Design and Testing of a Stewart Platform Augmented Manipulator for Space Applications

by

Terrence W. Fong

B.S., Massachusetts Institute of Technology (1988)

Submitted in Partial Fulfillment of
the Requirements for the Degree of

**Master of Science in
Aeronautics and Astronautics**

at the

**Massachusetts Institute of Technology
June 1990**

Signature of Author ______________________________
Department of Aeronautics and Astronautics
May 11, 1990

Certified by ______________________________
Professor David L. Akin
Thesis Supervisor
Department of Aeronautics and Astronautics

Accepted by ______________________________
Professor Harold Y. Wachman
Chairman, Departmental Graduate Committee
Department of Aeronautics and Astronautics

Design and Testing of a
Stewart Platform Augmented Manipulator
for Space Applications

by

Terrence W. Fong

Submitted to the Department of Aeronautics and Astronautics
on May 11, 1990 in partial fulfillment of the requirements for
the degree of Master of Science in Aeronautics and Astronautics

Abstract

An innovative nine degree-of-freedom robotic manipulator intended for neutral buoyancy space simulation research has been developed. The manipulator design specifications were driven by two primary goals; to duplicate and surpass the operational characteristics of the Shuttle Remote Manipulator System (SRMS) and to serve as a robust, neutral-buoyancy positioning system. The ability to provide fine tip positioning as well as large force production capability was realized by decoupling the system design into a three degree-of-freedom, revolute joint, manipulator arm augmented by a six degree-of-freedom, parallel-link, micromanipulator. Hybrid pneumatic/hydraulic actuators were developed to overcome neutral buoyancy constraints and to satisfy safety considerations. Finally, a variety of joint-based control schemes and Cartesian trajectory planning methods were implemented and studied.

This research was funded by the NASA Office of Aeronautics and Space Technology (NASA Grant NAGW-21).

Thesis Supervisor: Professor David L. Akin
Rockwell Assistant Professor of Aeronautics and Astronautics

Acknowledgements

Two years ago as a fledgeling graduate student, I approached Dave and asked "What would be a really neat thing to build?" He replied, "Ummm, well, I've got an idea for a pneumatic positioning arm. It would be like the big arm down at Marshall [Space Flight Center] but much stronger. And I figure that we can make it strong enough to use in *one-g*, so we can take it to freshman open houses and tap people on the shoulder with it."

At the time I had no idea what to call the arm. A long string of attempts at naming ensued with such dubious acronyms as *Big Large Arm Hopefully* (BLAH) and *Pneumatically Actuated Robotic Manipulator Equipped with a Synergetically Active Network* (PARMESAN). After many months, and uncountable groans from everyone in SSL, I finally decided the "big arm" would be called the *Stewart Platform Augmented Manipulator,* and thus, *SPAM* was born. Where the inspiration for *SPAM* came from, I am uncertain. Of course there is the processed meat product of the same name to consider. But I doubt that. There is alternatively my dear friend Cindy Shen, who picked up the nickname "SPAM" when she was a freshman. But I have my doubts about this also. Lastly, there is the skit by Monty Python ("Spam, spam, spam...") involving a restaurant that only serves SPAM. This MAY be where "SPAM" originated, but then again, one can never be sure...

There are a great number of people that I feel deserve my expressed appreciation and gratitude. First and foremost, I would like to thank Professor David Akin for his encouragement, support, and timely advice over the past few years. I am convinced that the prospect of working with Dave was the primary factor which drove me to pursue another degree at MIT. All that I have learned and gained during the past two years, I feel I owe to him. It is my sincere hope that we will be able to work together again in future days.

Second, I am indebted to all the undergraduates who worked on SPAM. Paul "Jello" Duncan, deserves endless acclamations since he toiled long hours building most of the hardware and being the "SPAM Diver". I could not have asked for a more capable "right-hand man". Cecelia "Too Tall" Park should be given an award for the amazing amount of foam and fiberglassing she had to endure under my dubious guidance. I cannot thank her enough for her unwavering commitment and dedication. Paul "Moonbeam" Stach will hopefully realize his ambition of "going to the moon on a craft of his own design", but until then, I hope he will take pride in the most excellent parts of SPAM (especially the "Stach Wrench") that he built. It was great fun and a tremendous pleasure to have worked with a fellow Illini! Additionally, I owe thanks (and probably several apologies for waking her) to Beth "Starlight" Kader, the "Bonus UROPer". I had no idea that hiring "Moonbeam" was a package deal, but I appreciate all of Beth's help at Pool Tests. Finally, gratitude is also due to Wisdom *Franchot* Coleman, Ronke Olabisi and Jen Tran.

To Professor Harold "Sandy" Alexander, I extend my heartfelt appreciation for his constructive suggestions, weekend shop supervision, and for introducing me to the Advanced Missions Group at NASA-Ames. To Ping Lee, I apologize profusely for the stream of inundating purchase orders and receipts that I constantly poured on her desk. Acknowledgement must additionally be given to Al Shaw, Earle Wassmouth, and Don Weiner for having the courtesy to deal with myself and everyone associated with SPAM.

To the community that is the "SSL family", my thanks go: to Rich Patten for teaching me everything about pneumatics; to Karl "Killer" Kowalski for always being amusing and wanting a good argument; to Jud "Cowardly Lion" Hedgecock for his friendship since Unified; to Ender St. John-Olcayto "Smythe" for being a great roommate, friend and confidant; to Ella "Mrs. Smith" Atkins for just being *Ella* (even though she decided my name should be "Fong Unit") and for always telling me *"that's nice, dear"*; to Rob Sanner-*Blah* for his invaluable advice in computer science and fabulous cookies; to Sayan "Interview-Man" Chakraborty for being an active member in the "Sleep 'Til Noon or Preferably Later Club" (may he always have a "birdhouse in his soul"); to Sam "Zorrrrrrrrtech" Druker for lending me Young M.C.'s Stone Cold Rhymin' and for sharing my disgust of Microsoft products; to Kurt "Smut-Man" Eberly for always amusing me with his "Hans and Franz" imitation; to Matt "Mattress" Machlis for throwing a frisbee (and for splattering me all over the slopes at Cannon Mt.); and to Russ "Fluffy" Howard for having the reference books on anything and everything I needed.

Finally, my thanks to my family and friends who have made the past two years the best years of my life. To my parents for always giving me the encouragement and support to do whatever I wanted to do my entire life, I will always be deeply indebted. I only hope that they know how much their love has meant to me. To my brother Tim, and sister Tammy, thanks for being silly and asking amusing questions. To Brian, Cindy, Ellen, Eric, Mini, Ryo, and Tex, thanks for everything. To Eriko, for being my friend, fellow "strobophobe" and for always amusing me. Most of all, thanks to Jessica for being the most wonderfully caring person in the world, for talking with me about anything and everything, and for giving me companionship and love.

This thesis is dedicated to my parents and Jessica.

"I didn't expect a kind of Spanish Inquisition."

"NOBODY EXPECTS THE SPANISH INQUISITION!!! Our chief weapon is surprise . . . surprise and fear . . . fear and surprise . . . our *two* weapons are fear and surprise . . . and ruthless efficiency. Our *three* weapons are fear and surprise and ruthless efficiency and an almost fanatical devotion to the Pope . . . Our *four* . . . no . . . *amongst* our weapons are such elements as fear, surprise . . . (arrrrrrrrrgh) I'll come in again . . ."

– Monty Python's Flying Circus

Table of Contents

Table of Figures

Chapter 1
Introduction

1.1 SSL Research Background

During the past decade, the MIT Space Systems Laboratory (SSL) has been engaged in the study of orbital productivity. Specifically, this research has been directed towards qualifying and quantifying methods of increasing the productivity of activities in space. In 1982, the *Automation, Robotics, and Machine Intelligence Systems* (ARAMIS) study was conducted for NASA to determine the feasibility of introducing man-machine systems and methods to space operations. The results of this study theorized that significant cost savings and increased productivity of specific activities could be achieved by augmenting humans with automation and machine intelligence (Miller *et al.*, 1983).

In the years following ARAMIS, the SSL has focused its efforts in the areas of teleoperation and expert systems. Recent activities have included research into the development of teleoperators for structural assembly, multiple sensor/control loop fusion techniques, autonomous docking and proximity operations, worksite integration of multiple humans and robots, and intelligent control systems. The vast majority of this research has been performed in neutral buoyancy simulations. A neutrally buoyant research environment was selected since it offered the capability to perform full-scale testing of man/machine operations. Furthermore, unlike other environments, neutral buoyancy does not restrict translational or rotational degrees of freedom, and thus offers a high degree of space simulation fidelity.

SSL's neutral buoyancy research has been conducted at two primary research sites, the MIT Alumni Swimming Pool and the NASA / Marshall Space Flight Center Neutral Buoyancy Simulator (MSFC-NBS). The MIT Alumni Swimming Pool, measuring 40 by 75 feet with a maximum depth of 13 feet, offers limited space for large scale activities. Research in this facility has been focused on localized activities such as assembly of structural nodes, proximity operations, and anthropometrics. The MSFC-NBS, on the other hand, is a 70 foot diameter by 40 foot deep tank, which allows for large space operation testing. Included at the MSFC-NBS are high-fidelity mockups of the STS payload bay and the Hubble Space Telescope, pressure suits and support facilities, and a Shuttle Remote Manipulator System (SRMS) simulator.

1.2 The SRMS Simulator

The SRMS simulator at the MSFC-NBS is a full scale kinematic simulation of the SRMS. To understand its capabilities and deficiencies, it is first necessary to understand the mechanism it mimics, the Shuttle Remote Manipulator System (SRMS).

1.2.1 Summary of the SRMS

The SRMS was conceived as part of a cooperative development effort between NASA and the National Research Council of Canada. Built by Spar Aerospace (CANADA), the SRMS is intended to provide a controlled means of cargo deployment and retrieval during on-orbit STS missions. It is essentially a six degree-of-freedom manipulator system comprised of a large, three degree-of-freedom anthropomorphic manipulator augmented by a three degree-of-freedom end-effector. The SRMS configuration is shown in diagrammatic form in Figure 1.2.1.

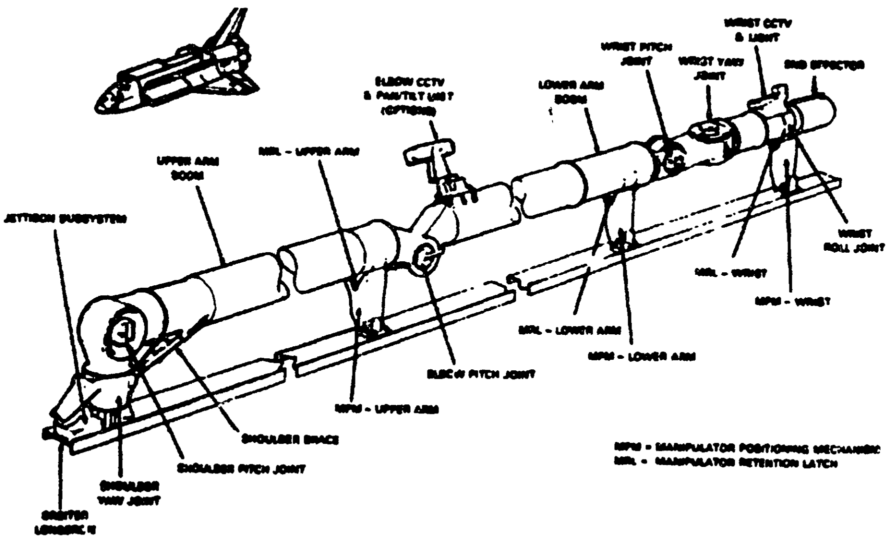


Figure 1.2.1: SRMS Configuration
(Ussher, T.H., and Doetsch, K. H., *An Overview of the Shuttle Remote Manipulator System*, NASA N85-16964, 1985)

The SRMS delivered to NASA, and currently used on the STS, is 50 ft. in length and weighs 950 lb. It is designed to maneuver a nominal payload of 32,000 lb. and to retrieve a maximum payload weight of 65,000 lb. The operational characteristics of the SRMS are summarized below in Table 1.1 (Ussher, T. H. 1985).

Table 1.1: SRMS Operational Parameters

Degrees-of-freedom	6
Max. tip extension	50 ft (15.24 m)
Min. tip force (straight-arm)	15 lb. (81.96 N)
Arm stiffness (straight-arm)	8.4 lb./in. (1,471 N-m)
Joint rate hold accuracy	±1 deg./sec.
Tip position hold accuracy	±2 in. (±0.025 m)
Max. tip translation rate	2 ft./sec. (0.61 m/sec)

The SRMS is controllable in any one of four standard, switch-selectable modes. These modes are (1) Manual Augmented, in essence open-loop tip control with translational and rotational hand controller (THC & RHC) Cartesian rate commands resolved into joint rates; (2) Single Joint, which controls an individual joint rate while freezing the other five joints; (3) Direct Drive, whereby fixed rate commands are sent to switch-selected joints; and (4) Automatic Mode, where preprogrammed end-effector position trajectories are used to generate joint rate command sequences. All of these modes except for Direct Drive utilize the General Purpose Computer (GPC) onboard the STS for computation of joint rate commands.

1.2.2 SRMS Simulator Characteristics

The SRMS simulator was intended to provide the MSFC-NBS with the capability of simulating STS tasks involving the SRMS. At the time of the simulator's conception, two hardware simulations of the SRMS had been constructed elsewhere, at the Real-Time Simulation Facility (SIMFAC) in Canada and at NASA's Johnson Space Center (JSC). The SIMFAC simulator was constructed with an air-bearing system to simulate zero gravity. Due to space limitations and logistics, this simulator was useful only for verifying performance of SRMS components. Conversely the JSC simulator, used for training STS crews, was designed to operate in one-g. Although this simulator provided kinematic similarity, its dynamic characteristics were extremely different since it operated in full gravity instead of zero-g. To address the shortcomings of both of these simulators, it was proposed to construct a SRMS simulator at MSFC. Operating a SRMS simulator in the

MSFC-NBS would theoretically be able to offer kinematic similarity as well as provide a high-fidelity simulation of on-orbit conditions.

Although the simulator eventually constructed at the MSFC-NBS is kinematically identical to the SRMS, it differs both from a physical and an operational standpoint. The simulator hardware (e.g. joints, boom structures, etc.) were assembled from surplus equipment at MSFC, waterproofed for operation in the NBS. Additionally, the four SRMS control modes were not implemented in the simulator, the only method being a direct mapping of translational and rotational hand controller offsets to joint rates. Moreover, no effort was made to compensate for the characteristics of a neutrally buoyant environment (e.g. water drag and currents).

As a result of these factors, the SRMS simulator at MSFC-NBS exhibits three prominent differences from the actual SRMS. First, because of the water environment, the system is much less accurate and moves quite sluggishly. This makes precise positioning and deployment/retrieval of payloads significantly harder. Secondly, since the joint actuators were not originally designed for underwater use, the force available in the NBS is much lower. Tests have shown that the maximum tip force that the SRMS simulator can exert is on the order of one pound. This exacerbates the positioning problem because payloads with suboptimal neutral buoyancy cannot be moved. Finally, since the arm cannot be controlled in a closed-loop sense, it is extremely difficult to position the tip at desired positions and orientations. Taken collectively, these shortcomings mean that the SRMS simulator is not a very realistic simulation of the capabilities of the SRMS. Although the SRMS simulator very closely matches the kinematics of the SRMS, in terms of reachable and dexterous workspace, it is severely deficient in its duplication of positioning and performance characteristics.

1.3 Motivation for Research

The current direction of SSL research indicates that future studies will focus to a greater and greater extent on enhancing man/machine system productivity. This may involve investigation of coordinating humans with multiple autonomous and teleoperated robots in performing activities in space. Some potential tasks for study include satellite servicing, structural assembly and maintenance. Additional research may entail quantifying worksite positioning requirements and finding means of reducing the expenditure of consumable supplies. To effect much of this research, however, a positioning system of some form will be required. Since SSL research is conducted primarily in neutral buoyancy, a logical choice to fulfill positioning needs would be a system similiar to the

SRMS simulator at the MSFC-NBS. As discussed in the previous section, however, this simulator has shortcomings which tend to obviate its usefulness.

The problem at hand, therefore, is to develop a neutral buoyancy positioning system that corrects the deficiencies of the MSFC-NBS SRMS simulator. The fundamental requirements for such a system are threefold. First, it should be capable of duplicating the SRMS workspace, in terms of reach and volume. Secondly, it should be designed for robust operation in a neutral buoyancy environment, able to generate large forces in the presence of water drag and currents. Finally, it should be operable by modern control methods and provide an efficient man/machine interface.

In the following chapters an innovative concept, the Stewart Platform Augmented Manipulator (SPAM), is described which fulfills these requirements. The system is in essence a three degree-of-freedom anthropomorphic manipulator arm with revolute joints, augmented with a six degree-of-freedom parallel-link platform micro-manipulator. The SPAM concept, as will be shown, is capable of simulating the SRMS with a high degree of fidelity in neutral buoyancy. Moreover, the system has the potential for higher performance and flexibility than even the actual SRMS can provide.

In Chapter 2 the SPAM concept is presented and a description of the system design specifications is given. Chapter 3 details the development of the anthropomorphic manipulator including the design of an underwater hybrid pneumatic/hydraulic rotary actuator, the implementation and testing of digital rate controllers, and the testing of revolute joints. Chapter 4 describes the design of a parallel-link platform micro-manipulator (often referred to as a *Stewart Platform*), the development of linear pneumatic/hydraulic actuators, and parallel-link control issues. Chapter 5 presents planned systems integration and operational methods for the complete SPAM system, including a distributed processing system architecture. Finally, Chapter 6 summarizes the findings of this thesis and suggests avenues for future research.

Chapter 2
System Design

2.1 Design Goals

The development of a manipulator-type positioning system for neutral buoyancy was driven by two goals. First, the system had to be kinematically similar to the SRMS. This indicated that the manipulator design should reflect dexterous and reachable workspaces that closely matched those of the SRMS. Secondly, the system had to operate in a robust manner in an underwater environment. This implied that the joint actuators be capable of producing large enough forces to compensate for water drag, and that link rigidity or joint controllability be sufficient to reject disturbing currents.

The design goals for the positioning system were initially based on the SRMS operational characteristics. During the early design phase, however, several modifications to these objectives were made. First, since the primary task is fine positioning, it was specified that the system be capable of fine tip control in six degrees-of-freedom. While the SRMS has a total of six degrees-of-freedom at the tip, the three translational degrees are provided by the large manipulator arm section; making fine translation difficult. Additionally, since the objects intended for positioning are humans and telerobots (which are much less massive than STS payloads), duplication of maximum SRMS tip force was not necessary. Finally, overall arm stiffness was determined not to be a critical design constraint. Although link flexing would have an impact on precise tip positioning, the conclusion was that micro-manipulator actions combined with active control would sufficiently compensate for lack of rigidity.

The easiest means of achieving SRMS react and workspace similarity is simply to duplicate the SRMS hardware arrangement of joints and links. In short, a large, three revolute joint (i.e. shoulder pitch, shoulder yaw, and elbow pitch) anthropomorphic arm augmented with a three revolute joint (wrist roll, pitch, and yaw) end-effector should be constructed. In this arrangement, the large arm provides large reach capability while the end-effector provides fine tip angular position control. This system is readily accomplished in neutral buoyancy, as demonstrated by the MSFC-NBS manipulator. Unfortunately though, this approach has a significant drawback since it utilizes a serial-link end-effector at the large arm's tip.

Serial-link mechanisms, though able to provide high positioning accuracy and positioning ease, typically are incapable of producing large forces. Generally speaking, this is not a problem since other devices or mechanisms can fulfill requirements for force. In an underwater environment, however, force production capability is a matter of concern since large water forces, such as drag and currents, are present.

In the case of the MSFC-NBS SRMS simulator, the serial-link end-effector attached to the large arm does not generate a large amount of force. Consequently, the large arm is relied upon to produce tip forces when positioning objects. Since this requires extremely minute joint control, which is difficult to achieve with long linkages, fine positioning is extremely hard to perform. In essence, any positioning advantages offered by the end-effector are totally offset by the inability to produce strong forces. As the MSFC-NBS simulator demonstrates, a serial-link end-effector is not a viable solution for fine positioning in neutral buoyancy.

2.2 The Stewart Platform Augmented Manipulator

2.2.1 SPAM Concept Development

An elegant solution to the tip positioning problem is to replace the 3 jointed, serial-link wrist with a parallel-link mechanism. Parallel-link, or closed kinematic chain, devices are alternatives to traditional serial type mechanisms in which two or more linkages connect the base of the manipulator with the end-effector. The direct benefits of this arrangement are (Ismail, A., 1984):

- High strength and stiffness-to-weight ratios can be achieved since the links do not carry moment loads but act only in tension and compression.

- Positioning of the end-effector is performed by actuators acting in parallel, resulting in a total force and moment capability greater than each individual servomechanism.

- Moving only the end-effector in space rather than massive servomechanisms results in economy of power, excellent dynamic performance, and low manipulator inertia.

- High accuracy and precision is possible since actuator errors are not magnified by lengthy linkages.

Since all of these characteristics are advantageous for underwater mechanisms, it seems natural to implement a parallel-link mechanism in neutral buoyancy.

The design task, therefore, is to create a positioning system that combines a large manipulator arm with a six degree-of-freedom parallel-link micromanipulator. One concept that matches this description is the *Stewart Platform Augmented Manipulator (SPAM).* In this system, a three degree-of-freedom, anthropomorphic manipulator arm is augmented by a parallel-link mechanism called a *Stewart Platform.*

The Stewart Platform is a space truss comprised of six prismatic actuators, each mounted by a universal joint to the manipulator base and by a spherical joint to the top platform. Figure 2.2.1 depicts a typical Stewart Platform. This arrangement of actuators allows the platform to be placed in any position and orientation (i.e. with six degrees-of-freedom) within a certain volume of space.

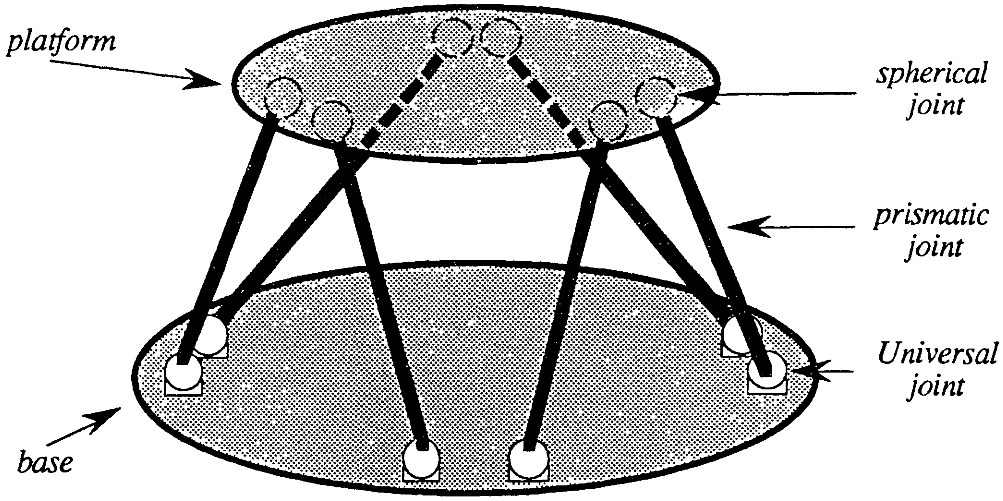


Figure 2.2.1: A Stewart Platform

In the SPAM concept, the large, anthropomorphic manipulator arm provides coarse positioning and large reach capability with three revolute joints. The Stewart Platform, which augments the manipulator, offers fine tip positioning in six degrees-of-freedom. Moreover, it has the characteristics (e.g. strength, stiffness, power, etc.) of parallel-link devices which are advantageous for underwater operation. The complete SPAM system is illustrated schematically in Figure 2.2.2.

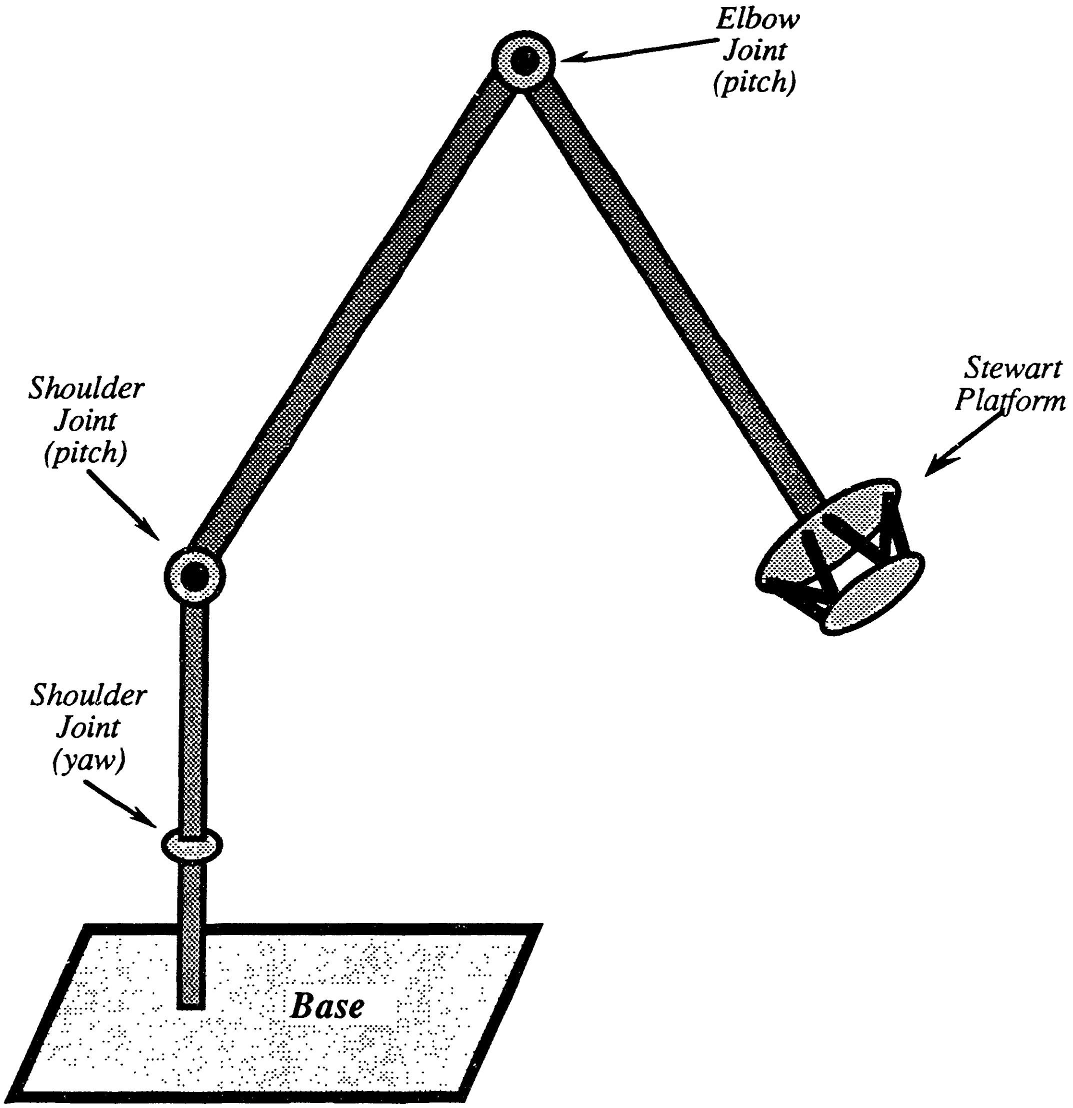


Figure 2.2.2: Stewart Platform Augmented Manipulator (SPAM)
(schematic)

Detailed design specifications for the two main components of SPAM, the manipulator arm and the Stewart Platform, are presented in the following two sections.

