotic system that permits high performance dynamic motion control, while also allowing direct control of environmental interactions. This is accomplished by controlling the manipulated *object* to react to external environmental stimuli with a programmable impedance. This facilitates motion direction by presenting a simple yet powerful interface; the strategic controller need only specify the impedance. Although "exact" inertial force compensation is achieved, the control structure does not require explicit formulation of the closed-chain dynamic equations of motion, and is amenable to parallel computation. Object internal forces are explicitly controlled. The object impedance controller has been implemented on a multiprocessor real-time computer system. Experimental results for a dual two-link arm robotic system are presented to verify the controller's performance, both for free-motion slews and environmental contact.

2.7.1 Background, Philosophy, and Literature Review

This section discusses our philosophy of cooperative manipulation. We analyze the cooperative manipulation control problem, and point out the desirable features of a cooperative controller. Finally, we review the recent results and proposals in the literature.

2.7.1.1 The dynamic controller as a module in the larger system

In a complex robotic system, the dynamic controller is an interface between "mid-level" commands (such as "slew to 2.3, 4.2") and the system's sensors and actuators. As such, the salient features of an effective controller are its dynamic performance capabilities, and the language set of "mid-level" commands it presents to the strategic control module. Of course, there are several ease of implementation issues of importance as well.

2.7.1.2 Control policies vs. control implementations

The distinction is made here between dynamic control *policies* and control *implementations*. We define a control policy as the interface the dynamic controller presents to the next higher level, the aforementioned "mid-level" instruction set. The policy determines the specifiable ideal behavior of the system. The control implementation, on the other hand, is the method by which controller actually causes the specified behavior to occur. Thus, for example, a perfect implementation of a position control policy would maintain a desired position regardless of external forces. While this division is somewhat artificial, it does provide a framework to discuss the various techniques being used and proposed in the literature. It also will lead us to several new approaches to the problem.

2.7.1.3 Policy objectives

A controller's policy determines its method of *directing* motion. An effective policy for specifying robotic motion should—at a minimum—provide for the specification of trajectories for the manipulated object to follow. It should also allow for the specification of the forces of interaction between the manipulated object and its environment. Lastly, and most importantly, it should present a simple, intuitive interface to the strategic controller. The instruction set should be powerful, yet easy to specify. The resulting interface will then allow simple specification of the actions required to perform most tasks demanded of a robotic system: transport, assembly, etc.

2.7.1.4 Implementation objectives

The controller's implementation is the means by which its policy is enforced. Several aspects of an implementation determine its efficacy. First, to permit accurate trajectory tracking performance, a control implementation should compensate for the inertia forces of the manipulator/object system. Second, an implementation should resolve the natural redundancy of the system in a beneficial manner. Two types of redundancy are present;

the same object behavior may be maintainable with different internal forces (static redundancy), and in multiple manipulator positions (kinematic redundancy). Third, a controller that interacts with its environment should incorporate force sensing, to more accurately control the forces of interaction. Fourth, a controller that is amenable to a parallel processing approach allows better performance with less expensive computational resources. Finally, an implementation should be verified experimentally.

2.7.1.5 A review of approaches

This section describes the various approaches to cooperative manipulation taken in previous work, as well as proposing several new approaches. The merits of both the policies and implementations of each approach are discussed. We note here that our system does not possess kinematic redundancy; thus we do not consider this type of redundancy resolution. We note also that several of these approaches were implemented on our experimental manipulators; the experimental results section discusses their performance.

Single arm (endpoint) control policies Currently, there are three major policies in use in traditional robotics—position control, hybrid position/force control, and impedance control. Simple position control policies are useful for transport problems, but often inadequate for assembly tasks. Adding force control increases environmental interaction capabilities. Hybrid control strategies [29] permit simultaneous intermixed force and position control on orthogonal axes.

Impedance control differs from these policies in that instead of controlling one state variable—position, velocity or force—it enforces a relation between them. This type of control has many desirable attributes; Neville Hogan's definitive three-part paper [16] presents them in detail. Chief among them are the ability to come into contact with a hard surface without losing stability, and the ability to specify directly the behavior of mechanical interactions with the environment. Impedance control is thus ideal for tasks requiring assembly or other contact with external systems.

Multiple arm control policies Multiple arm control policies can be further divided into three categories: coordinated motion, master/slave, and object motion policies. We define any algorithm that controls the arms independently—but possibly on coordinated trajectories—as coordinated arm control. The arms are "coordinated" rather than "cooperative" because they respond neither to each other's actions nor inputs except through the object's dynamics. Master/slave policies distinguish the arms, designating one the "master" and the other the "slave". Object motion policies specify the behavior of the manipulated object, the manipulators themselves are abstract motion generators. Each of the endpoint control strategies can be extended to the multiple arm case in two manners: as a coordinated motion strategy, or as an object motion strategy.

Coordinated motion approaches Each of the endpoint control strategies can be extended to the multiple arm case as a coordinated motion policy. Unfortunately, no coordinated motion policy presents an attractive cooperative control strategic interface. It is also difficult to compensate for the full system dynamics with a coordinated motion strategy.

Coordinated position control is inadequate for even simple object positioning tasks; since the closed-chain cooperative system is kinematically over-constrained, any minor endpoint positioning error will cause the build-up of large internal forces. Zheng and Luh present a coordinated motion approach [43], along with some preliminary experimental results. They solve the kinematic over-constraint problem by manipulating a deformable object; they do not consider inertial forces or environmental interactions.

Mason [26] developed constraint relations for multiple-armed coordinated hybrid control. His development solves the kinematic over-constraint problem, but presents a complicated strategic interface. The arms must never be allowed to conflict in a single degree of freedom—either both in position control mode or both in force control mode. Extension of this requirement to more than two arms is completely unwieldy. His work does not consider dynamic interactions, and no simulation or experimental results are given.

Coordinated impedance control [28] presents a simpler interface, and is capable of good performance in many tasks. Unfortunately, while the arm dynamic properties are easily taken into account, it is difficult to compensate for the object dynamics with this control scheme. It is also complex to specify external interaction forces.

Master/slave approaches Much of the pioneering work in cooperation utilized some variation of master/slave control [17, 12]. More recently, Alford and Belyeu [3] modified the slave's trajectory in real-time to minimize the error in the positions of the end effectors. These techniques do not consider environmental interactions. They also suffer from the artificially imposed asymmetry, and none take into account the dynamic behavior of the system. Moreover, from a strategic viewpoint, master/slave control is a difficult policy to direct. Free-space motions are the only simply specifiable actions.

Object motion approaches Object control policies specify the object's behavior—no independent arm action is specified. Each of the endpoint control policies can also be extended to the multiple arm case as an object control policy. Object control policies have the distinct advantage of providing the strategic controller with the ability to specify the object motion without regard to the manipulator details. Object control also raises some difficult implementation issues—no previously published experimental results consider inertial force compensation.

Several researchers propose object position control algorithms. Tarn, Bejczy, and Yun [37] developed the closed-chain dynamic equations of motion, and utilized nonlinear decoupling feedback to control the object position. This treatment compensates for the full system dynamics. However, it completely ignores the effect of external forces. Also, since it utilizes the closed-chain equations of motion, it must be implemented serially. Nakamura, Nagai, and Yoshikawa [27] develop a multi-fingered hand control scheme that specifies the dynamic behavior of the manipulated object. Their formulation also allows specification

of the object internal forces. They also propose a parallel dynamical compensation scheme based on D'Alembert's principle. However, they too assume the object is not in contact with its environment. No simulated or experimental verification is offered.

Hybrid object force/position control permits interactions with the external environment. Uchiyama, Iasawa, and Hakomori [39] present an experimental hybrid controller using two arms to carry an object under compression. They define an interesting workspace coordinate system that allows hybrid control of both external and internal positions or forces. Their algorithm is based on static analysis; it is incapable of dynamic compensation. The experimental task they present is a simple transport problem, with no environmental interactions.

Hayati [13] presented a theoretical extension of hybrid force/position control to the multiple manipulator case. He defines an arbitrary task reference frame, a useful concept incorporated into the controller developed in this paper. Although a parallel implementation is proposed, his controller requires the closed-chain equations of motion and is computationally complex. It also specifies locally closed force loops on each arm; in practice, this can make the control very sensitive to force offset errors.

Oussama Khatib, in [20], extends his operational space control methodology to incorporate multiple manipulators. His paper's main thesis is a proposed dynamic implementation technique based on summing the object inertia and the arms' effective tip inertias. The formulation allows utilizing operational space hybrid force/position control. The algorithm takes into consideration the full system dynamic response, but is not particularly well suited to parallel implementation. Internal forces are not controlled; the redundancy is used instead to minimize the actuator loads amongst the manipulators. This approach has not yet been experimentally verified.

Hybrid control does not, in the authors' opinion, provide an attractive strategic-level interface. Since separate force and position controlled subspaces must be maintained, control mode switching decisions must be made at many points during most tasks. Specifying sequences of these mode switches is often not intuitive. Unexpected situations make these decisions even more difficult; the set of "natural" constraints may not be easily determinable.

Impedance control provides compliant response at all times. Selection of significantly differing stiffnesses in different directions must be done only when required by the situation. The more general impedance compliant response also eases interactions with moving environments. [11]

Object impedance control is the subject of the next section. It provides a simple, powerful interface for directing motion and environmental interaction. Allowing the specification of an arbitrary task frame and effective object dynamic response permits intuitive and general task sequence specification. A parallel implementation that compensates fully for the system dynamics, and directly controls the object internal forces is presented below.

[11] This is especially important if the "environment" is an object being controlled by another robotic system—we are also concerned with extension to multiple *teams* of robots.

2.7.2 Object Impedance Controller Derivation

This section restates the motivation for object impedance control, then derives a dynamic implementation.

2.7.2.1 Control objective

Hogan's impedance control policy, described above, causes the endpoint of the manipulator to react to external forces with a programmable impedance. The simplest example both to understand and implement is a simple second order linear impedance—the endpoint behaves as a mass attached via a virtual spring-damper to the environment. In this section, we develop a controller that enforces a controlled impedance not of the arm *endpoints*, but of the manipulated *object* itself. Intuitively, the object behaves as if it were attached to its environment by linear spring-damper systems in the linear degrees of freedom, and also by uncoupled torsional spring-dampers to control rotational orientation.

Policy The object impedance controller enforces (for a simple linear second-order impedance) the relationship:

$$m_d(\ddot{x} - \ddot{x}_{des}) + k_v(\dot{x} - \dot{x}_{des}) + k_p(x - x_{des}) = f_{ext} \tag{2.1}$$

Here x denotes the coordinate of any one degree of freedom of an *arbitrary* point of the object. The constants m_d, k_v, and k_p are specifiable. The reference signal x_{des} denotes the desired position (or orientation) of the chosen point. The $\ddot{x}_{des}$ term represents acceleration feed-forward. Thus, the programmable impedance force corrects deviations from the desired trajectory.

Intuitively, this control policy completely supplants the actual dynamics of the object with a "virtual" object, with specifiable mass and inertia properties. The "virtual" object is attached at its (apparent) center of mass via an orthogonal set of imaginary damped springs to a selectable point in the environment. Thus, the object can be manipulated by simply moving the virtual spring endpoint. Controlled force interactions with environmental obstacles can be done by simply pressing the "spring" against the obstacle.

We note here that a more general impedance can be modeled as:

$$m_d\ddot{x} = f_{ext} + f_{imp}$$

Here f_{imp} is an imaginary impedance "force", it is usually a function of only position and velocity, but can be quite arbitrary. In particular, the non-linear "potential field" forces used in many obstacle avoidance techniques [22] can be easily incorporated by simple summation (for proof, see [16]).

This controller accomplishes all the policy objectives outlined above. The strategic interface is quite simple and straight-forward. The strategic module directly selects the object's behavior—the arms are very much an abstract manipulation system. In addition, to the bandwidth of the control system, the object dynamical properties are also selectable. Critical damping can be enforced in all degrees of freedom simultaneously. Contact of

the manipulated object with the environment is not a special case requiring control mode switching; the impedance controller handles it in a natural manner. Thus, both free motion slews and manipulation requiring contact can be done with the same strategic interface.

The position and orientation of the "virtual" object with respect to the actual object is also selectable. Thus, this controller is also capable of pseudo remote center of compliance (RCC) operation [41], permitting simple and efficient part mating and insertion operations. A particularly useful example is for performing connector insertions—by placing the "virtual" object frame at a fixed location in the connector frame, all assembly operations can be specified as connector motions only. Multiple connectors arranged on an object in arbitrary orientations can then be handled by the same simple connector insertion algorithm.

Implementation Implementation of this type of controller requires consideration of the dynamics of the entire system: both arms and manipulated object. Utilization of the full system equations of motion is one method of performing this control, but we have developed an implementation based on Nakamura's multi-fingered hand controller [27] that isolates the system into three sub-systems: arm1, arm2, and the common object. This vastly simplifies the computations required at runtime; the closed-chain equations of motion are not required, and the algorithm can easily be implemented by a parallel processor architecture. In addition, the internal force in the object is directly specifiable.

Force information at the arm endpoints enhances performance on two counts: better control of the internal forces and more accurate measurement of the forces of interaction between the object and its environment.

2.7.2.2 Derivation

The derivation presented below is rather straight-forward: The desired object acceleration is found, and substituted into the actual object equations of motion. The resulting accelerations and forces at the arm endpoints are found, and then implemented on the arms as a "computed-torque" (CT) control law.

Notation The symbols used are:

Symbol	Type	Description
$\mathbf{n_k}$	3×1	inertial frame unit vector
$\mathbf{b_k}$	3×1	object body frame unit vectors
m	scalar	mass of the manipulated object
M	3×3	m times the identity matrix
I	3×3	inertia matrix of the object relative to its center of mass for $\{\mathbf{b_i}\}$
$\mathbf{f}_{ext}$	3×1	external force on the object (not due to the arms)
τ_{ext}	3×1	external torque on the object (not due to the arms)
$\mathbf{x}$	3×1	inertial frame position of the object center of mass
$\mathbf{y}$	3×1	inertial frame position of the apparent center of mass
$\mathbf{r}$	3×1	$\mathbf{y} - \mathbf{x}$ (offset of apparent from actual center of mass)
r_k	3×1	$\mathbf{r} \cdot \mathbf{n_k}$
ω	3×1	object angular velocity
$\mathbf{g}$	3×1	gravity vector
U_n	$n \times n$	identity matrix
N	scalar	total number of arms contacting object
$\mathbf{a}_i$	3×1	inertial frame position of i^{th} arm's endpoint
$\mathbf{p}_i$	3×1	body frame position of i^{th} arm's endpoint
p_{ik}	scalar	$\mathbf{p}_i \cdot \mathbf{n_k}$
$\mathbf{f}_i$	3×1	force on object due to i^{th} arm's endpoint
$\mathbf{m}_i$	3×1	torque (moment) due to i^{th} arm's endpoint
$\mathbf{q}_i$	$n_{links} \times 1$	vector of i^{th} arm's joint angles
τ_i	$n_{links} \times 1$	vector of i^{th} arm's joint torques
$\mathbf{f}_{imp}$	3×1	imaginary impedance "force"
τ_{imp}	3×1	imaginary impedance "torque"

Actual object dynamics The object's free-body diagram is shown in Figure 2.14. The object's center of mass is offset in the inertial frame by $\mathbf{x}$. We assume there are N total manipulators in contact with the object, two are shown. Arm i exerts a force $\mathbf{f}_i$ and moment $\mathbf{m}_i$ at a point offset by the vector $\mathbf{p}_i$ from the center of mass. A second arm, j, acts in a similar manner. External forces and torques act on the object—they are modeled by their resultant force $\mathbf{f}_{ext}$ and torque τ_{ext} acting at the object's center of mass.

The object's equations of motion are:

$$m\ddot{\mathbf{x}} = \mathbf{f}_{ext} + \sum \mathbf{f}_i + m\mathbf{g}$$

$$I\dot{\omega} + \omega \times I\omega = \tau_{ext} + \sum \mathbf{p}_i \times \mathbf{f}_i + \sum \mathbf{m}_i$$

Using $\ddot{\mathbf{y}} = \ddot{\mathbf{x}} + \dot{\omega} \times \mathbf{r} + \omega \times (\omega \times \mathbf{r})$, the equations of motion can be expressed in matrix form as:

$$\begin{bmatrix} mU_3 & mR \\ 0 & I \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{y}} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} -m\mathbf{g} - m(\omega \times (\omega \times \mathbf{r})) \\ \omega \times I\omega \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{ext} \\ \tau_{ext} \end{bmatrix} + \begin{bmatrix} \sum \mathbf{f}_i \\ \sum \mathbf{p}_i \times \mathbf{f}_i + \sum \mathbf{m}_i \end{bmatrix}$$

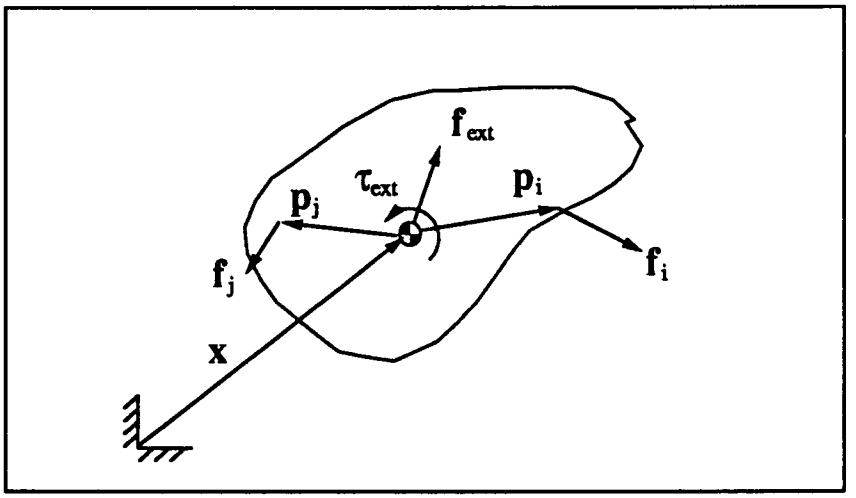


Figure 2.14: Object Free-Body Diagram

or, more compactly:

$$I_0\ddot{Y} + \mathbf{B}_0 = \mathbf{F}_{ext} + W\mathbf{F} \tag{2.2}$$

where

$$\ddot{Y} = \begin{bmatrix} \ddot{\mathbf{y}} \\ \dot{\omega} \end{bmatrix}, \quad W = \begin{bmatrix} U_3 & 0 & U_3 & 0 & \cdots & U_3 & 0 \\ P_1 & U_3 & P_2 & U_3 & \cdots & P_N & U_3 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{n}_1 \\ \vdots \\ \mathbf{f}_N \\ \mathbf{n}_N \end{bmatrix}$$

and

$$R = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix}, P_i = \begin{bmatrix} 0 & -p_{i3} & p_{i2} \\ p_{i3} & 0 & -p_{i1} \\ -p_{i2} & p_{i1} & 0 \end{bmatrix}$$

Desired object behavior The desired object behavior is depicted in Figure 2.15. The same external forces $\mathbf{f}_{ext}$ and τ_{ext} act on the object. Point C_v is the arbitrarily chosen apparent center of mass. It is offset from the center of mass by $\mathbf{r}$, and in the inertial frame by $\mathbf{y}$. The imaginary impedance "force" $\mathbf{f}_{imp}$ and "torque" τ_{imp} act at point C_v. The apparent object mass and inertia are also specifiable as m_d and I_d. Thus, the desired equations of motion are:

$$M_d\ddot{\mathbf{y}} = \mathbf{f}_{ext} + \mathbf{f}_{imp}$$

$$I_d\dot{\omega} + \omega \times I_d\omega = \tau_{ext} + \tau_{imp} - \mathbf{r} \times \mathbf{f}_{ext}$$

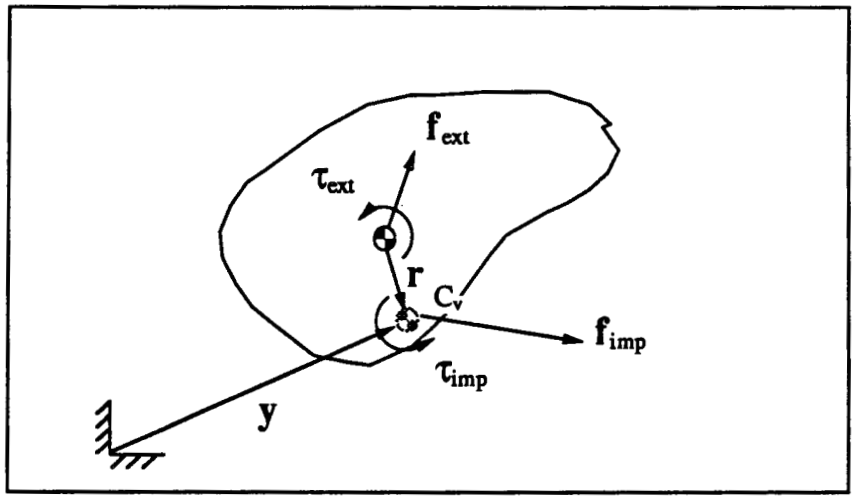


Figure 2.15: Desired Object Behavior

The desired behavior in matrix form is:

$$\begin{bmatrix} M_d & 0 \\ 0 & I_d \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{y}} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} 0 \\ \omega \times I_d \omega \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{ext} \\ \tau_{ext} - \mathbf{r} \times \mathbf{f}_{ext} \end{bmatrix} + \begin{bmatrix} \mathbf{f}_{imp} \\ \tau_{imp} \end{bmatrix}$$

$$I_{0d}\ddot{Y} + \mathbf{B}_{0d} = \tilde{\mathbf{F}}_{ext} + \mathbf{F}_{imp} \tag{2.3}$$

Controller Derivation So, from 2.3, the desired acceleration is:

$$\ddot{Y}_{cmd} = I_{0d}^{-1}[\tilde{\mathbf{F}}_{ext} + \mathbf{F}_{imp} - \mathbf{B}_{0d}]$$

This requires (using 2.2):

$$W\mathbf{F} = \mathbf{B}_0 - \mathbf{F}_{ext} + I_0(I_{od}^{-1}[\tilde{\mathbf{F}}_{ext} + \mathbf{F}_{imp} - \mathbf{B}_{0d}]) \tag{2.4}$$

To choose the commanded forces, $\mathbf{F}_{cmd}$, let

$$\mathbf{F}_{cmd} = W^{wpi}\{\mathbf{B}_0 - \mathbf{F}_{ext} + I_0(I_{od}^{-1}[\tilde{\mathbf{F}}_{ext} + \mathbf{F}_{imp} - \mathbf{B}_{0d}])\} + \mathbf{f}_{internal} \tag{2.5}$$

where W^{wpi} is a weighted pseudoinverse of W, and $\mathbf{f}_{internal}$ is any vector in the null space of W. The section *A note on load balancing* below details how these may be chosen.

We now have $\mathbf{f}_i$ for each arm. To calculate the desired arm tip accelerations $\ddot{\mathbf{a}}_i$, use:

$$\ddot{\mathbf{a}}_{i_{cmd}} = \ddot{\mathbf{x}} + \dot{\omega} \times \mathbf{p}_i + \omega \times (\omega \times \mathbf{p}_i)$$

Then, letting M_i and J_i denote the individual arm mass matrix and Jacobian, use the arm kinematics:

$$\ddot{\mathbf{q}}_{i_{cmd}} = J_i^{-1}[\ddot{\mathbf{a}}_{i_{cmd}} - \dot{J}_i\dot{\mathbf{q}}_i]$$

and the arm equations of motion:

$$\tau_i = M_i\ddot{\mathbf{q}}_{i_{cmd}} + \mathbf{C}(\mathbf{q}_i, \dot{\mathbf{q}}_i) + J_i^T\mathbf{f}_{i_{cmd}}$$

to calculate the desired arm torques.

External force calculation We have yet to estimate $\mathbf{F}_{ext}$. We can use equation 2.2, but we first need to estimate $\ddot{Y}$. We tried two approaches:

using the last commanded acceleration: $\hat{\ddot{Y}} = \ddot{Y}_{k-1}$
and using the desired acceleration: $\hat{\ddot{Y}} = \ddot{Y}_{des}$

Each of these yielded acceptable experimental results. In practice, the accelerations are small when the external forces are significant. Since the acceleration estimate is *only* used to estimate the external force, it is only critical when the object is in contact with an accelerating *environment.* This is a rare case, but may be useful, for example, to insert a battery pack into a rotating satellite. We are investigating more sophisticated acceleration estimation techniques.