2.2.2 3-DOF Manipulator Design Specifications

The preliminary design specifications for the large arm component of SPAM were derived from the SRMS. Three revolute joints arranged anthropomorphically (i.e. shoulder yaw, shoulder pitch, and elbow pitch) are connected by straight link sections. As with the SRMS, the revolute joints should have a working range of 180 degrees. To complete the

SRMS kinematic simulation, therefore, simply requires a duplication of SRMS linkage lengths.

If the joints and linkages are modularly designed, however, mechanisms other than the SRMS may be simulated. As a result, linkage lengths were not specified as a fundamental design specification, but modularity was included in the baseline. Tip accuracy was assumed to be within one percent of the maximum manipulator translational reach. The SRMS 15 lb. straight arm tip force (corresponding to a total linkage length of 50 ft.) resulted in a maximum shoulder joint torque specification of 750 ft-lb.

A summary of the baseline design specifications for the manipulator is presented in Table 2.1.

Table 2.1: Manipulator Baseline Specifications

Degrees-of-freedom	3		
Number of joints & type	3	revolute	
Joint range of motion	180	degrees	
Joint torque capability	750	ft-lb.	(1,017 N-m)
Joint rate hold accuracy	±1	deg./sec.	
Tip position accuracy	>1	% max reach	
Tip position hold accuracy	±2	in.	(±0.051 m)
Max. tip translation rate	2	ft./sec.	(0.61 m/sec)

2.2.3 6-DOF Stewart Platform Specifications

The design specifications for the Stewart Platform component of SPAM were based on SRMS tip characteristics with consideration given for operation in an underwater environment. To match SRMS tip positioning, the Stewart Platform must have, at a minimum, ±1 inch translational position accuracy. The translational workspace requirements were derived from the assumption that the large arm can position the end-effector to within one percent of the maximum translational reach. For a 50 ft arm, this indicates that the Stewart Platform should be capable of ±6 inches movement in three translational axes.

The SRMS wrist provides only three degrees-of-freedom in the end-effector workspace. Since the Stewart Platform provides a total of six degrees-of-freedom, SPAM has three more degrees of tip freedom than the SRMS. To offer a large and flexible end-effector workspace for positioning, it was decided that ±45 degrees of rotational movement in 3 axes would prove to be sufficient. To overcome water forces and to provide 15 lbs of end-effector force (equivalent to SRMS minimum tip force), each actuator had to be capable of generating at least 15 lbs of force.

A summary of the baseline design specifications for the Stewart Platform is presented below in Table 2.2.

Table 2.2: Stewart Platform Baseline Specifications

Degrees-of-freedom	6		
Number of joints & type	6	prismatic	
Joint force capability	15	lbs.	(66.7 N)
Translational accuracy	±1	inch	(±0.025 m)
Translational range	±6	inches	(±0.15 m)
Rotational range	±45	degrees	

Chapter 3
Manipulator Arm

The task of constructing the manipulator arm component of SPAM was separated into several design and testing phases. In the first phase, a revolute joint was developed for underwater operation. This entailed actuator research and development, linkage design, fabrication, instrumentation, control systems design, and testing. For the second phase, a 2-link manipulator was assembled and operated in neutral-buoyancy. Control systems were implemented via computer and tests conducted to determine viability of various control schemes. The third and final phase involved the construction and operational testing of the full 3-link manipulator. At the present time, however, this phase has not been completed.

3.1 Manipulator Kinematics

In this section, the kinematics for 2-link and 3-link serial-link manipulators are presented. A serial-link manipulator is essentially a series of rigid bodies connected in a chain structure. Such an arrangement is often referred to as an *open kinematic chain* (Asada, 1986). Two mappings completely characterize serial-link manipulator geometry. The *forward kinematics* mapping relates the natural coordinates of a manipulator to reference coordinates. Similarly, the *inverse kinematics* mapping transforms reference coordinates to natural coordinates. These two mappings are often used with respect to the end-effector. Given a description of natural coordinates (e.g. joint angles), for example, the forward kinematics are used to compute the end-effector position and orientation in some reference frame (e.g. Cartesian coordinates). Conversely, given position and orientation in a reference frame, the inverse kinematics are used to calculate the set of joint angles which will place the end-effector in the specified location.

The natural coordinates for a serial-link manipulator having *n* degrees-of-freedom is a set of *n* joint variables (Craig, 1986). If the manipulator is comprised only of revolute joints, for example, the set of joint variables is the set of all the joint angles. This may be expressed as a vector in *joint space*:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \tag{3.1}$$

The reference coordinates most frequently used are Cartesian coordinates, with an orthonormal frame to describe position and Euler angle rotations to describe orientation. This may be expressed as a 6x1 vector in *Cartesian space* (using Roll-Pitch-Yaw Euler angles):

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{3.2}$$

where ϕ, θ, ψ are right-handed rotations about the x, y, z axes respectively. The forward kinematics mapping, F, is then:

$$\mathrm{F} : \theta \Rightarrow \mathbf{X} \tag{3.3}$$

Similarly, the inverse kinematics mapping, G, is:

$$\mathrm{G} : \mathbf{X} \Rightarrow \theta \tag{3.4}$$

For serial-link manipulators, frames are assigned to the manipulator base, to each joint in the chain, and to the end-effector (tip). The mapping of positions from frame A to frame B is then performed with a 4x4 homogeneous transform matrix, ${}^{A}_{B}\mathbf{T}$:

$$\begin{aligned} \begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} &= {}^{A}_{B}\mathbf{T} \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} {}^{A}_{B}R & {}^{A}P_{B\,ORG} \\ 0\ 0\ 0 & 1 \end{bmatrix} \end{aligned} \tag{3.5}$$

where: ${}^{A}P$ is a 3x1 Cartesian position vector in frame {A}

${}^{B}P$is a 3x1 Cartesian position vector in frame {B}

${}^{A}P_{B\,ORG}$ is the location of the origin of {B} in {A}

${}^{A}_{B}R$ is a 3x3 rotation matrix describing the angular orientation of {B} in {A}

The forward kinematics of the manipulator is, therefore, the transformation from the manipulator base frame {0} to the end-effector frame $\{T\}$, ${}^{0}_{T}\mathbf{T}$.

Determining manipulator inverse kinematics is problematic since the kinematic equations are highly non-linear and a general closed-form solution does not exist. However, several approaches to solving the inverse kinematics problem are described in the literature for serial-link manipulators (i.e. algebraic manipulation, geometric decomposition, Pieper's method, etc.) which may result in the desired closed-form solution. It should be noted that a universal, numerical solution does exist for 6 degree-of-freedom, serial-link manipulators composed of revolute and prismatic joints (Craig, 1986). From an inverse kinematics applications perspective, however, a numerical method is considered to be suboptimal since it necessitates a significantly greater amount of iterative computation than a closed-form method, and because such a technique is not guaranteed to discover all the solutions of the problem.

3.1.1. 2-Link Manipulator

The 2-link manipulator was designed to have two revolute joints, s ioulder pitch and elbow pitch. Since the two joint axes are parallel, the manipulator is an extremely simple planar arm as shown in Figure 3.1.1.

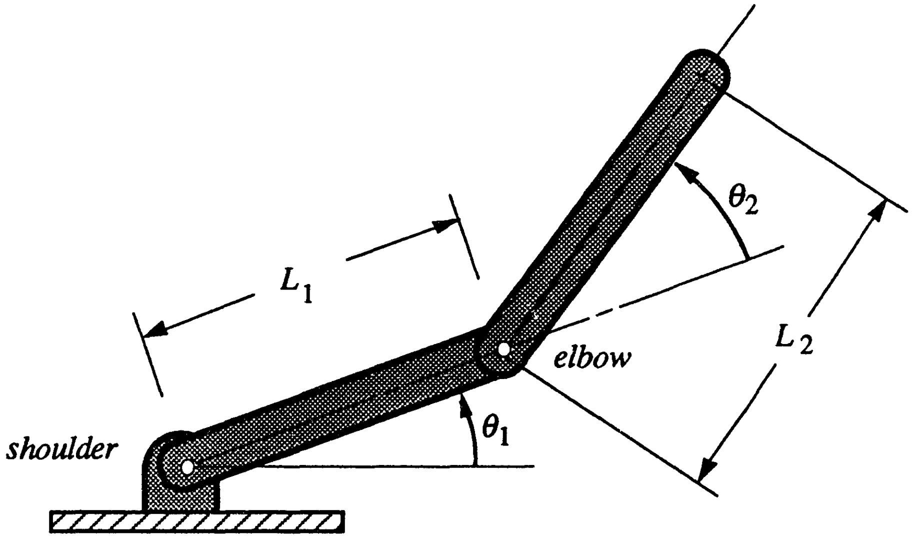


Figure 3.1.1: 2-link planar manipulator

The frames assigned to the manipulator joints follow the convention in (Craig, 1986). The reference frame {0} is affixed to the base. Frame {1} is attached to the first joint (shoulder) and aligns with {0} when the shoulder angle is zero. Frame {2} is attached to the second joint (elbow) and aligns with {1} when the elbow angle is zero. Frame {*T*}, the tip frame, is located at the end of the elbow link. The 2-link arm with frame assignments is shown in Figure 3.1.2

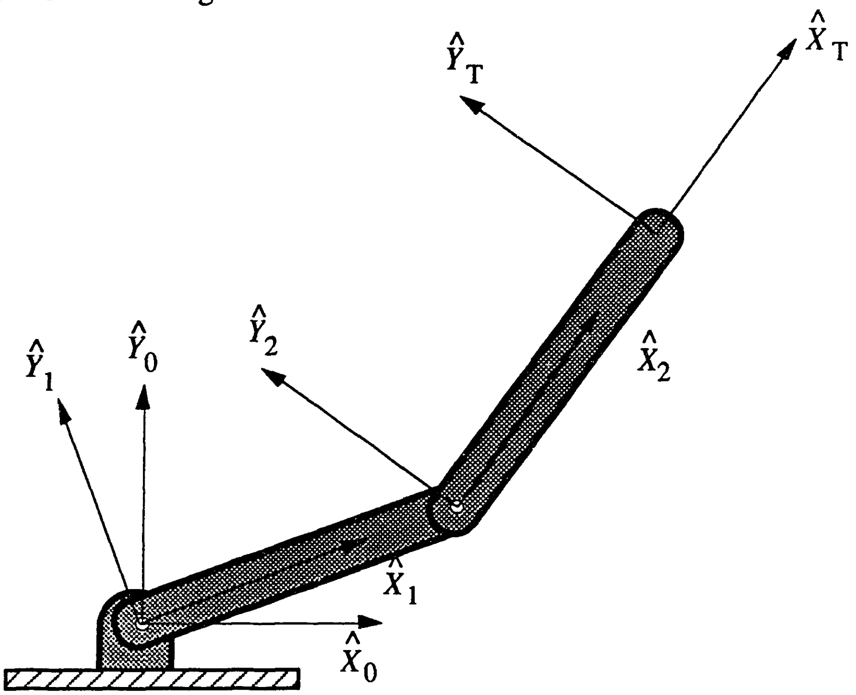


Figure 3.1.2: Frame assignments for 2-link planar arm

3.1.1.1. Forward Kinematics

The frame transformations for the 2-link planar arm are:

$$
{}^0_1T = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^1_2T = \begin{bmatrix} c_2 & -s_2 & 0 & L_1 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.6)
$$

$$
{}^2_T T = \begin{bmatrix} 1 & 0 & 0 & L_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

where c_1 is notational shorthand for $\cos(\theta_1)$.

Multiplying these matrices together, produces the desired forward kinematics transform:

$$ {}^{0}_{T}\mathrm{T} = \begin{bmatrix} c_{12} & -s_{12} & 0 & L_1c_1+L_2c_{12} \\ s_{12} & -c_{12} & 0 & L_1s_1+L_2s_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad (3.7a) $$

where $c_{12} = \cos(\theta_1 + \theta_2)$.

The kinematic equations relating the joint angles to the tip location, referenced to frame {0}, are found in column 4 of the above:

$$ \begin{aligned} x &= L_1c_1 + L_2c_{12} \\ y &= L_1s_1 + L_2s_{12} \end{aligned} \qquad (3.7b) $$

3.1.1.2. Inverse Kinematics

Following the algebraic solution method presented in (Craig, 1986) results in the desired inverse kinematics. The equations relating the tip position, referenced to frame {0}, to the joint angles are:

$$ \begin{aligned} \theta_1 &= \mathrm{Atan2}(y, x) - \mathrm{Atan2}(K_2, K_1) \\ \theta_2 &= \mathrm{Atan2}(s_2, c_2) \end{aligned} \qquad (3.8) $$

where:

$$ c_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2} \qquad (3.9) $$

$$ s_2 = \pm\sqrt{1 - (c_2)^2} \qquad (3.10) $$

$$ \begin{aligned} K_1 &= L_1+L_2c_2 \\ K_2 &= L_2s_2 \end{aligned} \qquad (3.11) $$

There are several items to note in the preceding. First, the two argument arc tangent function is used in (3.8) to determine the quadrant containing the resulting angle. Secondly, for a solution to exist, the the right-hand side of (3.9) must be in the range [-1, 1]. If this constraint is not met, then the specified tip position is outside the reachable workspace. Finally, the multiple solutions of (3.10) correspond to multiple geometric configurations of the arm that result in the same tip position.

3.1.2. 3-Link Manipulator

The 3-link manipulator was designed to have three revolute joints in an anthropomorphic configuration; shoulder yaw, shoulder pitch and elbow pitch. The shoulder yaw joint axis, located at the manipulator base, is normal to the shoulder joint pitch axis. Since the shoulder pitch joint and elbow joint axes are parallel, the shoulder pitch/elbow section is identical to the 2-link manipulator. The 3-link manipulator is shown in Figure 3.1.3.

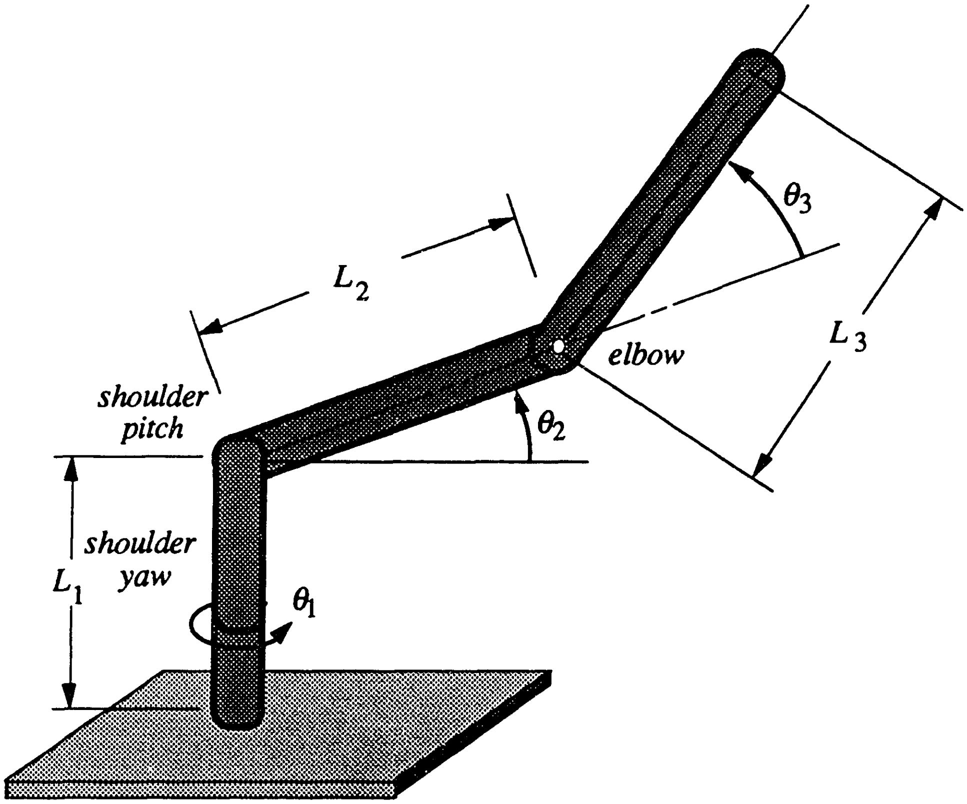


Figure 3.1.3: 3-link anthropomorphic manipulator

The reference frame, frame {0}, is affixed to the base. Frame {1} is attached to the shoulder yaw joint and aligns with the reference frame when the joint angle is zero. Frame {2} is attached to the shoulder pitch joint. Frame {3}, attached to the elbow joint, aligns with frame {2} when the elbow angle is zero. The tip frame, frame {4} is located at the end of the elbow link. The frame assignments for the 3-link manipulator are shown in Figure 3.1.4

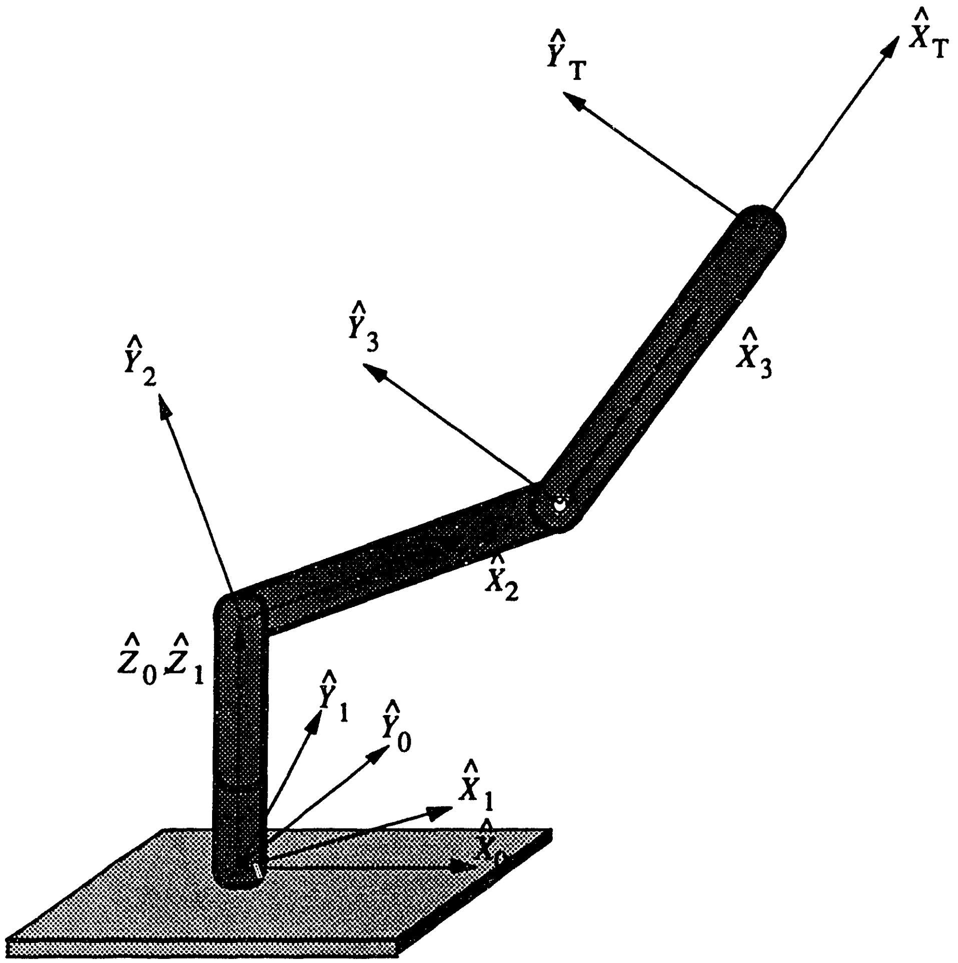


Figure 3.1.4: Frame assignments for 3-link arm

3.1.2.1. Forward Kinematics

The frame transformations for the 3-link anthropomorphic arm are:

$$
\begin{aligned}
{}^{0}_{1}\mathrm{T} &= \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^{1}_{2}\mathrm{T} &= \begin{bmatrix} s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ c_2 & -s_2 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^{2}_{3}\mathrm{T} &= \begin{bmatrix} c_3 & -s_3 & 0 & L_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^{3}_{T}\mathrm{T} &= \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\qquad (3.12)
$$

Multiplying these matrices together, produces the desired forward kinematics:

$$
{}^{0}_{T}\mathrm{T} = \begin{bmatrix} c_1s_{23} & c_1c_{23} & -s_1 & L_2c_1s_2 + L_3c_1s_{23} \\ s_1s_{23} & s_1s_{23} & c_1 & L_2s_1s_2 + L_3s_1s_{23} \\ c_{23} & -s_{23} & 0 & L_1 + L_2c_2 + L_3c_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.13a)
$$

The kinematic equations relating the joint angles to the tip location, referenced to frame $\{0\}$, are contained in column 4 of (3.13a):

$$
\begin{aligned}
x &= L_2c_1s_2 + L_3c_1s_{23} \\
y &= L_2s_1s_2 + L_3s_1s_{23} \\
z &= L_1 + L_2c_2 + L_3c_{23}
\end{aligned}
\qquad (3.13b)
$$

3.1.2.2. Inverse Kinematics

Since the 3-link anthropomorphic manipulator is essentially the 2-link planar arm augmented with a revolute base joint, the inverse kinematic mapping is easy to obtain. First, use the specified x, y tip position (in frame $\{0\}$) with the two argument arc-tangent to calculate the shoulder yaw angle. Next, rewrite the specified Cartesian tip position in cylindrical coordinates, accounting for the base to shoulder pitch joint separation distance.

Finally, apply the 2-link arm solution using the cylindrical coordinates. The complete inverse kinematic solution is given below:

$$\theta_1 = \mathrm{Atan2}(x, y) \tag{3.14}$$

$$\begin{aligned} R &= \sqrt{x^2 + y^2} \\ H &= z - L_1 \end{aligned} \tag{3.15}$$

$$\begin{aligned} \theta_2 &= \mathrm{Atan2}(R, H) - \mathrm{Atan2}(K_2, K_1) \\ \theta_3 &= \mathrm{Atan2}(s_3,\ c_3) \end{aligned} \tag{3.16}$$

where:

$$c_3 = \frac{R^2 + H^2 - L_2^2 - L_3^2}{2L_2L_3} \tag{3.17}$$

$$s_3 = \pm\sqrt{1 - (\cos\theta_3)^2} \tag{3.18}$$

$$\begin{aligned} K_1 &= L_2 + L_3c_3 \\ K_2 &= L_3s_3 \end{aligned} \tag{3.19}$$

Equations (3.14) to (3.19), of course, are subject to the same constraints on the 2-link solution given in section 3.1.1.2. As a result, only the shoulder yaw angle has a unique solution.

3.2. Manipulator Control Schemes

The control problem for serial-link manipulators is an inherently nonlinear one. Since a serial-link manipulator is a chain of rigid bodies, mass and inertia parameters are dependent on the manipulator configuration and are therefore time varying. Additionally, actuator non-linearities such as friction contribute to the difficulty of accurately modeling a manipulator system. Model based controllers, therefore, must compensate for changing manipulator characteristics by providing some form of adaptive or non-linear control. System linearization may be used to derive locally linear models if nonlinearities are not a significant factor about an operating point. This approach has limited usefulness, however, since it requires a changing linearized model as the manipulator moves from point to point.

For a manipulator operating in neutral buoyancy, modeling of the system is further complicated. Water forces (e.g. drag, damping and turbulence) are highly non-linear and strongly dependent on the velocity of moving components. As a result, it is even more difficult to derive a system model upon which to base a controller. Non-linear methods such as sliding-surface offer the potential to alleviate this problem since such techniques are able to produce extremely robust controllers.

The following sections describe several methods for controlling serial-link manipulators. Specifically, techniques for position and velocity control of the end-effector are detailed. Sections 3.2.1 to 3.2.4 present *joint-based controllers*, in which controller errors are represented in joint-space. Such controllers are fairly easy to implement and are found on a vast number of current commercial systems. The final section (3.2.5) describes a *Cartesian-based controller* which maps controller errors in Cartesian space rather than joint space.

It should be noted that in all of these methods the manipulator is controlled with simple, error-driven controllers. Furthermore, each joint of the manipulator is operated independently without regard to the other joints by a localized servo controller. A serious deficiency of this methodology is that no attempt is made to derive and utilize a system model. At first glance, such an approach would appear to be a poor choice since it is impossible to guarantee optimal, or even fair, performance of the arm in all configurations. However, since derivation of a model structure is problematic and because determination of model parameters (e.g. inertia moments, center of mass, etc.) is difficult, the benefits offered by model-based control may be few. Moreover, such controllers are likely to be computationally intensive, requiring fairly powerful computer power. On the other hand, the individual joint approach based on simple control laws requires few calculations so that low computational power will suffice.

3.2.1. Open-Loop Position

In this control scheme, the manipulator tip is controlled open-loop since there is no direct feedback of tip position. As a result, this method has limited appeal for direct tip positioning control. During operation of the manipulator, joint position commands are fed directly on a joint-by-joint basis to position servocontrollers, which receive joint position feedback. This is illustrated by block diagram in Figure 3.2.1. The resulting tip location may be determined by applying manipulator forward kinematics to the joint positions.

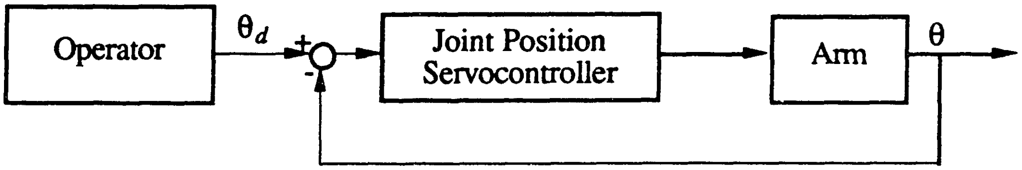


Figure 3.2.1: Open-Loop Position Control

For a human operator, this control scheme is feasible if the proper user interface is chosen. With an anthropometric manipulator, for example, one possible approach is to utilize similarity between a scaled model and the controlled arm. By controlling the tip of the scaled, *master* arm, inverse kinematics are implicitly performed and provide joint position commands for the *slave* manipulator. This technique has been used extensively for teleoperation tasks in the nuclear industry as well as for structural assembly research in the SSL.

Moreover, it is possible to "teach" a manipulator how to perform a specific task. This is accomplished by first having a human operator guiding (i.e. operating) the manipulator through a desired set of motions and sequentially storing sets of joint positions. Later, a sequencer plays back the series of joint sets, effectively commanding the manipulator to duplicate human actions. The primary application of *teaching by showing* is for programming a manipulator to perform repetitive tasks, reducing human operator workload.

3.2.2. Open-Loop Resolved Motion Rate

A more useful tip control scheme than open-loop position is referred to as *resolved rate* or *resolved motion rate* control and is attributed to D.E. Whitney. In resolved rate control, given desired manipulator tip velocities, joint rates are determined which will cause the manipulator tip to move at the desired velocities. One common approach is to utilize the inverse Jacobian mapping to *resolve* Cartesian velocities into joint rates:

$$\dot{\theta}_d = ({}^0\mathbf{J})^{-1}\,{}^0\dot{\mathbf{X}}_d \tag{3.20}$$

where the leading, superscripted zero indicates terms written with respect to the base frame.

However, since the Jacobian is dependent on manipulator configuration, it is possible for it to become singular. In this case, the inverse Jacobian does not exist, and a determination of joint rates via (3.20) is not possible. Furthermore, as the manipulator approaches a singular configuration, the denominator of (3.20) tends towards zero and may result in excessively large joint velocities.