A note on load balancing Equation 2.5 has two terms. The first term, $W^{wpi}\{\cdots\}$, represents a particular solution to equation 2.4. This vector of forces produces the desired motion while countering the inertial, gravitational, and external forces. The second term, $\mathbf{f}_{internal}$, is any member of the null space of W, the subspace of forces that produce only internal loading on the object. Normally, $\mathbf{f}_{internal}$ can be chosen simply to provide the desired internal object forces when the object is at rest.

Some latitude exists in choosing W^{wpi}, the weighted right pseudoinverse:

$$W^{wpi} = Q^{-1}W^T(WQ^{-1}W^T)^{-1}$$

Where Q is a diagonal matrix of task-space weights. If Q is the identity matrix, then the solution will be the minimum norm vector of endpoint forces and torques—all degrees of freedom will receive equal preference for loading. Since manipulators are usually capable of generating considerably more force than torque at their endpoints, Q will usually be used to weight the linear degrees of freedom more heavily. Task-space load distribution (in a static environment) is studied more completely in [1]. A comparison with joint space optimization techniques can be found in [36].

Controller structure The overall controller structure is depicted in figure 2.16. Note that the system divides naturally into $N + 1$ parallel computations: one to calculate the object dynamics, and one for each arm's dynamics[12]. Our experimental system employs three processors, configured in exactly this manner.

[12]The arm calculations could be similarly sub-divided, see chapter 3.

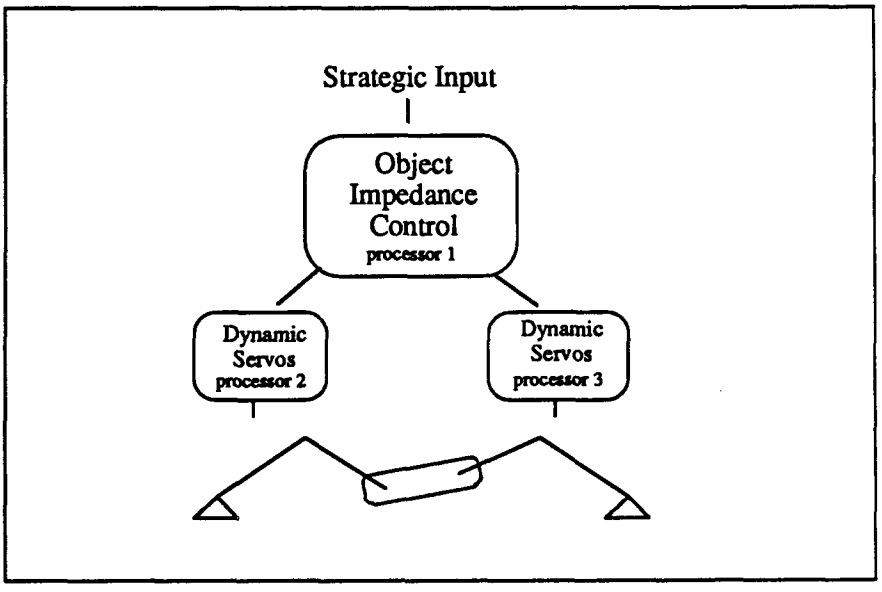


Figure 2.16: Controller Structure

Controller block diagram Figure 2.17 depicts the resulting controller block diagram. First, measured and desired object position and velocity are used to calculate the impedance "force", and the incoming measured forces at each manipulator endpoint are used to estimate the external force on the object. The desired behavior equations then produce a new desired acceleration command. Simple object kinematics then provide arm endpoint accelerations, and the object equations of motion are used to calculate consistent commanded endpoint forces. These commanded values are in turn used by each arm's computed torque controller to servo the arm actuators. Each of the three major boxes share no intermediate values; thus they can be run in parallel. (In fact, our experimental implementation runs the three loops at different, asynchronous rates.)

An intuitive alternate controller A slight variation in this derivation yields a controller with different properties. When the $\mathbf{r}$ is non-zero, the controller derived above above causes the object to behave as if its center of mass were moved to the impedance force action point. Thus, the object dynamic behavior is completely specifiable—the actual dynamics are completely canceled. This provides exact dynamic decoupling of all degrees of freedom, even under RCC operation. A simple change in the desired motion equations to:

$$M_d\ddot{\mathbf{x}} = \mathbf{f}_{ext} + \mathbf{f}_{imp}$$

$$I_d\dot{\omega} + \omega \times I_d\omega = \tau_{ext} + \tau_{imp} + \mathbf{r} \times \mathbf{f}_{imp}$$

leaves the object's apparent center at the actual center of mass. This controller does not completely decouple the object orientation from the linear motions. Although this is less appealing from a mathematical viewpoint, experiments with both manual operation and strategic programming showed that this controller is sometimes more intuitive to use. An

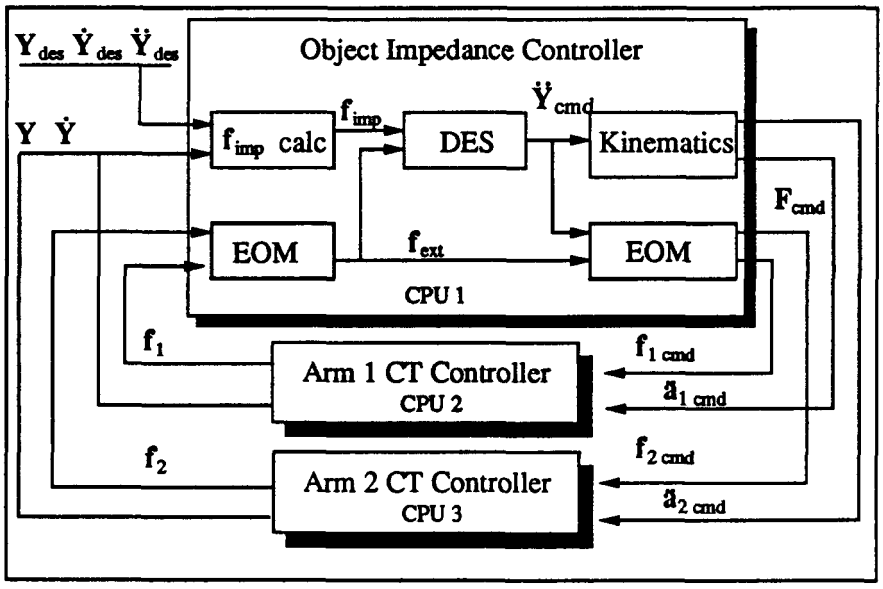


Figure 2.17: Controller Block Diagram

example is depicted in Figure 2.18. Users expect that the object pulled by a "spring" attached at the end will rotate—the real world is dynamically coupled. Moving the center of mass decouples these motions and results in a linear translation only. The strategic control module permits the user (or strategic programmer) to select either controller.

The two controllers are identical when $\mathbf{r} = 0$, and they have identical static behavior, since static balance requires $\mathbf{f}_{ext} = -\mathbf{f}_{imp}$. The coupled controller does however, have practical implementation advantages. It is less sensitive to biases in the force sensors (since $\mathbf{f}_{imp}$ does not depend on measured force). It also makes better use of the available actuator authority, since moving the apparent center of mass often increases required (peak transient) manipulation forces.

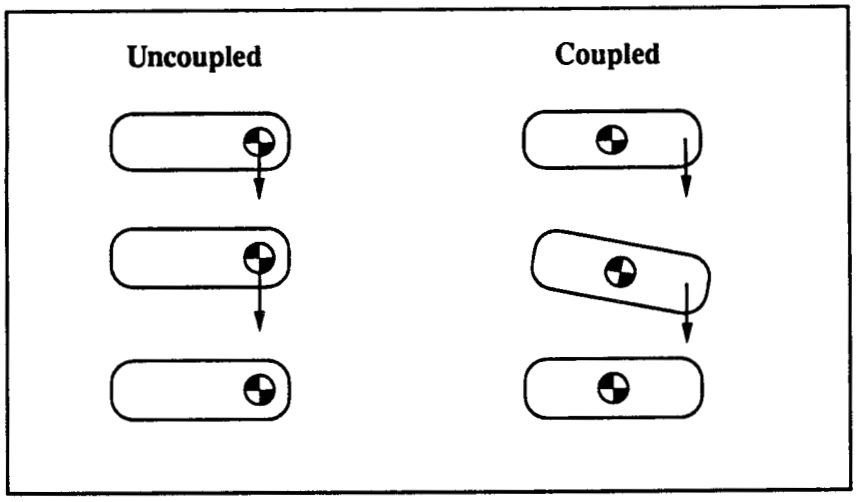


Figure 2.18: Intuitive Alternate Controller

2.7.3 Experimental Results

2.7.3.1 Experimental Dual Manipulator System

The experimental system is designed to emulate a dual-armed space robotic manipulator. The arms are each direct-drive, SCARA configuration, two-link manipulators. They are equipped with pneumatic force-sensing grippers. The grippers connect to "ports" on the floating object via a bearing pin joint. Thus, no torque can be delivered to the object by a single arm. The manipulated object floats over the granite surface plate with negligible friction. A picture of the experimental system is presented as Figure 2.19.

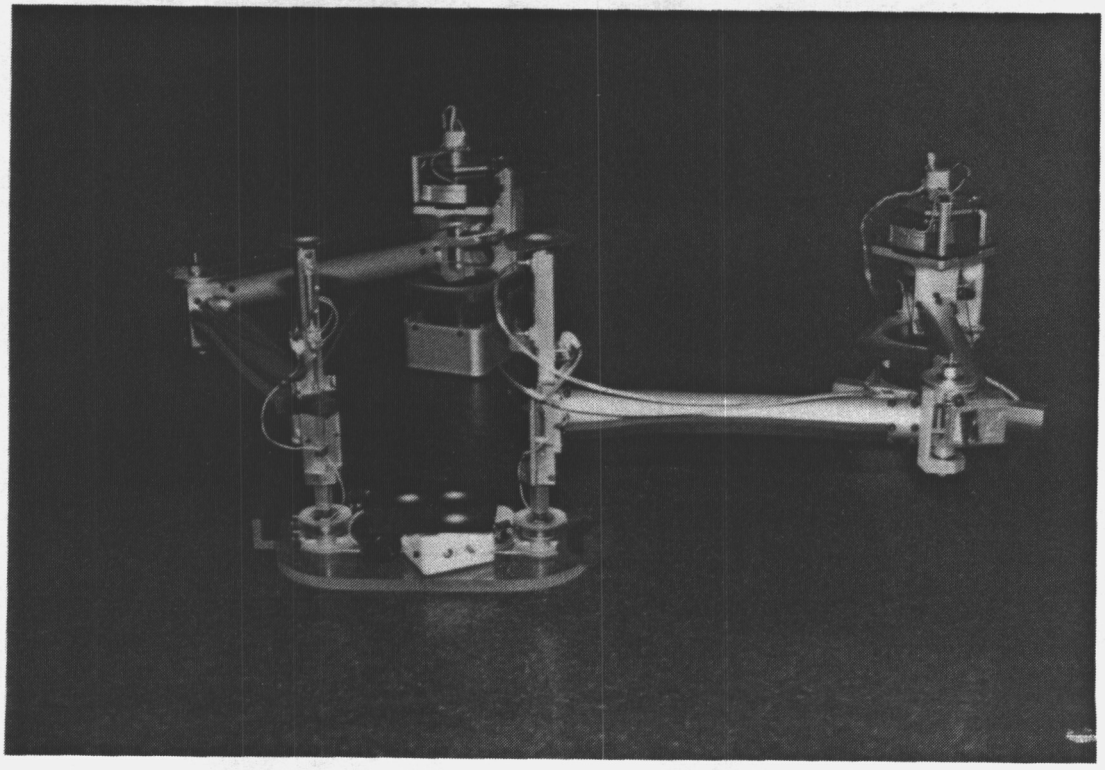


Figure 2.19: Experimental Dual Arm Manipulator System

2.7.3.2 Transport trajectory tracking

This section presents a comparison of the performance of three cooperative control algorithms for a cooperative transport task. The three algorithms are: co-located proportional-derivative joint-space coordinated position control, coordinated endpoint impedance control, and object impedance control. The starting and ending positions for the test slew are indicated in Figure 2.20. Note that all the arm joints undergo large angular changes, and that both position and orientation are effected. The commanded reference is a fifth-order trajectory of the center of mass of the object in each object degree of freedom: x, y, and

θ [10]. The slew reference takes 1.7 seconds to complete its path—faster slews caused our (rather weak) motors to saturate.

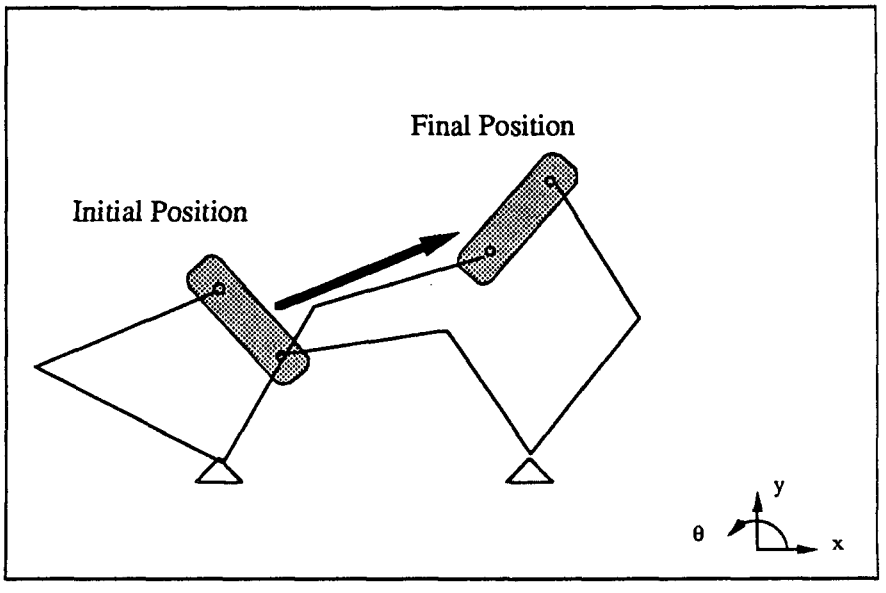


Figure 2.20: Test Trajectory

All algorithms were provided with the correct coordinated position, velocity, and acceleration references for the entire slew path. The gains on all the controllers were set for fairly stiff operation (to yield the best trajectory tracking), and are set as "fairly" as possible. For example, the PD joint angle controller gains were calculated to provide the same corrective force at the tip as the operational space controllers would provide for a similar positioning error. Several other gain combinations were studied. Where appropriate, effects of this variations are discussed below.

In the figures that follow, the upper-left plot depicts the motion of the center of mass of the y direction. The lower-left is the corresponding velocity. The upper-right plots x vs. y, and indicates the desired and actual object positions at 0.5 second intervals during the motion. The lower-right plot shows the magnitude of the "tension" between the arms, after being corrected for dynamic forces. All controllers are attempting to maintain zero tension. Note that the force sensors exhibit some orientation-dependent bias (maximum magnitude is approximately 0.2 Newtons), thus some "tension" is due to sensor error and is unavoidable.

Coordinated PD control Figure 2.21 presents the performance of a simple joint-space PD controller. Co-located PD control implements the algorithm:

$$\tau = K_p(\mathbf{q}_{desired} - \mathbf{q}) + K_v(\dot{\mathbf{q}}_{desired} - \dot{\mathbf{q}})$$

for each joint.

This controller does a poor job of following the desired trajectory and offers no control of the internal forces on the object. This controller also does not compensate for inertial

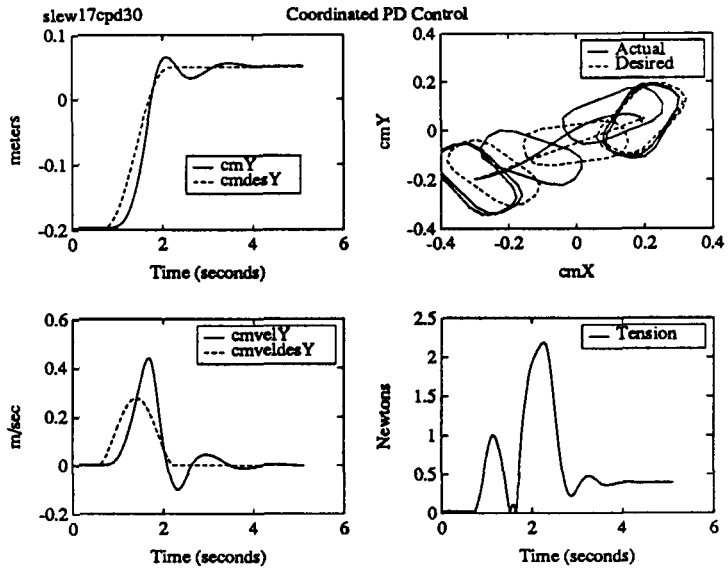


Figure 2.21: PD Controller Slew Performance

forces. Reducing the gains caused the trajectory performance to deteriorate badly, while failing to eliminate the tension buildup.

Coordinated endpoint impedance control Figure 2.22 depicts the performance of coordinated endpoint impedance control. This controller implements the dynamic spring relationship:

$$m_{desired}(\ddot{\mathbf{p}}_{desired} - \ddot{\mathbf{p}}) + K_v(\dot{\mathbf{p}}_{desired} - \dot{\mathbf{p}}) + K_p(\mathbf{p}_{desired} - \mathbf{p}) = \mathbf{f}_{tip}$$

on each arm, where $\mathbf{p}$ is the tip position, and $\mathbf{f}_{tip}$ is the measured force at the manipulator endpoint.

Substituting $\ddot{\mathbf{p}}$ into the arm kinematic equation

$$\ddot{\mathbf{q}} = J^{-1}[\ddot{\mathbf{p}} - \dot{J}\dot{\mathbf{q}}]$$

yields joint accelerations $\ddot{\mathbf{q}}$. The arm equations of motion:

$$\tau = \mathbf{M}\ddot{\mathbf{q}} + \mathbf{C} + \mathbf{J}^T\mathbf{f}_{tip}$$

then yields a dynamically compensated "computed torque" impedance controller. In this relationship, $\mathbf{f}_{tip}$ is the measured external force on the manipulator tip. This controller thus incorporates force feedback to enforce the endpoint impedance relationship.

This controller exhibits much better trajectory tracking performance (Figure 2.22). Although the object inertia forces are not compensated for, the feedback—coupled with acceleration feed-forward—is sufficient to cause very little trajectory tracking error. Unfortunately, to yield this level of performance, the impedance gains had to be fairly stiff. As a result, the inter-arm tension is not controlled well. Slight kinematic errors are sufficient to cause a large internal force at the end of the slew. Lowering the gains significantly reduced the tension problem, but also detracted from the trajectory accuracy.

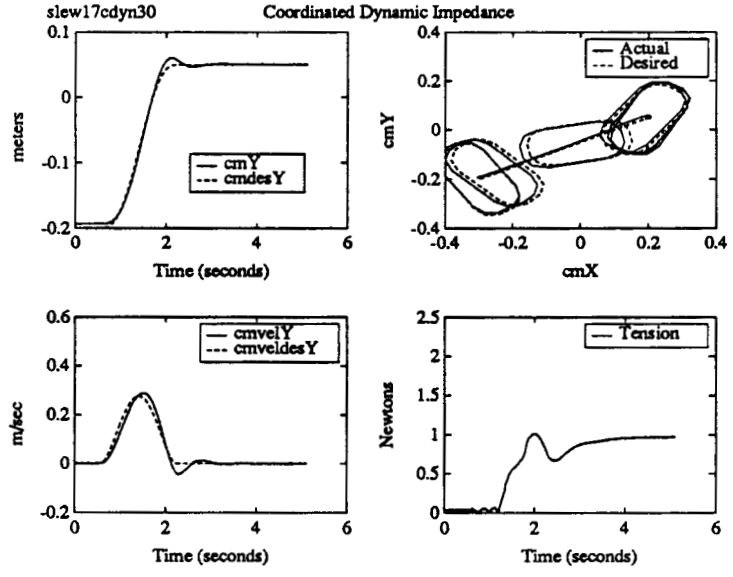


Figure 2.22: Coordinated Endpoint Impedance Slew Performance

Object impedance control Figure 2.23 displays the results of the object impedance controller. Since this controller correctly compensates for the object dynamics, the trajectory tracking performance is quite impressive. The inter-arm tension is controlled well also. Lower gains did not effect the trajectory accuracy much.

2.7.3.3 Force control performance

Figure 2.24 compares the response of the object impedance controller's external force measurement with the expected response of an ideal impedance (equation (POLICY)). These data were obtained by simply placing a hard, stationary object in the path of the object during a slew (in the x direction). Raw versus filtered (10Hz 2-pole digital Butterworth filter) data from a single arm (the right arm) is presented in Figure 2.25. The unfiltered data exhibits some ringing—this is an impact between two very hard objects—but the force level quickly settles to the desired. The important thing to note is that the object impedance controller successfully controls the forces of interaction, without switching control modes, even when it comes into contact with a very stiff environment.

2.7.3.4 RCC operation

Figure 2.26 shows a "strobe" picture of the object under control, with $\mathbf{r}$ non-zero. The point C_v is selected at the right edge of the object. The data used to generate this plot were generated by pushing the object (manually) in the positive y direction, and then releasing. The estimated external disturbance force is depicted in Figure 2.27. The "spring" gains chosen for this experiment are stiffer in the linear directions than in orientation, thus the object rotates about the point C_v with very little translation.