During resolved rate control the operator, in essence, "flies" the tip of the manipulator. Cartesian tip velocity commands are passed through a rate resolver to joint rate servos. The controller is shown schematically in Figure 3.2.2. Since there is no explicit feedback of tip position, tip control is technically in an open-loop sense. However, resolved motion rate control allows the operator to directly control the tip velocity in a

Cartesian sense. This is a significant advantage over open-loop position control since manipulator operation is more intuitive and places fewer demands on a human operator.

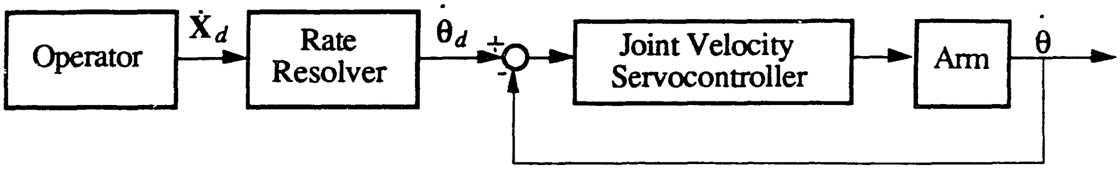


Figure 3.2.2: Resolved Motion Rate Control

An alternative to Jacobian-based resolved rate control is joint space differencing. In this technique, given a planned path of desired Cartesian tip states, **X** = f(t), joint positions are found by applying manipulator inverse kinematics. Joint rates are then determined by differencing of joint positions and are passed to the joint rate servos. This approach is perhaps easier to implement since fewer calculations are required than computing the inverse Jacobian. Differencing effects, however, warrant careful consideration (i.e. causality, sampling rate, noise, etc.) It should be noted that multiple solutions and singularities generated by inverse kinematics are also problems.

3.2.3. Closed-Loop Position

The simplest closed-loop scheme focuses on feedback of tip position. Desired Cartesian manipulator location, typically tip position with respect to the base frame, are converted to joint positions via inverse-kinematics. These positions are then directed to joint position servos. Since the servos operate on joint position, control errors are reflected in joint-space. This system operates in a closed-loop sense since tip position, transformed into joint positions by inverse-kinematics, is reflected in the joint-space controller error. Figure 3.2.3 shows a block diagram of closed-loop position control.

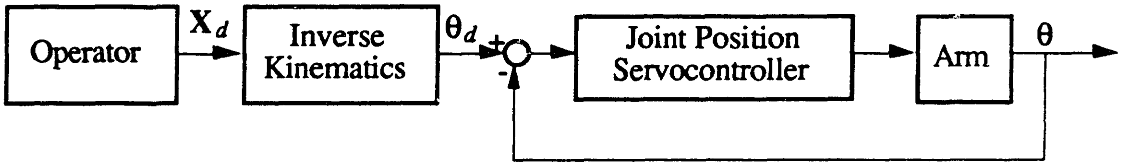


Figure 3.2.3: Closed-loop Position Control

In practice, this control scheme is generally augmented with a path planning module (either Cartesian or joint-space) which generates trajectories between specified endpoints or from symbolic specifications. A path planner is necessitated by several concerns including singularity handling, smoothness of trajectory (i.e. continuous first and second derivatives), and obstacle avoidance. The addition of a trajectory generator, however, dramatically increases computation demands on the manipulator controller and is not always feasible for real-time operations. The augmented closed-loop controller is shown below in Figure 3.2.4.

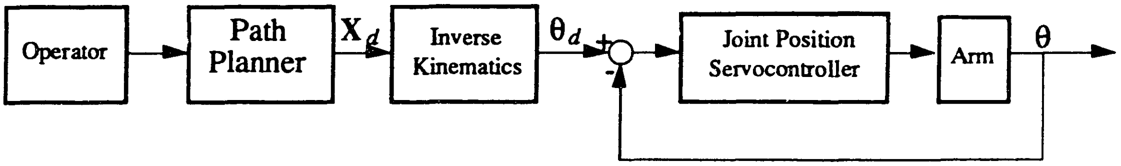


Figure 3.2.4: Augmented Closed-Loop Position Control

3.2.4. Closed-Loop Tip Control

Although the closed-loop controller described in the previous section has some usefulness, it cannot be used to explicitly control tip velocity. The only means of specifying velocities to this controller is through the augmented path planner (i.e. velocity transformed into trajectory positions). This approach is suboptimal, however, since the path planner must produce joint position commands closely spaced in time to achieve any degree of velocity precision. Moreover, there is no direct tracking of velocity error since the controller operates solely on joint position.

The preferred means of tip position and velocity control is to implement a multiple-input, multiple-output (MIMO) controller. In this scheme, desired cartesian position and velocities are resolved into joint angles and rates which are transformed into joint commands via some type of state-space control law structure. Transformation of position to joint angle is usually performed with inverse kinematics, while resolution of velocity to joint rate is accomplished by the inverse Jacobian. Since the controller is based in state-space, there is a vast assortment of possible control structure (e.g. pole-placement, LQR/LQG, etc.). Depending on the choice of control law, the low-level joint controller may be either a position or a velocity servo, though the latter is more common. A block diagram of this controller is shown in Figure 3.2.5.

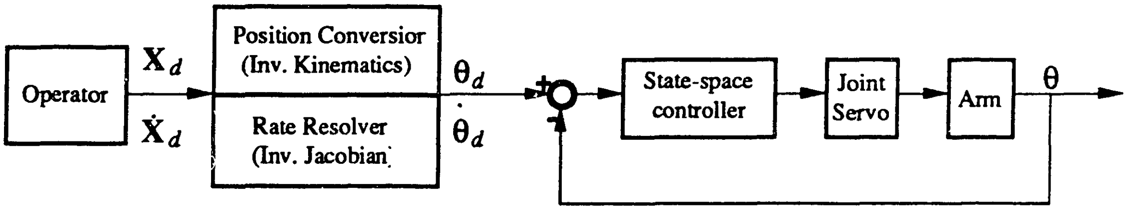


Figure 3.2.5: Closed-loop Tip Controller

As the figure shows, tip position and velocity errors are reflected in joint-space, with the MIMO controller inside the feedback loop.

3.2.5. Inverse Jacobian Controller

The tip control schemes described in previous sections (3.2.2 to 3.2.4) operate on errors described in joint-space. Since operator commands are generally given in Cartesian space, all of these methods require conversion to compute joint-space parameters. The primary difficulty with this approach is that the conversion process is computationally expensive (e.g. inversion of the Jacobian and its derivatives), particularly if higher order path derivatives are specified. In practice, typical current systems only transform position commands to joint angles. Velocity, and any other higher derivative, is then computed using differencing techniques. Differencing, however, may introduce undesirable side-effects such as lag and noise amplification. (Craig, 1986). Since commanding higher order derivatives (e.g. acceleration, jerk, etc.) is desirable, the problem is to suggest an alternative to joint-based control which requires fewer computations to obtain the required derivatives.

One approach to manipulator control which offers a potential solution is described in (Craig, 1986) as the *Inverse Jacobian controller.* In this scheme, controller errors are mapped to Cartesian space by introducing coordinate transformations directly into the servo loop. Cartesian errors (presumed to be small) may be mapped to joint-space displacements using the inverse Jacobian. These displacements are then converted to actuator commands which reduce the Cartesian errors. The structure of this control scheme is shown in Figure 3.2.6.

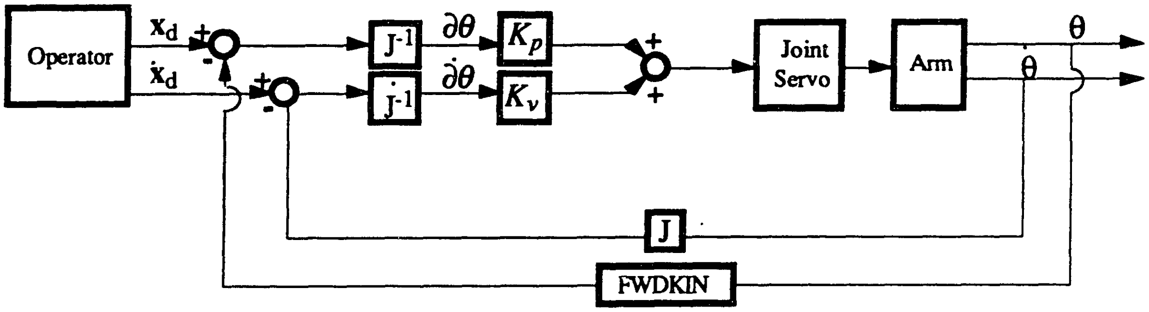


Figure 3.2.6: Inverse-Jacobian Controller.

3.3. Revolute Joint Design & Testing

In order to realize the serial-link manipulators described in section 3.1 in a neutral buoyancy environment, it was necessary to create a revolute joint for underwater operation. To implement the controllers detailed in section 3.2, it was additionally necessary to devise joint position and rate servocontrollers. Details of the development process and final design for both are given in the following.

3.3.1. Development of Neutral Buoyancy Actuators

The underwater setting associated with neutral buoyancy simulation is an extremely challenging environment for actuator design. Among the difficulties that must be considered in the design of such devices are corrosion, lubrication, and the electrical conductivity of water. The situation is further complicated when humans are introduced to the scene since consideration must be made for confinement of contaminants and safety factors. Moreover, since the baseline specifications for SPAM called for a high-torque revolute joint and a high-force linear actuator, it was mandated that any actuator design be capable of high power output. Taken collectively, these factors meant that the development of actuators for SPAM was not a trivial task.

Three fundamental criteria were selected to drive the actuator design. First, to minimize the deleterious effects of water, moving components and linkages were desired to be as mechanically simple as possible. Secondly, the means used to power the actuator had to be safe to submerged humans. Finally, the actuator was specified to be as cost-effective as possible.

To fulfill these requirements, a substantial research effort was made to determine the viability of various actuation schemes. Among the methods rejected after evaluation were electrical torque motors with gearing, pneumatically driven motors, and hydraulics. The first approach, electrical torque motor with gearing, was abandoned because of potential high voltage/currents and due to the gear lubrication problems. Pneumatically driven motors, though offering high torque at a low cost, had to be rejected since the required lubricated air supply presented a potential contaminant problem. Finally, hydraulics involving oil was dismissed since absolute prevention of oil leakage and spillage is extremely troublesome.

After rejecting traditional actuation methods, research was focused on novel approaches to actuator design. One such approach is the "water over air" cylinder. In this scheme, compressed air is used to supply actuating force and water, assumed to be incompressible, is used for braking and control. Since compressed air is clean, and because water is readily available in neutral buoyancy, it was decided to pursue development of a high-powered actuator using this concept.

The *Pneumatic and Hydraulic Actuating Device* (PHAD) is the result of these efforts. The PHAD, shown schematically in Figure 3.3.1, is a parallel arrangement of three cylinders, two pneumatic and one hydraulic. Air and water flow control is provided by solenoid valves. The air cylinders and solenoid valves are sized to meet force and supply demands. The hydraulic cylinder and solenoids are chosen according to braking and control considerations. Three cylinders were used in the PHAD design to guarantee balanced applied and braking forces.

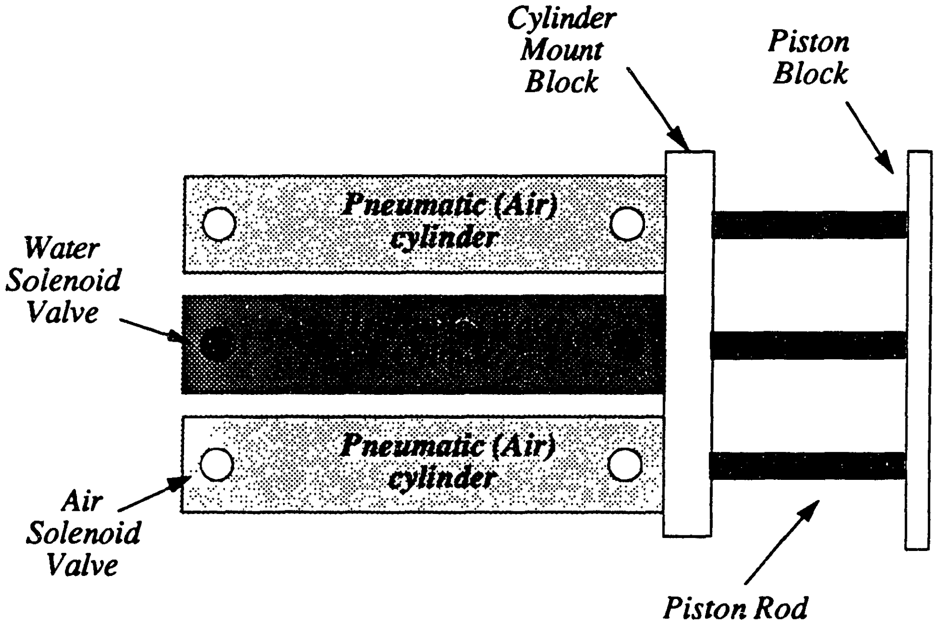


Figure 3.3.1: Pneumatic and Hydraulic Actuating Device (PHAD) *(schematic)*

During PHAD operation, compressed air is supplied to one side of the pneumatic cylinders. Since the hydraulic cylinder is sealed and contains an incompressible fluid (i.e. water), the PHAD pistons remain in a static position. Opening the water solenoid valves allows water flow, and hence, the pistons move. Moreover, if the period the water solenoids are open is regulated by some method (e.g. pulse-width modulation), precise linear actuation is possible. To illustrate this, consider the PHAD depicted in Figure 3.3.1. Suppling air to the left side of the air cylinders drives the pistons to the right, extending the piston rods. Controlling the water solenoids, and hence water flow through the hydraulic cylinder, regulates the piston rod extension speed. It should be noted that because the water solenoids are used as ON/OFF switches, and do not actually regulate flow volume, the PHAD is essentially a linear stepper.

For all of the PHAD implementations, compressed air is supplied to the pneumatic cylinders from a first-stage scuba regulator mounted on a 72 cu. ft. air tank. The regulator used, Scubapro model MK100, delivers 125-140 p.s.i. high-pressure air depending on ambient pressure and flow rate demand. Air flow control is provided by Koganei 110-4E1-F11 solenoid valves equipped with a 12 VDC coil. The supply of water is obtained directly from the surrounding environment. Laketown or KIP solenoids are used to control the intake and output flow of water from the hydraulic cylinders. The choice of Laketown or KIP solenoids is implementation specific and is dependent on hydraulic braking force considerations.

Three implementations of the PHAD concept were constructed. First, a revolute variant was developed to provide the in-line pitching required by the serial-link manipulator shoulder and elbow pitch joints. Since the PHAD is a linear device, it was necessary to employ a 5-bar linkage to convert motion from translation to rotation. This variant is described in section 3.3.2. Secondly, a different revolute variant was designed to provide the rotation required by the manipulator base-mounted shoulder yaw joint. For this variation, described in section 3.3.3, a 2-bar pivot was used. Finally, a linear PHAD was constructed for actuation of the Stewart Platform. This implementation is discussed in detail in section 4.3.1.

3.3.2. Revolute PHAD-A

To provide the pitching motion required by the SPAM manipulator shoulder pitch and elbow joints it is necessary to convert PHAD linear motion to rotation. Of the many potential means of performing this conversion, gearing and closed-chain linkages were studied. Gearing, using a toothed rack and gear system, is highly desirable since it produces constant rotation for constant linear motion and constant torque for constant linear force. Unfortunately, several problems (e.g. underwater lubrication, required gearing ratio, mounting location) made implementation unfeasible. Closed-chain linkages, on the other hand, are extremely simple mechanical devices and do not have these intrinsic difficulties. The disadvantage is that all closed-chain linkages have a non-linear relationship between linear and rotational motion. The logistical simplicity for implementation, however, resulted in a decision to utilize a closed-chain linkage with the PHAD, regardless of non-linearities.

As a result, a diagonally-driven 5-bar linkage was used to convert PHAD linear motion to rotation. The 5-bar linkage was chosen since it is has few links, provides good torque to force ratios, and is a compact spatial arrangement. As shown in Figure 3.3.2, the linkage is symmetrical with two upper link members and two lower link members joined by the diagonal drive.

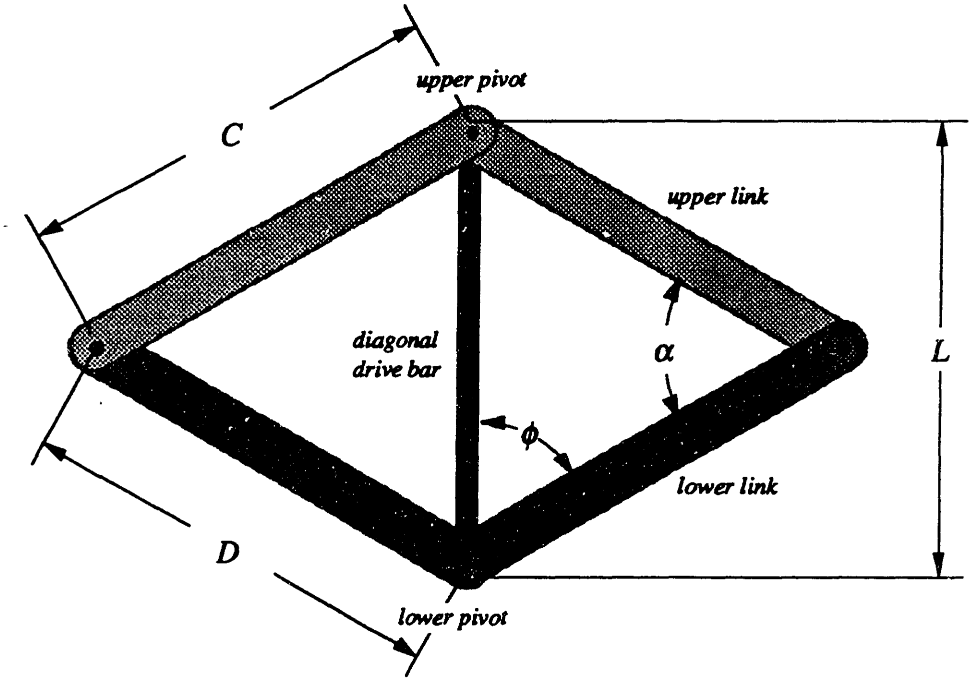


Figure 3.3.2: Diagonally-driven 5-bar linkage

In this configuration, extension of the center (drive) bar diagonally drives the linkage, causing the link bars to rotate about four pivot points. The motion is illustrated below in Figure 3.3.3

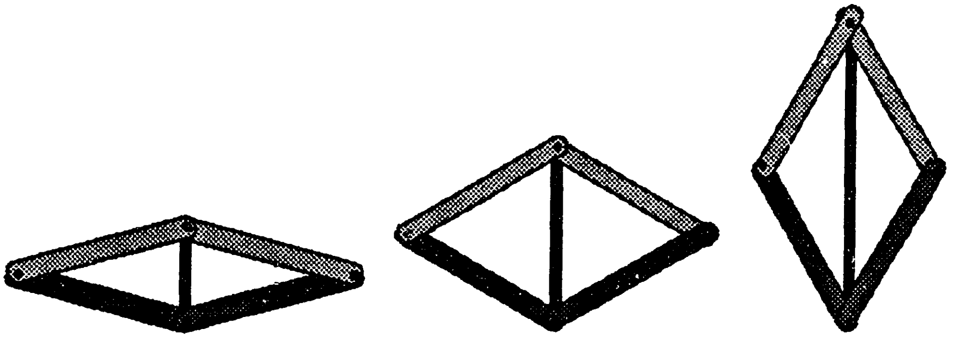


Figure 3.3.3: 5-bar linkage motion with diagonal drive
left to right – increasing drive bar extension

The extension of the diagonal drive member, L, is related to ϕ, the linkage half-angle at the lower pivot point, by:

$$\cos\phi = \frac{D^2 + L^2 - C^2}{2DL} \tag{3.21}$$

where **C** and **D** are lengths of the upper and lower pivot to linkage point link members respectively.

Force input through the diagonal drive, **F**, is converted to torque, τ, about the lower pivot point by:

$$\frac{\tau}{F} = \frac{CLD\sin\alpha}{C^2 + L^2 - D^2} \tag{3.22}$$

where α is the angle between upper and lower link members.

For the design of the PHAD 5-bar linkage, it was assumed that the range of motion for the diagonal drive was 12 inches. This corresponds to the largest stroke length available for small, cost-efficient pneumatic cylinders. Sizing of the link members was performed by studying rotational range and torque/force output as a function of drive member length. In order to utilize as much of the drive member range as possible, the upper and lower link bars were constrained to always sum to 12 inches. Characteristic curves for various link members lengths are shown below in Figures 3.3.4 (a–c):

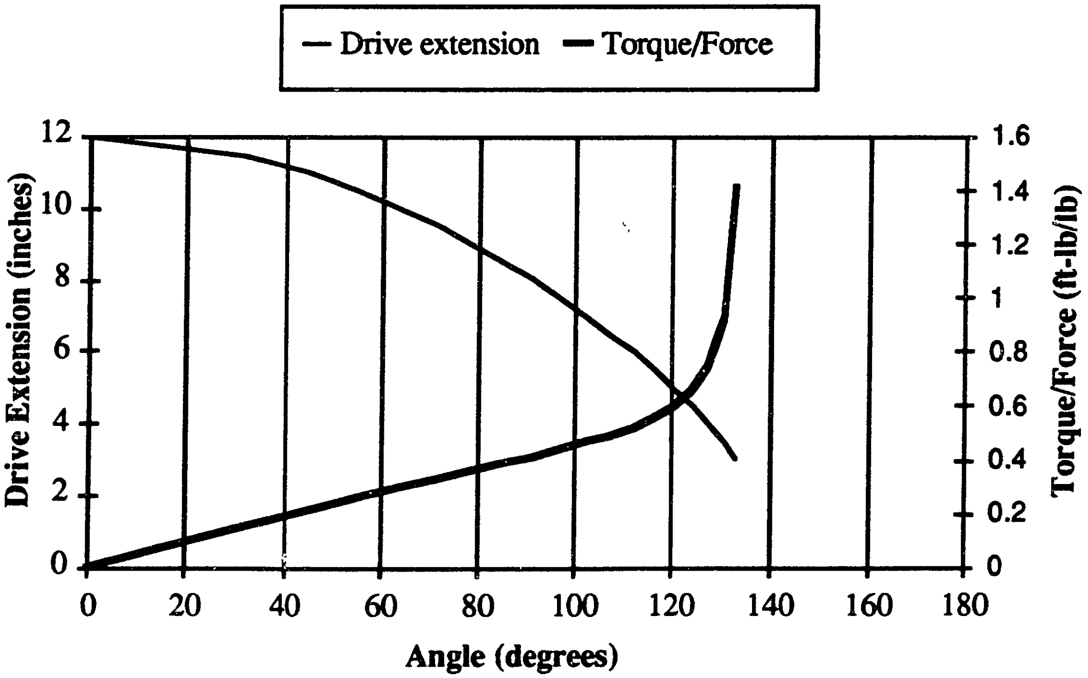


Figure 3.3.4 (a): Characteristic curve of 5-bar linkage
$(C = 5.75'', D = 6.25'')$

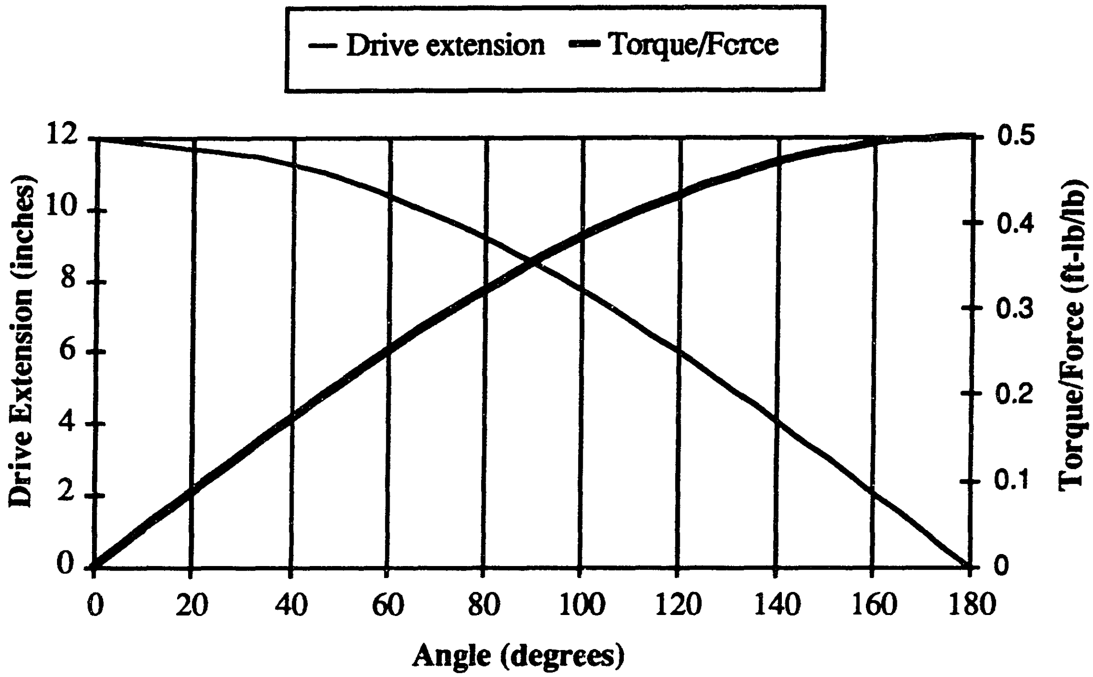


Figure 3.3.4 (b): Characteristic curve of 5-bar linkage
(C = 6.00", D = 6.00")