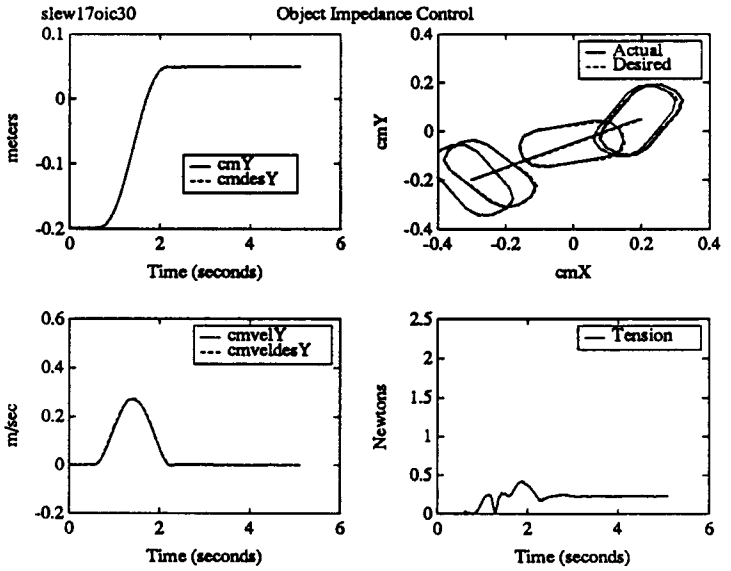


Figure 2.23: Object Impedance Controller Slew Performance

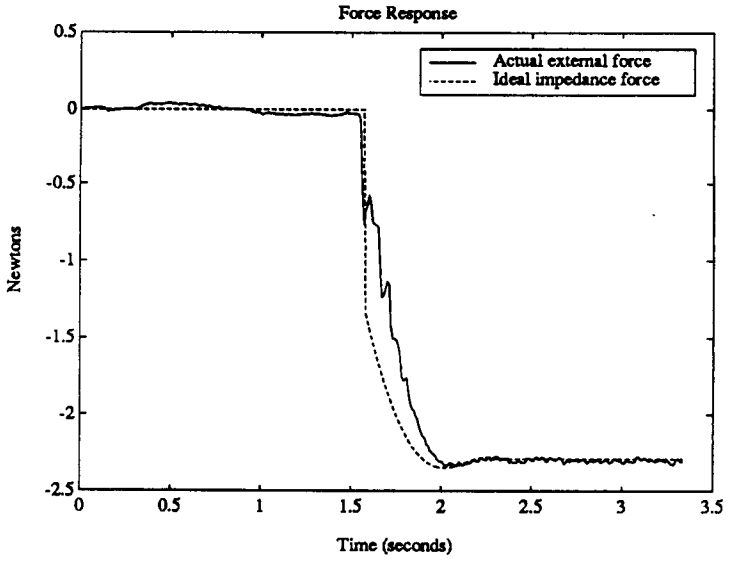


Figure 2.24: Estimated External Force

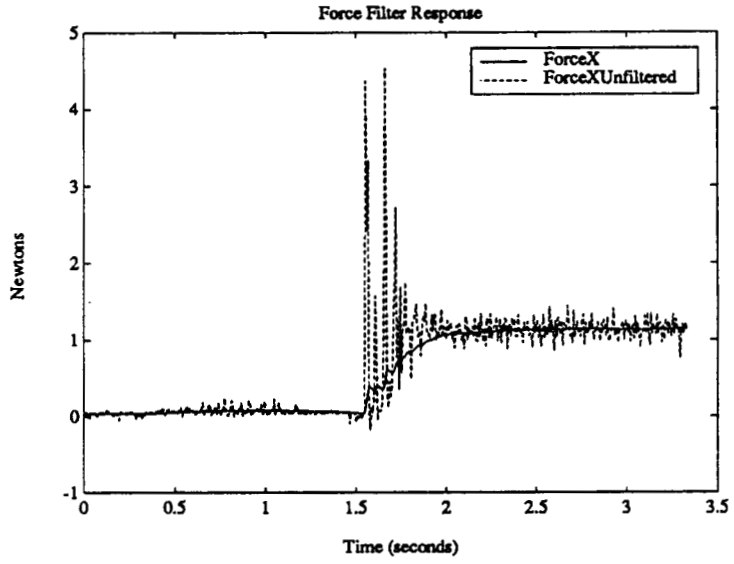


Figure 2.25: Right Arm Force Sensor Output

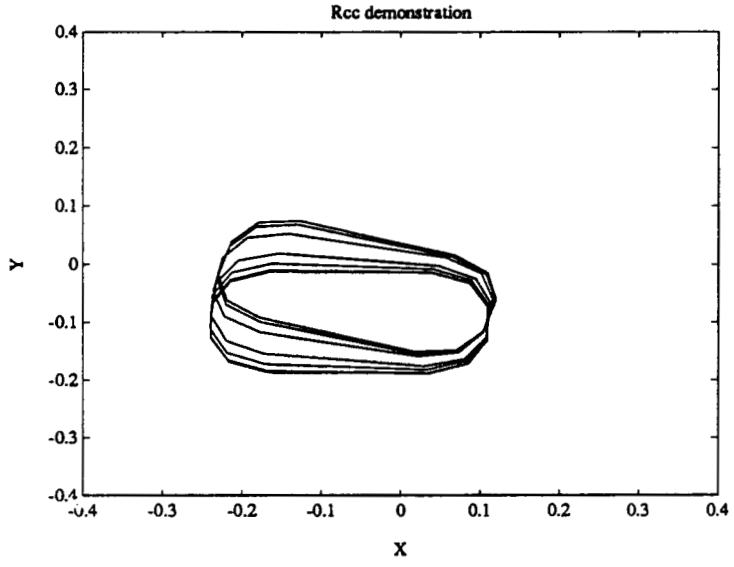


Figure 2.26: Remote Center Rotation

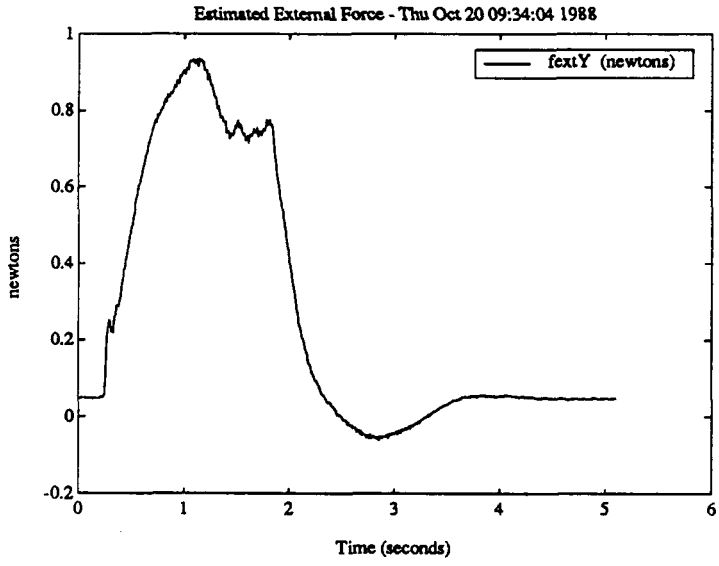


Figure 2.27: Estimated Force for Remote Center Rotation

2.7.4 Conclusions

This section has presented a dynamic control policy and implementation for multiple-armed manipulator systems that features:

- A simple, powerful *object* behavior specification interface.
- Flexible control specification: both for free motion positioning and environmental interaction.
- Good dynamic performance, both in free motion and in contact, without switching controllers.
- Exact dynamic compensation, without requiring closed-chain equations of motion.
- A naturally parallel structure, allowing simple implementation on a multiple processor computer system.
- Remote-centered compliance operation, facilitating assembly operations.

This controller has been implemented experimentally, and proven to be an effective dynamic control module in an overall cooperative manipulator system design.

2.8 Real-time Vision System

During this report period, a high-speed television camera-based point-tracking vision system was implemented. In addition, algorithms for identification and tracking of moving two-dimensional objects were developed, and used to demonstrate vision-guided dual-arm intercept and capture. This section discusses the results of that effort.

2.8.1 The vision system as a module in the larger system

The vision system has a well-defined function: to interpret the video data, and report information about the various objects in the workspace. This function is complicated somewhat by the varying nature of the different modules' need for the information. For example, some modules—such as the user interface—may simply require position information at rather slow update rates. On the other hand, the task of catching a moving object requires not only position measurements, but also velocity estimates. Finally, control modules require object and arm endpoint information at very high rates and with minimal delay. The vision system must provide the information in the proper form to each module.

2.8.2 Vision System Structure

The vision system software structure chart is presented as Figure 2.28. Video information from the CCD television camera is digitized and stored into a video frame buffer. A "UFO" searcher process scans the video data, looking for new points to track. When a point is located, it is "installed" with the point manager, which arranges for its being tracked in the future.

The object managers are each responsible for managing one type of object in the system. For example, the "body" manager is able to identify any of the various rigid bodies in the view. It also can calculate the body's position and orientation, given the location of its targets. Object managers can optionally employ the observer service to implement a simple Kalman filter, thus providing an estimate of the object's velocity as well as its position.

The object manager manager acts as a liaison between the rest of the vision system and the various object managers.

2.8.3 Sub-pixel real-time point tracking

Appendix A of the Fifth Semi-Annual Report presented the theoretical basis of a proposed sub-pixel accuracy real-time point tracking system. In summary, this system allows determining the location of special passive optical "targets". Preliminary simulation results suggested that the positions could be determined to approximately $1/20^{th}$ of a pixel accuracy, at the full frame rate (60 Hz) of the camera. Our experimental findings have confirmed this prediction.

The passive targets used vary in reflectivity, from black at the edges to white in the center. An example target is presented as Figure 2.29. They span an area of approximately 8 by 8 pixels. Thus, each pixel in the 8×8 grid contains information about the location

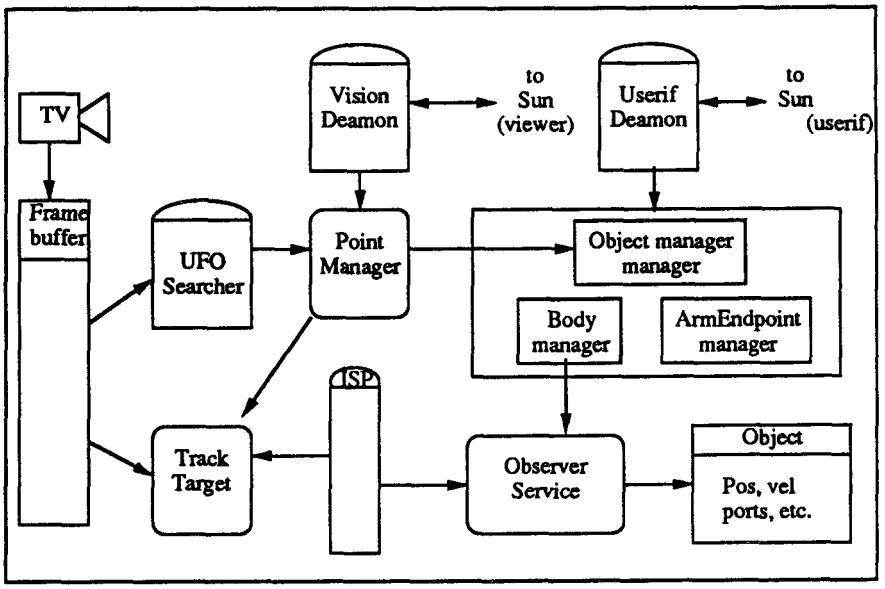


Figure 2.28: Vision System Software Structure

of the center. (In the noise-free case, one could calculate the distance to the center from each pixel exactly.) A simple, fast, centroid (center-of-brightness) calculation then yields an estimate of the actual center.

The point tracking module maintains a list of known target locations. At each frame interrupt, the new target location is calculated. To track quickly moving points, a simple digital filter is used to estimate each point's velocity. The system can then generate a better estimate of the target's expected location in the next frame. Thus, points should be "lost" only when they experience a high acceleration[13].

2.8.3.1 Viewer and the Vision Daemon

To assist in debugging the code, a program called "viewer" was developed. It is presented here to assist the reader in understanding the concepts involved. The "viewer" program runs on the Sun workstation. It opens a channel to the vision system; the "vision daemon" process exists solely to interact with the viewer. Two fields of information are displayed: the list of active point coordinates (in pixels), and the camera field of view. The active points flash in the camera view, so they can be easily located. A typical screen display is presented as Figure 2.30

2.8.4 "UFO" Searcher

The function of the "UFO" searcher module is to locate and identify new points in the field of view. The "UFO" searcher runs as a background task, completely asynchronously from the rest of the system.

[13]In practice, several effects can cause the point to be "lost". A more detailed analysis will be presented in the next report.

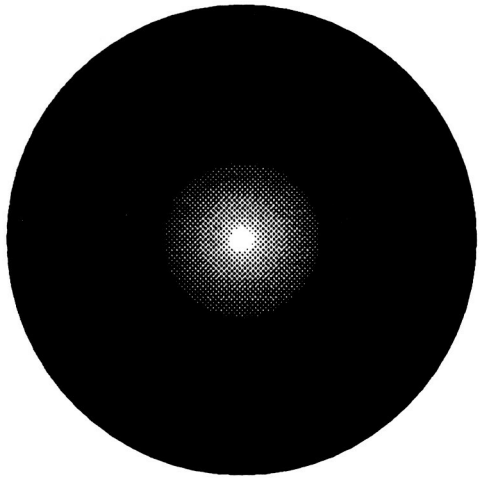


Figure 2.29: Example Target

To locate new points, it must first be able to distinguish the special optical targets from the other clutter in the scene. The algorithm for this is rather simple. First, the scene is scanned, looking for brightness transitions. When a transition is found, the coordinates are tested to see if this is already a tracked point. Next, the point-tracking centroid algorithm is executed once to find the center of the candidate point.

If the centroid algorithm returns a reasonable target center location, the following tests are made to determine if the point is indeed a target. First, a small "box" is drawn around the suspected target center. All pixels on this path (lightly shaded in Figure 2.31) should be above a programmable threshold value. Next, a larger box is drawn around the center. All pixels on this path (heavily shaded in Figure 2.31) should be below the threshold. If the point passes both tests, it is declared a new point, and installed with the point manager. This rather crude test only checks for points of a specified size, but suffices in our rather controlled environment. More sophisticated tests, such as matching the target reflectivity profile, could be easily performed if they were deemed necessary.

Scanning the camera's field of view by this method takes about 30 seconds. To increase the efficiency of the scanning process, the "UFO" searcher also accepts "hints" about where to look for new points. Any module may request that a Region Of Interest (ROI) be quickly scanned for a new point. All the ROI requests are serviced on a circular queue basis. Thus, each ROI is scanned for a short time; then scanning attention is turned to the next ROI. The entire view is always installed as one ROI to provide over-all scanning. With the exception of the entire view ROI, all other ROIs have a limited lifetime; they are removed from the circular list after a specified number of scans.

This scheme allows modules that have a good idea of "where to look" to install a very small ROI, and ensure that the point(s) of interest are quickly located. For instance, the arm endpoint managers install a small ROI around the location of the arm endpoint

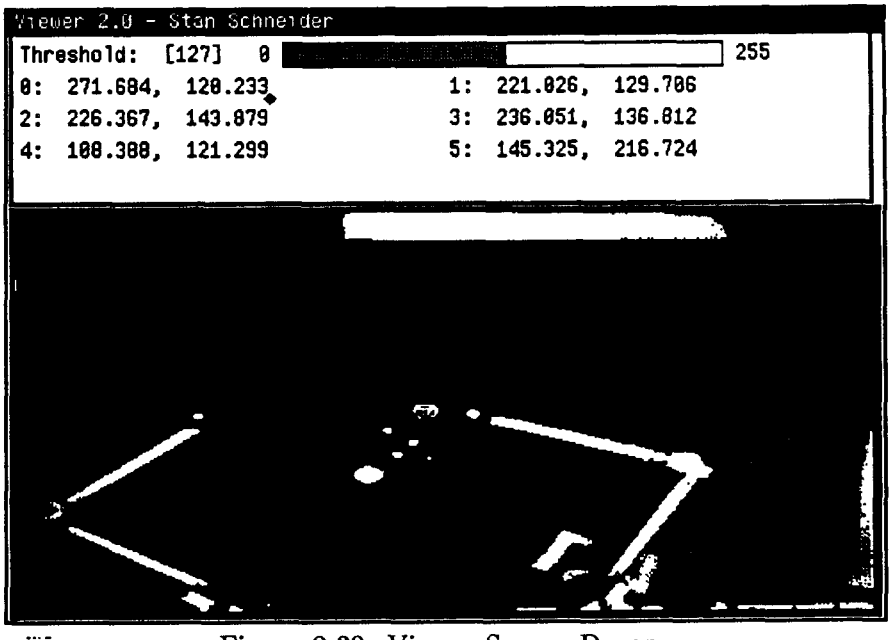


Figure 2.30: Viewer Screen Dump

calculated from the joint angle measurements. This point is known relatively well, thus it is usually found in a fraction of a second.

This capability is critical to the moving object acquisition problem. During a "catch" attempt, the object often passed beneath one of the approaching arm links. This obscures the targets, and causes the object to be "lost" at a critical time—just before the grippers are to engage. To remedy this, the body manager simply installs a ROI at the expected location of the object, based on its last known position and velocity. When the object comes back into view, it is located quickly and the grasping operation can continue.

2.8.5 The Point Manager

The function of the point manager is to maintain a list of tracked points, and ensure their positions are updated each frame interrupt. The point manager also performs the pixel-data-to-world-coordinates transformation, allowing all higher-level modules to work in natural (metric) units.

New points are provided by the "UFO" searcher. These are maintained on a list of active points. When installed, each new point is "unmanaged". Whenever the list of unmanaged points changes, the point manager passes it to the object manager manager for analysis by the various object managers. If an object manager recognizes a point, it claims the point (by setting its *manager* field). That point will then not be considered in future scene analysis attempts.

The location of each point on the active list is updated each frame interrupt [14]. When

[14] Actually, the point's object manager can request less frequent updates if the point is not expected to move much. This saves considerable processing time.

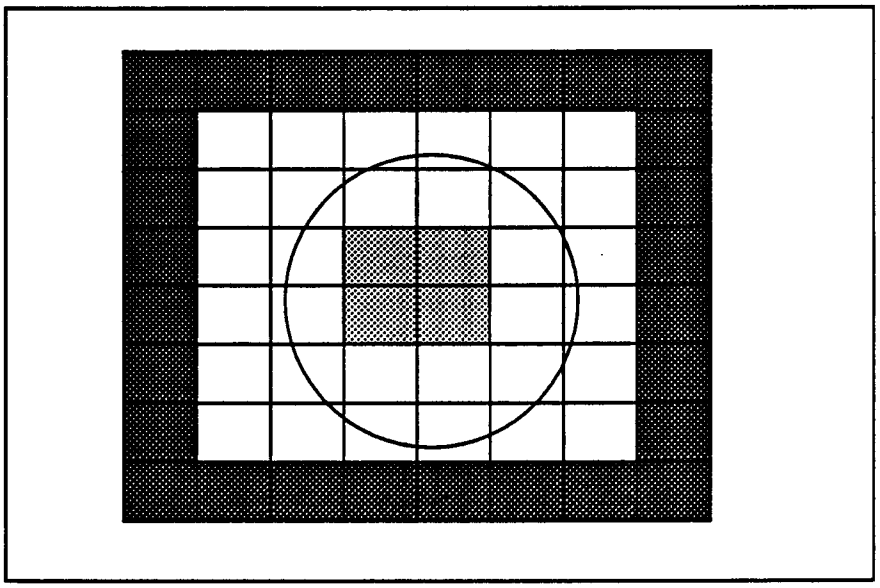


Figure 2.31: New point identification algorithm

a point is lost for some reason, the point manager notifies the point's object manager (if it has one), and removes it from the active point list.

To transform pixel locations to world coordinates, two operations are required: the pixel value must be converted to meters, and the effects of parallax must be corrected. At this point, this transformation is very fast and simple; pixels are converted to world frame coordinates via a simple scale and offset calculation, and the parallax effects are included in the scale. Figure 2.32 shows the parallax correction technique in one dimension. By similar triangles, each height above the table has a different pixel-to-meters conversion factor, given by the formula:

$$ParallaxCorrectedScale = ScaleAtTableHeight * \frac{(CameraHeight - height)}{CameraHeight}$$

Since all points in our system have a fixed height, this calculation need only be done once, when the point is identified.

2.8.6 The Object Managers

There are currently three object managers in the system: the body manager, the arm endpoint manager, and the point server. The body manager can identify and track rigid bodies in three degrees of freedom: translation in x and y, and rotation in θ. The arm endpoint manager is responsible for tracking the arm endpoints, and providing this information to the dynamic control module. Any module on any processor can specify a single point to be tracked by the point server. The point server is not used (yet) in this system (it will soon supplant the arm endpoint managers), but it allows the vision system to serve point location information to other experiments that share the vision system [15].

[15] In particular, the two-very-flexible-link endpoint control experiment.

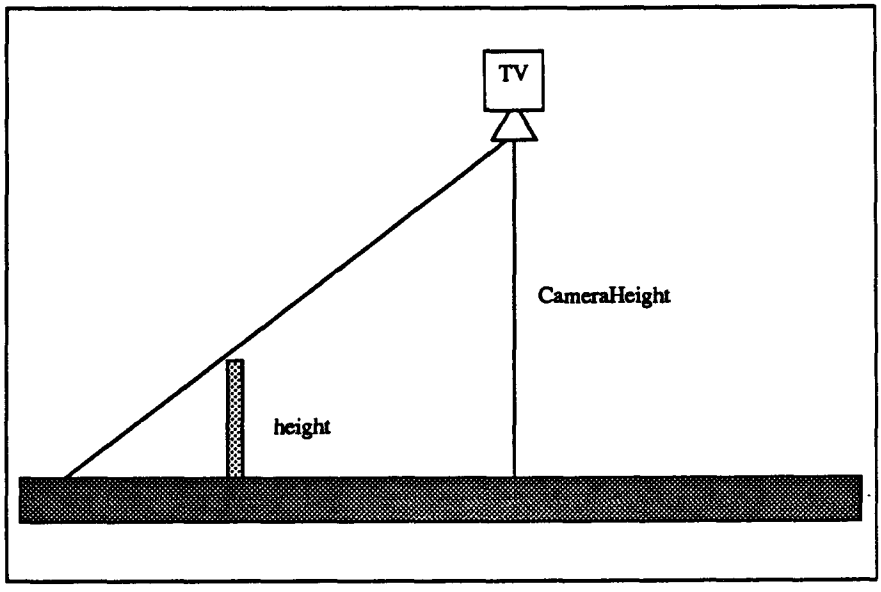


Figure 2.32: Parallax Correction

Each object manager must register with the object manager manager (OMM) at start-up time. The OMM then maintains a list of active object managers. Registration requires providing four routines:

- an identification procedure
- a lost point procedure
- a frame interrupt procedure
- and a report position procedure

The identification procedure is passed the list of unclaimed points when the "UFO" searcher changes it. The lost point procedure is called when a point claimed by this object manager is lost. The frame interrupt procedure is called each frame interrupt. Finally, the report position procedure is called to request that the managed object's positions be sent to the user interface.

2.8.7 The arm endpoint manager

The joint angle sensors can be used to estimate the endpoint position of each arm. Unfortunately, this estimate does not exactly match the vision system estimate of the same location. This is due to several effects, among them are angle measurement error, kinematic parameter errors, and optical distortion. Since the measurements of interest to the system (object locations, docking connector locations, etc.) are provided by the vision system, the *vision system* estimate of the arm endpoints must also be used by the dynamic controller. This *endpoint* control ensures that the controlled arm gripper position accurately matches the positions of the gripper ports on the objects of interest. It is the function of the arm

endpoint manager to provide the vision-sensed endpoint location to the dynamic control module.

The arm endpoint manager object procedures are simple. The identification procedure simply scans the unclaimed point list for a point near the kinematic estimate of the arm endpoint. The lost point procedure installs a small ROI around the arm endpoint expected position, and directs the dynamic controller to use the kinematic endpoint estimate. The frame interrupt procedure reports the endpoint location to the dynamic controller. The report position procedure is currently null, it will be added when the user interface is expanded to allow direction of individual arm motions.

2.8.8 The Body Manager

The body manager is responsible for identifying and tracking all the rigid bodies in the system[16]. At least three target points are affixed to each body. These points are in a unique pattern on each different type of body. The body manager is provided a database describing each body at initialization time, this database includes:

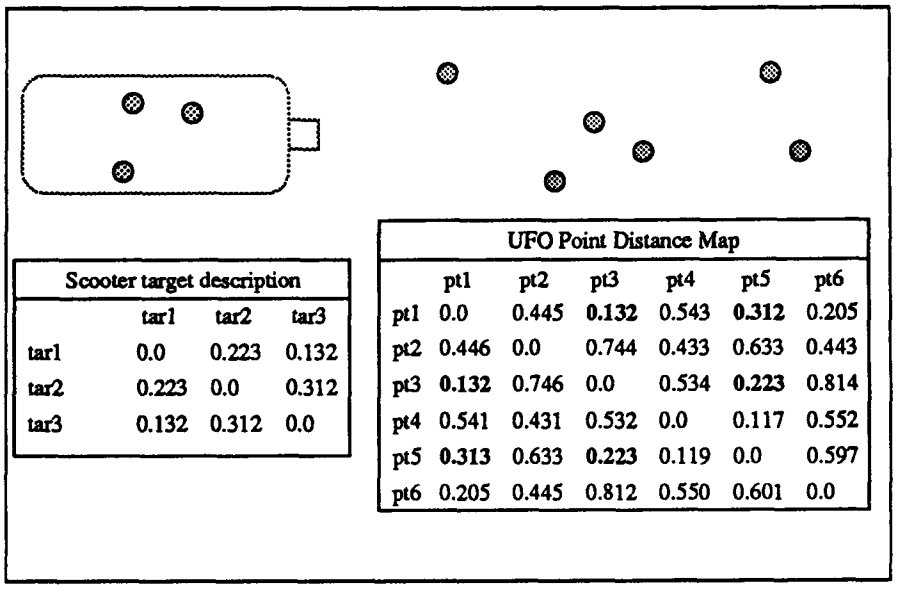
- the body's mass and inertia
- the location (and height) of the targets on the body
- the location of any gripper ports on the body
- and the location and type of any connectors on the body

As an ancillary function, the body manager provides any needed data about the body to other modules in the system. For instance, the dynamic controller requires the mass of any object it manipulates; the body manager provides this information.

Identification The body identification algorithm exploits the fact that the target configuration on each object is unique. When passed a list of unidentified points, the body manager first constructs a "UFO" point map, specifying the distances between all the points in the system. Figure 2.33 presents an example. This point map is then scanned for matches with the known distances between the targets on each object. If a sufficiently close match can be found for each target on an object, a quick consistency check is performed to ensure the identification is correct. If the matches are consistent, then the object is considered found.

Position and orientation update Each frame interrupt, the position and orientation of the object is calculated. The algorithm is based on the fact that the "center of brightness" of the targets is independent of orientation. More precisely, the average of the positions of all the object's targets can be calculated without regard to the object's orientation. It can then be used as a basis for determining the object's orientation.

[16]The term *body* in this section refers to any single rigid body whose position and orientation are both of interest. This includes the floating "object", as well as the docking ports, etc.



Scooter target description

	tar1	tar2	tar3
tar1	0.0	0.223	0.132
tar2	0.223	0.0	0.312
tar3	0.132	0.312	0.0

UFO Point Distance Map

	pt1	pt2	pt3	pt4	pt5	pt6
pt1	0.0	0.445	**0.132**	0.543	**0.312**	0.205
pt2	0.446	0.0	0.744	0.433	0.633	0.443
pt3	**0.132**	0.746	0.0	0.534	**0.223**	0.814
pt4	0.541	0.431	0.532	0.0	0.117	0.552
pt5	**0.313**	0.633	**0.223**	0.119	0.0	0.597
pt6	0.205	0.445	0.812	0.550	0.601	0.0

Figure 2.33: Body Identification Algorithm

The algorithm is depicted schematically in figure 2.34. During initialization, the "center of brightness" $\mathbf{r}_{cen}$ of the object's targets (in the object's frame) is calculated. The vector to each target $\mathbf{a}_i$ in the body frame is also calculated.

To calculate the object position and orientation, the vector $\mathbf{r}$ and the angle θ must be found. When the target positions are known, the new "center of brightness" $\mathbf{d}$ is simply the average of all the coordinates. Let $\mathbf{b}_i$ be the vector from $\mathbf{d}$ to the i^{th} target. i estimates of $\cos(\theta)$ are then available as:

$$\cos(\theta) = (\mathbf{a}_i \cdot \mathbf{b}_i)/|\mathbf{a}_i||\mathbf{b}_i|$$

These estimates are combined in a weighted average to yield an estimate of $\cos(\theta)$. The optimal weights to use are[17]:

$$w_i = |\mathbf{a}_i|^2/(|a_{ix}| + |a_{iy}|)$$

The angle θ is then calculated via a single arc cosine subroutine call.

This procedure will result in a θ value in the range $[0, \pi]$. To determine the sign of θ, note that:

$$b_{ix} = \cos(\theta) * a_{ix} - \sin(\theta) * a_{iy}$$

Since the sign of θ is the same as the sign of $\sin(\theta)$, *theta* is positive if:

$$a_{iy} > 0, and \cos(\theta) * a_{ix} - b_{ix} > 0$$

These equations, along with similar ones for b_{iy}, yield $2i$ independent estimates of the sign of θ. The estimates are combined via a simple voting method: if more estimates indicate negative sign than positive sign, θ is assigned the negative value.

[17] Proof will be supplied in a future report.

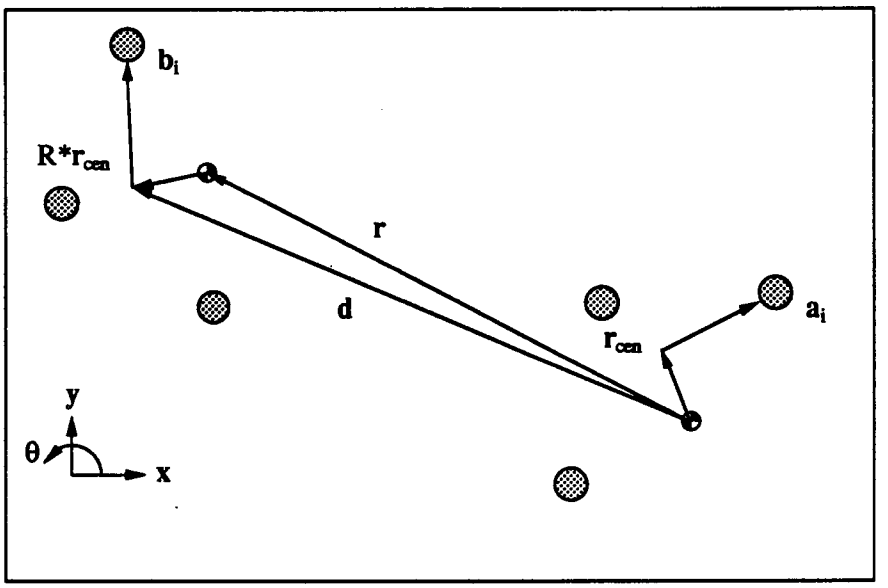


Figure 2.34: Body Position Calculation

Once θ has been calculated, the position of the center of mass is calculable as:

$$\mathbf{r} = \mathbf{d} - R * \mathbf{r}_{cen}$$

Where R is the rotation matrix:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Lost point procedure In a dynamically changing system, object points are often obscured. Once a body's points have been identified, it is possible to calculate it's position and orientation as long as any two points are visible. The body manager's lost point algorithm makes this possible. The technique is quite simple: at initialization time, the $\mathbf{a}_i$ vectors and w_i weights are calculated not only for the "all targets visible" case, but also for *every* possible configuration of two or more visible targets. This calculation is rather complex, but it need only be done once, during initialization. The position and orientation algorithm described above is then provided with a data structure which describes the currently visible target configuration. This structure is updated as the body's points are lost and re- found. The entire set of points need only be visible long enough for the body to be identified. The only effect of occluded points is a slight degradation in the accuracy of the position and orientation estimates.

If fewer than two points are visible, then the body is truly lost. When this happens, the lost point algorithm notifies the system that the body is lost, and installs a ROI at the body's last known position. The position of the ROI is updated for a short time, based on the body's last known velocity. In most cases, this causes the body to be re-found as soon as it becomes visible again.

Reporting positions to the user interface The final task of the body manager is to provide position updates to the user interface. The names and positions of all tracked objects are reported to the user interface whenever requested. This activity is executed as a background task, as it is of low priority.

2.8.9 The Observer Service

The object managers convert the raw point positions to the more useful object positions. However, many functions of the system require estimates of object velocities as well. The observer service allows each body manager the option of installing a simple linear state estimator [11] to provide object position and velocity estimates. By utilizing a model of the plant dynamics, the estimator also provides a one-step prediction of the object position, thus compensating for the inherent delay in the CCD camera sensing system.

The observer system implements the equations:

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + L(\mathbf{y}_k - H\bar{\mathbf{x}}_k) \tag{2.6}$$

$$\bar{\mathbf{x}}_{k+1} = \Phi\hat{\mathbf{x}}_k + \Gamma\mathbf{u}_k \tag{2.7}$$

Equation 2.6 is referred to as the measurement update, and equation 2.7 is the time update equation. This is a predictive estimator; $\hat{\mathbf{x}}$ provides an estimate of the state one sample period after the last measurement. One independent set of these equations is executed for each degree-of-freedom for every moving object in the system. The plant dynamic model for our frictionless rigid-body system is quite simple; each degree-of-freedom is a textbook double-integrator plant. The observer gains L were chosen to provide a time constant of approximately 0.5 second[18]. This value was chosen empirically; it provides fairly quick recovery from unmodeled plant disturbances, yet provides low-noise estimated position and velocity.

2.8.10 Experimental Results

Point tracking performance The simulations reported in the Fifth Semi-Annual Report predicted point tracking resolution on the order of $1/20^{th}$ of a pixel (average case). These predictions are well supported by the experimental data. A typical time history data sample of a single stationary point location is presented in Figure 2.35. The ordinate axis units are pixels in the vertical direction. The point's vertical position is indeed stable to about $1/20^{th}$ of a pixel.

Our simulation also predicted that resolution was a rather strong function of target size, with the maximum obtained when the target radius was about 3 pixels. This prediction was also confirmed. Since the camera's resolution, 440 x 240 pixels, is greater in the horizontal direction, the circular targets cannot have the same pixel radius in both directions. While the targets radius is about 2.8 pixels in the vertical direction, it is nearly 4.5 pixels in the

[18]The observer poles are Butterworth at 0.5 Hz

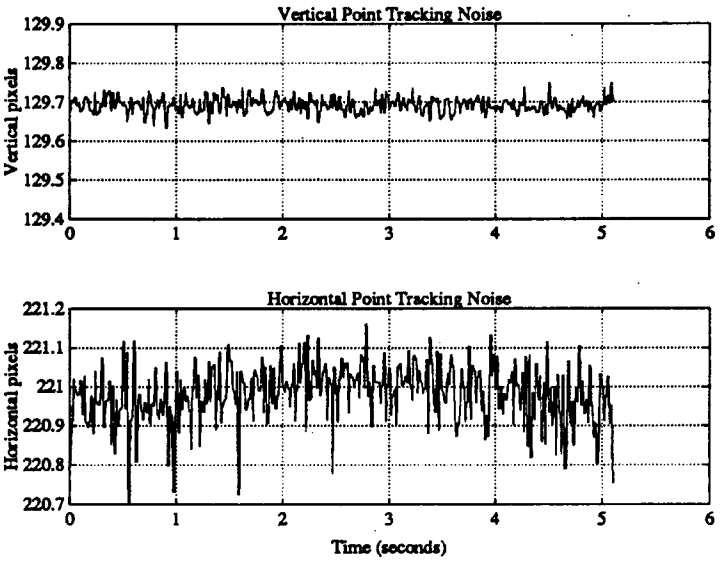


Figure 2.35: Typical Point Position Calculation Noise

horizontal direction. As a result, the horizontal resolution is slightly worse, about $1/10^{th}$ pixel[19].

Since a vertical pixel corresponds to about 5 mm, and a horizontal pixel to about 3 mm, the actual resolution in both dimensions is about 0.3 mm. The field of view is about 1 square meter. Thus, this represents 3000 : 1 resolution—quite acceptable for our task.

The system's global accuracy was not carefully measured, but is clearly limited by the optical distortion in our rather wide-angle lens. The lens is advertised to suffer distortion of about 1% over its field of view; this corresponds roughly to the distortion experienced. Resolution is far more important to our tasks than global accuracy; precise manipulation of adjacent objects requires only accurate difference measurements.

Observer performance The observer performance is depicted in Figure 2.36. These data were produced by bouncing the object off a piece of rather stiff foam. This causes a "step" change in the object velocity. Note that the recovery to the correct state is rather quick, and exhibits the expected Butterworth overshoot characteristic response.

Changes in the estimated state values each sample are readily apparent in Figure 2.36. The noise in the observed measurements is actually extremely small; velocity measurements for the approach in Figure 2.36 vary by only 1.5 mm/second. It is this high accuracy that permits the "catch" algorithm, described below, to predict accurately where the object will be in the future. The trajectory still requires periodic updates; even the 1.5 mm/second error, when projected 5 seconds forward, produces a 7.5 mm error in predicted position—enough to miss the gripper port.

[19]There may be other effects at play here as well, such as horizontal sampling jitter. They were not investigated.

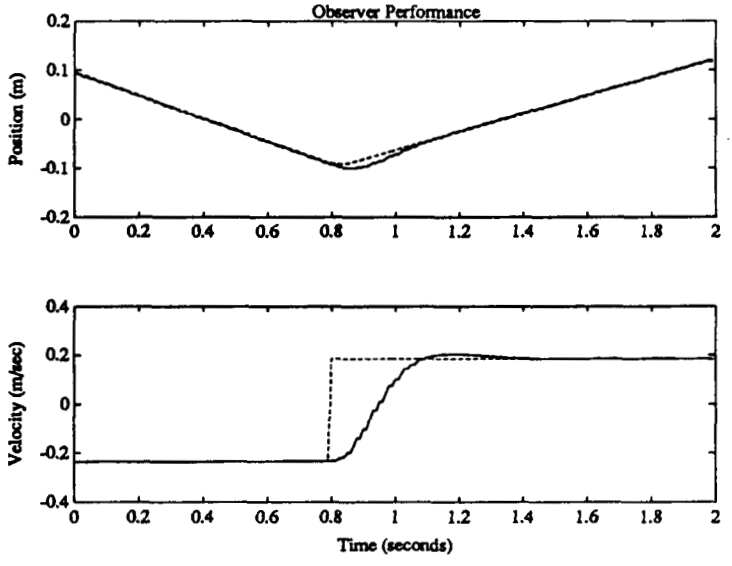


Figure 2.36: Observer Performance

An example: the catch The most demanding task required of the vision system is the interception and capture of a moving object. A "strobe" sequence picture of a typical catch is presented as Figure 2.37. The vision system must provide high-speed data for three subsystems to perform this task: the two arms, and the body manager. Each arm is under vision-guided endpoint control. The desired trajectory for each arm takes it from its initial "home" position, to an intercept state that matches the object's gripper port in both position and velocity.

To perform a successful capture, the arm endpoint must then be held over the gripper port for the duration of the time required for the gripper mechanism to engage. The positioning must be fairly accurate. A schematic side view of the gripper mechanism is shown in Figure 2.38. To engage the port, the gripper is driven downward by a pneumatic cylinder. Since the bevel in the port is only 4 mm, the position of the arm must track the port accurately for the 0.5 seconds to 0.9 seconds required for the downward motion of the gripper.

Figure 2.39 shows the vertical positions and velocities of the right arm and right gripper port during the above catch. The object is being accelerated toward the arm system for the first second. During the next second, the arm tip accelerates to match the port position and velocity at about the 3.2 second mark. The gripper's downward motion occupies the next second, after which the object is brought to a halt. After the grippers have engaged (at about 4.5 second mark) the observer has an incorrect plant model; this accounts for the apparent difference in position and velocity after the capture.

This figure also underscores the importance of the predictive estimates; a $1/60^{th}$ second delay in the position estimate at the 0.2 meters/second intercept velocity is 3.3 mm. This is nearly enough to cause the arm to miss its mark. Any additional accumulated errors would ensure a missed attempt.

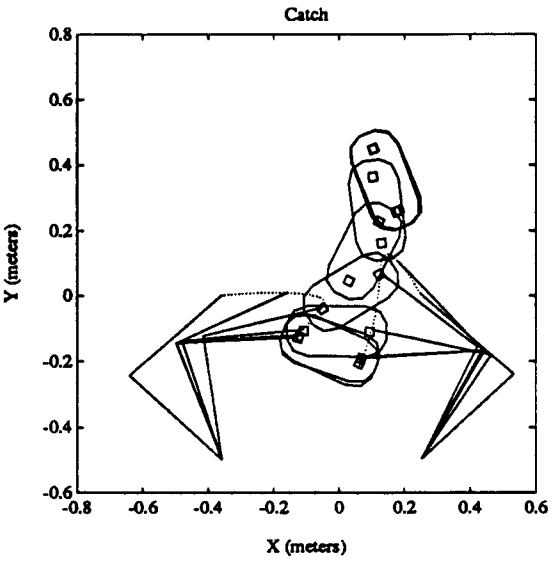


Figure 2.37: A Two-handed Catch

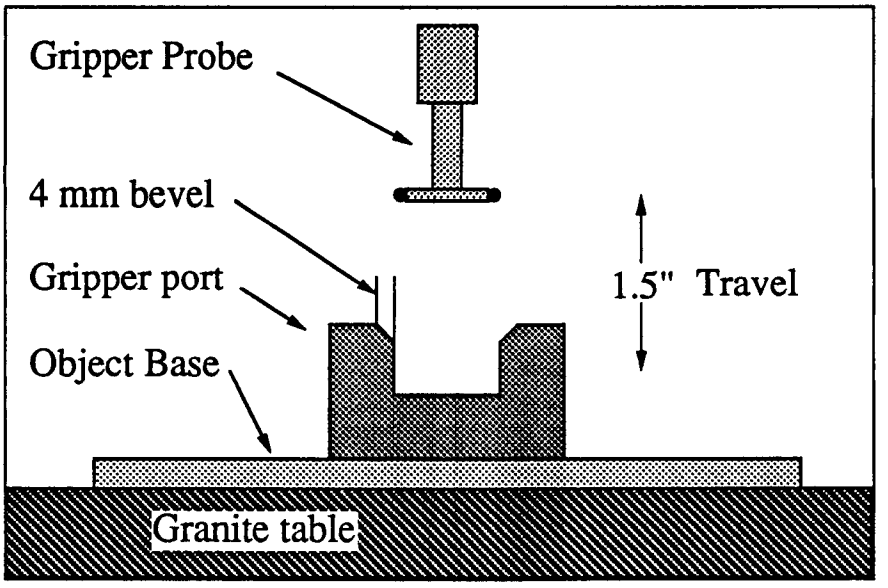


Figure 2.38: Gripper Schematic

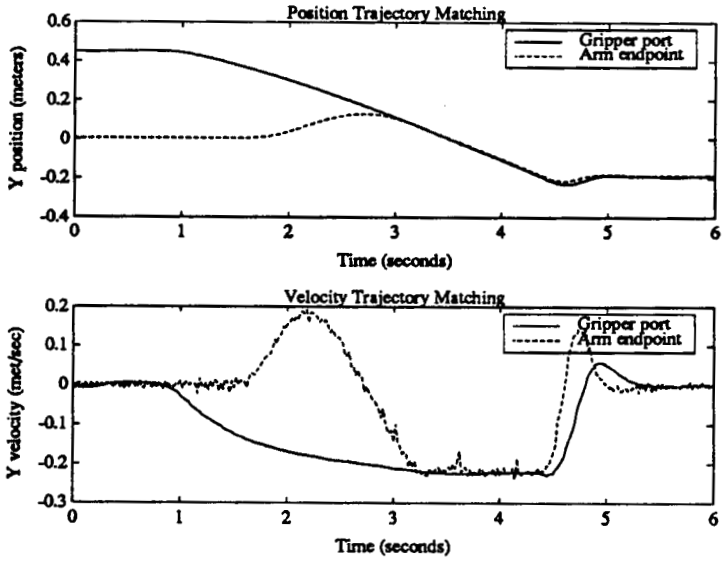


Figure 2.39: Catch Trajectory Matching

The task is complicated by the need for both arms to arrive at their intercept locations simultaneously. The arm trajectories are also updated as the object position and velocity estimates improve. The strategic control section below provides the details of both these issues.

2.9 Future Work

The fixed-base cooperative manipulation experiment is now essentially complete. During the next report period, the technology developed on this facility will be transferred to the floating base experimental apparatus.

We are also beginning the technology documentation process in earnest. During the next period, we expect to publish several technical papers—as well as a Ph.D. thesis—on the results of this effort.

Chapter 3

Multiple Arm Cooperation on a Free-Flying Robot

Ross Koningstein

3.1 Introduction

This chapter summarizes the work performed on multiple arm cooperation on a free-flying robot under NASA grant NCC-2-333 for the period February 1988 to August 1988. Multiple arm cooperation from a free-flying robot is one of the basic technologies required for space based manipulation. Ongoing work at the Stanford Aerospace Robotics Laboratory is yielding insights into the fundamental nature of free-flying manipulator robots, both in the understanding of the dynamics, and the understanding of the control problem. Extensions have been made to previous work on the dynamic modelling of a typical space robot configuration, the kinematic chain, to demonstrate the simple formulation of desired endpoint accelerations for control purposes. Also, given the joint accelerations, the computation of the control torques can be computed via an efficient $\mathcal{O}(n)$ algorithm. The control systems being considered for implementation are based on the computed torque formulation introduced by Craig[9].

3.2 Motivation

Computed torque formulations of the past have found the redundant degrees of freedom possessed by floating base robots awkward to deal with. Standard computed torque schemes rely on the inverse and derivative of the system's Jacobian, $\mathbf{J}$ as expressed by

$$\mathbf{v}^{\text{endpoints}} = \mathbf{J}\dot{q}$$

where $\mathbf{v}$ is a vector of the speeds of the manipulator endpoints, measured in some coordinate system and $\dot{q}$ are the derivatives of the system generalized coordinates. When this conventional approach is taken to the formulation of the Jacobian of a free-flying robot,

it is non-square[2] and the standard techniques to solve for the desired joint accelerations cannot be used. We will demonstrate that a square Jacobian for a floating based robot can be formulated, and that the joint accelerations can be solved for in a manner similar to the standard technique.

Continuing work in the analysis of robot dynamics by Rosenthal[31], Rodriguez[30] and others have shown that robot dynamics can be solved in $\mathcal{O}(n)$ computations. The ability to solve the inverse dynamics equations for control torques in $\mathcal{O}(n)$ computations would give the controls engineer the same benefits available to the simulation community: tractability of large problems. Our work has led us to develop an $\mathcal{O}(n)$ algorithm for control torque calculation suitable for kinematic chain robots.

3.3 Free Flying Robot Jacobian

The space robot being considered falls into the class of objects called kinematic chains. The mathematical model for kinematic chains has a special structure allowing an algorithm to easily formulate endpoint acceleration equations. The work on the formulation of the Jacobian for a free-flying robot draws on previous work[7], which formulated equations of motion, endpoint accelerations and closed chain constraint equations. Notation used is that introduced by Kane[18].

This analysis is performed for a free-flying robot torso with one or more kinematic chain manipulator arms. The derivations are for 3D systems, and have been specialized to 2D for our experimental work. A generic free-flying robot has n degrees of freedom, which account for the 3 degrees of freedom of each arm, and 6 degrees of freedom of the robot's torso. The robot's torso's degrees of freedom are not directly controllable by the arm torquers if we wish to achieve arm endpoint control.

3.3.1 Concepts used in Analysis

This theory for serial chain manipulators is derived using Kane's dynamical analysis techniques. The analysis that follows assumes that the velocities $\mathbf{v}$ of points and angular velocities ω of bodies in the system under consideration can be expressed in a Newtonian reference frame as follows:

$$\mathbf{v}^i = \sum_{s=1}^{p} \mathbf{v}_s^i u_s$$

$$\omega^i = \sum_{s=1}^{p} \omega_s^i u_s$$

where the generalized speeds $u_{1..n}$ are linear combinations of the derivatives of the generalized coordinates $\dot{q}_{1..n}$. This will be true if no part of the system is undergoing prescribed motions. The partial angular velocities of bodies, and partial velocities of points, as defined by Kane[18], can be shown to be:

$$\begin{aligned} \mathbf{v}_r &= \frac{\partial}{\partial u_r}\mathbf{v} \\ \omega_r &= \frac{\partial}{\partial u_r}\omega \end{aligned}$$

3.3.2 Jacobian Element Equations

In this endpoint acceleration control specifications will be expressed in terms of joint speeds and accelerations. The standard computed torque method, which relies heavily on the Jacobian, will be briefly overviewed. We wish to be able to determine the Jacobian scalars j_{rs} which form a Jacobian matrix as follows:

$$\mathbf{v}^{\text{endpoint}} = \mathbf{J}u_{1..n}$$

The endpoint acceleration can then be expressed as:

$$\mathbf{a}^{\text{endpoint}} = \mathbf{J}\dot{u}_{1..n} + \dot{\mathbf{J}}u_{1..n}$$

and the joint accelerations are typically solved for by rearranging these equations:

$$\dot{u}_{1..n} = \mathbf{J}^{-1}(\mathbf{a}^{\text{endpoint}} - \dot{\mathbf{J}}\ u_{1..n})$$

This matrix equation can be broken down into components which are dependent upon the partial velocities and partial angular velocities of the endpoint of the each kinematic chain in the system. Each endpoint velocity can be expressed in terms of its partials as:

$$\mathbf{v}^{\text{endpoint}} = \sum_{r=1}^{n} \mathbf{v}_r^{\text{endpoint}} u_r$$

and therefore endpoint velocity can be expressed in terms of speeds along some established inertial x,y and z directions, for example, along unit vectors which we define as $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$:

$$\begin{aligned} \mathbf{v}^{\text{endpoint}} \cdot \mathbf{x} &= \sum_{r=1}^{n} \mathbf{v}_r^{\text{endpoint}} \cdot \mathbf{x}\ u_r \\ \mathbf{v}^{\text{endpoint}} \cdot \mathbf{y} &= \sum_{r=1}^{n} \mathbf{v}_r^{\text{endpoint}} \cdot \mathbf{y}\ u_r \\ \mathbf{v}^{\text{endpoint}} \cdot \mathbf{z} &= \sum_{r=1}^{n} \mathbf{v}_r^{\text{endpoint}} \cdot \mathbf{z}\ u_r \end{aligned}$$

the elements of the Jacobian due to the endpoint velocity values are therefore:

$$\begin{aligned} j_{1r} &= \mathbf{v}_r^{\text{endpoint}} \cdot \mathbf{x} \\ j_{2r} &= \mathbf{v}_r^{\text{endpoint}} \cdot \mathbf{y} \\ j_{3r} &= \mathbf{v}_r^{\text{endpoint}} \cdot \mathbf{z} \\ &\vdots \end{aligned}$$

These partial velocities are the same as the ones used in the dynamical derivations, which were discussed in previous work[7].