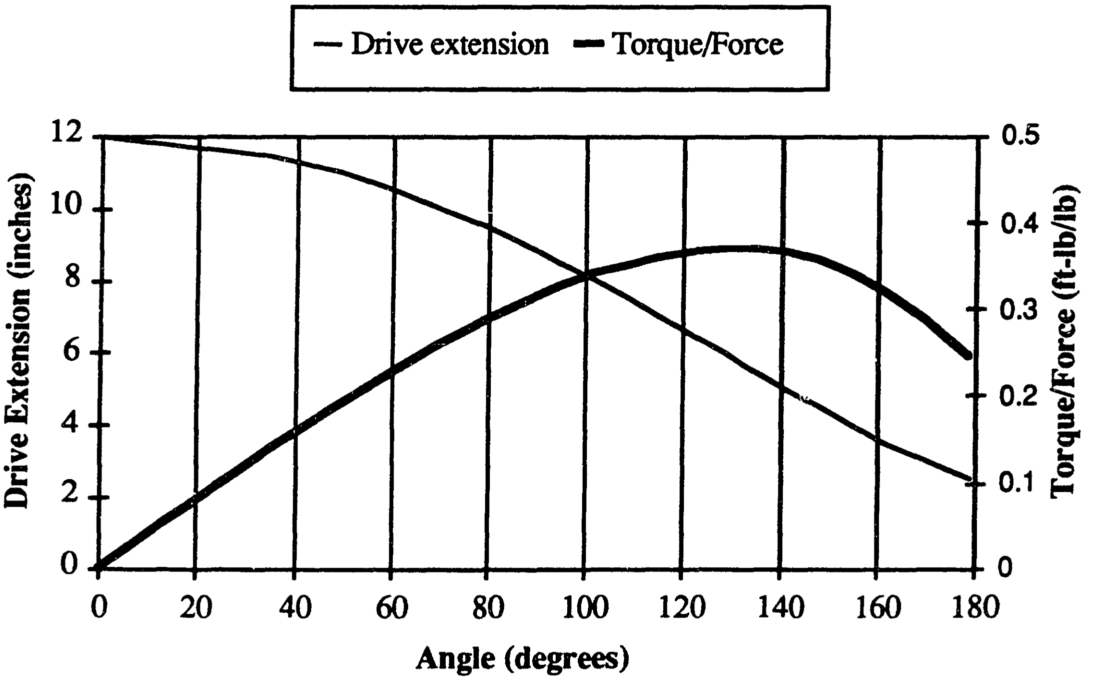


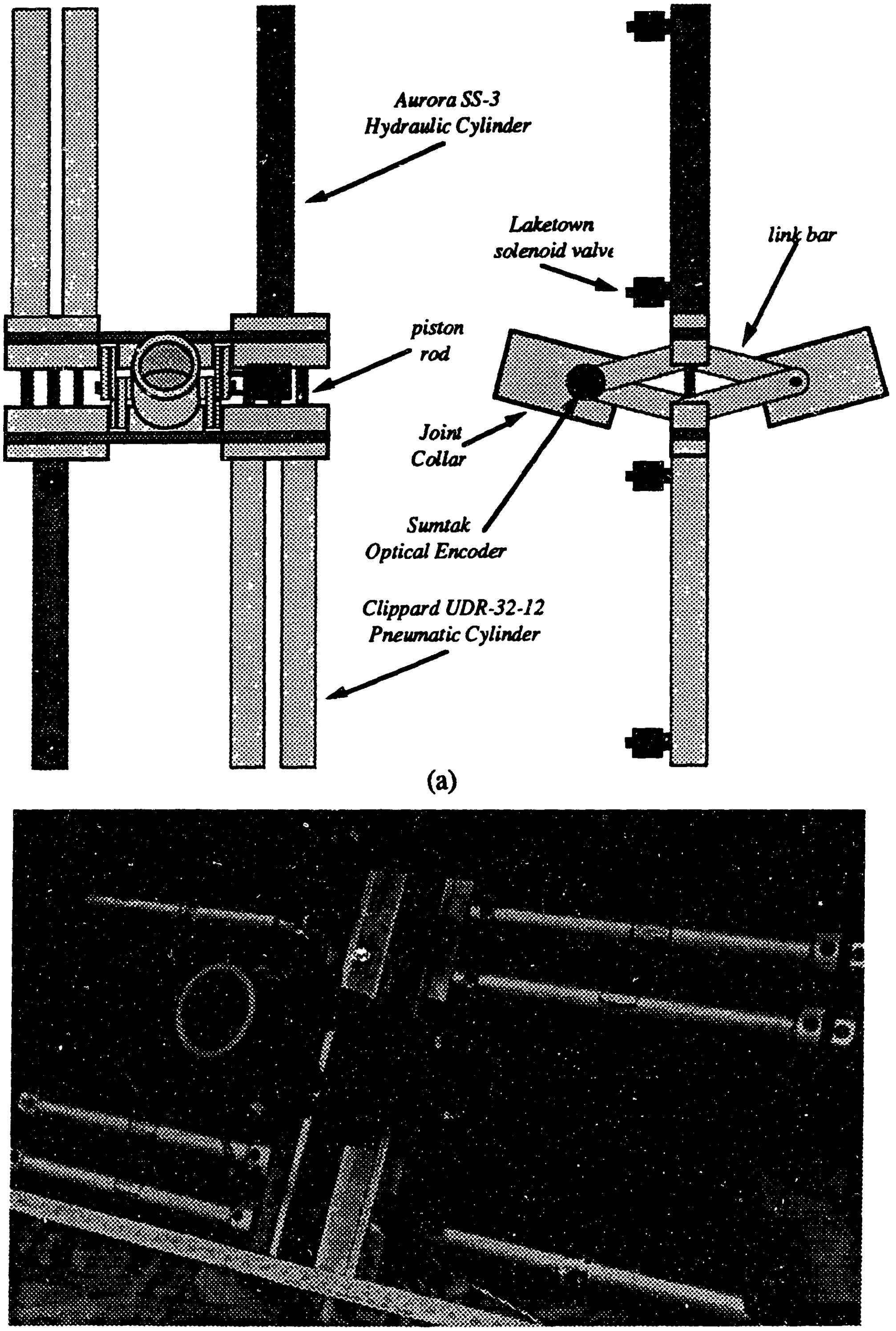
Figure 3.3.4 (c): Characteristic curve of 5-bar linkage
(C = 6.25", D = 5.75")

As a result of these studies, symmetrical 6 inch links were chosen for the nominal 5-bar linkage design. The characteristic curve is shown above in Figure 3.3.4 (b). The motivating factor for selecting these link sizes was that both the drive extension-to-link angle and the input force-to-output torque curves were smooth, monotonic, functions over the full drive member range. Moreover, the theoretical rotational range covered a full 180°.

PHAD cylinder sizing was governed initially by an attempt to meet the baseline torque specification. For most of its rotational range, the 5-bar linkage offers a torque-to-force ratio of 0.167 or better, indicating that at a minimum 4,500 lbs. of force is needed to generate 750 ft-lb of torque. The scuba compressed air supply will deliver at least 125 p.s.i., so that 36 square inches of pneumatic cylinder bore area is required. Discussions with cylinder manufacturers revealed that 2 inch bore cylinders had the best cost-effectiveness and logistics, as well as being readily obtainable. This meant, however, that a total of twelve, 2 inch bore, cylinders would be needed to provide the required bore area. Since this number of cylinders would have been exceedingly difficult to integrate, a decision was made to forgo the torque specifications in favor of joint compactness. Sizing of hydraulic cylinders was performed in a similar manner, with the emphasis placed on producing 4,500 lbs. of braking force.

The final design of the revolute PHAD (R-PHAD-A) for in-line pitching utilizes 4 Clippard UDR-32-12 pneumatic cylinders and 2 Aurora SS-3 hydraulic cylinders. Each of these cylinders has a 2 inch bore with a 12 inch stroke. The Clippard cylinders, driven by 125 p.s.i. air, produce a total force of 1,570 lbs corresponding to a minimum joint torque of 262 ft-lbs. The Aurora cylinders are equipped with hydraulic piston and shaft seals and are capable of 2,000 lbs. maximum breaking force. Laketown solenoids were chosen for control water flow because of flow rate capability and pressure rating. A schematic of the R-PHAD-A is shown in Figure 3.3.5. Detailed drawings are included in Appendix B.

There are a few items of note in the figure. First, the right view clearly illustrates the integrated five-bar, diagonally driven linkage. Secondly, the four Laketown solenoids are directly mounted on the Aurora cylinders. This allows instantaneous water flow control during joint operation. Finally, the joint/arm boom interface (labeled as "Joint Collar") is shown. This connector is described in section 3.4.2.



(a)

(b)

Figure 3.3.5: Revolute PHAD-A
(a) schematic (b) assembled R-PHAD-A

Submerged, the R-PHAD-A joint weighs approximately forty pounds. To simulate zero-gravity, foam and fiberglass flotation panels were installed to help make the joint neutrally buoyant. The resulting joint is shown below:

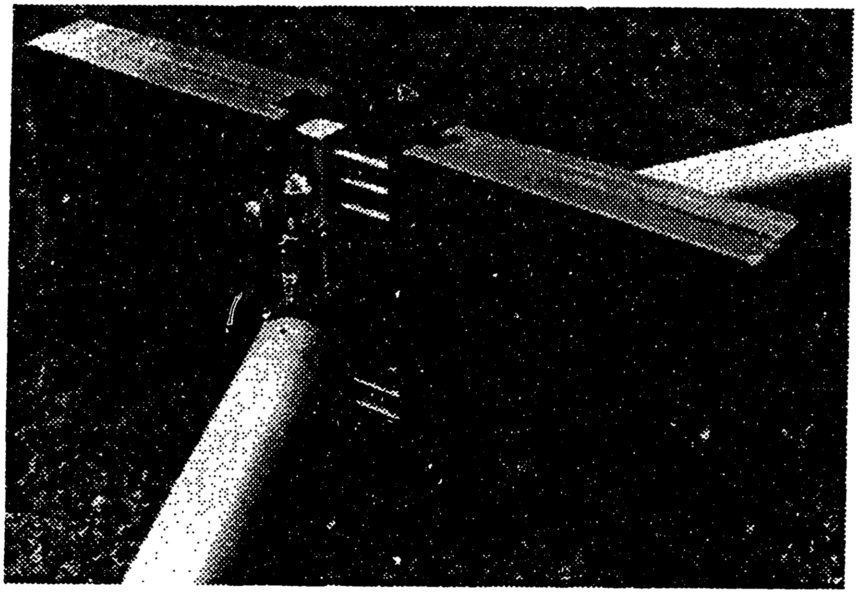


Figure 3.3.6: Neutrally-buoyant R-PHAD-A

3.3.3. Revolute PHAD-B

The SPAM manipulator arm shoulder yaw joint is mounted at the base of the system. This base is intended to be mounted on a variety of surfaces such as a worksite platform, a STS payload bay mockup, and on telerobots. Additionally, the joint rotation axis is located normal to the link axis since the joint provides yaw motion. Considering these factors, the revolute PHAD design described in the previous section could not easily be used for this joint. As a result, it was necessary to design a different rotational PHAD implementation. The approach taken, however, attempted to utilize as much of the previous design as possible. Therefore, the same number and type of cylinders and solenoid valve used on the R-PHAD-A were targeted for use in the R-PHAD-B design. Similarly, a closed-chain linkage would be employed instead of gearing.

Since the R-PHAD-B joint had to provide yawing about the link axis rather than pitching through (as the R-PHAD-A does), a two-bar crank was chosen to convert linear motion to rotary motion. The crank configuration, shown in Figure 3.3.7, utilizes a cylinder piston rod as the drive member.

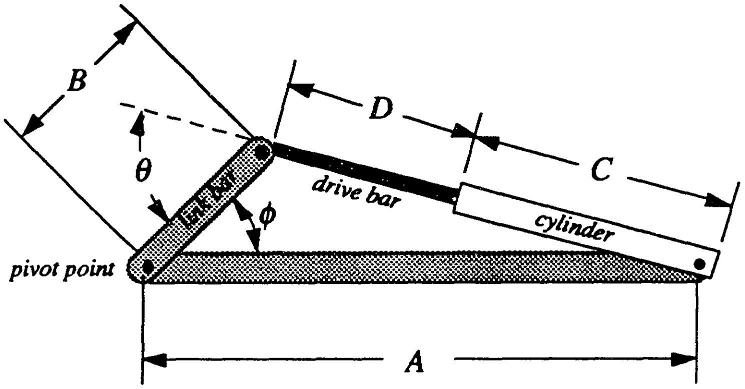


Figure 3.3.7: Pivoting 2-bar crank

For this closed-chain linkage, extension of the drive cylinder piston rod rotates the link bar about the pivot point. This motion is illustrated below in Figure 3.3.8

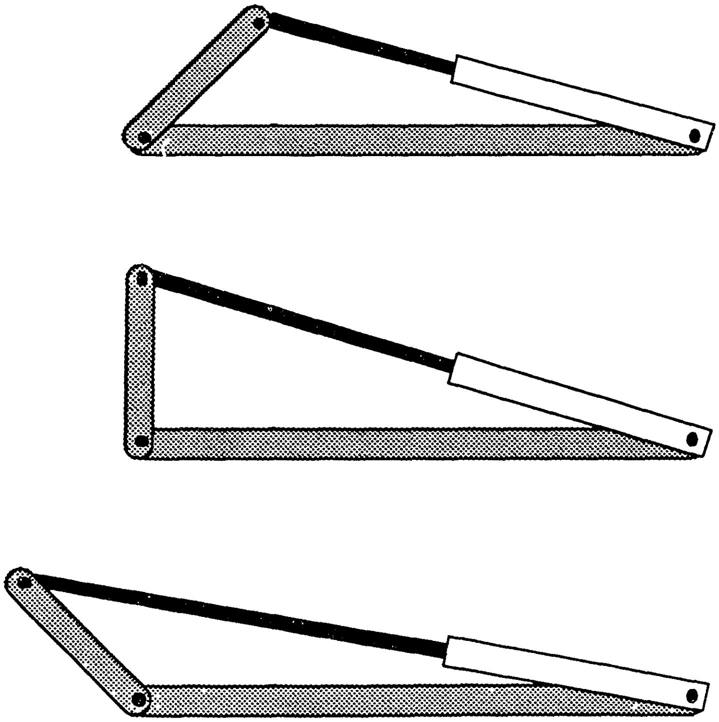


Figure 3.3.8: 2-bar cranking motion
top to bottom–increasing drive bar extension

The rotation produced by the 2-bar crank is related to the drive extension, **D**, by the following:

$$\cos\phi = \frac{A^2 + B^2 - C^2 - D^2}{2AB} \qquad (3.23)$$

where **A** is the cylinder mount to joint axis distance, **B** is the length of the link bar, and **C** is the length of the drive cylinder.

Force applied through the drive rod is converted to torque, τ, through the link bar. The torque-to-force ratio is:

$$\frac{\tau}{F} = B\sin\theta \qquad (3.24)$$

where θ is the exterior drive-to-link bar angle.

Characteristic curves of joint rotation and torque-to-force are presented below in Figures 3.3.9 (a) to (c) for different actuator mount to joint axis and link bar lengths.

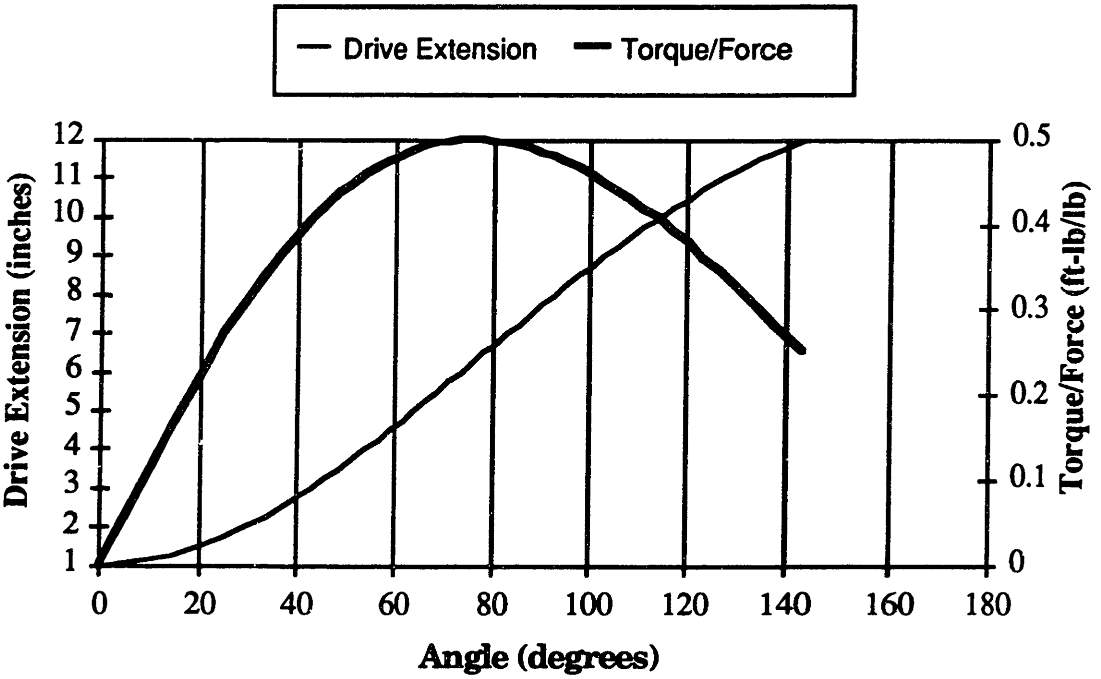


Figure 3.3.9 (a): Characteristic curve of 2-bar crank
(A = 25, B =6)

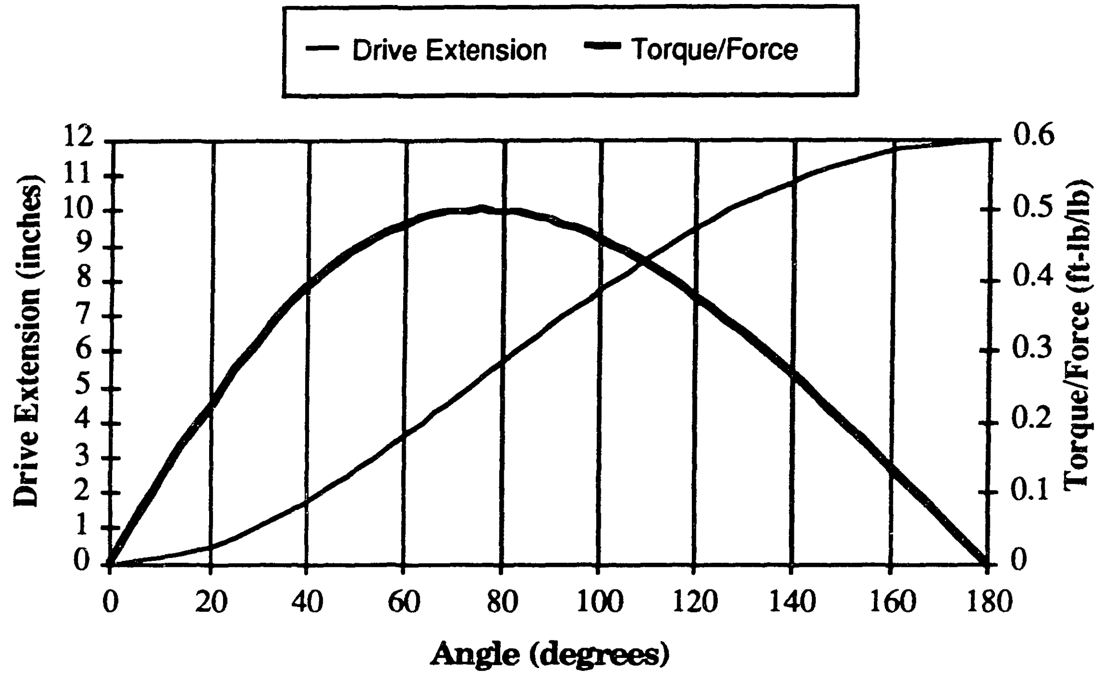


Figure 3.3.9 (b): Characteristic curve of 2-bar crank
(*A* = *24*, *B* =*6*)

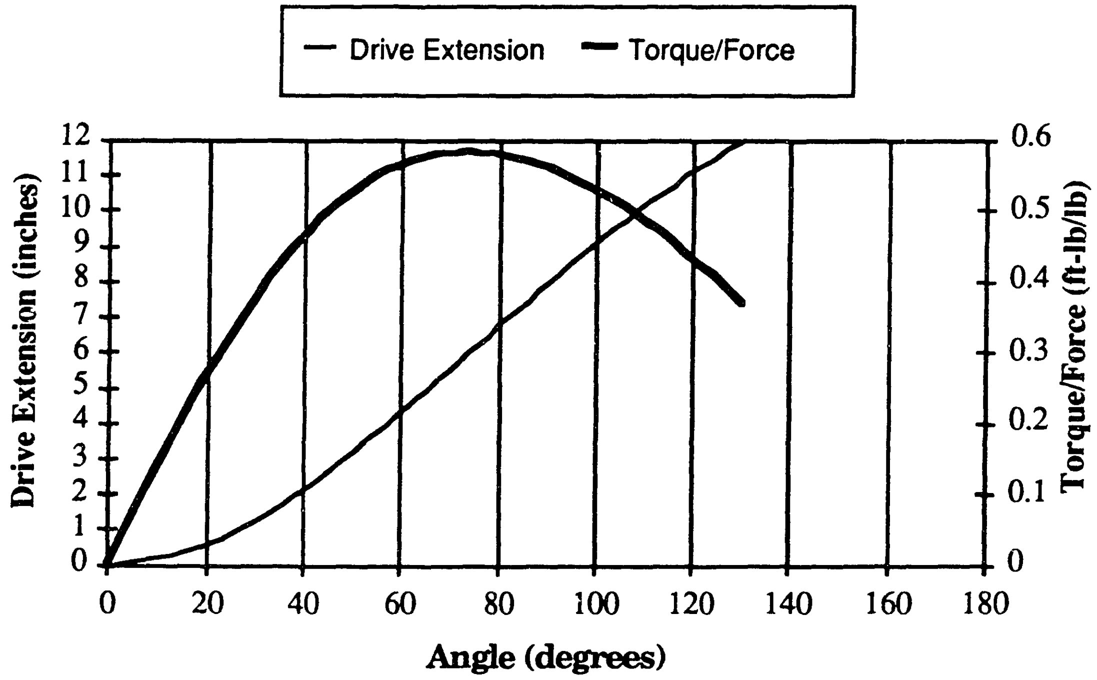


Figure 3.3.9 (c): Characteristic curve of 2-bar crank
(*A* = *25*, *B* =*7*)

The curves shown in Figure 3.3.9 (b) resulted in 2-bar crank linkage lengths of $A=24$ and $B=6$ inches. Although Figures 3.3.9 (a) and (c) demonstrate higher average and peak torque output, the angular range is smaller than that of Figure 3.3.9 (c). Using the lengths from Figure 3.3.9 (b) gives a smooth torque curve over a theoretical 180 degrees.

Due to the yawing motion produced by the 2-bar crank, it was necessary to include a bearing in the R-PHAD-B design. Off-the-shelf bearings, however, could not be used because of lubrication and corrosion problems inherent with a water environment. Moreover, since the bearing is mounted along the arm link axis, it had to be capable of carrying large loads. For a base-to-tip distance of 50 feet (similar to the SRMS) with a maximum dynamic tip load of 50 lbs., the approximate bearing loading is 50 lbs axial and 2,500 lbs radial.

The final R-PHAD-B design (Figure 3.3.10) utilizes a metallic-plug type bearing custom manufactured by the Spadone-Alfa Corp. This bearing contains a pattern of solid-lubricant filled plugs covering the load carrying surfaces. The solid-lubricant used is Metaline, a dry lubricant composed primarily of metallic oxides and graphite, which provides bearing lubrication in hostile environments including chlorinated water. Design drawings of the R-PHAD-B are given in Appendix B.

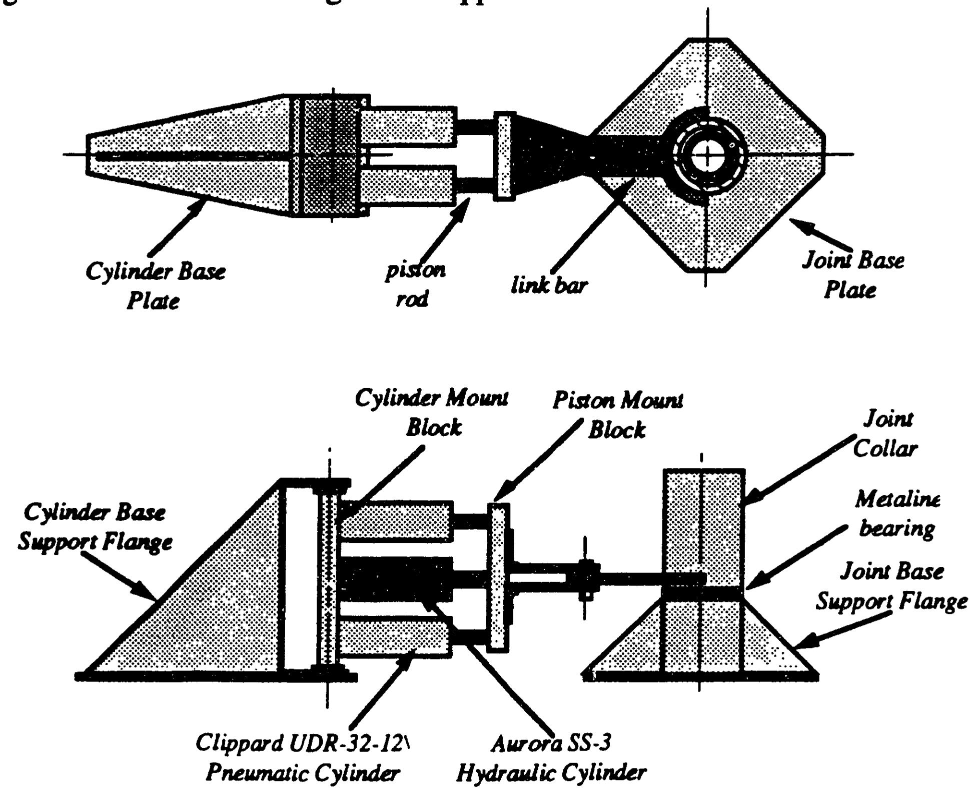

Cylinder Base Plate
piston rod
link bar
Joint Base Plate
Cylinder Mount Block
Piston Mount Block
Joint Collar
Cylinder Base Support Flange
Metaline bearing
Joint Base Support Flange
Clippard UDR-32-12\ Pneumatic Cylinder
Aurora SS-3 Hydraulic Cylinder

Figure 3.3.10: R-PHAD-B schematic

3.3.4. Position Sensor

To provide angular information from each joint, an optical encoder (Sumtak LEI-037-2048) was waterproofed and mounted on each R-PHAD. Since this encoder is manufactured with "O"-ring casing and shaft seals, waterproofing involved simply the attachment of a low-pressure air line.

The Sumtak LEI-037-2048, produces 2048 pulses per revolution of quadrature output. Since the R-PHAD designs have a theoretical range of 180 degrees (0.5 revolutions), angular position covers a 12-bit range:

$$\begin{aligned}\left(2048\,\frac{\text{pulse}}{\text{revolution}}\right)\left(4\,\frac{\text{counts}}{\text{pulse}}\right)(0.5\text{ revolution}) &= 4096\text{ counts}\\ &= 2^{12}\text{ bits}\end{aligned} \qquad (3.25)$$

The design of the R-PHAD-A prevented mounting of the optical encoder directly along the joint pivot axis. Instead, the encoder was mounted at the 5-bar linkage link bar pivot point as shown in Figure 3.3.11 and a trigonometric transformation used to derive joint angle from measured pivot angle.

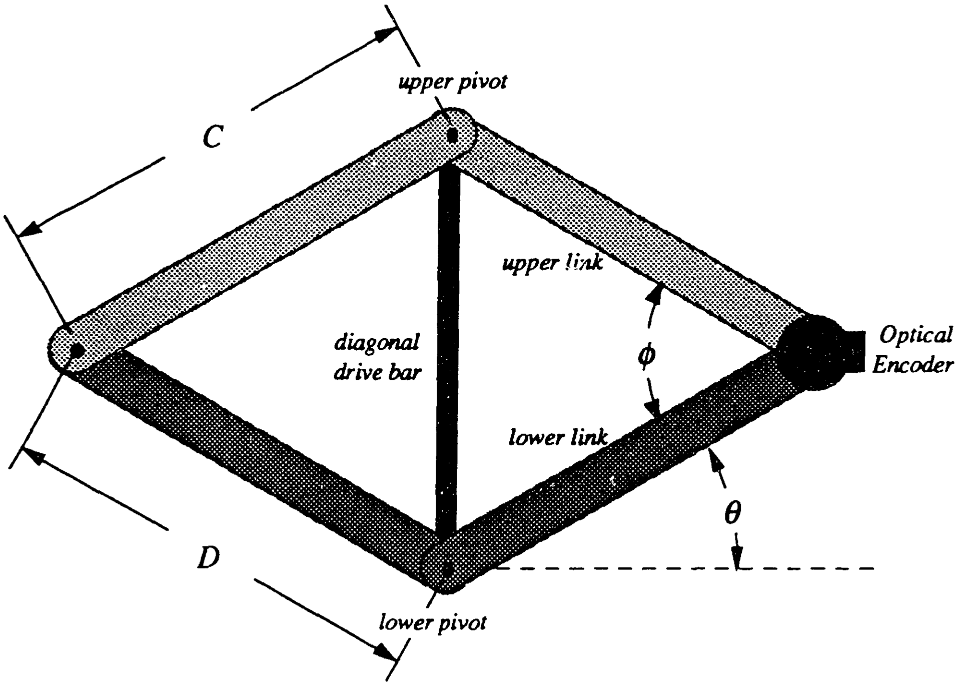


Figure 3.3.11: R-PHAD-A Optical Encoder Mounting

The transformation used is:

$$\theta = \pi - 2\tan^{-1}\left[\frac{D\sin\phi}{C\text{-}D\cos\phi}\right] \tag{3.26}$$

where C and D are the lengths of the upper and lower link bars, and ϕ is the angle between these bars measured by the optical encoder.