Endpoint acceleration control specification can be expressed in terms of the Jacobian, its derivative, and the generalized speeds and their derivatives:

$$\mathbf{a}^{\text{endpoint}} = \mathbf{J}\dot{u}_{1..n} + \dot{\mathbf{J}}u_{1..n}$$

The derivatives of the elements of the Jacobian can also be determined from quantities used in the dynamical equation formulation:

$$\begin{aligned} \dot{j}_{1r} &= \dot{\mathbf{v}}_r^{\text{endpoint}} \cdot \mathbf{x} \\ \dot{j}_{2r} &= \dot{\mathbf{v}}_r^{\text{endpoint}} \cdot \mathbf{y} \\ \dot{j}_{3r} &= \dot{\mathbf{v}}_r^{\text{endpoint}} \cdot \mathbf{z} \\ &\vdots \end{aligned}$$

If we assume a system, $\mathcal{S}$, consists of a free-floating kinematic chain, with robot arms which possess 3 independent links with controls, then each endpoint will possess 3 translational degrees of freedom and the torso will possess 6 degrees of freedom, since it is free to translate and rotate in space. We are able to specify the desired endpoint acceleration, but not the torso motion. The joint accelerations cannot be solved for purely by knowing the endpoint motion, as is possible on a fixed base robot. When controlling a free-flying robot, the motions of the torso must be accounted for. These motions, expressed by the generalized speeds and coordinates of the torso, need to be solved simultaneously with the motions of the arms. This can be done by including additional equations which ensure system consistency given its dynamic properties.

3.3.3 Jacobian Augmentation Equations

The Jacobian needs to be augmented with several equations to reflect the relations between joint speeds in a free-floating kinematic chain. Constraint relations can be used in this manner to solve joint accelerations in constrained systems (ie. closed chain), as reported earlier[7], or they can be used to solve underdetermined systems, such as free-floating robots with redundant degrees of freedom. In this section, we will add equations to the Jacobian in order to be able to solve the case of the underdetermined system. Augmentation equations equations could be taken directly from the system dynamics, however, they contain motor torques which are not known a priori. Instead, we will investigate the linear and angular momenta conservation equations. The linear and angular momenta in free flying robots are either conserved quantities, or vary according to the settings of system thrusters. We will assume that these thruster settings are known a priori.

First, the linear momentum, then the angular momentum of the system will be examined. The linear momentum of body i in the system is

$$\begin{aligned} \mathbf{L}^i &= m^i \mathbf{v}^{i*} \\ &= m^i \sum_{s=1}^{n} \mathbf{v}_s^{i*} u_s \\ &= \sum_{s=1}^{n} \mathbf{L}_s^i u_s \end{aligned}$$

where the partial linear momentum of body i is defined by

$$\mathbf{L}_s^i \triangleq m^i \mathbf{v}_s^{i*}$$

The linear momentum of system of a system of ν bodies is the sum of the linear momenta of each body i in the system:

$$\begin{aligned} \mathbf{L} &\triangleq \sum_{i=1}^{\nu} \mathbf{L}^i \\ &= \sum_{i=1}^{\nu} m^i \mathbf{v}^{i*} \\ &= \sum_{i=1}^{\nu} m^i \sum_{s=1}^{n} \mathbf{v}_s^{i*} u_s \\ &= \sum_{i=1}^{\nu} \sum_{s=1}^{n} m^i \mathbf{v}_s^{i*} u_s \\ &= \sum_{i=1}^{\nu} \sum_{s=1}^{n} \mathbf{L}_s^i u_s \\ &= \sum_{s=1}^{n} \mathbf{L}_s u_s \end{aligned}$$

where the partial linear momentum of the system is defined by

$$\mathbf{L}_s \triangleq \sum_{i=1}^{\nu} m^i \mathbf{v}_s^{i*}$$

The partial linear momenta of the system can be formulated using the system masses and center of mass partial velocities. The integration of these vector quantities into the Jacobian is similar to the process used for the partial velocities discussed in the previous section, and will be discussed further at the end of this section, after the angular momentum terms are examined.

The angular momentum of each body i, about its center of mass is:

$$\mathbf{H}^i \triangleq \mathbf{I}^{i/i*} \boldsymbol{\omega}^i$$

$$
\begin{aligned}
&= \mathbf{I}^{i/i*} \sum_{s=1}^{n} \omega_s^i u_s \\
&= \sum_{s=1}^{n} \mathbf{I}^{i/i*} \omega_s^i u_s \\
&= \sum_{s=1}^{n} \mathbf{H}_s^i u_s
\end{aligned}
$$

where the partial angular momentum of each body is

$$\mathbf{H}_s^i \triangleq \mathbf{I}^{i/i*} \omega_s^i$$

The angular momentum of the system, consisting of ν bodies, about the system's center of mass point, is:

$$
\begin{aligned}
\mathbf{H} &= \sum_{i=1}^{\nu} \mathbf{H}^i + \sum_{i=1}^{\nu} (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times m^i \mathbf{v}^{i*} \\
&= \sum_{i=1}^{\nu} (\mathbf{I}^{i/i*} \omega^i + (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times m^i \mathbf{v}^{i*}) \\
&= \sum_{i=1}^{\nu} (\sum_{s=1}^{n} \mathbf{I}^{i/i*} \omega_s^i u_s + \sum_{s=1}^{n} (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times m^i \mathbf{v}_s^{i*}) \\
&= \sum_{i=1}^{\nu} \sum_{s=1}^{n} (\mathbf{H}_s^i u_s + (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times \mathbf{L}_s^i u_s) \\
&= \sum_{s=1}^{n} \mathbf{H}_s u_s
\end{aligned}
$$

where the partial angular momentum of the system is then

$$\mathbf{H}_s \triangleq \sum_{i=1}^{\nu} (\mathbf{H}_s^i + (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times \mathbf{L}_s^i)$$

The equations which describe the partial linear and angular momenta can now be used to augment the system Jacobian. This will result in a full rank Jacobian, allowing the joint accelerations to be solved for. The resultant Jacobian when the system's linear and angular momenta are taken into consideration is:

$$\mathbf{J} = \begin{bmatrix} \mathbf{H}_1 \cdot \mathbf{x} & \mathbf{H}_2 \cdot \mathbf{x} & \dots & \mathbf{H}_n \cdot \mathbf{x} \\ \mathbf{H}_1 \cdot \mathbf{y} & \mathbf{H}_2 \cdot \mathbf{y} & & \mathbf{H}_n \cdot \mathbf{y} \\ \mathbf{H}_1 \cdot \mathbf{z} & \mathbf{H}_2 \cdot \mathbf{z} & & \mathbf{H}_n \cdot \mathbf{z} \\ \mathbf{L}_1 \cdot \mathbf{x} & \mathbf{L}_2 \cdot \mathbf{x} & & \mathbf{L}_n \cdot \mathbf{x} \\ \mathbf{L}_1 \cdot \mathbf{y} & \mathbf{L}_2 \cdot \mathbf{y} & & \mathbf{L}_n \cdot \mathbf{y} \\ \mathbf{L}_1 \cdot \mathbf{z} & \mathbf{L}_2 \cdot \mathbf{z} & & \mathbf{L}_n \cdot \mathbf{z} \\ \mathbf{v}_1 \cdot \mathbf{x} & \mathbf{v}_2 \cdot \mathbf{x} & & \mathbf{v}_n \cdot \mathbf{x} \\ \mathbf{v}_1 \cdot \mathbf{y} & \mathbf{v}_2 \cdot \mathbf{y} & & \mathbf{v}_n \cdot \mathbf{y} \\ \mathbf{v}_1 \cdot \mathbf{z} & \mathbf{v}_2 \cdot \mathbf{z} & & \mathbf{v}_n \cdot \mathbf{z} \\ \vdots & & & \vdots \end{bmatrix}$$

The partial linear momenta can be normalized by the sum of the masses of the bodies in the system. These normalized quantities have the same dimensions as partial velocities, and can be treated just like the partial velocities already forming part of the Jacobian. The normalized partial linear momenta are defined as:

$$l_s \triangleq \frac{1}{\sum m^i} \mathbf{L}_s$$

The convenient feature of this normalization is that these normalized partial linear momenta of the system for the indices which describe the translation of the torso, $s = \nu, \nu+1, \nu+2$, are unit vectors equal to those appearing in every endpoint partial velocity. By performing simply subtracting the normalized partial linear momenta rows from the endpoint partial velocity rows, the size of the Jacobian can be reduced by three rows, creating a reduced set of $\dot{q}_{1..3\nu-3}$ to solve for.

3.4 Order n Inverse Dynamics

In this section we will demonstrate a simple and straightforward algorithm to solve the inverse dynamics equation for the control torques along a serial chain. Traditional computed torque control schemes have used the following equation to compute the joint torques:

$$\mathbf{M}\dot{u} = \mathbf{N}u + \mathbf{PT}$$

This method requires $\mathcal{O}(n^2)$ computations, and requires that the mass matrix and non-linear terms of the system $\mathcal{S}$ be computed, then desired joint accelerations and known joint rates be used to generate a vector from which the control torques are easily derived. We will present an alternate method of computing these joint torques in $o(n)$ computations. This method is based on the Newton-Euler method of formulating robot equations of motion, but instead of generating equations symbolically, we will generate numerical values for accelerations, joint forces and torques, and actuator torques as we traverse the robot's chain manipulator.

The algorithm consists of two phases:

First, the joint accelerations are used to determine the accelerations of all the joints and each of the center of mass points of the ν bodies in the system. We can use the link recursion relation that the acceleration at the end of a link is related to the acceleration at the start of a link as follows:

$$\mathbf{a}^{\text{end}} = \mathbf{a}^{\text{start}} + \boldsymbol{\alpha}^{\text{link}} \times \mathbf{r}^{\text{start to end}} + \boldsymbol{\omega}^{\text{link}} \times \boldsymbol{\omega}^{\text{link}} \times \mathbf{r}^{\text{start to end}}$$

where the following components are derived as follows:

$$\boldsymbol{\alpha}^{\text{link i}} = \boldsymbol{\alpha}^{\text{link i-1}} + \ddot{q}_i \cdot \lambda^{\text{i}}$$

The axis direction λ^{i} is a positive rotation, in a right handed sense, along q_i. The torso center of mass acceleration can be constructed using $u_{\nu,\nu+1,\nu+2}$, while the torso angular acceleration can be constructed using $u_{\nu+3,\nu+4,\nu+5}$. All other accelerations in the system can be determined using the torso accelerations and the link recursion relations.

Second, the forces and moments are propagated back from the end of each chain. We assume the force and moment at the end of the chain is a known value (typically zero at the end of the arm). We take moments about the joint at the start of the link, and consider only the components along the joint's axis λ^{i}. The moments due to the center of mass acceleration and the link's angular acceleration are easily evaluated given its mass properties. The joint motor torque will be the only unknown in the equation

$$\mathbf{T}^{i} \cdot \lambda^{\text{i}} = -(\mathbf{T}^{\text{link end}} + \mathbf{r}^{\text{start to end}} \times \mathbf{F}^{\text{end}} - \mathbf{r}^{\text{start to *}} \times \mathbf{m}^{\text{i}}\mathbf{a}^{\text{i*}}) \cdot \lambda^{\text{i}}$$

Now take moments about the link start point, which are the moments applied to the end of the next link in. Likewise, the sum of the forces will yield the forces applied by this link to the end of the next link in. The focus of reference can now be shifted to the next link in, where this process can be repeated until all of the control torques have been determined. If linear actuators are being used, then the actuator force solution is done using the sum of forces along the actuator axis.

If the chain is closed, then a 'squeeze' force can be assumed as a starting internal force at the link end by conceptually cutting the closed chain, and the same procedure can be followed but with the two chains generated by the cut.

The process of solving for the joint control torques or forces is fairly straightforward, and if the robot has two or more arms, the solution for the control values for the various arms can be done in parallel.

3.5 Status

The Jacobian formulation method introduced here has been used to generate the joint acceleration specification matrix equation necessary in order to solve the computed torque control problem for the general 3D case of a free-flying robot with kinematic chain manipulators. The $\mathcal{O}(n)$ inverse dynamics solution has also been derived for this general 3D case. A specialized and partially optimized derivation for 2D has been done to allow testing on our experimental free-flying robot model.

3.6 Further Research

The underlying theory will allow position control of the endpoints of robot arms on a free-flying robot platform, for the purposes of positioning and reaching out to grasp and hold an object. The methods presented here along with work previously presented[7] will yield a controller capable of position and force control for the closed loop 'capture' configuration. The algorithms presented generally require $\mathcal{O}(n)$ computations, except for the solution for the values of the joint accelerations required for the desired control, which require $\mathcal{O}(n^2)$ computations. This last computational bottleneck will be examined over the course of our further research. Our physical simulation, a two dimensional satellite robot simulator which floats on an air bearing, is very near completion, with initial experimental results being presented January 1989.

Chapter 4

Navigation and Control of Free-Flying Space Robots

Marc Ullman

4.1 Introduction

This chapter summarizes the progress to date in our research on global navigation and control of free-flying space robots. This work represents one of the key aspects of our comprehensive approach to developing new technology for space automation. Ultimately, we envision groups of fully-self contained mobile robots making up the core work force in space.

4.1.1 Motivation

Although space presents us with an exciting new frontier for science and manufacturing, it has proven to be a costly and dangerous place for people. Space is therefore an ideal environment for sophisticated robots capable of performing tasks that currently require the active participation of astronauts.

While earth based robots have not always proved to be cost effective solutions to manufacturing inefficiencies (due to the abundance of cheap labor), the tremendous cost associated with putting men in space, especially when EVA is required, makes the economics of robots in space particularly attractive.

4.1.2 Research Goals

The immediate goals of this project are to:

- demonstrate the ability to simultaneously control robot base position and arm orientation so that a free-flying robot can navigate to a specified location in space while manipulating its arm(s).

- demonstrate the ability to capture a (possibly moving) free-floating target "on-the-fly" using the manipulator arm while the base is in transit.
- provide a suitable platform for the eventual addition of A.I. based path planning and obstacle avoidance algorithms which will enhance the robustness of task execution.

4.1.3 Background

This work emphasizes the modeling of robot dynamics and the development of new control strategies for dealing with problems of:

- a non-inertially fixed base (i.e. free-floating base)
- redundancy with dissimilar actuators
- combined linear and non-linear actuators
- highly non-linear dynamics
- unstructured environments

Our laboratory work involves the use of a model satellite robot which operates in two-dimensions using air-cushion technology. We have developed a series of satellite robots which, in two dimensions, experience the drag-free and zero-g characteristics of space. These robots are fully self-contained vehicles with onboard gas supplies, propulsion, electrical power, computers, and vision systems. The latest generation of robots is also equipped with a pair of two-link arms for acquiring and manipulating target objects.

4.2 Summary of Progress

The following advances have been achieved during the past report period:

- The low pressure gas subsystem has been revised to incorporate two-stage regulation in order to provide low pressure air for pneumatically actuated end effectors.
- A revised version of the Power Control Unit (PCU) that corrects the problems associated with the original implementation has been designed and built.
- The Safety Cut-Out/Solenoid Drivers Board has been designed and manufactured.
- The board level components for the on-board computer system have been obtained along with the VxWorks real-time operating system. Successful operation of the computer system has been demonstrated.
- Initial device drivers have been written and tested to verify the functionality of the I/O system.
- A VME-bus based version of the "point-grabber" vision system used in our earlier work has been designed and is currently being fabricated.

- An enhanced network architecture incorporating gateways and subnets that facilitates off-board processing using multiple parallel processors has been proposed and tested.
- Our plan to migrate all of our design, analysis, and software development activities to a network of Sun Workstations has been proceeding smoothly.

4.3 Experimental Hardware

4.3.1 Gas Subsystem

We have revised the on-board gas subsystem to incorporate several new features including a miniature low pressure regulator, a corresponding pressure gauge, and a pair of two-way solenoid valves. (See Figure 4.1). The solenoid valves are used for controlling the air supply to a set of pneumatically actuated end effectors. These end effectors or grippers are of essentially the same design as that used in the fixed base cooperation experiments described in Chapter 1. They incorporate a simple z-axis plunger which is driven by a double-acting piston. The new regulator provides the source of low pressure air (10-60 psi) for operating these pistons while also providing much better regulation of the flotation air supply during thruster operation. This later benefit is derived by now using two-stage regulation ahead of the flotation control flow meter whereby the second stage of regulation isolates the flow meter from the step changes in supply pressure which occur as a result of the bang-bang thruster operation.

4.3.2 Power Distribution Subsystem

Two new custom printed circuit boards were developed to facilitate operation of the robot. The first was a new version of the Power Control Unit (PCU) which supports the following improvements:

- Simplified control circuitry implemented with discrete device logic for much improved noise immunity from power-on transients.
- More convenient operation including the addition of a master power-on switch.
- Support for independent operation of the battery charging circuits even when the remainder of the power distribution system is off.

The second was a new board that performs the combined functions of providing a safety cut-out circuit and solenoid driver logic and amplifiers. The safety circuit operates in several different switch selectable modes. In its nominal mode of operation, the absence of a computer generated "heartbeat" (presumable signaling that the computer has crashed) disables power to the manipulator motors and the thruster solenoids in order to prevent damage to the robot or its surroundings. Manual overrides are possible to either enable or disable the output circuitry. A capability for remote manual disable, i.e. a "kill switch," is also supported.

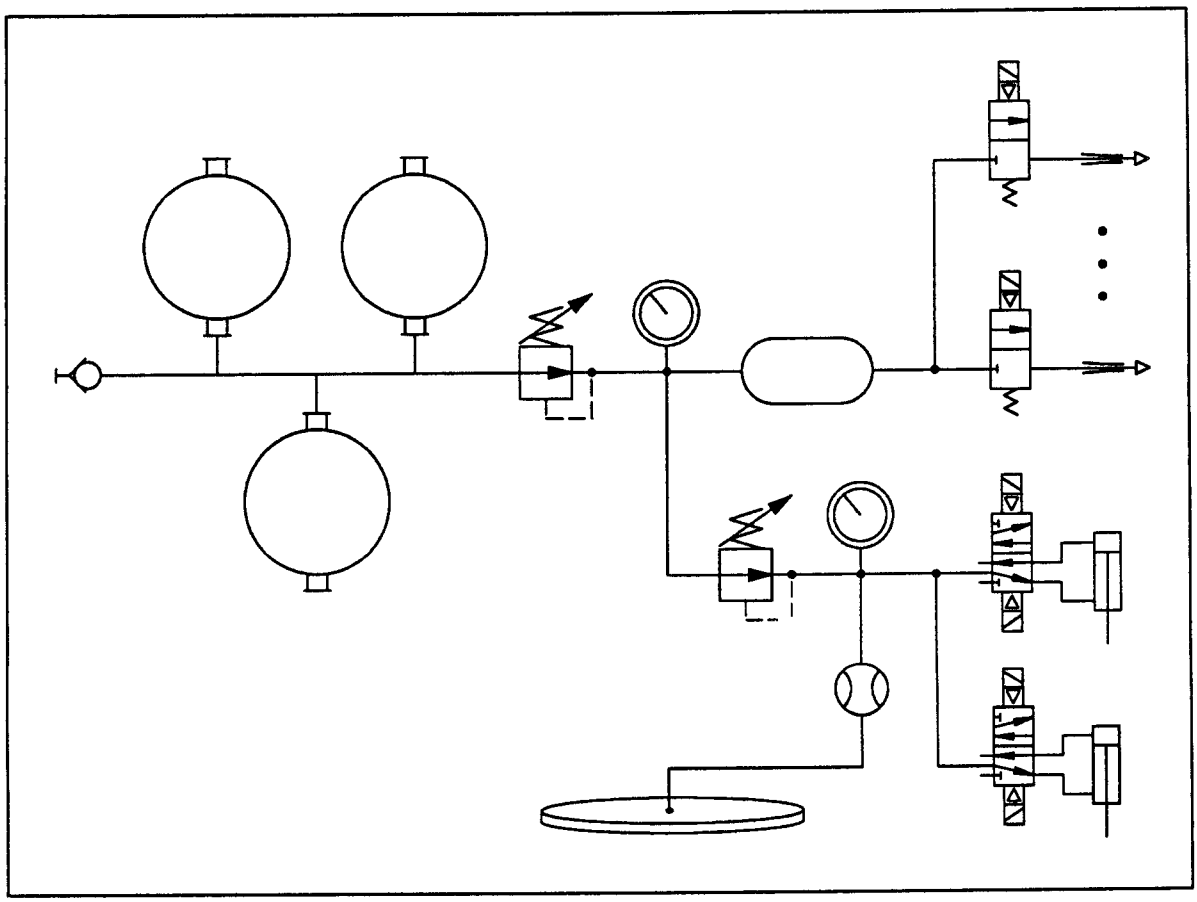


Figure 4.1: Schematic diagram of satellite robot onboard gas subsystem.

Printed circuit board implementations of these boards have been manufactured and tested and are now operating properly on board the robot.

4.3.3 Real-Time Computer

The onboard computer subsystem is now essentially complete and operational. We have connected a pair of horizontally mounted five-slot VME card cages via a 96 conductor ribbon cable. The "I/O" card cage contains a 16 channel 12 bit A/D board, a 4 channel 12 bit D/A board, and a 32 bit digital I/O board while the "computer" card cage houses a Motorola MVME 147 single board computer. This unit features a 68030 microprocessor running at 20 MHz along with a 68882 Floating Point Coprocessor. It also contains 4 MB of dynamic RAM, an Ethernet controller, a SCSI bus interface, four serial communications ports, a Centronics parallel interface, and a complete VME bus controller.

We are using Wind River Systems' VxWorks[1] as our realtime operating system and now have a device driver customized for the MVME 147. The computer uses an EPROM based boot monitor incorporating this device driver and configuration settings stored in

[1] VxWorks is described in depth in Chapter 3 of our Sixth Semi-Annual Report

EEPROM to boot over the Ethernet network connected to our Sun 3/160 file server.

4.3.4 I/O Interface Modules

Device drivers incorporating basic functionality have been written and tested to demonstrate the proper operation of the I/O system. These drivers are non-interrupt driven C-callable routines. Each device driver typically has three callable functions: An initialization call, a read procedure, and a write procedure. The drivers for output devices (DO and D/A) save the current state of the outputs (when the hardware does not) thus enabling the read calls to return the current output states.

4.3.5 Point Grabber Vision System

A new version of the "point-grabber" vision system used in our earlier experiments has been designed which incorporates the following new features:

- A VME bus compatible interface.
- The ability to handle up to four cameras simultaneously.
- Jumper selectable generation of sync signals for cameras which allow for external syncing.
- Programmable adjustment of the threshold setting

A wire-wrap prototype has been built and debugged to the point of demonstrating basic functionality; however, it has yet to been used with higher level code to preform actual vision services.

A revised version is being designed using a schematic capture and PCB layout system. As a PCB it will be feasible to produce boards for each robot as well as for possible off-board vision.

4.3.6 Multilayer Network Architecture

In light of the fact that VxWorks supports a complete implementation of the TCP/IP networking protocol including gateway and subnet capabilities, we have devised a multilayer network architecture that allows us to do parallel processing using both on-board and off-board computers.

Figure 4.2 illustrates the topology of our three levels of subnets. Our main laboratory network (ARL-Net) connects our diskless Sun Workstations and our VxWorks gateway machine(s) to our central file server.[2] Our VxWorks gateway machine consists of an open frame 20 slot VME card cage and a pair of MVME 147 single board computers[3] each with an onboard Ethernet controller. A virtual network implemented in software using shared memory transparently connects the two MVME 147s together (ACV-BP-Net). The second

[2]This fileserver also serves as a gateway to the campus wide network.

[3]These are the same computers as described above in the section entitled Real Time Computer.

MVME 147 serves as gateway to a third network (ACV-Net)[4] connecting to the on-board computer.