3.3.5. Joint Servo Control

The controllers discussed in this section provide the lowest level of control for the SPAM manipulator arm. Since the control schemes attempt to minimize errors between commanded and actual joint states (i.e. angle and angular velocity), the controllers may be classified as servocontrollers. In particular, the following sections describe design and testing of simple discrete-time controllers; section 3.3.5.1 gives details of a proportional-plus-derivative position servo and section 3.3.5.2 lists a number of velocity servo schemes.

All of the servo schemes were tested with a R-PHAD-A joint using an IBM microcomputer. The controllers were implemented in software with the Microsoft C v5.1 optimizing compiler. A complete C code listing is provided in Appendix C. Joint position was obtained using the optical encoder described in section 3.3.4 via a Hewlett-Packard HCTL-2000 quadrature decoder/counter. Joint rates were inferred from three step backwards differencing. All servo outputs were transformed into PHAD solenoid control signals via two methods. The Koganei air solenoids were sent binary signals specifying on/off air flow. The Laketown water solenoids received 4-bit pulse width modulated (PWM) signals to control water flow rate.

Testing revealed that the Laketown solenoids used on the R-PHAD-A cannot be driven faster than 9 Hz. with a 4-bit PWM. Consequently, the servos described in the following sections were run at a 9 Hz. control loop rate.

3.3.5.1. Position Servo Design & Testing

Position servocontrollers have limited applicability to manipulator arms. This is not to say that position servos are not used, in fact most industrial robots utilize such controllers, simply that velocity servos have a wider range of usefulness for manipulator control schemes (see Section 3.2). Since overshoot of joint angle may place the arm in a singular configuration or cause collision with some obstacle, most position servos are critically damped or overdamped. Consequently, these servos generally have long response times and poor overall response, though this typically is not a problem.

As was discussed in Section 3.2, the non-linearities inherent with manipulators and an underwater environment make accurate modelling a difficult task. Although non-linear model-based control has reasonable potential, model-based control in the linear sense is certainly of dubious merit. This does not mean, however, that linear control laws cannot be applied to SPAM joint control. In fact, if one considers that position servos need to be heavily damped, that slow response time is not problematic, and that an underwater environment tends to quickly damp out high frequency effects, then it is indeed feasible to attempt some form of linear control in the absence of system modelling. To this end a linear position servo, specifically proportional-plus-derivative (P-D), was been implemented for SPAM joint position control.

The classical P-D controller is classified as having a phase-lead structure since it attempts to improve system stability and response by increasing the phase-lead angle. Although the P-D does not have the steady-state accuracy of either a proportional-plus-integral (P-I) or proportional-plus-integral-plus-derivative (P-I-D) controller, it tends to produce better time response characteristics than purely proportional (P) control. Additionally, from a digital perspective, a P-D controller is only slightly harder to implement than a P controller, and much less computationally expensive than a P-I-D.

The "textbook" P-D controller is expressed in continuous-time Laplace form as:

$$G_c(s) = \frac{U(s)}{E(s)} = K_p + K_d s \tag{3.27}$$

where *U(s)* is the Laplace transformed controller output and *E(s)* the Laplace transformed error signal. This may be expressed in the discrete-time domain by applying a backwards-difference transformation:

$$s = \frac{1 - z^{-1}}{T} \tag{3.28}$$

where T is the sampling period. This results in:

$$G_c(z) = \frac{U(z)}{E(z)} = K_p + K_d\left(\frac{1 - z^{-1}}{T}\right) \tag{3.29}$$

Z-transforming results in the difference equation for control:

$$
\begin{aligned}
u_k &= K_p e_k + \frac{K_d}{T}(e_k - e_{k\text{-}1}) \\
&= \left(K_p + \frac{K_d}{T}\right)e_k - \frac{K_d}{T}e_{k\text{-}1} \\
&= K_1 e_k - K_2 e_{k\text{-}1}
\end{aligned}
\tag{3.30}
$$

For the joint position servo, the error signal is derived by minimizing differences between commanded angle and actual angle, so the P-D control law is:

$$u_k = K_1(\theta_d - \theta)_k - K_2(\theta_d - \theta)_{k-1} \tag{3.31}$$

A block diagram of the complete P-D joint position servo is given in Figure 3.3.12:

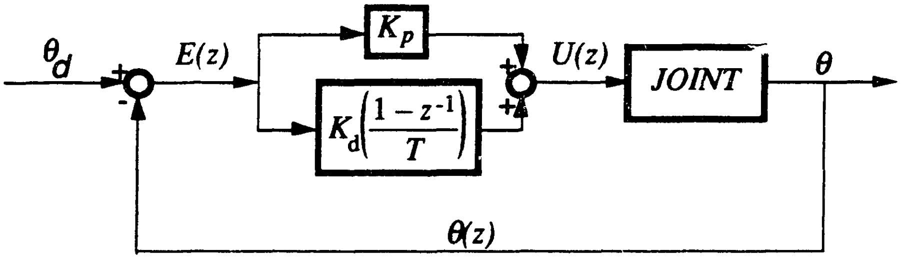


Figure 3.3.12: P-D Joint Position Servo

Testing of the position servocontroller was performed in neutral-buoyancy using a R-PHAD-A joint. Six foot arm booms were attached to either side of the joint, and the assembly vertically mounted by attaching one of the booms to a floor mount. To simulate loaded operation, a ten pound weight was placed at the tip of the moving arm boom. Step testing was performed by commanding the joint to move from zero to 90 degrees. Testing results for two sets of P-D gains are shown in Figure 3.3.13.

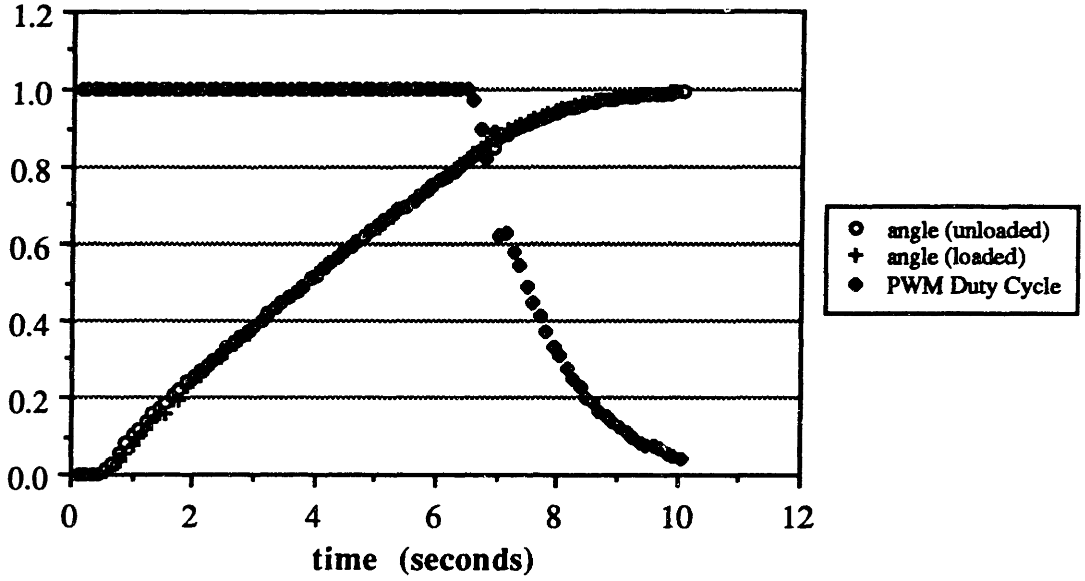


Figure 3.3.13 (a): P-D Position Servo ($K_p = 1.0$, $K_d = 0.1$)
90 degree step response (normalized)

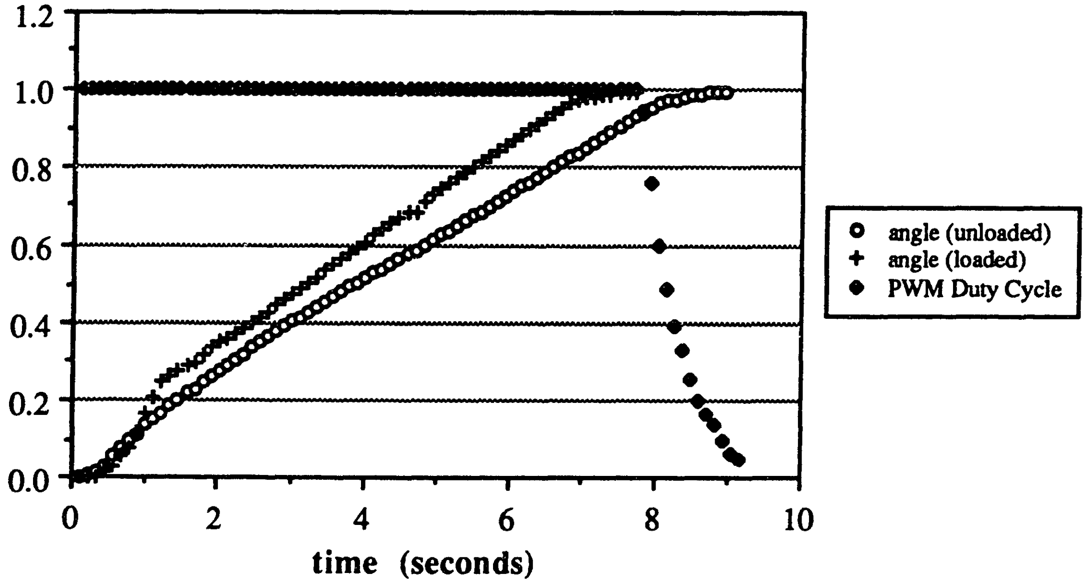


Figure 3.3.13 (b): P-D Position Servo ($K_p = 2.0$, $K_d = 0.1$)
90 degree step response (normalized)

As the figures show, the P-D servo produces a smooth response to the step input. For much of the step, the joint position changes linearly with time in response to the saturated actuator command. Near the desired angle, the position curve is nicely overdamped as a result of the low P-D gains and water damping effects.

3.3.5.2. Velocity Servo Design & Testing

The design of a joint angular velocity servo followed an approach similar to that of the position servo previously described. The fundamental assumption made was that the damping characteristic of an underwater environment tends to smooth response and maintain stability by limiting the system bandwidth to very low frequencies. In the absence of system modelling, therefore, velocity servos using simple control laws and/or heuristics would be sufficient to provide adequate joint control.

The first approach to velocity servo design was the *Basic Linear Actuator Heuristic* (BLAH) controller. This controller utilized the simple heuristic:

- If the joint is not moving at the desired rate, increment or decrement the actuator control signal by a fixed quantity.

The rationale for this heuristic is that at the lowest control level, the control signal output to the PHAD solenoids is coarsely discretized (a 4-bit PWM signal). If the controller and PWM generator can operate at a higher frequency than the water damped system, therefore, it should be possible to regulate joint velocity simply by flipping PWM bits. One control law formulation is then:

$$U_k = U_{k-1} + \mathrm{sgn}\left[\left(\dot{\theta}_d - \dot{\theta}\right)_k\right] \tag{3.32}$$

where "sgn" is the sign of the error term. Figure 3.3.14 shows a block diagram of the complete BLAH controller.

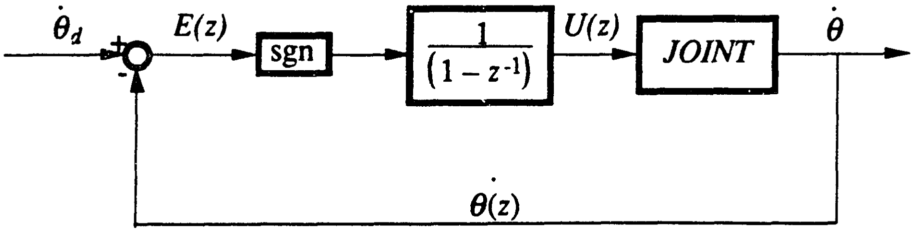


Figure 3.3.14: BLAH Controller Logic

In practice, however, the controller bandwidth is limited by the response time of the pulse-width modulated solenoids. Since the Laketown solenoids restrict the control loop to 9 Hz. sampling, the BLAH controller bandwidth is on the same order as the physical system. As a result, the controller exhibits large overshoot and severe chattering. Step response of the unloaded joint, for a 1 deg/sec command, is shown in Figure 3.3.15:

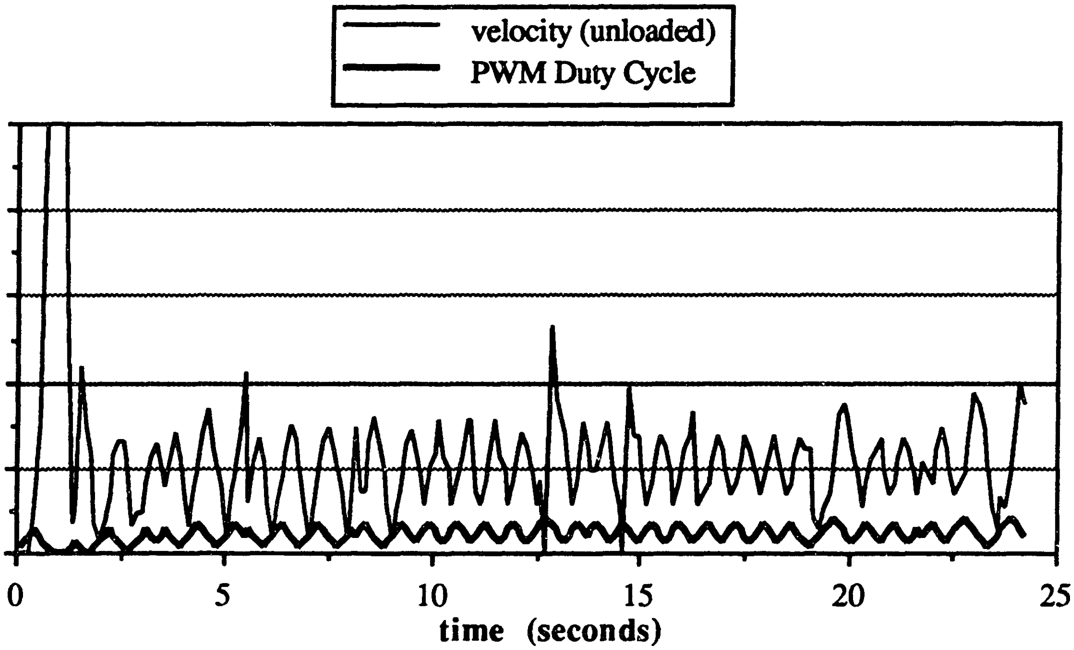


Figure 3.3.15: BLAH Velocity Servo
1 deg/sec step response (normalized)

The second velocity servo was designed to be a simple proportional controller. Error, defined as the difference between commanded and actual velocities, is multiplied by a proportional gain and combined with the current control signal. The effect of this is to drive the control output to a setpoint and produce the commanded velocity. The control law expressed as a difference equation is:

$$u_k = K_p\left(\dot{\theta}_d - \dot{\theta}\right)_k + u_{k-1} \qquad (3.33)$$

Figure 3.3.16 shows a block diagram of the complete proportional velocity error servocontroller.

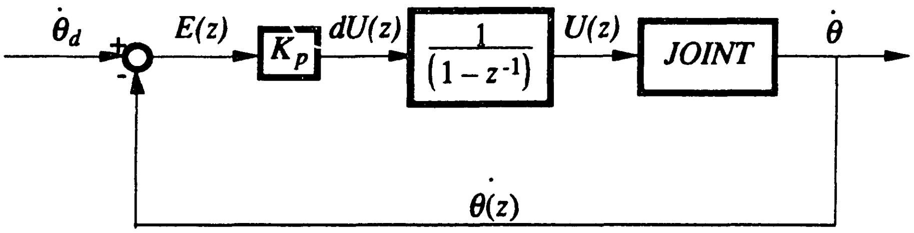


Figure 3.3.16: Proportional Velocity Servo

Step testing revealed that the proportional velocity error controller has better command tracking characteristics than the BLAH controller. Response time, however, is a bit slow taking approximately 25 sampling intervals. Additionally, overshoot is very high; on the order of 1,000 percent for the unloaded joint and 600 percent with ten pounds loading. Step responses (1 deg/sec step) of this controller, for both unloaded and loaded operation, are shown below in Figure 3.3.17.

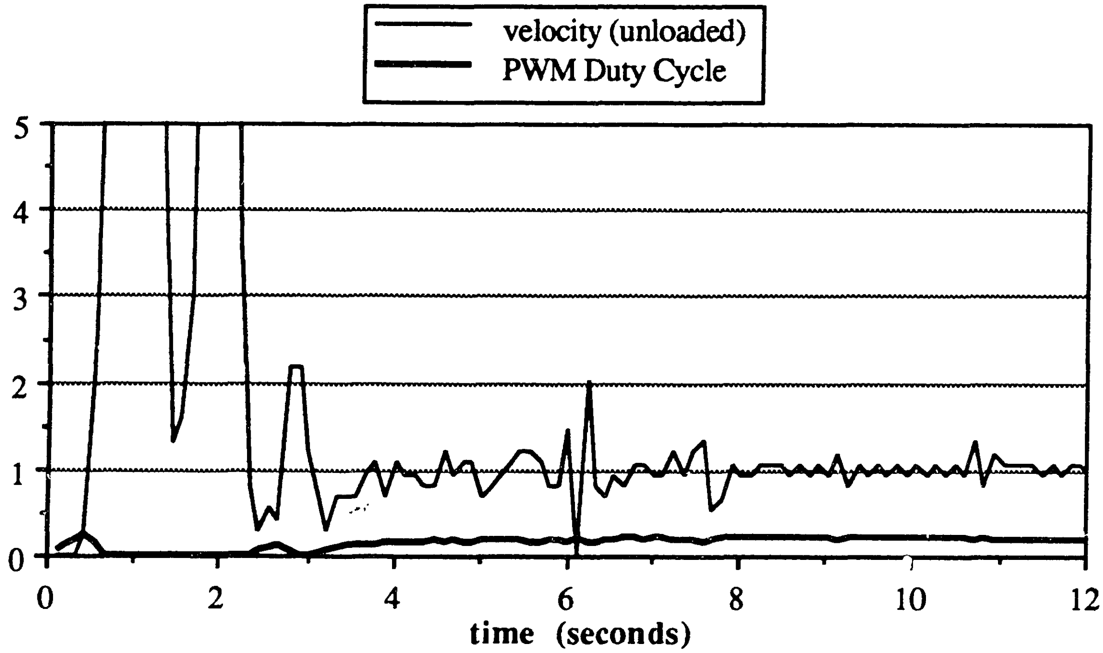


Figure 3.3.17 (a): P Velocity Servo (K_p = 1.0)
1 deg/sec step response (normalized), unloaded operation

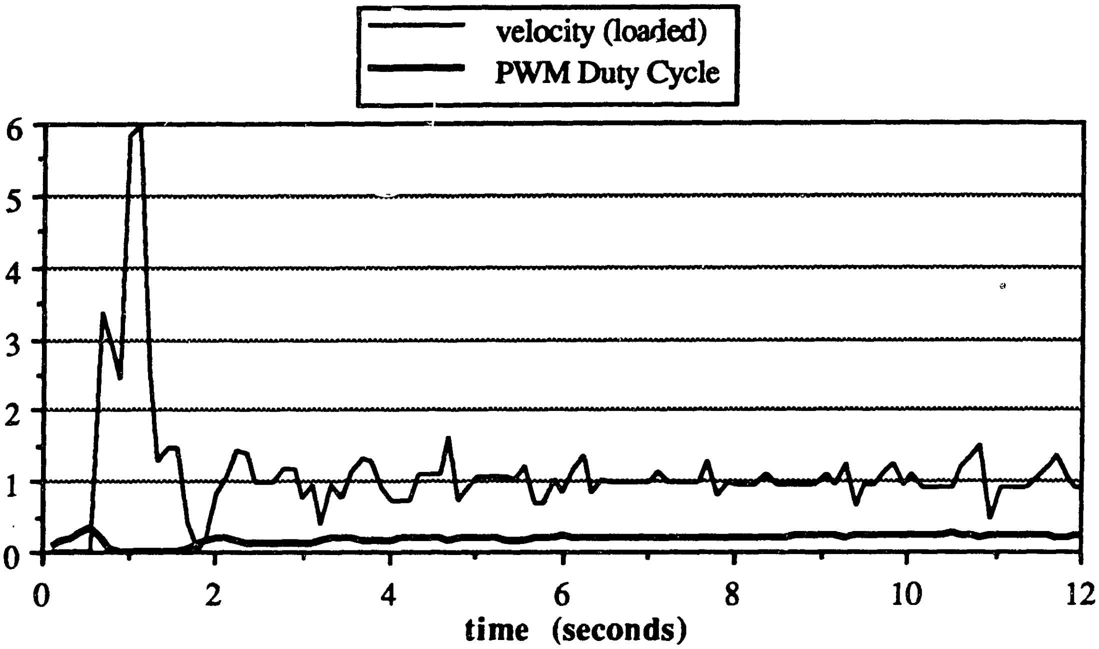


Figure 3.3.17 (b): P Velocity Servo (K_p = 1.0)
1 deg/sec step response (normalized), loaded operation

The P-D velocity servo was the final controller implemented. It was designed as an attempt to improve the transient response characteristics of the P velocity servo by the addition of an error derivative term in the control law. Similar to the P velocity servo, the error to be minimized is defined as the difference between commanded and actual velocities. As expected, the P-D velocity servo has a control law identical to the P-D position servo except for the error term:

$$\begin{aligned} u_k &= K_1 e_k - K_2 e_{k-1} \\ &= K_1\left(\dot{\theta}_d - \dot{\theta}\right)_k - K_2\left(\dot{\theta}_d - \dot{\theta}\right)_{k-1} \end{aligned} \tag{3.34}$$

Figure 3.3.18 shows a block diagram of the complete P-D servocontroller.

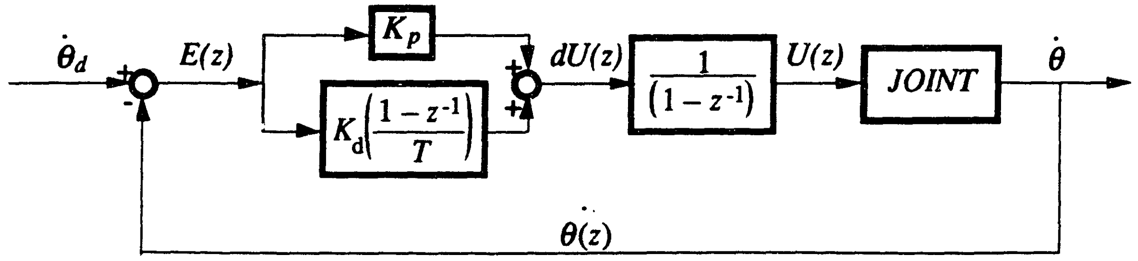


Figure 3.3.18: Proportional-plus-derivative Velocity Servo

Step testing revealed that the P-D controller has the best command tracking characteristics of all three velocity servos. Step responses of this controller are shown below in Figure 3.3.19 (1 deg/sec step) and Figure 3.3.20 (3 deg/sec step).