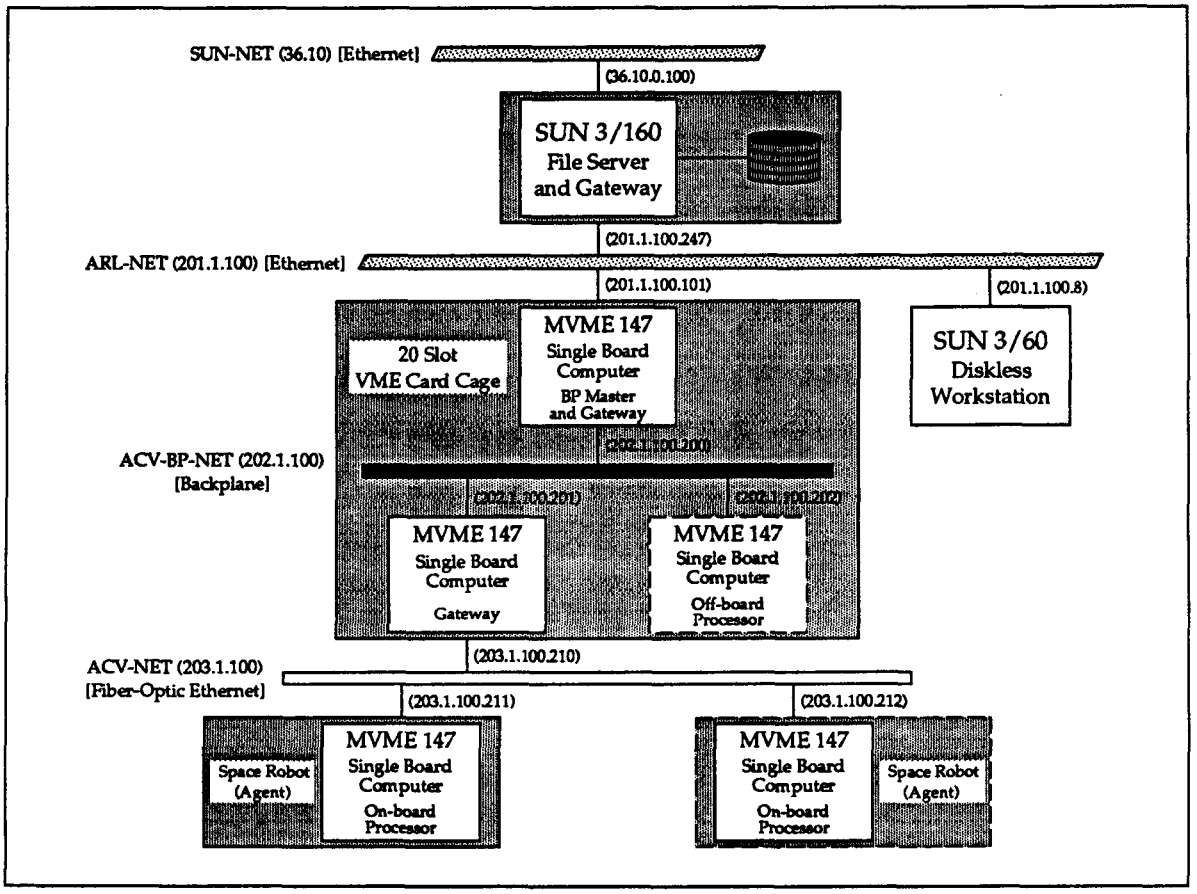


Figure 4.2: Distributed Realtime Computer System Network Topology.

This topology offers a number of benefits:

- Since these devices support TCP/IP as well as NFS, all communications between processors are transparent at the socket, RPC, or NFS level. This means that software can be constructed (except for timing considerations) without regard to whether two processors are in the same card cage or even on the same network. Once the system is configured, all routing though the gateway machines is completely transparent to the user.

- By isolating the communications between the on-board and off-board processors with a private subnet, the likelihood of collisions is greatly reduced due to the limited

[4]This network is to be implemented using a fiber optic Ethernet to minimize the effects of the cable on the robot dynamics. Ideally the link would be made using RF transmitters and receivers; however, we have been unable to find a low cost system capable of handling the 10 Mbps bandwidth with an acceptable level of reliability

access hence improving the real-time performance of the network.[5]

- As we extend our work from cooperating manipulators to cooperating robots it will become necessary to provide a communications channel between the cooperating agents. The proposed network topology supports this requirement in a very natural manner via the simple addition of new nodes (one for each new robot) to the third level (ACV-Net) network. The robots will then be able to communicate with each other as well as with the off-board computers that may be providing global coordination and planning.

- A desire to do sophisticated planning and obstacle avoidance will naturally lead to a requirement for greater off-board processing power. This need is also met in straight forward manner through the addition of more processor boards to the second level (ACV-BP-Net) or backplane network

The topology described above has been set up and configured and is now operating successfully. As the first gateway processor boots, it is configured as a gateway machine. This in turn allows the successive subnet computers to boot transparently through the higher level gateway machines.

4.4 Real-Time Development System

Our entire laboratory has been in process of migrating all of our software development, testing, downloading, and debugging activities to a network of Sun Microsystems diskless workstations. The network environment, based on an Ethernet LAN, couples our computers together so that all software and data can be shared transparently. This facilitates technology transfer and enhances information sharing among the many different projects in our laboratory.

The selection of the VxWorks realtime operating system software package is one of the important corner pins making this consolidation possible. To date, the transition has been proceeding smoothly. We have sold our pair of VaxStation II's and upgraded three Fujitsu Eagle Disk Drives to a pair of new 2382 drives, thereby nearly doubling our on-line storage capacity while improving performance and reducing maintenance costs. We hope to be able to upgrade our server and continue to add new workstations to the network as our computing requirements increase.

[5]Because Ethernet is a CSMA/CD based network with uses a combined exponential and random backoff algorithm to avoid collisions it offers no guaranteed upper limit on packet delivery time; however, as network traffic is decreased, the statistical likelihood of an unresolvable collision is greatly decreased.

It should also be noted that it is our intent to use off board computers in a client/server paradigm whereby a delayed response would not catastrophic. That is to say, asynchronous processes would be running on the off board computers that would not be tied "lock-step" to the real-time control loops running on-board.

4.5 Modeling and Simulation

The complete dynamical equations of motion have been derived and verified for a single-armed version of the robot. These equations have been coded up and simulated for both free and forced motion.

4.5.1 Analytical Model

The robot has initially been modeled with only one arm, since the global control and target capturing problems can be addressed with this somewhat simpler configuration. (See the section on Multi-Arm Cooperation for a derivation of the equations of motion for the two-armed version.) The model consists of three planar rigid bodies connected by two torque motors. (See Figure 4.3). The base body is capable of translation and rotation in the plane via eight on-off-on thrusters mounted as 90° opposed pairs on each of four corners.

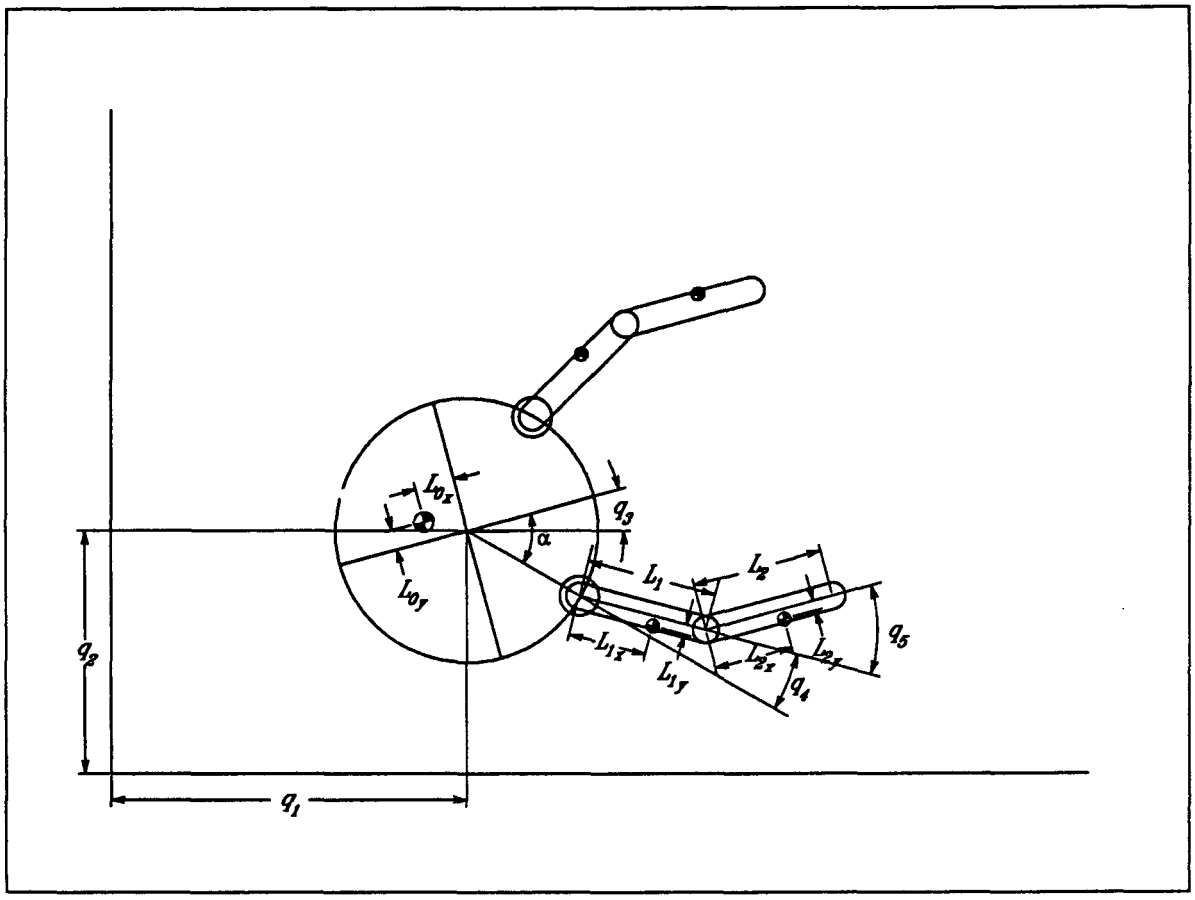


Figure 4.3: Free body diagram of space robot indicating nomenclature used for dynamic modelling.

4.5.2 Equations of Motion

The equations of motion for this five-degree-of-freedom system were derived using Kane's method[19] and for verification purposes were also derived using the symbolic equation generation program SDEXACT[32]. The joint space equations of motion can be expressed in terms of a vector of generalized coordinates $\mathbf{q}$ (corresponding to the joint positions and angles) and a vector of generalized speeds $\mathbf{u}$. They are of the form:

$$F_r + F_r^* = 0$$

or

$$\mathbf{F} - M(\mathbf{q})\dot{\mathbf{u}} - V(\mathbf{q},\mathbf{u})\mathbf{u} = 0$$

where

$$\mathbf{F}^* = -(M(\mathbf{q})\dot{\mathbf{u}} + V(\mathbf{q},\mathbf{u})\mathbf{u})$$

$M(\mathbf{q})$ is the configuration dependent mass matrix, $V(\mathbf{q},\mathbf{u})$ is the configuration and velocity dependent matrix of non-linear terms, and $\mathbf{F}$ is the vector of generalized active forces. The $\mathbf{u}$'s or generalized speeds are defined in terms of the state derivatives, $\dot{\mathbf{q}}$, by the relation

$$\mathbf{u} = Y\dot{\mathbf{q}}$$

where

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

In order to implement a simulation, we must solve the previous set of equations to obtain

$$\dot{\mathbf{q}} = Y^{-1}\mathbf{u}$$

$$\dot{\mathbf{u}} = M(\mathbf{q})^{-1}\left(\mathbf{F} - V(\mathbf{q},\mathbf{u})\mathbf{u}\right)$$

4.5.3 Operational Space Computed Torque Controller

As discussed in our fifth Semi-Annual Report, a computed torque[9] controller was implemented as a first cut at closed-loop control. This controller has since been extended to operate in "operational space"[21] so that trajectories are now specified in terms of the manipulator tip positions rather than in terms of joint angles. This later approach is much preferred when one is generating trajectories with a path planning algorithm.

The extension of the computed torque controller into operational (or cartesian) space is fairly straight forward. We begin by defining a state vector of coordinates which describe

the robot configuration in terms of variables we are directly interested in controlling. In our case, we have selected

$$\mathbf{x} = \begin{bmatrix} x_{B_c} & y_{B_c} & \theta_B & x_{tip} & y_{tip} \end{bmatrix}^T$$

where (x_{B_c}, y_{B_c}) is the position of the center of the base in inertial space, θ_B is the orientation of the base in inertial space, and (x_{tip}, y_{tip}) is the position of the end point of the manipulator in inertial space. A set of basic kinematic equations relates this state vector to our original set of generalized coordinates $\mathbf{q}$ so we can write:

$$\mathbf{x} = KIN(\mathbf{q})$$

Typically one defines the relation between the time derivatives of these two state vectors in inertial space as the Jacobian yielding:

$$\dot{\mathbf{x}} = J\dot{\mathbf{q}}$$

However, since our equations have been cast in terms of generalized speeds, $\mathbf{u}$, we find it more convenient to make the following definition:

$$\mathcal{J} = JY^{-1}$$

so that

$$\dot{\mathbf{x}} = JY^{-1}\mathbf{u} = \mathcal{J}\mathbf{u}$$

Differentiating this relationship leads to

$$\ddot{\mathbf{x}} = \dot{\mathcal{J}}\mathbf{u} + \mathcal{J}\dot{\mathbf{u}}$$

from which we can solve for $\dot{\mathbf{u}}$

$$\dot{\mathbf{u}} = \mathcal{J}^{-1}(-\dot{\mathcal{J}}\mathbf{u} + \ddot{\mathbf{x}})$$

We can replace $\ddot{\mathbf{x}}$ with a desired acceleration $\mathbf{a}_{des}$ composed of both feed forward and feedback terms resulting from our commanded trajectory and our feedback control law respectively. With a simple proportional-derivative (PD) control law our desired acceleration vector consists of the following terms

$$\mathbf{a}_{des} = \ddot{\mathbf{x}}_{cmd} + K_v(\dot{\mathbf{x}}_{cmd} - \dot{\mathbf{x}}) + K_p(\mathbf{x}_{cmd} - \mathbf{x})$$

where K_p and K_v are diagonal matrices containing the proportional and derivative feedback gains respectively.

Substituting the resulting expression for $\dot{\mathbf{u}}$ back into our original equations of motion yields a set of generalized forces which represent our "inverse dynamics" control vector of forces and torques.

$$\begin{aligned}\mathbf{F} &= M(\mathbf{q})\{\mathcal{J}^{-1}(-\dot{\mathcal{J}}\mathbf{u}+\mathbf{a}_{des})\}+V(\mathbf{q},\mathbf{u})\mathbf{u}\\ &= M(\mathbf{q})\mathcal{J}^{-1}\mathbf{a}_{des}+[-M(\mathbf{q})\mathcal{J}^{-1}\dot{\mathcal{J}}+V(\mathbf{q},\mathbf{u})]\mathbf{u}\end{aligned}$$

The resulting generalized forces can then be mapped onto the available actuators (thrusters and arm torque motors) using the pseudo-inverse and thresholding techniques described in our previous report[6].

4.5.4 Trajectory Generation

Currently we are using commanded trajectories in the form of time and amplitude scaled unit step fifth order polynomials. Now that we have implemented an operational space controller, these trajectories are described in our cartesian work space rather than in joint space as was required with our joint space controller. Fifth order polynomials were selected so that we can match a desired position, velocity, and acceleration at both ends of the trajectory. The trajectory specification is given as a matrix T_{spec} of the form:

$$T_{spec} = \left[\begin{array}{ccccc} \mathbf{x}_0 & \dot{\mathbf{x}}_0 t_f & \ddot{\mathbf{x}}_0 t_f^2 & \mathbf{x}_f & \dot{\mathbf{x}}_f t_f & \ddot{\mathbf{x}}_f t_f^2 \end{array}\right]$$

The actual trajectories are then computed using the equations

$$\begin{aligned}\mathbf{x}_{cmd} &= a\tau^5+b\tau^4+c\tau^3+d\tau^2+e\tau+f\\ \dot{\mathbf{x}}_{cmd} &= (5a\tau^4+4b\tau^3+3c\tau^2+2d\tau+e)/t_f\\ \ddot{\mathbf{x}}_{cmd} &= (20a\tau^3+12b\tau^2+6c\tau+2)/t_f^2\end{aligned}$$

where $\tau = t/t_f$ is normalized time and $[\ a\ \ b\ \ c\ \ d\ \ e\ \ f\ ]$ is a vector of coefficients given by

$$\left[\begin{array}{cccccc} a & b & c & d & e & f \end{array}\right] = T_{spec}C^{-1}$$

where C is the matrix of constant coefficients that results from evaluating a generic fifth order polynomial and its derivatives at 0 and 1.

4.5.5 Simulation Results

In order to verify the effectiveness of the control strategy outlined above, several test cases were run in simulation. The algorithm was coded up as a series of "M-files" using the matrix manipulation program *Matlab*. Figure 4.4 shows a simulation run in which the robot base was commanded to move from the origin to a location 1 meter to the right and 1/2 meter up while undergoing a 90 degree counter-clockwise rotation over a 10 second time interval. At the same time the arm tip was commanded to move from (-0.25,-0.5)[6] to (1.5,0.5) and to arrive with a velocity of 10cm/s in the positive y direction. Figure 4.5

[6] All coordinate pairs refer to dimensions in meters relative to the origin.

shows the time history of the actuator forces and torques used for this trajectory. It shows the resulting pulse width modulation (PWM) of the eight thrusters along with the zero order hold (ZOH) torque outputs for the manipulator motors. Thus these results show that given a "reasonable" trajectory, we can control both the base and the manipulator using the control formulation presented above.

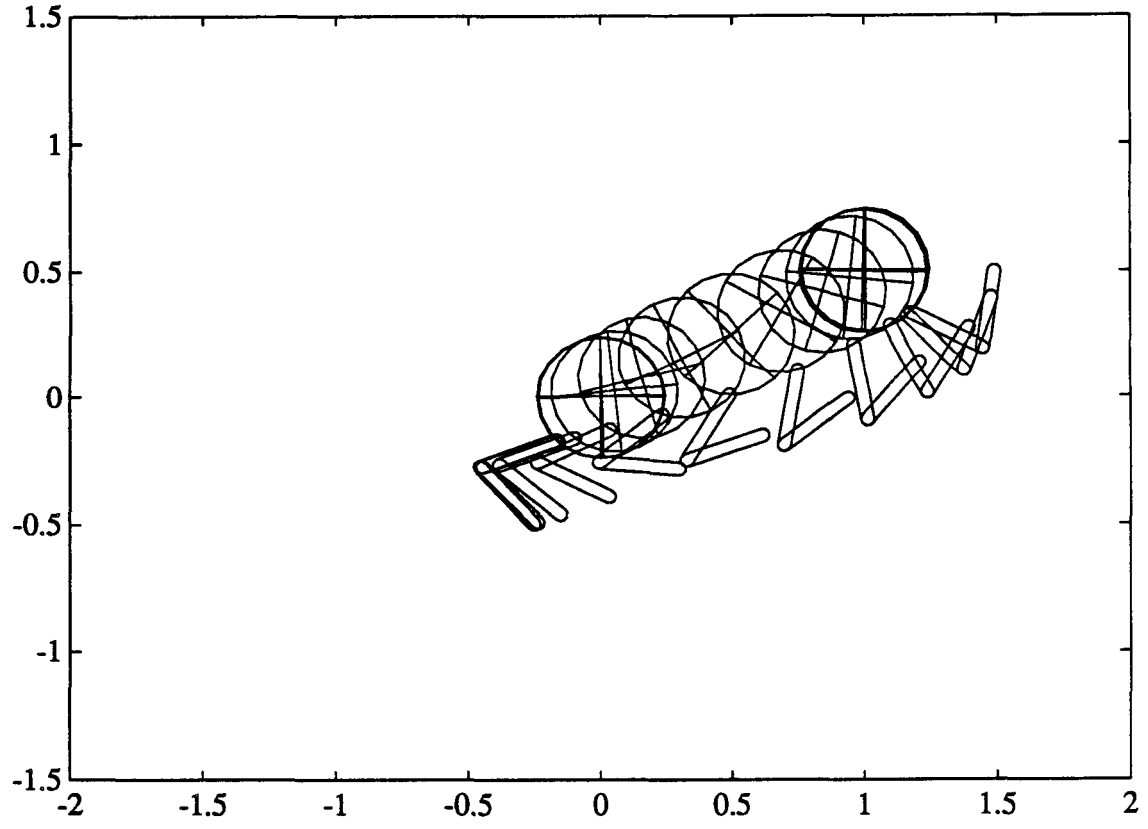


Figure 4.4: Time-lapse plot from simulation of space robot executing combined base and manipulator motion under closed loop control.

4.6 Summary

Significant progress has been made during the past report period on both the hardware and the analysis fronts. We now have an operational real-time computer system that supports remote login, remote debugging, multiple processors, distributed processing, and transparent networking. We have demonstrated successful operation of the gas subsystem (with its improvements), the power distribution system, the I/O subsystem, and the prototype vision system.

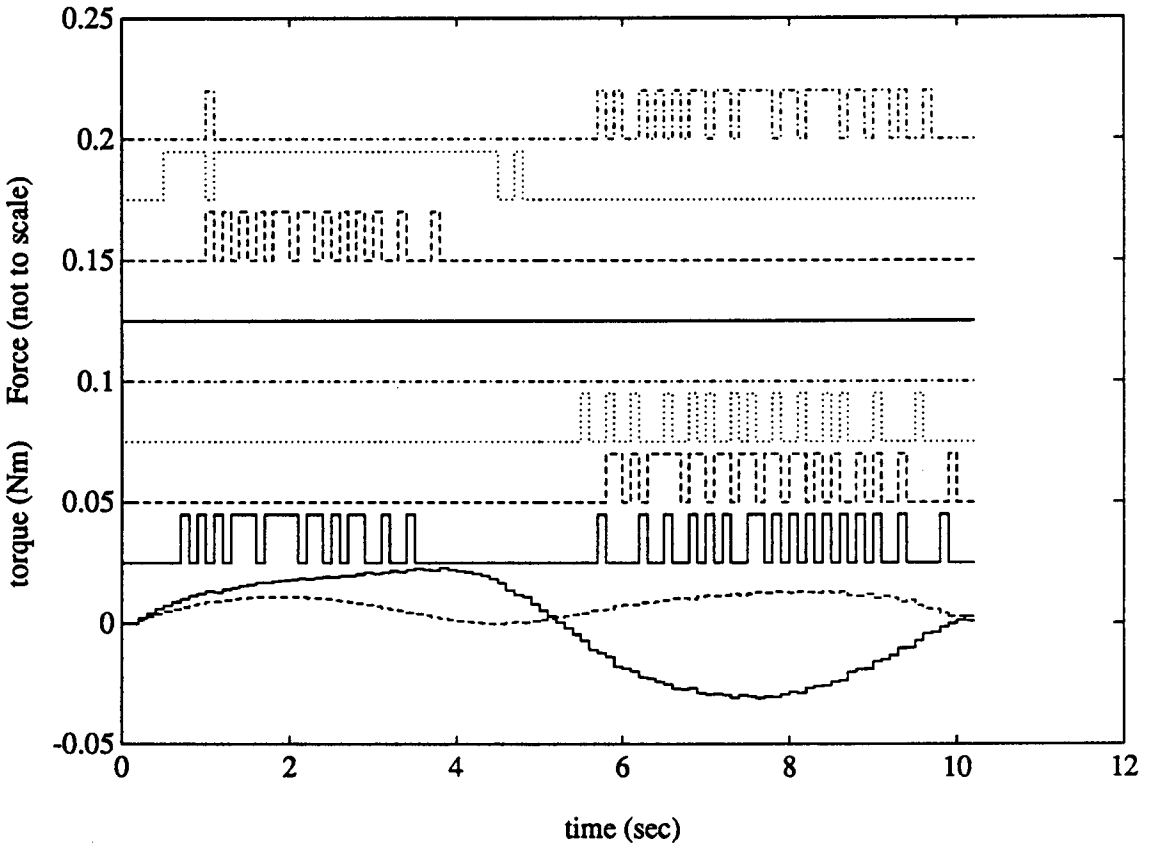


Figure 4.5: Thruster and torque motor time histories used in executing space robot trajectory shown above.

4.7 Future Work

Our robot is ever closer to becoming operational. All major subsystem components are now in place. The principle work remaining entails wiring and interconnecting the vast array of components. Specific items yet to be completed include:

- Completing the I/O Transition Module—an interface/patch board that provides a generic connection between the I/O Subsystem and the Analog Subsystem while providing custom configuration and signal breakout/monitoring.
- Wiring the sensors and actuators.
- Installing the manipulator end effectors (grippers).
- Mounting the onboard and off-board cameras.
- Mounting the angular rate sensor and the x-y accelerometers which will serve as the core of an onboard INS system for tracking vehicle position and orientation.

The modular design philosophy, which has been a guiding principle for this project since its inception, will continue to apply as our focus begins to shift away from hardware toward software.

Chapter 5

Multiple-Vehicle Cooperation

William C. Dickson

5.1 Introduction

This chapter introduces a new line of research being conducted in the area of multiple-vehicle cooperation. This work will eventually unite the various lines of research presently being conducted in fixed- and floating-base cooperative manipulation, and in global navigation and control of space robots. Our goal is to demonstrate multiple free-floating robots working in teams to carry out tasks too difficult or complex for a single robot to perform. Achieving this cooperative ability will involve solving specialized problems in dynamics and control, high-level path planning, and communication.

Progress Summary

Activities completed from March 1988 to August 1988 were:

- Far-range research goals were defined
- Initial model of the physical system was developed
- Candidate path/motion planning algorithm was developed
- Closed-loop control for two-robot manipulation was successfully simulated

5.2 Research Goals

Some of the goals of this project are:

- Cooperative manipulation and assembly by multiple robots
- Fine cooperative manipulation in presence of on-off control

- Development of control strategies for path following in presence of obstacles and large disturbances
- Path generation considering dynamic constraints and known geometric boundaries
- Task planning for complex assemblies

5.3 Experimental Hardware

The vehicles to be used in this research are a pair of the two-armed free-floating robots used in the LEAP experiment (see Chapter 6). As with the Navigation and Control research, only one of the two arms on each of the robots will be used, removing the issue of cooperation between arms on the same robot and allowing the research to focus on cooperation between independent vehicles.

The vehicles will be manipulating a free-floating object having either two receptacles for vehicle dockings (as with the object used in the fixed-base cooperation research) , or a bar to be grasped by grippers similar to those used in the LEAP experiment. The mass and inertia of the object will be adjustible so that studies can be made concerning a given controller's performance over a range of these parameters.

Experiments will be performed on the 9' × 12' granite table. This large size is necessary if the robots and manipulated object are to perform interesting configuration changes or maneuvers (such as moving around obstacles).

5.4 Modelling

The robot is modelled as a three-link chain consisting of a free-floating base and a single two-link arm—resulting in the five-degree-of-freedom system described by $q_i(i=1,\ldots,5)$ as shown in Fig. 5.1. The set of actuators consists of eight on-off thrusters (mounted as 90° opposed pairs on each of four corners of the base), a momentum wheel on the base, and torque motors at the shoulder and elbow of the arm. For modelling and control purposes, the thrusters are grouped into two perpendicular, multi-directional, on-off-on sets. With this simplification, base control forces and torques are conveniently separated between the thrusters and momentum wheel, respectively. Thus, each vehicle has five controls—two thruster forces (F_1 and F_2) for two-dimensional translation, and three torques (T_1, T_2, and T_3) for orientation. f_x and f_y are manipulation forces.

The Equations Of Motion (EOM) for the five-degree-of-freedom system were derived using Kane's method [18] and can be written as:

$$\begin{aligned} \mathbf{M}\dot{\mathbf{u}} &= \mathbf{b} + \mathbf{G}\boldsymbol{\tau} + \mathbf{H}\mathbf{f} \\ \dot{\mathbf{q}} &= \mathbf{u} \end{aligned} \tag{5.1}$$

where $\mathbf{q}$, $\mathbf{u}$, $\boldsymbol{\tau}$, and $\mathbf{f}$ are defined as:

$$\mathbf{q} \triangleq [\, q_1\ q_2\ q_3\ q_4\ q_5\,]^T,$$

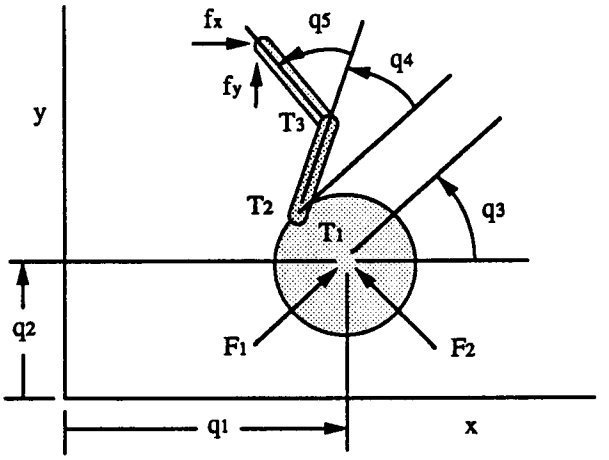


Figure 5.1: Modelling of single-arm vehicle

$$\begin{aligned}
\mathbf{u} &\triangleq [\, u_1\ u_2\ u_3\ u_4\ u_5\,]^T, \\
\tau &\triangleq [\, F_1\ F_2\ T_1\ T_2\ T_3\,]^T, \\
\mathbf{f} &\triangleq [\, f_x\ f_y\,]^T.
\end{aligned}$$

5.5 Controller

Computed torque controllers solve systems' EOM for the forces and torques required to produce desired accelerations. With the present system, given by the EOM in (5.1), **G** is always invertible so that the τ necessary to produce the desired acceleration vector $\dot{\mathbf{u}}_{des}$ can be given as:

$$\tau = \mathbf{G}^{-1}(\mathbf{M}\dot{\mathbf{u}}_{des} - \mathbf{b} - \mathbf{H}\mathbf{f}).$$

However, in our case, the first two elements of τ, F_1 and F_2, are available only in the discrete values of 0 and $\pm F_{max}$, thus not all $\dot{\mathbf{u}}_{des}$ can be produced. More precisely, only three linear combinations of the five terms in $\dot{\mathbf{u}}_{des}$ can be arbitrarily specified—the other two vary discretly due to the on-off-on values of F_1 and F_2. This problem of specifying desired accelerations was resolved as follows.

First, we define a new velocity vector **v** :

$$\mathbf{v} \triangleq [\, u_1\ u_2\ u_3\ v_x\ v_y\,]^T,$$

where v_x and v_y are the manipulator tip speeds in the x and y directions. Defining **a** as the time derivative of **v**,

$$\mathbf{a} \triangleq \frac{d}{dt}\mathbf{v} = [\, \dot{u}_1\ \dot{u}_2\ \dot{u}_3\ a_x\ a_y\,]^T,$$

we find that **a** can be written as

$$\mathbf{a} = \mathbf{R}\dot{\mathbf{u}} + \mathbf{s},$$

and since $\mathbf{R}$ is invertible,

$$\dot{\mathbf{u}} = \mathbf{R}^{-1}(\mathbf{a} - \mathbf{s}).$$

We can now rewrite (5.1) in terms of $\mathbf{a}$:

$$\mathbf{M}\mathbf{R}^{-1}(\mathbf{a} - \mathbf{s}) = \mathbf{b} + \mathbf{G}\tau + \mathbf{H}\mathbf{f}.$$

Using the substitutions

$$\begin{aligned} \mathbf{N} &\triangleq \mathbf{M}\mathbf{R}^{-1}, \\ \mathbf{c} &\triangleq \mathbf{b} + \mathbf{N}\mathbf{s}, \end{aligned}$$

we arrive at a new set of EOM expressed in terms of $\mathbf{a}$:

$$\mathbf{N}\mathbf{a} = \mathbf{c} + \mathbf{G}\tau + \mathbf{H}\mathbf{f}. \tag{5.2}$$

The motivation for writing the EOM as in (5.2) is that the acceleration vector $\mathbf{a}$ now contains terms that directly describe the motion of the manipulated object — namely a_x and a_y, the manipulator tip accelerations in the x and y directions. Precise object control requires that arbitrary values of these accelerations be achievable. As before, only three of the five terms in $\mathbf{a}$ can be arbitrarily chosen. With two being the important tip accelerations a_x and a_y, the remaining choice is the base angle acceleration $\dot{u}_3$.

Thus, $\mathbf{a}$ can be partitioned into determined and arbitrary parts:

$$\mathbf{a} = [\, \mathbf{a}_1^T \mid \mathbf{a}_2^T \,]^T = [\, \dot{u}_1 \; \dot{u}_2 \mid \dot{u}_3 \; a_x \; a_y \,]^T.$$

τ, $\mathbf{N}$, and $\mathbf{G}$ are similarly partitioned:

$$\tau = [\, \tau_1^T \mid \tau_2^T \,]^T = [\, F_1 \; F_2 \mid T_1 \; T_2 \; T_3 \,]^T,$$

$$\begin{aligned} \mathbf{N} &= [\, \mathbf{N}_1 \mid \mathbf{N}_2 \,], \\ \mathbf{G} &= [\, \mathbf{G}_1 \mid \mathbf{G}_2 \,]. \end{aligned}$$

We can recombine the equations in (5.2) according to these partitionings to arrive at:

$$[\, \mathbf{N}_1 \mid -\mathbf{G}_2 \,] \begin{bmatrix} \mathbf{a}_1 \\ - \\ \tau_2 \end{bmatrix} = \mathbf{c} + [\, \mathbf{G}_1 \mid -\mathbf{N}_2 \,] \begin{bmatrix} \tau_1 \\ - \\ \mathbf{a}_2 \end{bmatrix} + \mathbf{H}\mathbf{f}.$$

Defining $\mathbf{W}$ and $\mathbf{P}$ as:

$$\begin{aligned} \mathbf{W} &\triangleq [\, \mathbf{N}_1 \mid -\mathbf{G}_2 \,], \\ \mathbf{P} &\triangleq [\, \mathbf{G}_1 \mid -\mathbf{N}_2 \,], \end{aligned}$$

we arrive at the EOM expressed in a convenient form for control purposes:

$$\mathbf{W} \begin{bmatrix} \mathbf{a}_1 \\ - \\ \tau_2 \end{bmatrix} = \mathbf{c} + \mathbf{P} \begin{bmatrix} \tau_1 \\ - \\ \mathbf{a}_2 \end{bmatrix} + \mathbf{H}\mathbf{f}. \tag{5.3}$$

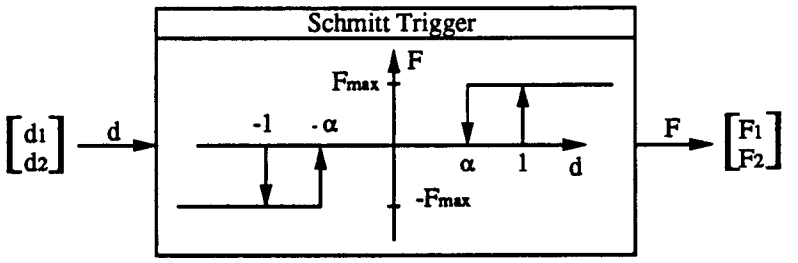


Figure 5.2: Schmitt trigger used to determine thruster forces

Given the discrete thruster forces τ_1 and the desired accelerations $\mathbf{a}_2$, we can determine the resulting base accelerations $\mathbf{a}_1$ and the required control torque vector τ_2:

$$\begin{bmatrix} \mathbf{a}_1 \\ - \\ \tau_2 \end{bmatrix} = \mathbf{W}^{-1}(\mathbf{c} + \mathbf{P}\begin{bmatrix} \tau_1 \\ - \\ \mathbf{a}_2 \end{bmatrix} + \mathbf{Hf}). \tag{5.4}$$

The control problem is thus divided between the separate tasks of first determining τ_1 (F_1 and F_2), then using these values with the desired accelerations $\mathbf{a}_2$ ($\dot{u}_3, a_x$, and a_y) to calculate the motor torques τ_2 and the resulting base accelerations $\mathbf{a}_1$.

The first candidate scheme under analysis for determining the base forces is to make F_1 and F_2 the outputs of a Schmitt trigger, as shown in Fig. 5.2. The inputs to this filter are the weighted sums of the errors in base position and velocity resolved in the two perpendicular thrust directions:

$$\begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} \cos(q_3) & \sin(q_3) \\ -\sin(q_3) & -\cos(q_3) \end{bmatrix} \begin{bmatrix} K_q & K_u & 0 & 0 \\ 0 & 0 & K_q & K_u \end{bmatrix} \begin{bmatrix} q_{1des} - q_1 \\ u_{1des} - u_1 \\ q_{2des} - q_2 \\ u_{2des} - u_2 \end{bmatrix}.$$

The parameters α, K_q, and K_u are chosen to yield desired response characteristics.

5.6 Path and Motion Planning

A path planner is required to find a viable corridor through the workspace to the desired terminal location. The path generated by the path planner will then be used to define a curvilinear coordinate system that a motion planner will use to describe the desired motion of the vehicles and manipulated object. In order to accomodate the possibility of unforeseen obstacles or a changing workspace geometry, the path will be frequently updated as the vehicles and manipulated object move toward their destination.

The visibility graph (VG) [40] is the first path-planning algorithm under study. Given a geometric representation of the workspace, the VG method searches for the shortest path from an initial to final point. Fig. 5.3 shows the path found by the VG path planner in a workspace with several obstacles that were "grown" by a safety margin to prevent collisions.

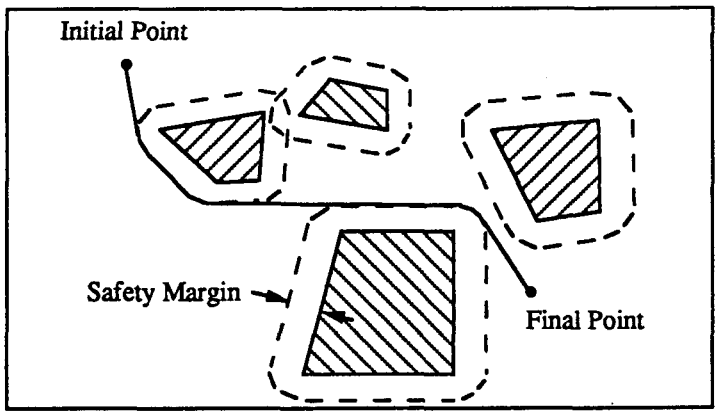


Figure 5.3: Path found by the Visibility Graph path planner

A motion planning algorithm has been developed that generates desired positions and speeds for the vehicles and object along a path found by the path planner. Where the path planner was concerned only with the workspace geometry, the motion planner takes into consideration performance limitations due to dynamic factors such as the vehicles' mass, inertia, and thrust levels. Other important factors are the vehicles' fields of view and the presence of unknown obstacles.

Fig. 5.4 shows a schematic of the control loop for the multiple-vehicle manipulation system.

5.7 Simulations

Simulations of a two-vehicle object manipulation have been successfully carried out using the matrix manipulation program Pro-Matlab [8]. One such manipulation task, shown in Fig. 5.5 , utilizes two single-armed robots transporting a beam along a straight path from A to B. An object impedance controller assigns values of a_x, a_y, f_x, and f_y to each of the two robots, then the control forces and torques are determined as discussed in Section 5.5.

At present, the path-planning algorithm has not been implemented in software, so Fig. 5.5 represents closed-loop control along a predefined path. However, since the method of generating the path is transparent to the controller, the motion of the vehicles and object along the predefined path will be identicle to the motion along the same path generated by the visibility graph or any other path planner. This independence between path planning and control will be convenient for developing both aspects of the closed-loop system.

5.8 Summary

Work has begun on the problem of multiple-vehicle cooperation. Far-range goals have been defined to give direction for progressing research. An initial control scheme has been

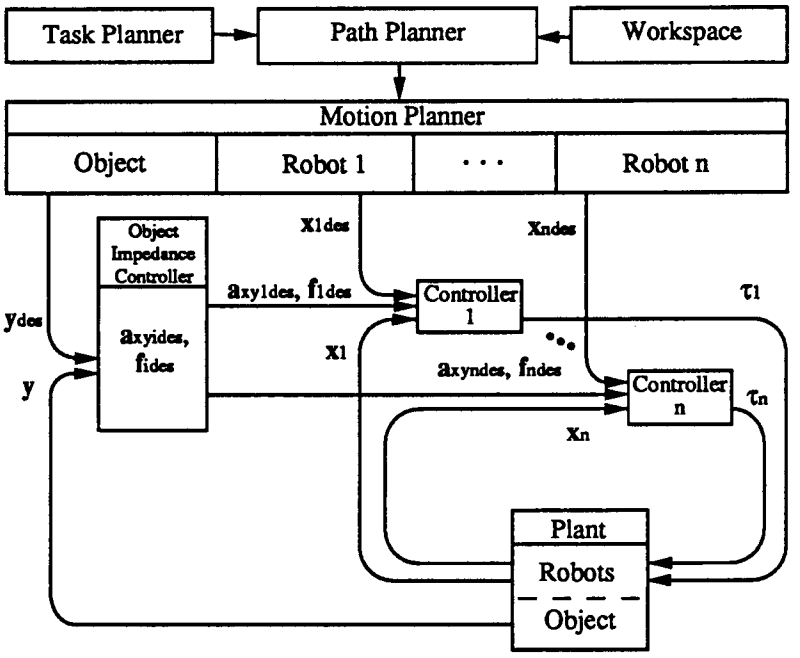


Figure 5.4: Schematic of control loop

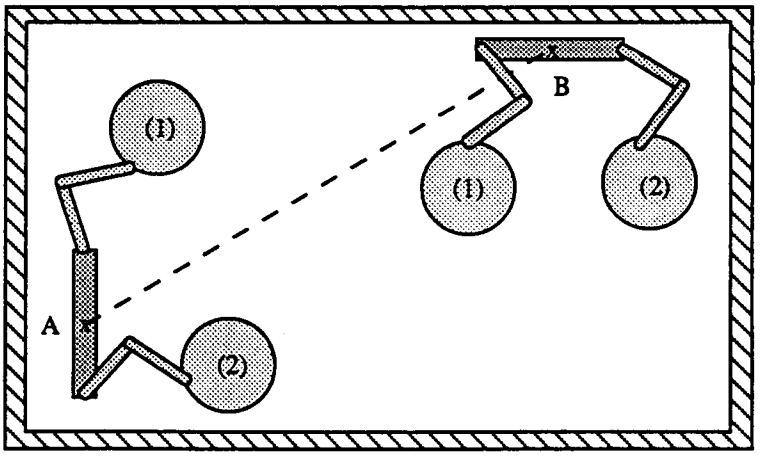


Figure 5.5: Simulation of two-vehicle object manipulation

successfully simulated, and candidate path- and motion-planning algorithms are being developed.

5.9 Future Work

The next major step in software development is to implement revised versions of the simulation code in C for eventual use with our real-time system.

The path-planning algorithm is near completion. Once the planner is operational, a study will be undertaken to determine the stability issues associated with the vehicles following a periodically changing path.

Chapter 6

Locomotion Enhancement via Arm Pushoff (LEAP)

Warren J. Jasper

6.1 Introduction

To perform complex assembly tasks, an autonomous vehicle needs to move from one place to another. There is a high premium on the use of propellants, for every kilogram of propellant is provided in space only at extreme cost. Also, the use of thrusters may disturb the environment by impacting a target which the robot is trying to grasp. Our proposal for reducing substantially the use of propellant is an approach called LEAP: Locomotion Enhancement via Arm Pushoff. In LEAP, the vehicle pushes itself off from a large space object and "leaps" to the desired resting place or simply "crawls" along an object. This is the common mode of locomotion used by the astronauts while in the Space Shuttle.

We believe that space robots can use this idea to good advantage. That is why this new project was added to investigate the problems and issues involved in autonomous space locomotion. The first phase of the project involves: devising the experiment, deriving the equations of motion and candidate control laws, and then simulating the model to size physical parameters for the actual experiment. The second phase encompasses design and fabrication of the vehicle, while the third phase is to verify experimentally the theoretical development. The following paragraphs describe the progress on phase two. Phase one is already completed.

Progress Summary

The major activities started or completed during the period March, 1988 through August, 1988 were:

- Completion of "Ballerina Bar" around the granite table.

- Completed assembly of high pressure plumbing.
- Completed fabrication of major hardware (machined) components. This includes battery packs, battery racks, analog racks, mounting plates, posts and a variety of mounting brackets. About 90% of the parts have been machined for two air cushion vehicles.
- Ordered digital electronics. This includes the CPU boards, A/D and D/A boards, and a Parallel I/O board.
- Started assembly of low pressure plumbing, thruster subsystem, and power system. Also began wiring vehicle.

6.2 The Experiment

A new air-cushion vehicle is being designed to study LEAP. This vehicle should simulate the motions that an autonomous space robot would perform while in the space station or maneuvering out in space. The experiment will consist of the vehicle pushing off a bar located on one side of the granite table, rotating 180 degs, and catching itself by grasping a bar located at the other end of the table. Ideally, one would like to complete this task without the use of thrusters. However, at the point of initial release from the bar, errors in the velocity of the center of mass of the vehicle can only be corrected using thrusters. To enhance the robustness of this approach, thrusters will be incorporated into the control laws for midcourse correction. Figure 6.1 shows the robot in three configurations: pushing off the bar, rotating, and catching itself at the other end. By incorporating crawling and leaping, the robot can position itself anywhere on the table with a minimum amount of propellant. This investigation complements current work done at the Stanford Aerospace Robotic Laboratory [5, 7] by incorporating global navigation and object manipulation into a general study of locomotion.

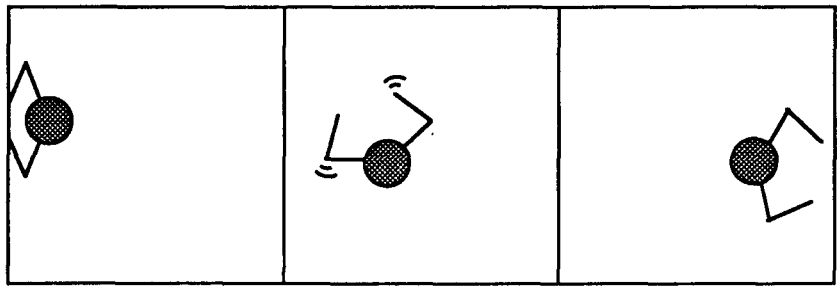


Figure 6.1: The LEAP Demonstration

6.3 Fabrication

Fabrication and assembly of two vehicles was the major emphases during the past six months. As mentioned in the Fourth semi-annual report [5], the overall design objective was to create a modular vehicle which consists of cylindrical layers. Each layer incorporates a major system, with five layers in all. Table 6.1 describes the systems in each layer and the fabrication/assembly status. This table only includes the progress/status for mechanical fabrication, and not wiring, testing or system integration.

The completion of the "ballerina bar" was another interesting task. To attach the bar to the granite table, plastic nodules with heli-coil inserts were glued to the side of the table. This was by far the cheapest method and preferable to drilling holes in the side of the table. Unfortunately, the first attempt using Loctite 401, a cyanoacrylate, failed. This is because the glue was too brittle, and fractured when a torque was applied to the nodules. The nodules were then glued using Epoxy 2216 and tightened down with strap clamps. It appears that this second approach works fine.

Task	Layer	Started	Completed
Ballerina Bar		√	√
Robotic Arms		√	√
High Pressure Plumbing	I	√	√
Low Pressure Plumbing	I	√	
Base Plate Fabrication	I	√	√
Thruster System	II	√	
Momentum Wheel System	II	√	√
Battery Packs	III	√	√
Battery Rack	III	√	√
Analog Rack	III	√	√
Digital Euro Card Cage	IV	√	
Vision System	V		

Table 6.1: Fabrication Completed

6.4 Inertial Sensing Unit

One area of concern was sensing the inertial position and rates of the robot base with respect to the inertial or lab frame. Although a vision system is being explored, it is unclear at this time whether velocity and acceleration values can be derived at a high enough bandwidth from a vision system, which essentially measures position. Acceleration data is needed to determine the amount of velocity or Δv imparted to the robot during thruster firing, while angular rate information is directly used in the computed torque control law. To sense these quantities, an accelerometer and an angular rate sensor were acquired. The angular rate sensor, made by Watson Industries, is a solid state electronic device which produces

Power supply:	±15 VDC ±5% 20 mA maximum
Output:	± 10 VDC at full scale angular rate
Sensitivity:	±100 °/second full scale
Output current:	±10 mA maximum
System frequency:	360 Hz nominal
Scale factor error:	2%
Linearity:	< 0.1% full scale
Frequency response:	DC to 30 Hz
Output noise:	5mV RMS maximum
Life:	50,000 hours MTBF minimum
Shock:	200G
Weight:	110 grams

Table 6.2: Angular Rate Sensor Specifications Model ARS-C131-1A

an output voltage when Coriolis forces causes bending in a piezoelectric bender element. This rate sensor, also known as a "tuning fork gyro" has the advantages over conventional gyros of low cost and high reliability. Table 6.2 gives some important specifications for the instrument.

The accelerometers used on the LEAP vehicle are two Systron Donner 4310 liner servo accelerometers. As Table 6.3 shows, these are high quality instruments. The output will be filtered for scale factor and bias, and integrated twice to give velocity and position. At this time, it is unclear whether the integration will be done in hardware or software.

One of the challenges will be to detect acceleration on the order of 0.1mg to 10mg in a 1g environment. Although accelerations in the x and y directions are orthogonal and theoretically decoupled from each other and the 1g gravity force in the z axes, there is some cross axis coupling on the order of <0.002g/g of applied acceleration. The major source of error will occur in detecting these small accelerations in a 1g environment. By way of example, if the accelerometers are set to read 10mg full scale, a tilt in the base of 20 arc seconds (which corresponds to a change in height of the base with respect to the granite table of 0.0009 inches) causes an error of 1% or 0.1mg. This will only become a problem if one integrates acceleration twice to derive position, for errors in acceleration grow as t^2 in position.