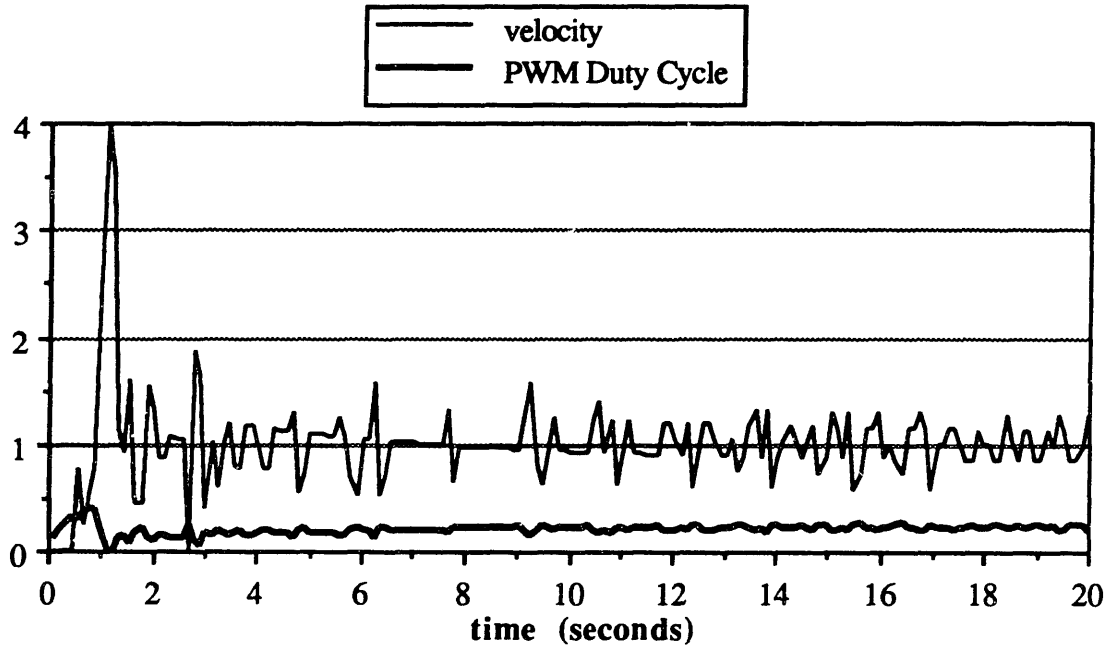


Figure 3.3.19 (a): P–D Velocity Servo (K_p = 1.0, K_d = 0.1)
1 deg/sec step response (normalized), unloaded operation

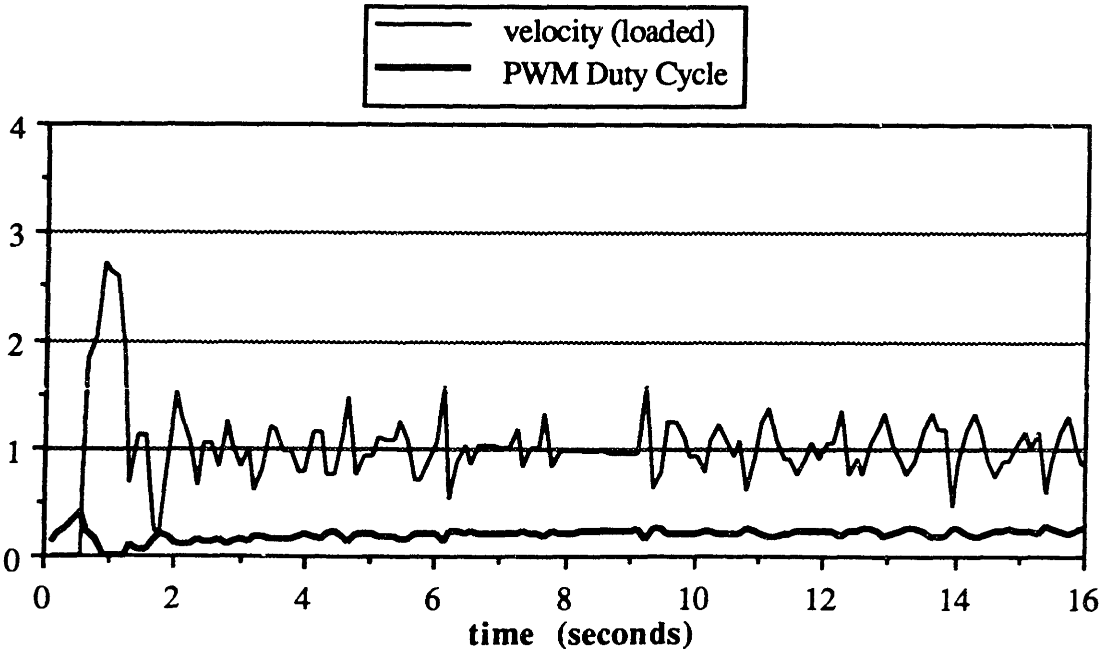


Figure 3.3.19 (b) P–D Velocity Servo (K_p = 1.0, K_d = 0.1)
1 deg/sec step response (normalized), loaded operation

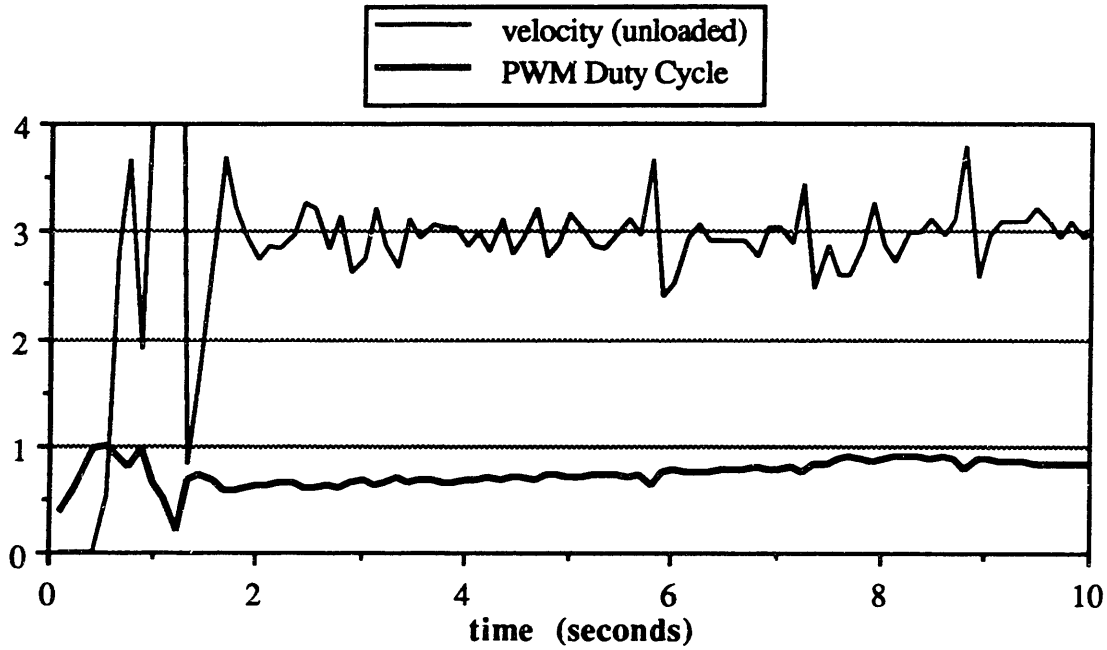


Figure 3.3.20 (a): P–D Velocity Servo ($K_p = 1.0$, $K_d = 0.1$)
3 deg/sec step response, unloaded operation

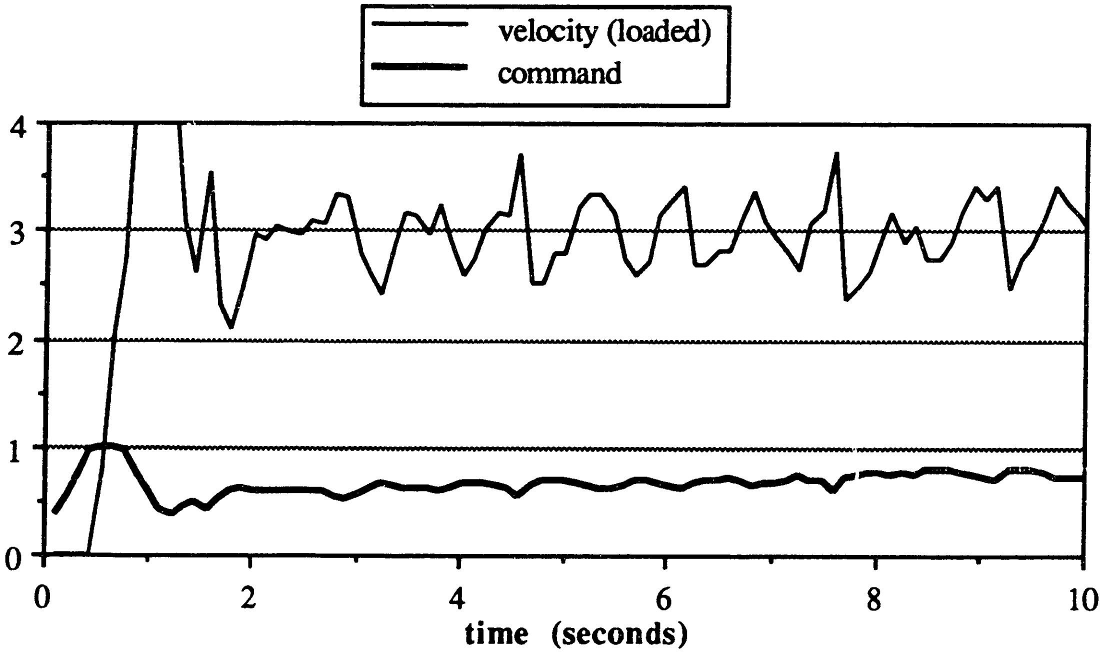


Figure 3.3.20 (b): P–D Velocity Servo ($K_p = 1.0$, $K_d = 0.1$)
3 deg/sec step response, loaded operation

As the figures show, the P-D servo performs fairly well. Overshoot is much less than the P servo and response time is better than either of the two servos. A small amount of steady-state error, however, exists in the step responses. This may be attributed to limit cycling inherent in the 4-bit PWM signal.

Comparison of the three velocity servo schemes revealed that the P-D velocity servo exhibits the best performance characteristics. As a result, this controller was chosen to serve as the joint servocontroller for all manipulator control schemes requiring joint velocity control.

3.4. SPAM Manipulator Hardware

This section describes the physical hardware of the SPAM manipulator arm. At the time of this writing, the two-link serial arm has been fully assembled and tested. Both arm joints, elbow pitch and shoulder pitch, were implemented using the R-PHAD-A design. The three-link arm, however, has not been completed due to time constraints and incomplete construction of the shoulder yaw joint (R-PHAD-B design). The following sections detail the design of peripheral joint hardware, modular joint interfaces and manipulator power systems.

3.4.1. Neutrally Buoyant Arm Booms

The arm booms in the SPAM manipulator provide rigid, straight-line connections between adjacent revolute joints. Each boom, independent of length, is neutrally buoyant to meet space simulation requirements. Additionally, the booms contain electrical and pneumatic control circuitry to drive the PHAD joints.

Since the baseline specifications called for joint-to-joint connections of various distances, booms were built in lengths of 6, 12, and 25 feet. Each boom was fabricated using two seamless aluminum tubes (6061-T3) concentrically aligned. The inner tube in this design, which is thin-walled (0.03125") and pressurized, provides the flotation necessary to achieve neutral buoyancy. The outer tube, with a thicker wall (0.125") than the inner tube, provides boom rigidity and strength. A schematic of this configuration is shown below in Figure 3.4.1.

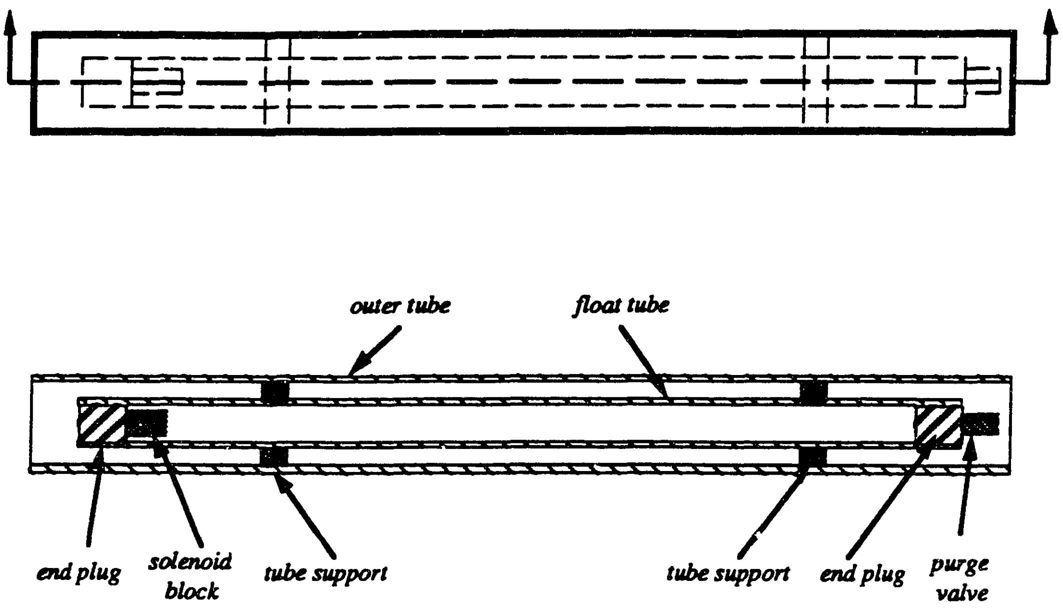


Figure 3.4.1: Arm Boom Tube Configuration

The inner tube (referred to as the "float" tube) is sealed on both ends with a PVC plug and 'O'-rings. The tube contains two Koganei solenoids and electrical hookups for the four Laketown solenoids used by the R-PHAD joints. Pressure and electrical lines are fed through connectors mounted to one of the PVC end plugs. During joint operation, air purged from the R-PHAD cylinders is vented into the float tube via the Koganei solenoids. Excess pressure is vented through a Plastomatic purge valve mounted in the other end plug. Photographs of the two PVC end plugs are shown in Figure 3.4.2.

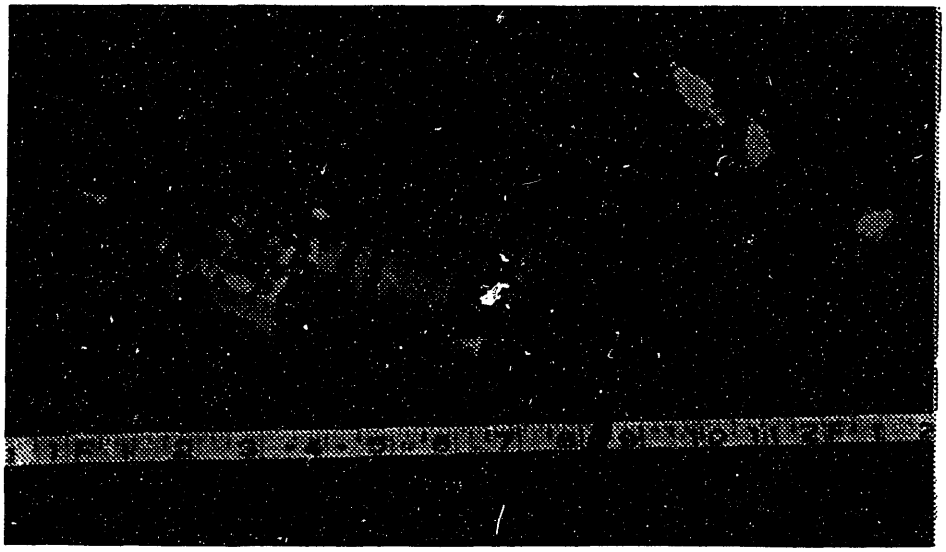


Figure 3.4.2 (a): End plug with Koganei solenoids

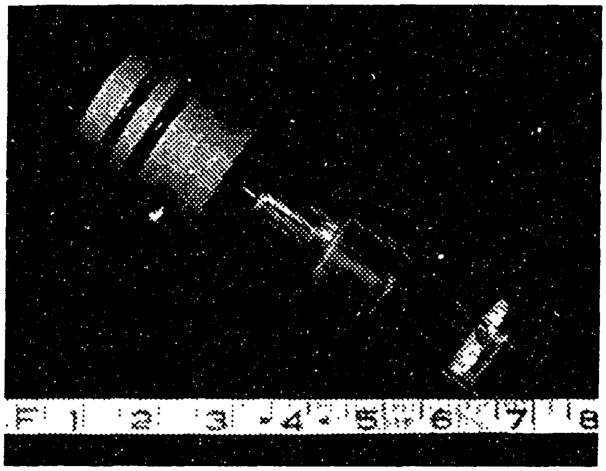


Figure 3.4.2 (b): End plug with purge valve

Approximate sizing of float tubes was based on buoyancy calculations using the densities of air, water, and aluminum listed in Table 3.1. Although each arm boom should ideally have zero net buoyancy, conservative calculations resulted in positive buoyancy figures. The resulting tube lengths and associated buoyancies are shown in Table 3.2.

Table 3.1 Assumed material densities

Material	Density (lb/cu in)
Air	0.00004
Aluminum (6061-T4)	0.98000
Water	0.03613

Table 3.2: Arm Boom Parameters

Boom Size (ft)	Outer Tube (ft)	Inner Tube (ft)	Net Buoyancy (lb)
6	6	5	1
12	12	10	5
24	24	20	14

It should be noted that positive buoyancy was not considered to be a problem for neutral buoyancy operation since compensation is easily achieved with balancing weights.

3.4.2. Joint/Arm Boom Interface

To facilitate change out of arm booms and reconfiguration of the manipulator arm, the R-PHAD joint and arm boom interface was designed to be modular and uncomplicated. Since the R-PHAD pneumatic and electric hookup is contained in the arm boom float tube, pressure and electrical lines were designed to utilize simple connector hardware. Specifically, CPC quick disconnect pressure fittings, Amphenol 165 series connectors, and ITT Cannon Sure-Seal connectors were used.

Mechanical coupling of the joint and arm boom was constructed as a slide fitting, with the arm boom outer tube sliding over a "joint collar" (shown in Figure 3.4.3). The coupling arrangement is shown in Figure 3.4.4.

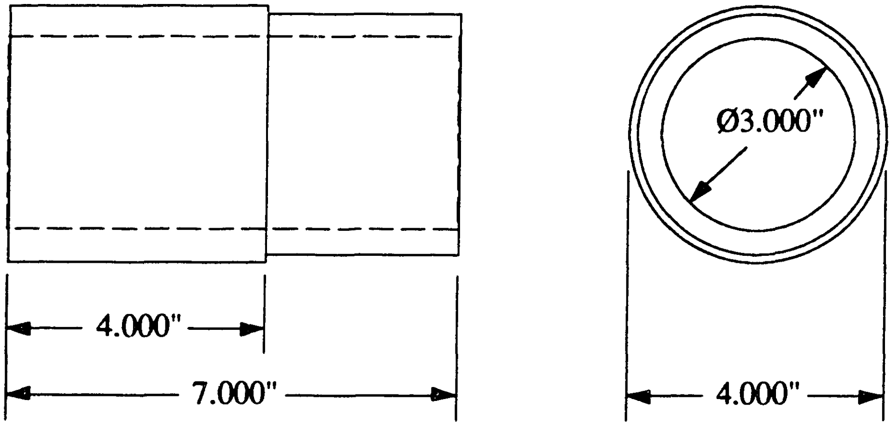
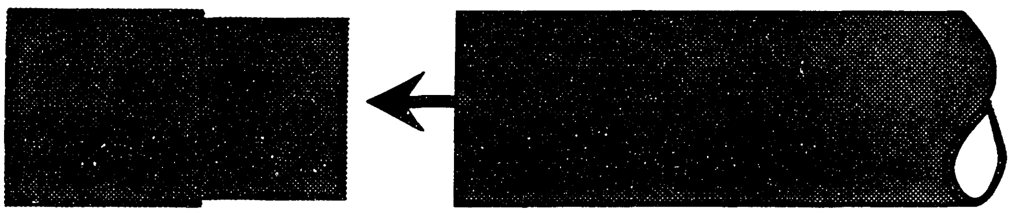


Figure 3.4.3: Joint Collar



Joint Collar *Arm Boom*

Figure 3.4.4: Joint/Arm Tube Interface

3.4.3. R-PHAD-A Power System

Each R-PHAD-A joint in the manipulator arm operates with a combination of electrical signals, compressed air, and water flow. Electrical signals generated by the *Electronic Control System* (ECS), described in the following section, operate the PHAD solenoid valves. Opening an air solenoid valve supplies compressed air to the PHAD pneumatic cylinders. Toggling the water solenoids provides controlled flow of water (from the surrounding environment) through the PHAD hydraulic cylinders. The resulting PHAD piston rod extension drives the R-PHAD-A 5-bar linkage, producing rotary joint motion.

As discussed in section 3.3.1, the compressed air for the SPAM manipulator arm is supplied from scuba tanks regulated through a Scubapro MK100 first-stage regulator. The 125-140 p.s.i. high pressure air from the regulator is fed from the water surface through the primary supply line (Clippard 1/8" urethane tubing) to the solenoid block end plug contained in each arm boom's float tube. Passing through a bulkhead connector, the compressed air is routed to the two Koganei solenoid supply ports. A secondary line, tapped off the primary supply line, is reduced to +2 lbs. over ambient pressure with a Go, Inc. regulator. This line is used to pressurize the sensor (optical encoder) on each joint. When the Koganeis are opened, the high pressure output is channeled through CPC quick disconnect fittings, filling the PHAD air cylinders. Closing the Koganeis allows cylinder to purging, sending the compressed air back through the float tube and out a check valve.

A schematic of the R-PHAD-A power system is provided in Figure 3.4.5:

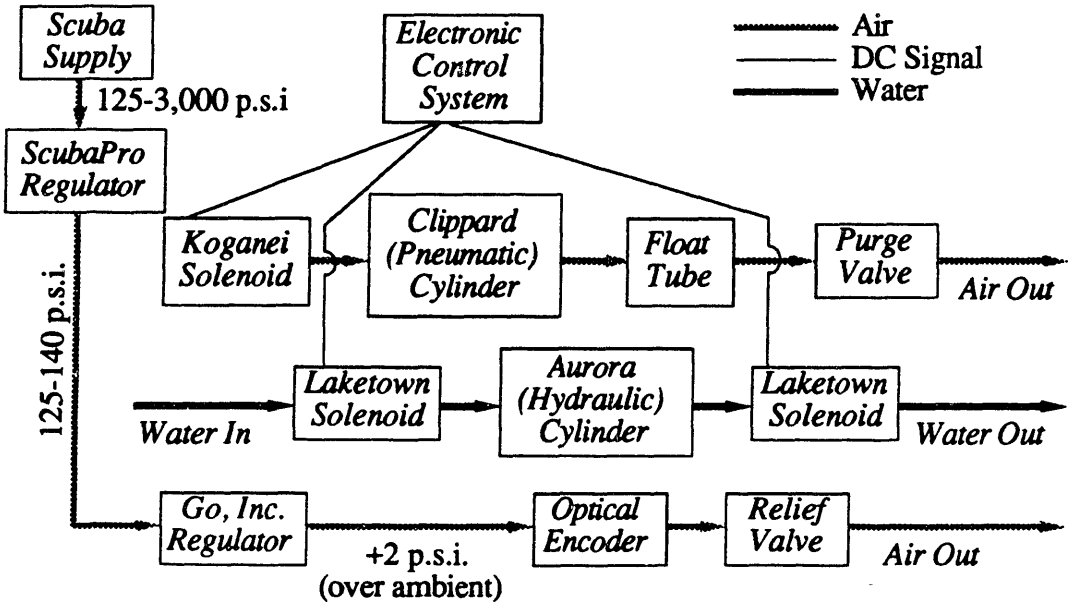


Figure 3.4.5: R-PHAD-A Power System

3.4.4. Electronic Control System

Actuator commands from joint servocontrollers are transformed into electric signals via the SPAM Electronic Control System (ECS). The ECS consists of two primary subsystems, computer input/output (I/O) and power switching. Computer I/O is handled through circuitry installed on a Prototype Expansion Card connected to the IBM PC bus. Encoder quadrature signals are decoded with HP HCTL-2000 counter/decoder chips. Solenoid commands, generated from software controllers, are output through an Intel 8255 and 4-bit PWM generator circuit. A schematic of this circuitry is given Appendix A.

Power switching circuitry is implemented on the Manipulator Hookup Board (MHB) shown in Figure 3.4.6. The MHB is wired to the computer I/O board and transforms switching commands from TTL levels to the 12 VDC level expected by R-PHAD solenoids. The MHB circuitry utilizes TIP-31 and MJ11016 npn-type transistors and is shown schematically in Appendix A. LED's mounted on the MHB chassis provides visual indication of solenoid state.

The complete ECS is shown schematically in Figure 3.4.7.

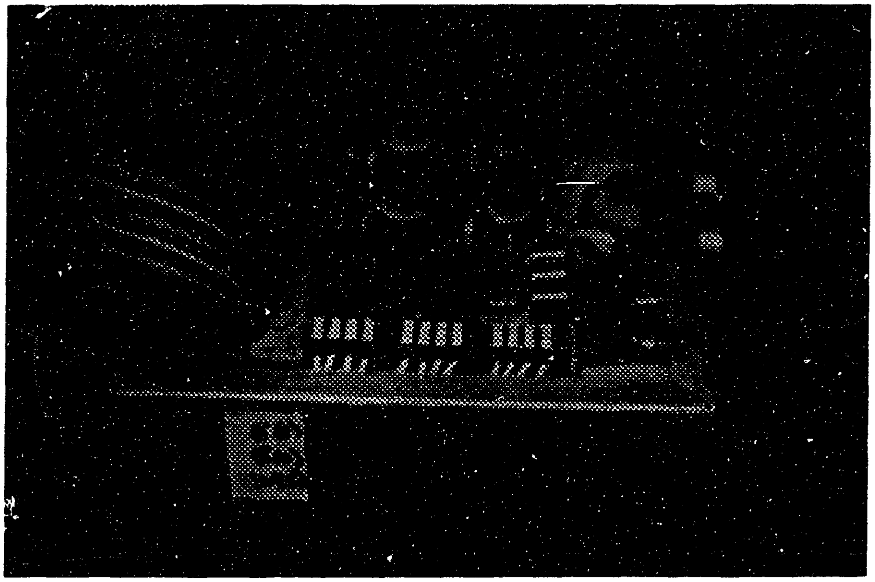


Figure 3.4.6: Manipulator Hookup Board (MHB)

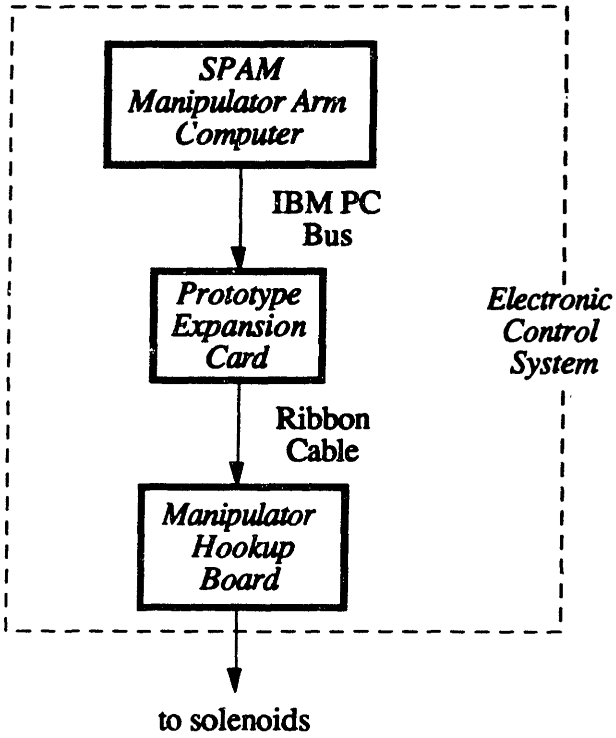


Figure 3.4.7: SPAM Electronic Control System (schematic)

3.5. 2-link Manipulator Testing & Results

The 2-link manipulator was constructed by assembling two R-PHAD-A joints and two six foot arm booms together. All testing of the manipulator was conducted in neutral buoyancy. To facilitate ease of handling and assembly, the arm was mounted to a vertical support stand. Since the 2-link manipulator is a planar arm, all tip movement was constrained to the *X*-*Z* plane of the base frame. Figure 3.5.1 illustrates the vertically mounted 2-link arm with frames. The completed planar arm is shown in Figure 3.5.2.

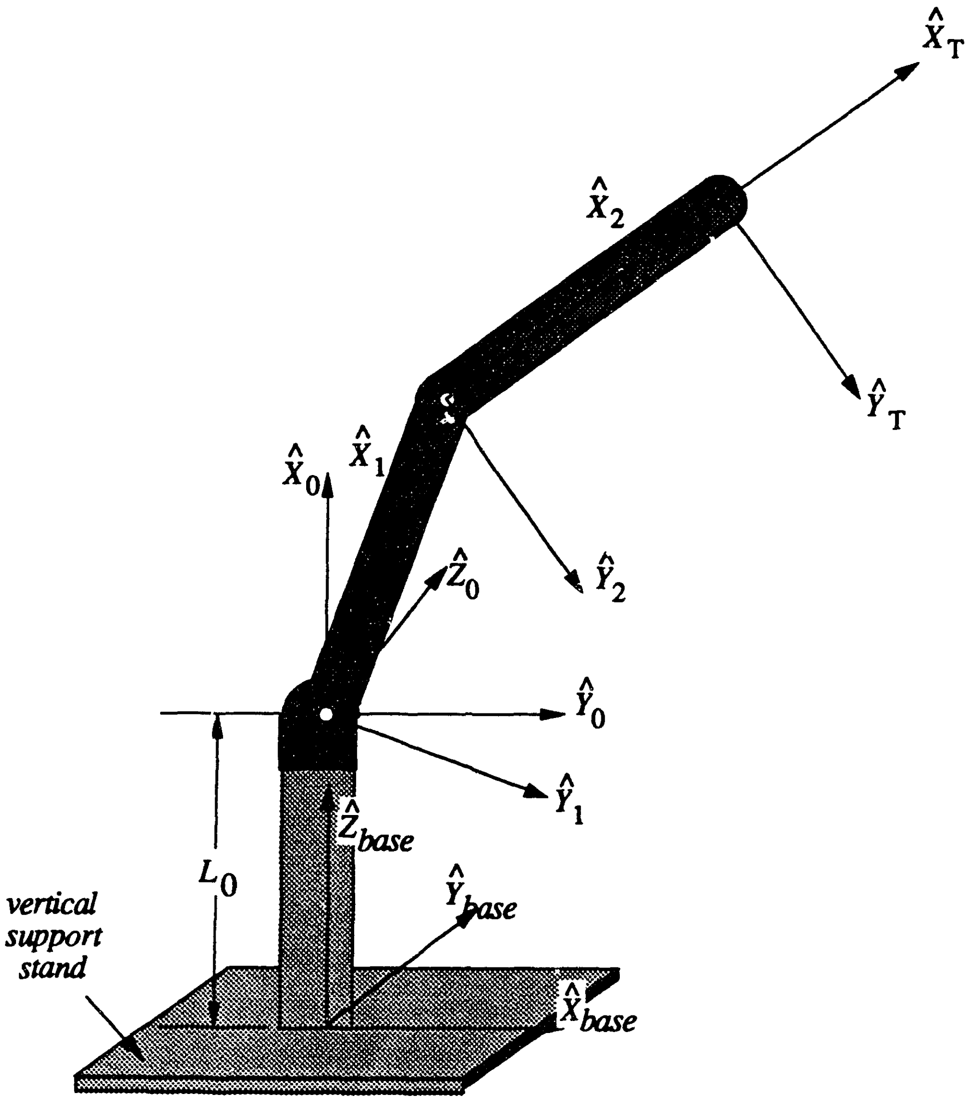


Figure 3.5.1. 2-link arm frames (vertical test mounting)

Figure 3.5.2: SPAM 2-link planar arm

In this configuration, tip commands specified in the base frame {B} must be transformed to the manipulator {0} frame using the mapping:

$$ {}^{B}_{0}T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -L_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.35) $$

3.5.1 Open-Loop Control Testing

Open-loop testing of the arm verified that two R-PHAD-A joints could be operated simultaneously. The forward kinematics mapping was validated by comparing calculated tip position with measured (physical) quantities. Since this method of manipulator control has limited usefulness, open-loop operation was brief.