Power supply:	±15 VDC ±10% at 10 mA maximum
Output:	±7.5 VDC at full range
Sensitivity:	±1 g full scale
Output current:	±3 mA maximum
Zero Output (Null):	< 0.05% full range
Linearity:	< 0.05% full scale
Natural frequency:	50 - 250 Hz
Output noise:	< 7.5mV RMS
Resolution	< 0.001% full range
Shock:	100G 11msec
Weight:	128 grams

Table 6.3: Linear Servo Accelerometer Model 4310

6.5 Future Work

The robotic vehicles should be near completion in the next five months. The major tasks yet to be completed are wiring, system integration and test. Fortunately, many of these tasks can be done in parallel and so experiments should begin by the summer of 1989.

Chapter 7

Adaptive Control of LEAP

Roberto Ernesto Zanutta

7.1 Introduction

A major task of free-flying robots is to aid in the construction, maintenance and repair of space structures (e.g. the space station and satellites) while in orbit. Because of the high costs of placing mass into orbit, it is desirable to reduce the amount of propellant consumed by the free-flying robots while carrying out their tasks. Typical tasks performed by the robots will require them to transport and retrieve various objects. To minimize the amount of propellant required, the robots will have to know accurately the inertia properties of the objects they carry. Since it is not practical to specify the object mass properties each time a robot performs a task, some method of identification is necessary. This can be done through the use of adaptive control.

The investigation of adaptive control for a two-armed free-flying robot is a recent project. The previously reported work has been in the investigation of adaptive control schemes and vehicle modeling and simulation. A major part of this was a literature search. The following is a summary of the work done on the project during the last six months. This work consists mainly of control law and adaptation law simulation and analysis. Also included is a brief description of future work.

7.2 Control Law Development

The first step in the development of an adaptive controller for the vehicle is the determination of a suitable control law. This has been the main thrust of the recent research on this project. As was reported in the previous semi-annual report two approaches were found in the literature which had desireable characteristics for real-time applications. These were presented by Slotine and Li of MIT [34] and Wen and Bayard of JPL [4]. These approaches must be extended and modified for implementation on a two-armed free-flying robot. The following is a description of the modifications made and preliminary simulation results.

The control law chosen to command the robot is one proposed by Slotine and Li [34]. It is:

$$\tau = \hat{\mathbf{M}}(\mathbf{q})\ddot{\mathbf{q}}_\mathbf{r} + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_\mathbf{r} - \mathbf{K}_v\mathbf{s} \tag{7.1}$$

where

$$\dot{\mathbf{q}}_\mathbf{r} = \dot{\mathbf{q}}_\mathbf{d} - \Delta\tilde{\mathbf{q}} \tag{7.2}$$

$$\tilde{\mathbf{q}} = \mathbf{q} - \mathbf{q}_\mathbf{d} \tag{7.3}$$

$$\mathbf{s} = \mathbf{q} - \mathbf{q}_\mathbf{d} + \Delta(\mathbf{q} - \mathbf{q}_\mathbf{d}) \tag{7.4}$$

Δ and $\mathbf{K_V}$ are positive definite diagonal matrices. $\mathbf{q}$ is the vector of the system states (vehicle position and arm orientations). $\mathbf{q}_d$ is the vector of desired states. $\tilde{\mathbf{q}}$ is the vector of tracking errors. τ is the vector of forces and torques associated with the system states (applied forces and torques). $\hat{\mathbf{M}}$ is the estimated system mass matrix. $\hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_\mathbf{r}$ is the estimated system non-linear terms.

This control law does not consider the closed-loop kinematic condition that occurs when the robot is holding an object with both arms. To deal with this problem the non-holonomic equations of motion were used [18].

The control law was modified by treating the sliding mode term (7.4) as an applied force and incorporating the closed-loop kinematic constraints (see Appendix C of the Fifth Semi-Annual Report) forming the non-holonomic equations of motion. The resulting control law is:

$$\begin{aligned}\tau_\mathbf{s} + A_{rs}^T\tau_\mathbf{r} &= \mathbf{C_s} - \mathbf{K_V s_s} + A_{rs}^T(\mathbf{C_r} - \mathbf{K_V s_r}) + (\mathbf{M_{sr}} + A_{rs}^T\mathbf{M_{rr}})\mathbf{B_r} \\ &+ (\mathbf{M_{ss}} + \mathbf{M_{sr}}A_{rs} + A_{rs}^T\mathbf{M_{rs}} + A_{rs}^T\mathbf{M_{rr}}A_{rs})\dot{\mathbf{u}}_\mathbf{s}\end{aligned} \tag{7.5}$$

The s and r subscripts refer to the independent and dependent degrees of freedom respectively.

The choice of independent degrees of freedom is fairly arbitrary. The independent degrees of freedom were chosen to be the base orientation, shoulder angles and momentum wheel orientation (s = 3,4,6,8). This choice results in the applied torque distribution:

$$\tau_\mathbf{s} + A_{rs}^T\tau_\mathbf{r} = \begin{pmatrix} a_{31}\tau_5 + a_{41}\tau_7 \\ \tau_4 + a_{32}\tau_5 + a_{42}\tau_7 \\ \tau_6 + a_{33}\tau_5 + a_{43}\tau_7 \\ \tau_8 \end{pmatrix} \tag{7.6}$$

These equations provide the relationship between the torques required to achieve the desired motion, but do not give a unique solution. There is still some freedom in how the torques can be distributed.

As a first cut the two shoulder motor torques were set equal. Simulation results using the control law (7.1) are shown on the next page. The graphs show the error between the desired and actual position of the left shoulder and elbow motors (tracking error). The tracking errors are less than 0.03 degrees. Similar results were obtained for the right arm and momentum wheel. These results demonstrate that, under ideal conditions the control law works well.

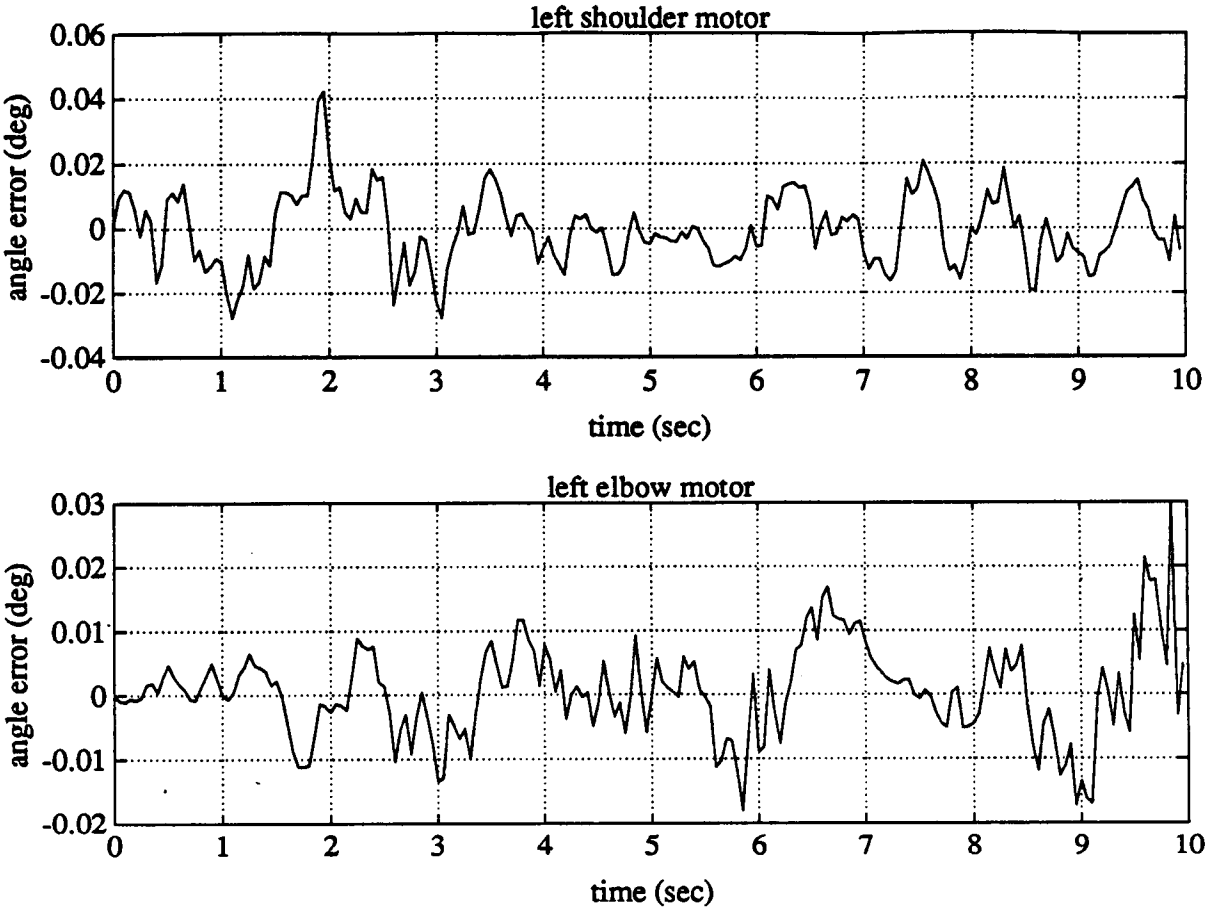


Figure 7.1: Tracking error using the control law

7.3 Adaptation Law

The control law and adaptation laws are closely related. The first adaptation law simulated is also one proposed by Slotine and Li. The adaptation law is:

$$\dot{\mathbf{a}} = -\Gamma^{-1}\mathbf{Y}^T(\mathbf{q}, \dot{\mathbf{q}}, \dot{\mathbf{q}}_r, \ddot{\mathbf{q}}_r)\dot{\tilde{\mathbf{q}}}_r \tag{7.7}$$

This law has some strong and weak points. Its main advantage is that it is very simple to implement using the control law (7.1). It uses the tracking error which is readily available and therefore adds to the simplicity of the algorithm. At the same time tracking error presents a problem in identification because it is small when good tracking is achieved, regardless of the parameter estimation. As a result when the desired trajectory is accurately tracked the parameter estimates will not change even when they are off the true values. In order to obtain a good parameter estimation a rough "sufficiently exciting" trajectory is desired. This presents a conflict with good tracking which requires smooth trajectories. Because of this conflict and the importance of good parameter identification a two-phase approach appears desireable. In the first phase the robot will "shake" itself about to determine its inertial parameters. During the second phase the robot will perform the desired task using a smooth trajectory.

This adaptation law with the control law (7.1) was simulated. Various trajectories and control parameters were tried. Two cases are presented: the first when adapting to one parameter (total mass) and the second when adapting to eight parameters (the minimum number when adding a mass to the base). The trajectory used for adaptation was generated using filtered random torques for the actuators. This generated a trajectory which was "exciting" and therefore good for identification.

As can be seen the scheme worked well when adapting to one parameter, but failed when adapting to eight parameters. In both cases the tracking error was small (less than 2.5 degrees). But, in the eight parameter case the parameters did not adapt well. The following figure shows the mass estimates to be off by more than an order of magnitude. The motors saturated when large adaptation gain values were used to try to improve the performance of the adaptation law. Once the motors saturate little, if any improvement is achievable.

More effort is required to improve this adaptation scheme. But, considering the past results there appears little hope that the scheme will yield promising results. An alternate method is presently being investigated. This new method will require either more hardware or a more complicated adaptation law.

7.4 Adaptation Law II

A new approach for adaptation, as suggested by Li and Slotine [25] is being considered. The method uses prediction error in addition to tracking error to drive the adaptation process. As a result twice as much information is used during the identification phase. The method has some very desireable properties. The approach has exponential convergence for persistently exciting trajectories and guarantees bounded parameter estimates. The

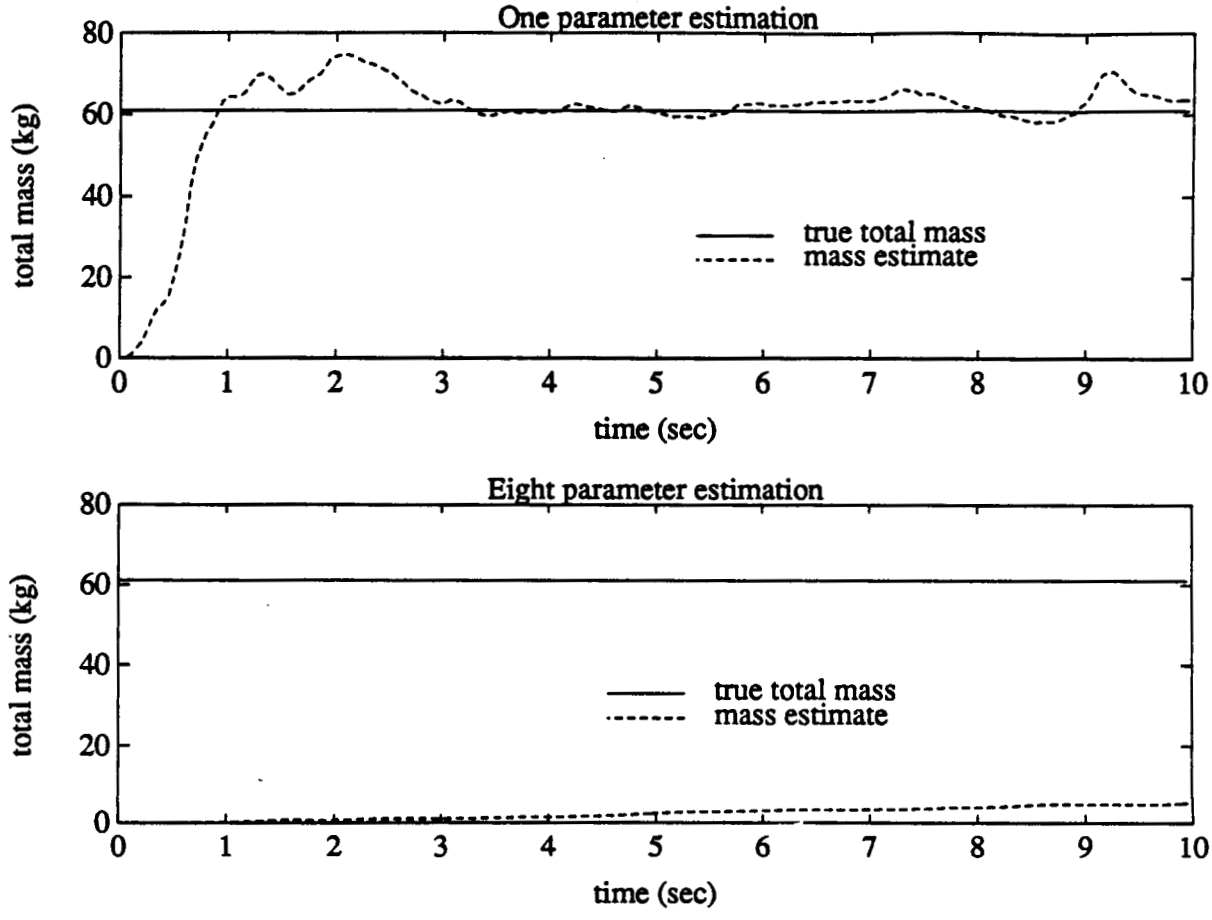


Figure 7.2: Adaptation law parameter estimates

method is based on the standard least-squares and exponential forgetting least-squares methods. But, as in the case of the control law, modifications have to be made.

There are two approaches which are being considered for modifying the adaptation law. The first is to use force sensors at the robot arm tips and to treat the system constraints only through these forces. The second approach uses the constrained system equations directly in the identification scheme. This latter method does not require force sensors, but the unknown parameters appear as non-linear terms in the control and identification equations. This will slow down the rate of convergence and may even prevent parameter convergence. The necessary modifications have been made to the algorithm for simulation.

7.5 Future Work

The simulation studies on the adaptation laws must be completed. These studies will determine the next step of research since it may be necessary to add force sensors to the arm tips. Upon completion of the simulation studies the real-time implementation aspects of the chosen algorithm will be addressed. Ongoing work in the hardware development will continue.

Bibliography

[1] T. E. Alberts and D. I. Soloway. Force control of a multi-arm robot system. In *Proceedings of the International Conference on Robotics and Automation*, pages 1490 – 1496, Philadelphia, PA, April 1988. IEEE.

[2] Harold L. Alexander. *Experiments in Control of Satellite Manipulators.* PhD thesis, Stanford University, Department of Electrical Engineering, Stanford, CA 94305, December 1987.

[3] Alford and Belyeu. Coordinated control of two robot arms. In *Proceedings of the International Conference on Robotics and Automation*, Atlanta, GA, 1984. IEEE.

[4] David S. Bayard and John T. Wen. Simple adaptive control laws for robotic manipulators. In *Proceedings of the Fifth Yale Workshop on the Applications of Adaptive Systems Theory*, pages 244–251, 1987.

[5] Robert H. Cannon, Jr., Harold Alexander, Marc Ullman, and Ross Koningstein. NASA Semi-Annual Report on Control of Free-Flying Space Robot Manipulator Systems. Semi-Annual Report 4, Stanford University Aerospace Robotics Laboratory, Stanford, CA 94305, February 1987.

[6] Robert H. Cannon, Jr., Marc Ullman, Ross Koningstein, Stan Schneider, Warren Jasper, and Roberto Zanutta. NASA Sixth Semi-Annual Report on Control of Free-Flying Space Robot Manipulator Systems. Semi-Annual Report 6, Stanford University Aerospace Robotics Laboratory, Stanford, CA 94305, February 1988.

[7] Robert H. Cannon, Jr., Marc Ullman, Ross Koningstein, Stan Schneider, Warren Jasper, and Roberto Zanutta. NASA Semi-Annual Report on Control of Free-Flying Space Robot Manipulator Systems. Semi-Annual Report 5, Stanford University Aerospace Robotics Laboratory, Stanford, CA 94305, August 1987.

[8] Cleve Moler, John Little, Steve Bangert, and Steve Kleiman. *PRO-MATLAB User's Guide for Sun Workstations.* The MathWorks, Inc., Sherborn, MA, August 1987. Version 3.2-SUN.

[9] John J. Craig. *Introduction to Robotics Mechanics and Control.* Addison-Wesley, Reading, MA, 1986.

[10] Tamar Flash and Neville Hogan. The coordination of arm movements: An experimentally confirmed mathematical model. A. I. Memo 786, MIT Artificial Intelligence Laboratory, November 1984.

[11] Gene F. Franklin and J. David Powell. *Digital Control of Dynamic Systems.* Addison-Wesley, Reading, MA, 1980.

[12] S. Fujii and S. Kurono. Co-ordinated computer control of a pair of manipulators. *Industrial Robot*, pages 155–161, December 1975.

[13] S. Hayati. Hybrid position/force control of multi-arm cooperating robots. In *Proceedings of the International Conference on Robotics and Automation*, pages 82 – 89, San Francisco, CA, April 1986. IEEE.

[14] Vincent Hayward and Richard P. Paul. Robot manipulator control under unix rccl: A robot control "c" library. *International Journal of Robotics Research*, 5(4):94–111, Winter 1986.

[15] G. Hirzinger and J Dietrich. Multisensory robots and sensor-based path generation. In *Proceedings of the International Conference on Robotics and Automation*, pages 1992 – 2001, San Francisco, CA, April 1986. IEEE.

[16] N. Hogan. Impedance control: An approach to manipulation, parts i, ii and iii. *Trans of the ASME, Journal of Dynamic Systems, Measurement, and Control*, 107:1–24, March 1985.

[17] T. Ishida. Force control in coordination of two arms. In *Proceedings of the 5th International Joint.Conference on Artificial Intelligence*, pages 717–722, Aug 1977.

[18] Thomas R. Kane and David A. Levinson. *Dynamics: Theory and Application.* McGraw-Hill Series in Mechanical Engineering. McGraw-Hill, New York, NY, 1985.

[19] Thomas R. Kane, Peter W. Likins, and David A. Levinson. *Spacecraft Dynamics.* McGraw-Hill, New York, NY, 1983.

[20] O. Khatib. Object manipulation in a multi-effector robot system. In *ISRR*, Santa Cruz, CA, 1987.

[21] Oussama Khatib. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Obstacles.* PhD thesis, (E)cole Nationale Supérieure de l'Aéronautique et de l'Éspace (ENSAE), Toulouse, France, 1980.

[22] Oussama Khatib. A unified approach to motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1), February 1987.

[23] J. C. Latombe, C. Laugier, J. M. Lefebvre, E. Mazer, and J. F. Miribel. The lm robot programming system. In H. Hanafusa and H. Inoue, editors, *Second International Symposium on Robotics Research*, chapter 7, pages 377–391. MIT Press, Cambridge, MA, 1985.

[24] M. A. Lavin and L. I. Lieberman. Aml/v: An industrial machine vision programming system. *International Journal of Robotics Research*, 1(3):42–56, Fall 1982.

[25] Weiping Li and Jean-Jacques E. Slotine. Parameter estimation strategies for robotic applications. In *A.S.M.E. Annual winter Meeting - Modeling and Control of Robotic Manipulators and Manufacturing Processes*, pages 213–218, Boston, MA, 1987.

[26] M. T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-11(6):418–432, June 1981.

[27] Y. Nakamura, K. Nagai, and T. Yoshikawa. Mechanics of coordinative manipulation by multiple robotic mechanisms. In *Proceedings of the International Conference on Robotics and Automation*, pages 991 - 998, Raleigh, NC, April 1987. IEEE.

[28] L. Pfeffer. Cooperation via coordinated endpoint impedance control. Informal technical discussions.

[29] Mark H. Raibert and John J. Craig. Hybrid position/force control of manipulators. *Transactions of the ASME*, 102(6):126–133, June 1981.

[30] G. Rodriguez and K. Kreutz. Recursive mass matrix factorization and inversion. Technical report, NASA Jet Propulsion Laboratory, Pasadena, CA, March 1988. JPL Publication 88–11.

[31] Dan E. Rosenthal. Order N Formulation for Equations of Motion of Multibody Systems. In G. Man and R. Laskin, editors, *Proceedings of the Workshop on Multibody Simulation*, pages 1122–1150, Pasadena, CA, April 1988. NASA Jet Propulsion Laboratory. JPL D-5190, Volume III.

[32] Michael Sherman and Dan Rosenthal. *SDEXACT User's Manual.* Symbolic Dynamics, Inc., Mountain View, CA, February 1988.

[33] B. E. Shimano, C. C. Geschke, and C. H. Spaulding III. Val-ii: A robot programming language and control system. In M. Brady and R. Paul, editors, *First International Symposium on Robotics Research*, chapter 10, pages 918–940. MIT Press, Cambridge, MA, 1984.

[34] Jean-Jacques E. Slotine and Weiping Li. On the adaptive control of robot manipulators. In *Proceedings of the ASME Winter Annual Meeting*, pages 51–56, Anaheim, CA, 1986.

[35] Software Components Group, Inc., 4655 Old Ironsides Drive, Santa Clara, CA 95054. *pSOS - 68K Real-Time, Multi-processing Operating System Kernel User's Manual*, 4.1 edition, 12 1986.

[36] D. I. Soloway and T. E. Alberts. Comparison of joint space versus task force load distribution optimization for a multi-arm manipulator system. In *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, February 1989. NASA.

[37] T. J. Tarn, A. K. Bejczy, , and X. Yun. Design of dynamic control of two cooperating robot arms: Closed chain formulation. In *Proceedings of the International Conference on Robotics and Automation*, pages 7 - 13, Raleigh, NC, April 1987. IEEE.

[38] R. H. Taylor, P. D. Summers, and J. M. Meyer. Aml: A manufacturing language. *International Journal of Robotics Research*, 1(3):19–41, Fall 1982.

[39] M. Uchiyama, N. Iwasawa, and K. Hakomori. Hybrid position/force control for coordination of a two-arm robot. In *Proceedings of the International Conference on Robotics and Automation*, pages 1242 - 1247, Raleigh, NC, April 1987. IEEE.

[40] E. Welzl. Constructing the Visibility Graph for n line segments in $O(n^2)$. *Information Processing Letters*, 20(4):167–171, May 1985.

[41] D. E. Whitney and J. L. Nevins. What is the remote centre compliance (rcc) and what can it do? *Robot Sensors*, 2:3–15, 1986.

[42] B. Yin. Using vision data in an object-level robot language—rapt. *International Journal of Robotics Research*, 6(1):43–58, Spring 1987.

[43] Y. F. Zheng, J. Y. S. Luh, and P. F. Jia. A real-time distributed computer system for coordinated motion control of two industrial robots. In *Proceedings of the International Conference on Robotics and Automation*, pages 1236 - 1241, Raleigh, NC, April 1987. IEEE.