3.5.2 Closed-Loop Position Control Testing

Closed-loop tip position control testing was performed using the 2-link manipulator inverse kinematics mapping to transform desired tip commands in {base} to joint positions. P-D servos were used to control the position of each R-PHAD-A joint. For both tests described below, both shoulder and elbow joints started at zero degrees; so that the arm was initially colinear with the {base} *Z*-axis.

In the first test, the Cartesian tip command of {L_2, 0, L_0+L_1} was specified. Geometrically speaking, this tip position corresponds to an elbow angle of 90° and a shoulder angle of 0°:

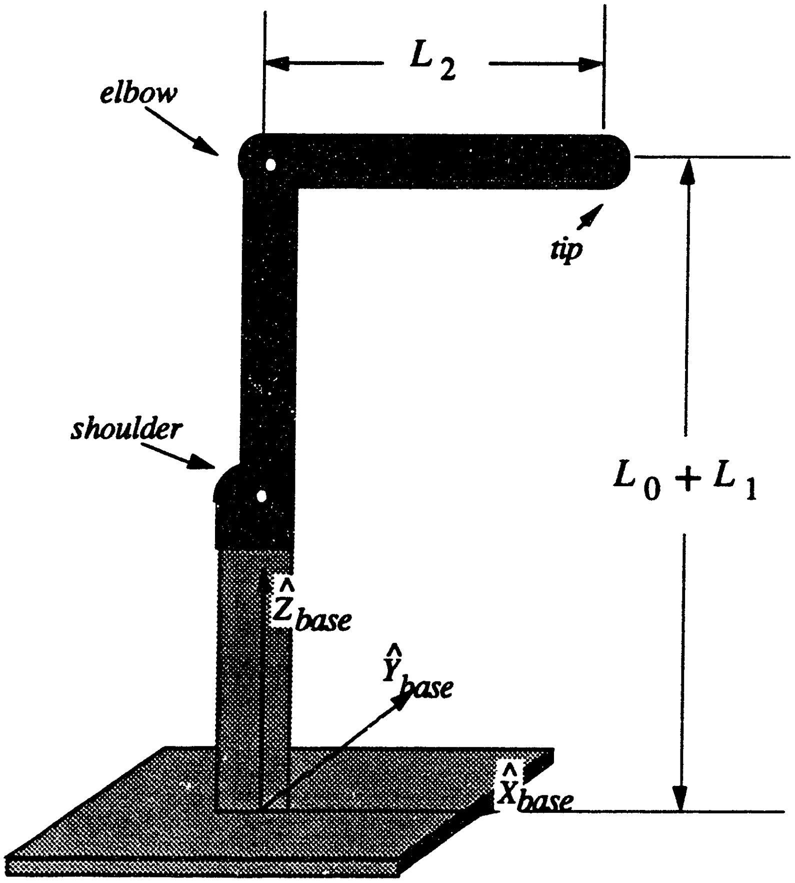


Figure 3.5.2: 2-link arm geometry for {L_2, 0, L_0+L_1} command

Execution of this command resulted in the correct joint angles, and hence, the desired tip position was achieved. Figure 3.5.3 shows the action of the P-D joint servos on elbow and shoulder position:

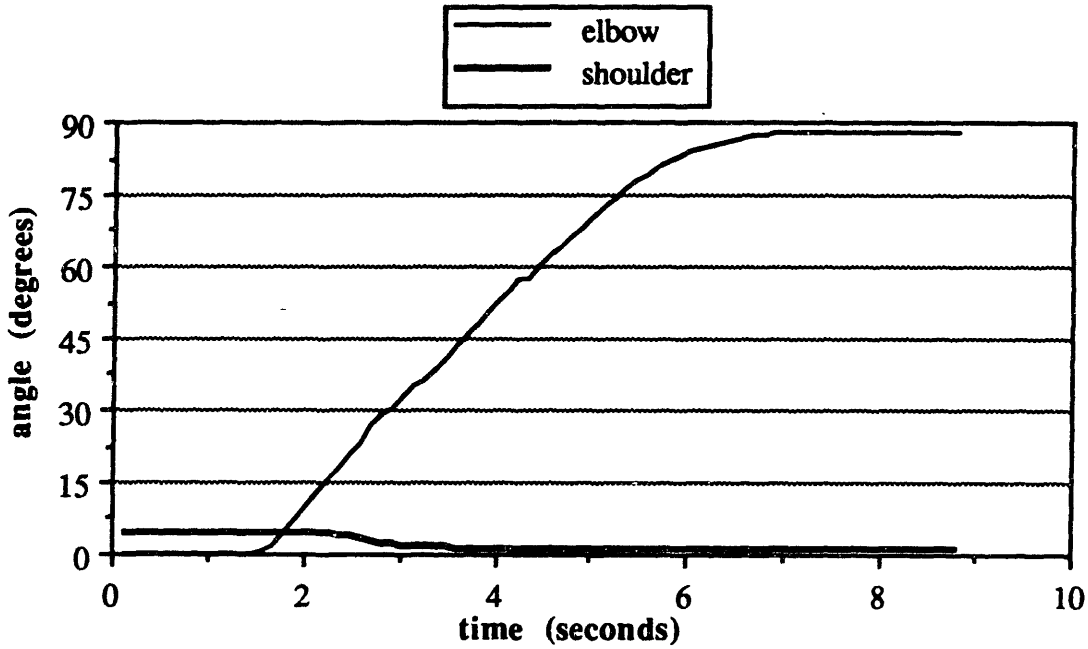


Figure 3.5.3: Joint angles resulting from $\{L_2, 0, L_0+L_1\}$ command

For the second test, the Cartesian tip command of $\{L_2+0.707L_1, 0, L_0+0.707L_1\}$ was specified. Geometrically speaking, this position corresponds to both joints at 45°.

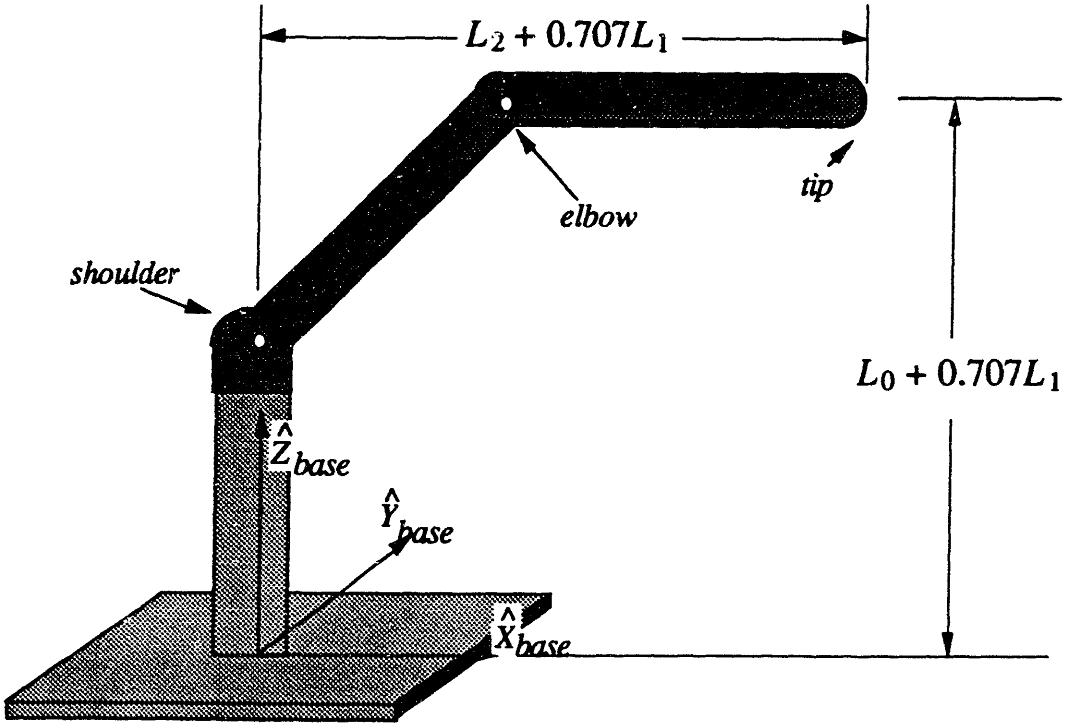


Figure 3.5.4: 2-link arm geometry for $\{L_2+0.707L_1, 0, L_0+0.707L_1\}$ command

From this command, both joints were correctly driven to 45 degrees as shown below in Figure 3.5.5:

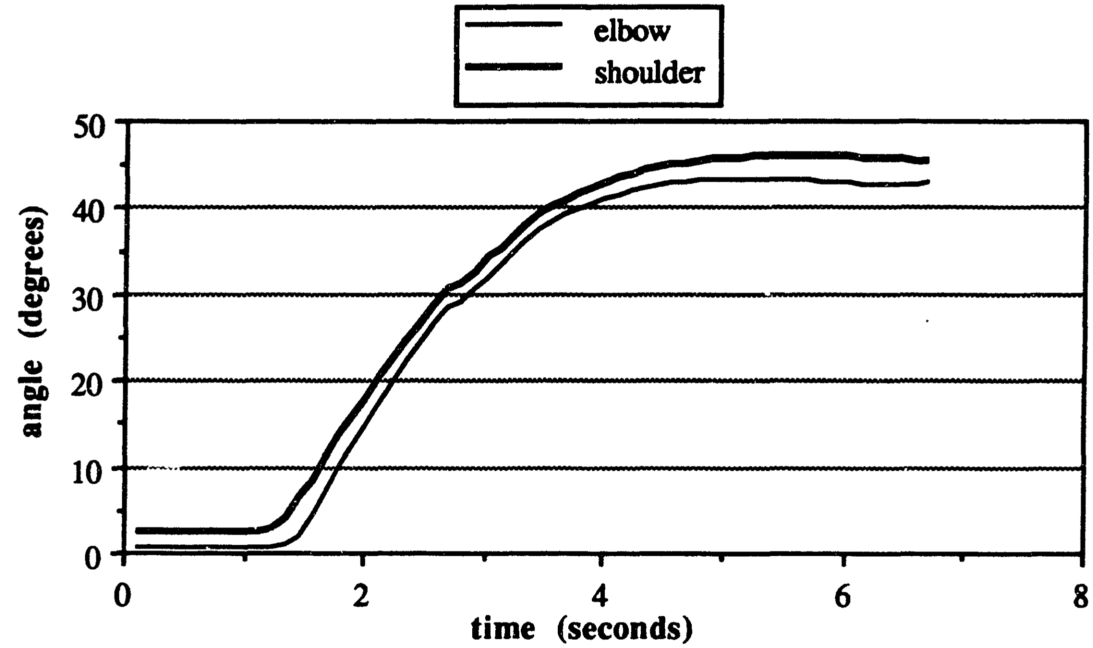


Figure 3.5.4: Joint angles from {L_2+0.707L_1, 0, L_0+0.707L_1} command

As a result of these tests, it was concluded that the closed-loop tip position control scheme could be operated successfully given tip commands in the reachable workspace.

Chapter 4
Stewart Platform

The task of developing the SPAM Stewart Platform component was divided into three phases. In the first phase, research was focused on acquiring knowledge of general Stewart Platform theory. This involved study of parallel-link chains, kinematics and control schemes. The second phase was concerned with the design of a Stewart Platform to meet baseline design specifications. Geometric design and kinematic modeling for this phase was performed with computer assistance. Finally, the Stewart Platform hardware was designed and constructed. This phase involved joint and linear actuator design, underwater sensor development, and implementation of a microprocessor system.

4.1 The Stewart Platform

The *Stewart Platform* is fundamentally described as a platform mechanism with 6 degrees-of-freedom. It has been suggested for numerous applications including motion based flight simulation, machining, and most recently, robotic manipulation. Though the Stewart Platform has been the subject of a great deal of analysis and study, the exact origin of the device remains unclear. A survey of literature reveals that the mechanism is most commonly attributed to D. Stewart's paper published by the Institution of Mechanical Engineers (Stewart, 1965). In this paper, Stewart proposes a platform mechanism for use as a helicopter flight simulator. Stewart's platform, however, is remarkably similiar to at least two other devices: Peterson and Cappel's *Hexapod* system, which was patented in the United States in 1965; and Gough's *Universal tyre test machine*, designed in 1949 and operational in 1954. In fact, the platform mechanism described in this chapter most closely resembles this latter device. The mechanism, however, is referred to as a "Stewart Platform" to maintain consistency with recent papers.

A common Stewart Platform implementation is shown in Figure 4.1.1. The mechanism is a space truss with an upper and lower plate and six links in an octahedral arrangement. The six mounting points on both plates are arranged in semi-regular hexagons. The Stewart Platform is classified as a six-degree-of-freedom platform mechanism of type 6-SPS, where S and P denote spherical and prismatic joints. (Yang, D.C.H., and Lee, T.W., 1984).

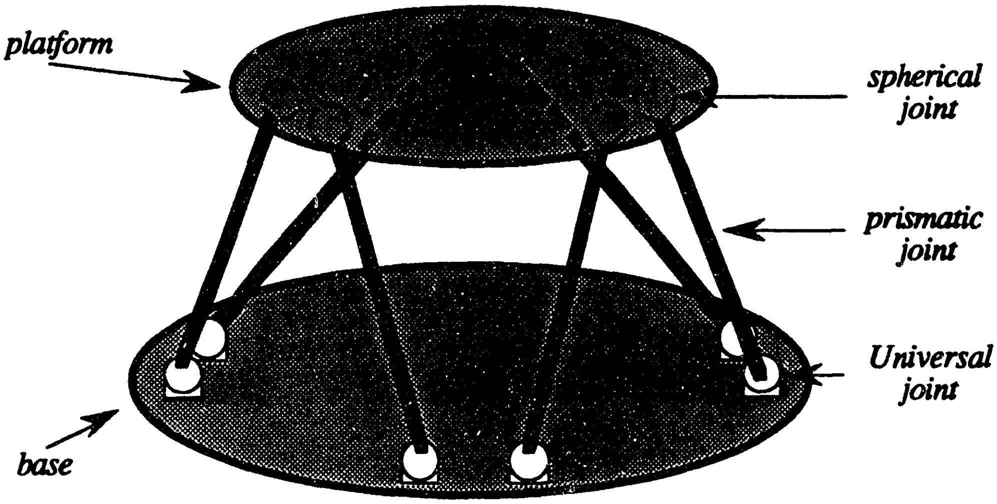


Figure 4.1.1: Common Stewart Platform Implementation

The Stewart Platform is a parallel-link or closed kinematic chain device. As discussed in section 2.2.1, parallel-link devices offer numerous advantages over serial-link or open kinematic chain devices including higher strength, force capability, and positioning accuracy. There are, however, also several disadvantages inherent in parallel-link mechanisms (Ismail, 1984). Specifically,

- Workspace is restricted because of parallel-links. There is difficulty in reaching around corners and/or into small spaces.
- Maneuverability and range of motion is much smaller than an open kinematic chain, given equivalent amounts of hardware.
- Near singular points, manipulator loading may result in excessive tensile/compressive actuator stresses.

In spite of these difficulties, a Stewart Platform manipulator "would be particularly feasible in applications where dynamic loading is severe and yet the demand on workspace and maneuverability is low" (Yang, D.C.H., and Lee, T.W., 1984).

It is important to note, however, that there is a significant difference in the mathematics between open and closed chain manipulators. For the former, the forward kinematic transform (from joint space to Cartesian space) is straightforward and the inverse kinematics (from Cartesian space to joint space) solution is difficult. In the case of closed chain manipulators, however, the exact converse is true; the forward kinematics transform is difficult and the inverse kinematics transform is straightforward. The difference between the manipulators lies in the degree of solution difficulty. While open chain inverse kinematics involves simultaneous non-linear equations, the problem is frequently solvable in closed-form. Solution of forward kinematics for closed-chain manipulators, however, is a different matter entirely since the mapping is not globally well-behaved and is ill-defined (Ismail, 1984). As a result, a simple closed-form solution is not possible and numerical methods, discussed in section 4.1.2.2, must be used.

4.1.1 Geometric Analysis

The general Stewart Platform has two bodies connected by six links which may vary in length. The lower body (in whatever coordinate system is used as reference) is called the *base* and the upper body the *platform*. The assignment of these names is, of course, completely arbitrary since the system is topologically symmetrical. Each of the six links has one end located in the base and the other end in the platform. Though the location of the six mounting points in either base or platform is not restricted, not all configurations result in controllable mechanisms. In each of the bodies, a coordinate frame is located; {B} denotes the base and {P} the platform. The location of mounting points in each body are then described as vectors to these coordinate frames. Figure 4.1.2 illustrates the general Stewart Platform (Fichter, 1986).

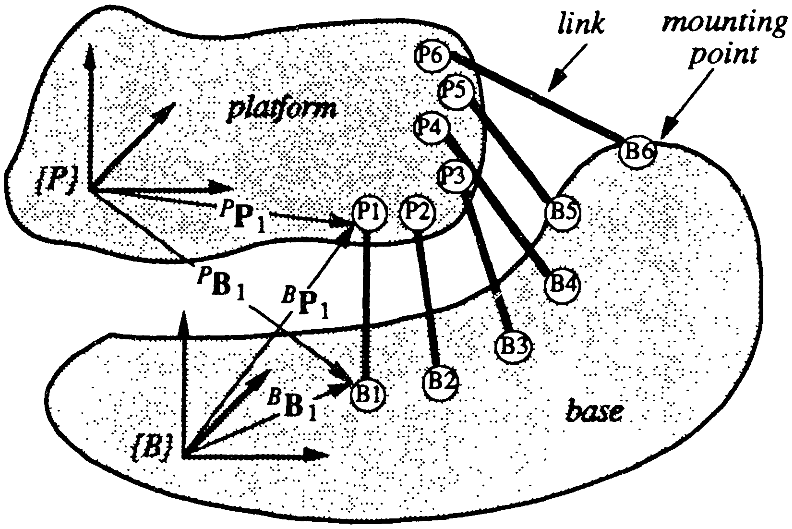
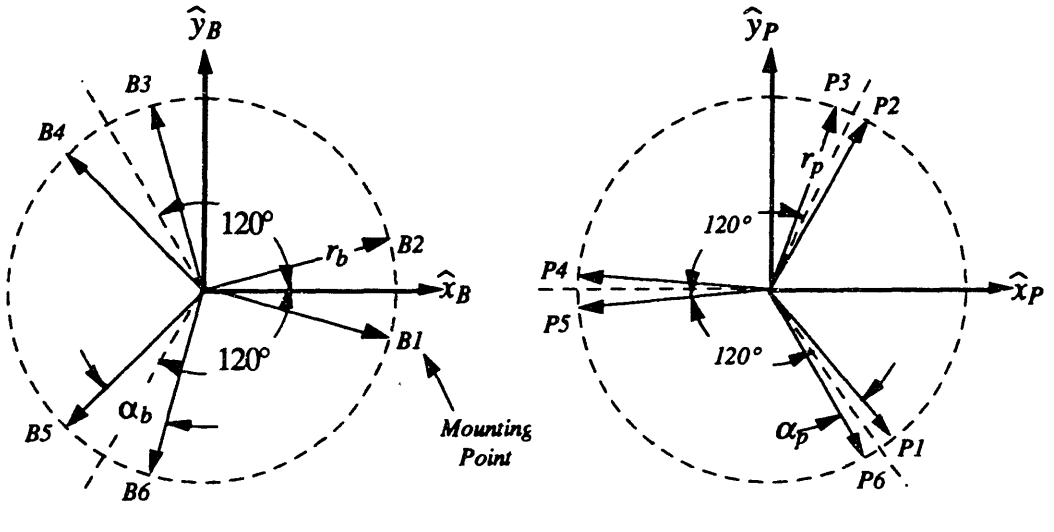


Figure 4.1.2: General Stewart Platform

In practice, Stewart Platforms are not completely arbitrary arrangements, but rather are implemented as geometrically symmetrical mechanisms due to mechanical design constraints. As shown previously in Figure 4.1, the most common configuration uses six links mounted in semi-regular hexagons on two plates. The resulting symmetrical Stewart Platform appears spatially as an octahedron. Moreover, since the vertices of the semi-regular hexagons may be inscribed in a circle, the geometry of both the base and platform plates can be described using only 4 parameters as shown in Figure 4.1.3. It should be understood that for this paper only this configuration is considered to be of interest and that all subsequent discussion implicitly refers to this particular type of Stewart Platform.



(a) (b)

Figure 4.1.3: Symmetrical Stewart Platform Geometry
(a) Base Plate, (b) Platform Plate

where:

r_b	=	base plate radius
r_p	=	platform plate radius
α_b	=	base mounting angle
α_p	=	platform mounting angle

It is interesting to note that this mechanism configuration is exactly the same as Gough's *Universal Tyre Test Machine*. The device presented in Stewart's paper, though similiar, uses a different arrangement of links.

A Stewart Platform with this octahedral arrangement has a large number of internal degrees-of-freedom. Each link contains a prismatic joint with one degree-of-freedom. This linear joint is attached to the base plate with a two degree-of-freedom Universal joint and to the top platform with a three degree-of-freedom spherical joint. This gives six degrees-of-freedom per link. Since there are six links connecting the base plate and top platform, the Stewart Platform has a total of 36 internal degrees-of-freedom. Table 4.1 (Ismail, 1984) presents a summary of these degrees-of-freedom.

Table 4.1: Stewart Platform Internal Degrees-of-Freedom

Joint Type	joint D-O-F	# of joints	D-O-F
prismatic	1	6	6
Universal	2	6	12
spherical	3	6	18
		18 total	36 total

The above appears to indicate that the Stewart Platform has an excessive number of degrees-of-freedom. In actuality, the free degrees-of-freedom are sharply limited by the base and platform plate. To demonstrate this, it is worthwhile to conduct a spatial mobility analysis of the joint structure. Grodzinski and M'Ewen's general mechanism degree-of-freedom equation has the form:

$$F = \lambda(n-1) - \sum_{i}^{g} (6-f) \qquad (4.1)$$

where:

F = effective degree-of-freedom of mechanism

λ = degree-of-freedom of operational space (6 for spatial motion, 3 for planar)

n = number of members

g = number of joints

f = degrees-of-freedom in joints

The Stewart Platform operates spatially so that λ is equal to 6. The base plate and top platform are each single members. Combined with six links, this gives a total of 8 members. If the six prismatic joints are locked in place, we have six Universal and six spherical free joints giving $g = 12$. Applying (4.1) gives:

$$F = 6(8-1) - \left[\underbrace{\sum_{1}^{6} (6-0)}_{\substack{\textit{PRISMATIC}\\ \textit{JOINTS}}} + \underbrace{\sum_{1}^{6} (6-2)}_{\substack{\textit{UNIVERSAL}\\ \textit{JOINTS}}} + \underbrace{\sum_{1}^{6} (6-3)}_{\substack{\textit{SPHERICAL}\\ \textit{JOINTS}}} \right] \qquad (4.2)$$
$$= 0$$

If the six prismatic joints are free to move we have six additional members, giving n=14. Applying (4.1) in this case gives:

$$F = 6(14-1) - \left[\underbrace{\sum_{1}^{6} (6-1)}_{\substack{\textit{PRISMATIC}\\ \textit{JOINTS}}} + \underbrace{\sum_{1}^{6} (6-2)}_{\substack{\textit{UNIVERSAL}\\ \textit{JOINTS}}} + \underbrace{\sum_{1}^{6} (6-3)}_{\substack{\textit{SPHERICAL}\\ \textit{JOINTS}}} \right] \qquad (4.3)$$
$$= 6$$

In summation, if the six prismatic joints are locked in a fixed positon, then the Stewart Platform will be fully constrained to a fixed location and orientation. More importantly though, if the six prismatic joints can be controlled it will be possible to position the Stewart Platform in a full six degrees-of-freedom.

4.1.2 Kinematics

Due to semantical ambiguity of parallel-link device mappings, it is important to clarify the nomenclature before discussing kinematic specifics. Many authors, for example, consider the transformation from given Cartesian coordinates to link extensions as forward kinematics since the mapping is well defined. The inverse kinematics problem, therefore, is concerned with the ill-defined task of determining Cartesian frame coordinates given a set of link extensions. This convention parallels serial-link manipulator nomenclature since the forward kinematics are straightforward and the inverse kinematics difficult for those mechanisms. This approach, however, is less than optimal because it makes the naming (and direction) of mappings device dependent. An alternate approach is to *always* define forward kinematics as the mapping from joint space to reference coordinate space, and inverse kinematics as the converse. For this paper, this latter nomenclature has been adopted for two reasons. First, the direction of mappings is always well defined. For example, discussion of forward kinematics *always* implies a transformation from *natural* manipulator coordinates to *imposed* reference coordinates. Secondly, and more importantly, it is *consistent* since the convention operates *independent of the device* under consideration.

In the following two sections, the forward and inverse kinematics of the symmetrical Stewart Platform are presented. Since inverse kinematics is the natural mapping for the Stewart Platform, this transform is presented first. As will be shown, application of inverse kinematics is straightforward and algebraic. The associated forward kinematics mapping is discussed in section 4.1.2.2. Since this transformation is highly non-linear and cannot be solved in closed-form, an iterative numerical solution approach is utilized.

4.1.2.1 Inverse Kinematics

The natural coordinates for a parallel-link device having *n* links is the set of *n* joint variables. For the Stewart Platform, this is the set of six prismatic joint lengths. Expressed as a vector in joint space:

$$\mathbf{L} = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \\ l_5 \\ l_6 \end{bmatrix} \tag{4.4}$$

The reference coordinate frame, as with the manipulator arm, is Cartesian and is composed of an orthonormal position frame and Roll-Pitch-Yaw Euler angles:

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{4.5}$$

The inverse kinematics mapping, G, is thus:

$$\mathrm{G} : \mathbf{X} \Rightarrow \mathbf{L} \tag{4.6}$$

For the symmetric Stewart Platform (using conventions of section 4.1.1) the base frame {B} and platform frame {P} are located at the centers of the base plate and platform plate respectively. The location of the base mount points are then expressed as vectors in {B} and are numbered in a counterclockwise fashion about the positive $\widehat{z_B}$ axis. Similarly, the location of the platform mount points are expressed as vectors in {P} and numbered about the positive $\hat{z}_P$ axis. Link vectors, representing actuator linkages, connect the base and platform mount points. Numbering is such that for actuator i, link vector $\mathbf{L}_i$ connects base mount vector $\mathbf{B}_i$ to platform mount vector $\mathbf{P}_i$. A summary of these frame and vector assignments is presented in Figure 4.1.4.

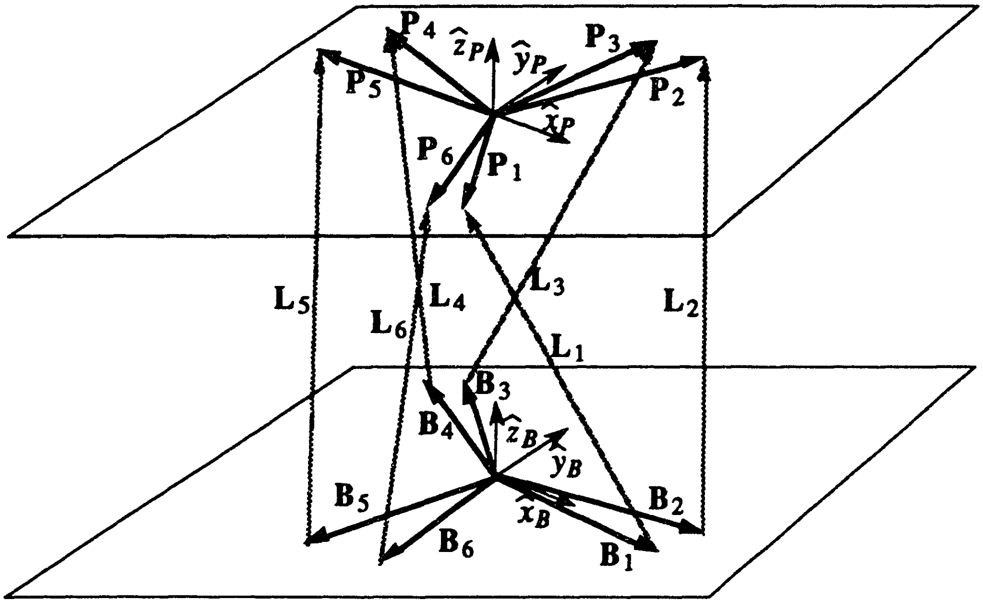


Figure 4.1.4: Frame and Vector Assignments
Symmetrical Stewart Platform

In terms of the symmetrical Stewart Platform parameters given in section 4.1.1, the base mount vectors in $\{B\}$ are:

$$\begin{aligned}
\mathbf{B}_1^T &= \begin{bmatrix} r_b\cos\left(\frac{\alpha_b}{2}\right) & -r_b\sin\left(\frac{\alpha_b}{2}\right) & 0 \end{bmatrix}^T \\
\mathbf{B}_2^T &= \begin{bmatrix} r_b\cos\left(\frac{\alpha_b}{2}\right) & r_b\sin\left(\frac{\alpha_b}{2}\right) & 0 \end{bmatrix}^T \\
\mathbf{B}_3^T &= \begin{bmatrix} -r_b\cos\left(60°+\frac{\alpha_b}{2}\right) & r_b\sin\left(60°+\frac{\alpha_b}{2}\right) & 0 \end{bmatrix}^T \\
\mathbf{B}_4^T &= \begin{bmatrix} -r_b\cos\left(60°-\frac{\alpha_b}{2}\right) & r_b\sin\left(60°-\frac{\alpha_b}{2}\right) & 0 \end{bmatrix}^T \\
\mathbf{B}_5^T &= \begin{bmatrix} -r_b\cos\left(60°-\frac{\alpha_b}{2}\right) & -r_b\sin\left(60°-\frac{\alpha_b}{2}\right) & 0 \end{bmatrix}^T \\
\mathbf{B}_6^T &= \begin{bmatrix} -r_b\cos\left(60°+\frac{\alpha_b}{2}\right) & -r_b\sin\left(60°+\frac{\alpha_b}{2}\right) & 0 \end{bmatrix}^T
\end{aligned} \qquad (4.7)$$

Similarly, the platform mount vectors in $\{P\}$ are:

$$\begin{aligned}
\mathbf{P}_1^T &= \left[\begin{array}{ccc} r_p\cos\left(60^\circ-\frac{\alpha_p}{2}\right) & -r_p\sin\left(60^\circ-\frac{\alpha_p}{2}\right) & 0 \end{array}\right]^T \\
\mathbf{P}_2^T &= \left[\begin{array}{ccc} r_p\cos\left(60^\circ+\frac{\alpha_p}{2}\right) & -r_p\sin\left(60^\circ+\frac{\alpha_p}{2}\right) & 0 \end{array}\right]^T \\
\mathbf{P}_3^T &= \left[\begin{array}{ccc} r_p\cos\left(60^\circ+\frac{\alpha_p}{2}\right) & r_p\sin\left(60^\circ+\frac{\alpha_p}{2}\right) & 0 \end{array}\right]^T \\
\mathbf{P}_4^T &= \left[\begin{array}{ccc} r_p\cos\left(60^\circ-\frac{\alpha_p}{2}\right) & r_p\sin\left(60^\circ-\frac{\alpha_p}{2}\right) & 0 \end{array}\right]^T \\
\mathbf{P}_5^T &= \left[\begin{array}{ccc} -r_p\cos\left(\frac{\alpha_p}{2}\right) & -r_p\sin\left(\frac{\alpha_p}{2}\right) & 0 \end{array}\right]^T \\
\mathbf{P}_6^T &= \left[\begin{array}{ccc} -r_p\cos\left(\frac{\alpha_p}{2}\right) & r_p\sin\left(\frac{\alpha_p}{2}\right) & 0 \end{array}\right]^T
\end{aligned} \qquad (4.8)$$

If the desired position of the platform frame is specified with respect to the base frame, ${}^{B}\mathrm{X}_{PORG} = (x_d, y_d, z_d)$, then the vector relationship for link vector i is:

$$\mathbf{L}_i = \mathbf{P}_i + {}^{B}\mathbf{X}_{PORG} - \mathbf{B}_i \qquad (4.9)$$

This relationship is illustrated in Figure 4.1.5.

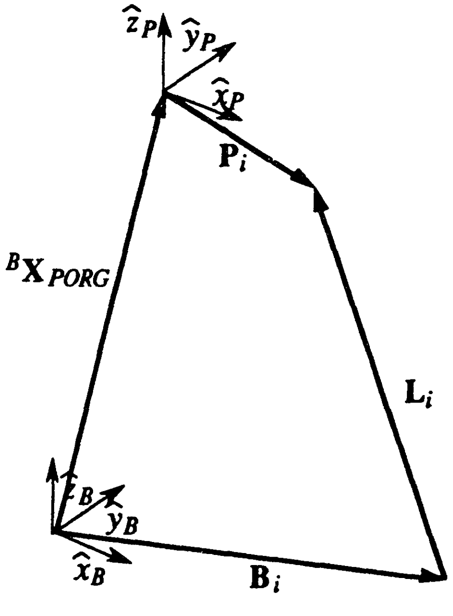


Figure 4.1.5: Actuator Vector Relationships

Expressing (4.9) as vectors in {B} gives the desired inverse kinematics relationship for each actuator:

$${}^{B}\mathbf{L}_i = {}^{B}_{P}\mathbf{R}\,{}^{P}\mathbf{P}_i + {}^{B}\mathbf{X}_{PORG} - {}^{B}\mathbf{B}_i \qquad (4.10)$$

where ${}^{B}_{P}\mathbf{R}$ is the 3-2-1 (Yaw-Pitch-Roll) Euler rotation transformation using desired platform orientation (ϕ, θ, ψ) specified in the base frame:

$$ {}^{B}_{P}\mathrm{R} = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{4.11} $$

The actual prismatic joint extension, l_i, is simply the 2-norm of (4.10):

$$ l_i = |\mathbf{L}_i| = \sqrt{\mathbf{L}_{i_x} + \mathbf{L}_{i_y} + \mathbf{L}_{i_z}} \tag{4.12} $$

4.1.2.2 Forward Kinematics

The forward kinematic mapping, F, for the Stewart Platform maps the six prismatic joint lengths to Cartesian position and orientation of the platform. The mapping is thus:

$$ \mathrm{F} : \mathbf{L} \Rightarrow \mathbf{X} \tag{4.13} $$

This map F is ill-defined and is not well behaved since multiple solutions for **X** may exist for a specified **L**. It has been shown, however, that locally F exists and is differentiable, provided that the local state is non-singular. (Ismail, 1984).

If one considers the forward kinematics mapping as purely the inverse of the mapping G (the inverse kinematics) then determination of F simply requires inverting (4.10) or (4.12) simultaneously with all six prismatic joints. Typically, however, only l_i and not $\mathbf{L}_i$ is known so that (4.12) must be used. Unfortunately, the inversion of (4.12) is quite difficult in a closed-form sense since the problem involves solving six simultaneous non-linear equations for the six unknown components of **X** ($x, y, z, \phi, \theta, \psi$). One means of solving this problem is given in (Dieudonne, 1972) and involves the application of the iterative Newton-Raphson method.

To apply Newton-Raphson to the inverse kinematics problem, begin by defining a vector function as the difference between calculated and actual (measured) length of joint *i*:

$$ \mathrm{f_i}(\mathbf{X}) = \mathbf{L}_i^T\mathbf{L}_i - |\mathbf{L}_i|_a^2 \tag{4.14} $$

where $|\mathbf{L}_i|_a^2$ is the actual (measured) length of joint i. Substituting (4.10) into the above yields:

$$\mathrm{f}_i(\mathbf{X}) = \left({}^B_P\mathbf{R}\,{}^P\mathbf{P}_i + {}^B\mathbf{X}_{PORG} - {}^B\mathbf{B}_i\right)^T\left({}^B_P\mathbf{R}\,{}^P\mathbf{P}_i + {}^B\mathbf{X}_{PORG} - {}^B\mathbf{B}_i\right) - \left|\mathbf{L}_{ia}\right|^2 \tag{4.15}$$

Expanding (4.15) and taking partial derivatives with respect to components of **X** ($x, y, z, \phi, \theta, \psi$) gives the following set of equations:

$$\begin{aligned}
\frac{\partial \mathrm{f}_i(\mathbf{X})}{\partial x} &= 2\left(x + {}^P\mathbf{P}_{i_x}\mathbf{T}_{11} + {}^P\mathbf{P}_{i_y}\mathbf{T}_{12} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{13} - {}^B\mathbf{B}_{i_x}\right) \\
\frac{\partial \mathrm{f}_i(\mathbf{X})}{\partial y} &= 2\left(y + {}^P\mathbf{P}_{i_x}\mathbf{T}_{21} + {}^P\mathbf{P}_{i_y}\mathbf{T}_{22} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{23} - {}^B\mathbf{B}_{i_y}\right) \\
\frac{\partial \mathrm{f}_i(\mathbf{X})}{\partial z} &= 2\left(z + {}^P\mathbf{P}_{i_x}\mathbf{T}_{31} + {}^P\mathbf{P}_{i_y}\mathbf{T}_{32} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{33} - {}^B\mathbf{B}_{i_z}\right) \\
\frac{\partial \mathrm{f}_i(\mathbf{X})}{\partial \phi} &= 2\left(x - {}^B\mathbf{B}_{i_x}\right)\left({}^P\mathbf{P}_{i_y}\mathbf{T}_{13} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{12}\right) \\
&\quad + 2\left(y - {}^B\mathbf{B}_{i_y}\right)\left({}^P\mathbf{P}_{i_y}\mathbf{T}_{23} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{22}\right) \\
&\quad + 2\left(z - {}^B\mathbf{B}_{i_z}\right)\left({}^P\mathbf{P}_{i_y}\mathbf{T}_{33} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{32}\right) \\
\frac{\partial \mathrm{f}_i(\mathbf{X})}{\partial \theta} &= 2\left(x - {}^B\mathbf{B}_{i_x}\right)\left(-{}^P\mathbf{P}_{i_x}\sin\theta\cos\psi + {}^P\mathbf{P}_{i_y}\sin\phi\cos\theta\cos\psi\right. \\
&\quad \left. + {}^P\mathbf{P}_{i_z}\cos\phi\cos\theta\cos\psi\right) + 2\left(y - {}^B\mathbf{B}_{i_y}\right)\left(-{}^P\mathbf{P}_{i_x}\sin\theta\sin\psi\right. \\
&\quad \left. + {}^P\mathbf{P}_{i_y}\sin\phi\cos\theta\sin\psi + {}^P\mathbf{P}_{i_z}\cos\phi\cos\theta\sin\psi\right) \\
&\quad - 2\left(z - {}^B\mathbf{B}_{i_z}\right)\left({}^P\mathbf{P}_{i_x}\cos\theta + {}^P\mathbf{P}_{i_y}\sin\phi\sin\theta + {}^P\mathbf{P}_{i_z}\cos\phi\sin\theta\right) \\
\frac{\partial \mathrm{f}_i(\mathbf{X})}{\partial \phi} &= -2\left(x - {}^B\mathbf{B}_{i_x}\right)\left({}^P\mathbf{P}_{i_y}\mathbf{T}_{21} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{22} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{23}\right) \\
&\quad + 2\left(y - {}^B\mathbf{B}_{i_y}\right)\left({}^P\mathbf{P}_{i_y}\mathbf{T}_{11} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{12} + {}^P\mathbf{P}_{i_z}\mathbf{T}_{13}\right)
\end{aligned} \tag{4.16}$$

where $\mathbf{R}_{ij}$ are components of (4.11). The Newton-Raphson iteration formula for obtaining a solution for **X** is straightforward:

$$\begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix}_{n+1} = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix}_{n} - \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial \phi} & \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial \psi} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\partial f_6}{\partial x} & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{n}^{-1} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix}_{n} \qquad (4.17)$$

The complete forward kinematics solution is as follows:

1) Measure lengths of the joints (l_i).
2) Guess platform position and orientation (X_0).
3) Calculate (4.11), (4.15), and (4.16) for each joint.
4) Iterate on (4.17) until ($X_{n+1}-X_n$) meets a convergence criterion.

It should be noted that if forward kinematics has to be performed in real-time, (4.17) will be the limiting factor for computational speed. Since Newton-Raphson is an iterative technique, convergence of (4.17) requires a variable number of iterations, depending on local state gradients. Consequently, if the local state is near a singularity, convergence (which is not guaranteed) may require a large number of iterations. Furthermore, because (4.17) involves calculating the inverse of a 6x6 matrix, iteration speed will depend primarily on how efficiently the inversion operation can be performed. From an implementation perspective, the complete 6x6 matrix inversion is not necessary since back substitution using LU factorization will suffice. Finally, nominal convergence of (4.17) may be such that a small, fixed number of iterations will give sufficient accuracy for the platform state, X.

4.1.2.3 Kinematic Constraints

Forward and inverse kinematics aside, there is a topic which concerns all manipulators, but is particularly important to parallel-link manipulators. Specifically, this is the problem of kinematic constraints imposed by the manipulator hardware. For all manipulators, the fundamental constraint is joint position limitations (i.e. angular range or extension). Parallel-link devices, however, have the additional constraint of link interference. As a result, the symmetrical Stewart Platform has three kinematic constraints which limit the range of operation.

First, prismatic joint extension is restricted to a minimum and maximum length. As a result, not all **X** generated by the forward map F can physically be reached since all l_i may not be achievable. During operation of the Stewart Platform, this implies that the forward kinematic outputs (prismatic joint extensions) must be checked for validity. This constraint may be expressed algebraically as:

$$0 \leq (l_i - l_{min}) \leq l_{max} \tag{4.18}$$

Secondly, the prismatic joints are connected to the base and platform plates with spherical joints. In practical terms, no "spherical" joint is truly spherical since there are always limitations on joint range-of-motion. A ball-and-socket joint, for example, offers unrestricted rotation about one axis but is limited to a maximum angle around the other two axes. Other implementations, moreover, may have non-uniformly shaped workspaces. A Universal-joint attached to a swivel, for instance, has large angular capability, but only in specific orientations. In terms of the Stewart Platform, this means that not all platform **X** are achievable, even if the commanded prismatic joint extensions are physically possible.

Finally, the Stewart Platform is constrained by parallel-linkage interference. Specifically, since the device is a space truss with six links, changes in position and orientation of the platform with respect to the base will result in changes for all the connecting links. Consequently, the possibility of link interference exists, which makes certain base/platform configurations unachievable. Calculation of such configurations during real-time operation is a difficult problem due to the complexity of the Stewart Platform spatial geometry. One approach, which treats the links as solid cylinders and searches for spatial intersection, is given in (Stelman, 1988).

4.2 Stewart Platform Control

The problem of controlling the Stewart Platform offers many challenges to the control system engineer. Not only does the mechanism have six degrees of freedom requiring active control, but because of the parallel-linkage there are strong relationships between each link. Although it is possible to choose any subset of joints for control, providing that six degrees of platform freedom are maintained, the prismatic joint set is desirable for three reasons. First, this group contains the minimum number of joints (6) which produces full spatial mobility (as demonstrated in section 4.1.1). Secondly, since the joints are physically separated from one another, independent servo control may be used for each joint. Lastly, both the forward and inverse kinematic maps are concerned with link length, which is naturally controlled by varying prismatic joint extensions.

During the past decade, there have been a number of research efforts directed towards high-level control of a symmetrical Stewart Platform. Fichter of Oregon State University, for example, has been developing force control methods by deriving models of actuator loading and force output (Fichter, 1986). Additionally, research into variable admittance techniques has been pursued at M.I.T. by Dubowsky's group (Fresko, 1986, Ismail, 1988, and Stelman, 1988).

For both the O.S.U. and M.I.T. groups high-performance linear actuators were implemented for Stewart Platform operation. Zero-backlash lead screws driven by electric motors were chosen by the former and a fast hydraulic system by the latter. In the case of SPAM, however, such actuator systems are unsuitable for use in neutral buoyancy. Consequently, the linear actuator used for the SPAM Stewart Platform (described in section 4.3.3) is extremely modest from a performance perspective, especially in terms of achievable velocity and acceleration. The actuator, however, is capable of achieving fine positioning.

Due to this limitation on actuator performance, the decision was made to focus initial control system design efforts purely on position control. Though this simplifies the control task a great deal, the problem still involves controlling six degrees-of-freedom. In the following sections, two approaches to position control of the Stewart Platform are described. In the sequel, a controller based on inverse kinematics is proposed. In section 4.2.2. a controller utilizing forward kinematics is described.

4.2.1 Joint-Based Control

Contrary to serial-link manipulators, the Stewart Platform inverse kinematics mapping is the fundamental transform and is straightforward to compute. It seems natural, therefore, to attempt to utilize the inverse mapping for the position control problem. As described previously (section 4.1.2.1), the inverse kinematics transforms a desired platform state **X** to a set of link lengths **L**. Achieving these commanded lengths is accomplished by controlling the extension of each prismatic joint. If it is assumed that this joint control can be accomplished by actuator position servos, then the high-level control problem reduces to the task of specifying smooth trajectories. As we shall see, however, it is important to understand exactly what constitutes "smoothness".

Given the task of moving the Stewart Platform from the current position and orientation to another position and orientation, it is necessary to change the current link lengths to a different set. If the change in state is small, then it is reasonable to assume that the lengths of the links will not change radically. However, if at least one state component (either translational position or angular orientation) changes significantly or if the platform is near a singularity, then one may expect large jumps in link lengths. Consequently, the platform may move wildly through space, abruptly rotating and translating, as it transitions from the current to desired state. This type of behavior is, of course, extremely undesirable for a manipulator or end-effector involved in proximity operations! To prevent this from occuring, it is necessary to generate a trajectory of joint states such that the platform transitions smoothly between end-points.

Two methods of trajectory generation have been considered for the SPAM Stewart Platform. In both approaches, current and desired platform states are described in Cartesian space. A number of intermediate *way points* between the two end points are then calculated by interpolation. In the first method, the interpolation is performed in Cartesian space, resulting in a Cartesian trajectory which is converted to joint states via inverse kinematics. The advantage of this approach is that the platform state varies smoothly. For the second method, the end points are immediately converted to joint states and interpolation performed in joint space. This approach is characterized by constant actuator steps, but the resulting Cartesian trajectory may not be smooth. Since both methods result in continuous joint-space trajectories, it is left to operational testing to determine which approach is more desirable.

The inverse kinematic, or joint-based, controller for the Stewart Platform has the following characteristics:

- Desired platform state, specified in Cartesian space, is transformed into a trajectory in joint space.

- The joint space trajectory is used to command a sequence of joint states.

- The control of joint state is performed by independent joint position servos.

Figure 4.2.1 illustrates the joint-based controller in block-diagram form:

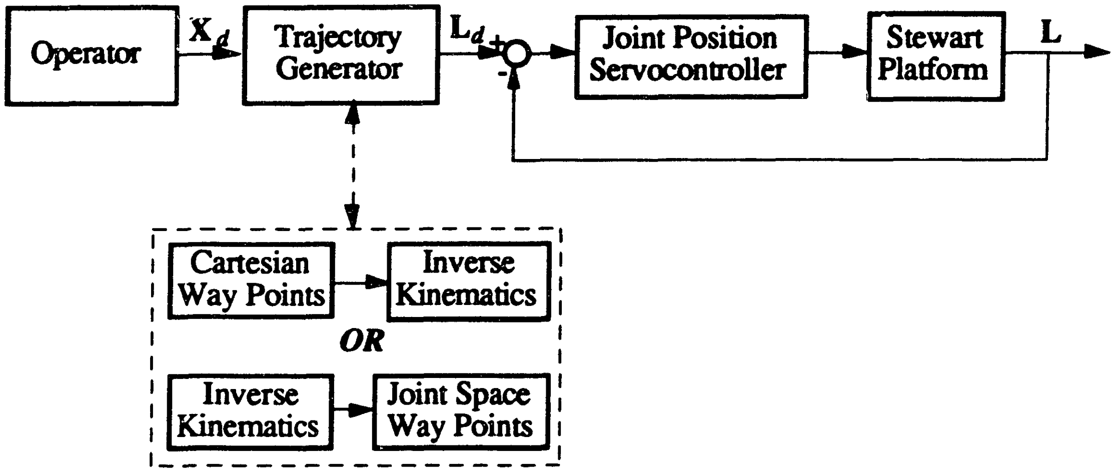


Figure 4.2.1: Stewart Platform Joint Based Control

The joint based controller is attractive because it requires relatively few computations. In fact, if trajectory generation is performed before platform movement is started, then the only calculations that need to be done in real-time are in the six joint position servo loops.

4.2.2 Cartesian-Based Control

Although the joint based controller has many desirable characteristics, it has one glaring deficiency. Specifically, there is no actual feedback of platform state in the control loop. Minimization of error, therefore, depends on trajectory efficacy and performance of the joint position servos. The obvious solution to this problem is to include platform state in the loop. This task, however, is complicated for two reasons. First, it is extremely difficult to instrument the platform with a six degree-of-freedom sensor, especially underwater. It is much easier to sense prismatic joint extension with a linear position sensor. Secondly, if only the link lengths are available (via joint extension sensors), then the platform state must be inferred using forward kinematics. As explained in section 4.1.2.2, however, the forward mapping is difficult to solve and numerical methods require significant amounts of computation.

In spite of these difficulties, a control scheme may be developed that accounts for platform state. The proposed Cartesian based controller utilizes inverse kinematics in the forward loop and Newton-Raphson forward kinematics in the feedback loop. Errors between commanded and current (estimated) position, expressed in Cartesian space, are transformed by the inverse kinematics to joint position errors. These errors, in turn, command the joint position servocontrollers. The resulting platform state is obtained by converting measured joint positions via forward kinematics. A block diagram summarizes this control scheme in Figure 4.2.2.

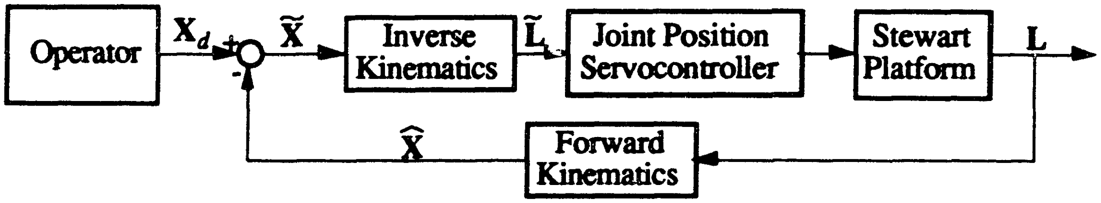


Figure 4.2.2: Stewart Platform Cartesian Based Controller

It is important to realize that the feedback platform state is only an estimate of the actual platform position and orientation because it is obtained with N-R forward kinematics. Since convergence of this numerical method is dependent on state gradients, precise state estimation may only be achievable with a large number of N-R iteration cycles. Testing of the N-R forward kinematics over a wide range of platform states is required to determine the minimum number of iterations for acceptable state precision. Alternatively, it may be possible to replace the numerical forward kinematics with some form of state observer such as a Kalman Filter. Implementation, however, would require modelling of the Stewart Platform and actuator dynamics.

4.3 Joint Design & Testing

In the following sections, design of joint hardware for the SPAM Stewart Platform is discussed. To assemble a symmetrical Stewart Platform, it is necessary to design and construct three types of joints. The final designs for prismatic, Universal, and spherical joints are given in the following three sections. To provide feedback of prismatic joint extension, a linear position sensor was developed for underwater operation. This sensor is described in section 4.3.4. Finally, servo control methods for the prismatic joint are presented in section 4.3.5.

4.3.1 PHAD Revisited

As discussed in Chapter 3, designing high-performance or high-precision actuators for neutral buoyancy operation is non-trivial. The task is not impossible, however, as demonstrated by the joints used in the SPAM manipulator arm. These joints utilize the PHAD actuator augmented with motion converting (linear-to-rotary) linkages. Since the basic PHAD is linear, it seems natural to utilize this actuator design for linear motion. Consequently, the Linear-PHAD (L-PHAD) prismatic joint was developed for use in the

Stewart Platform. Though the name "L-PHAD" is perhaps redundant, it was chosen to maintain consistency with the revolute joint (R-PHAD) name.

In sections 3.3.2 and 3.3.3, sizing of the R-PHAD cylinders and solenoid valves was performed by considering joint torque requirements. Since the joints were required to produce large torque levels to satisfy tip force specifications, large pneumatic and hydraulic cylinders were required. For the Stewart Platform, however, prismatic joint forces are directly applied to the platform plate at full strength. As a result, smaller cylinders and solenoids are sufficient to meet prismatic joint force production goals.

The final design of the prismatic L-PHAD joint utilizes two Clippard UDR-12-12 (3/4" bore) and one Clippard UDR-17-12 (1-1/16" bore) cylinders. Each of these cylinders has a 12 inch stroke, which is the longest length available for small class cylinders. The two pneumatic UDR-12-12 cylinders produce 110 lbs of force when driven by 125 p.s.i. compressed air. The UDR-17-12 cylinder was chosen for hydraulic use since it approximates the combined bore area of the two pneumatic cylinders. Similiar to the R-PHAD implementations, Koganei 110-4E1-F11 solenoid valves provide air flow control for the L-PHAD. Control of water flow, however, is performed with Kip 141010 Series 1 solenoid valves instead of Laketown valves. The Kips were selected because lower flow rate and pressure requirements permitted the usage of smaller valves. A schematic of the L-PHAD is given in Figure 4.3.1; detailed drawings are included in Appendix B. The assembled L-PHAD is pictured in Figure 4.3.2.

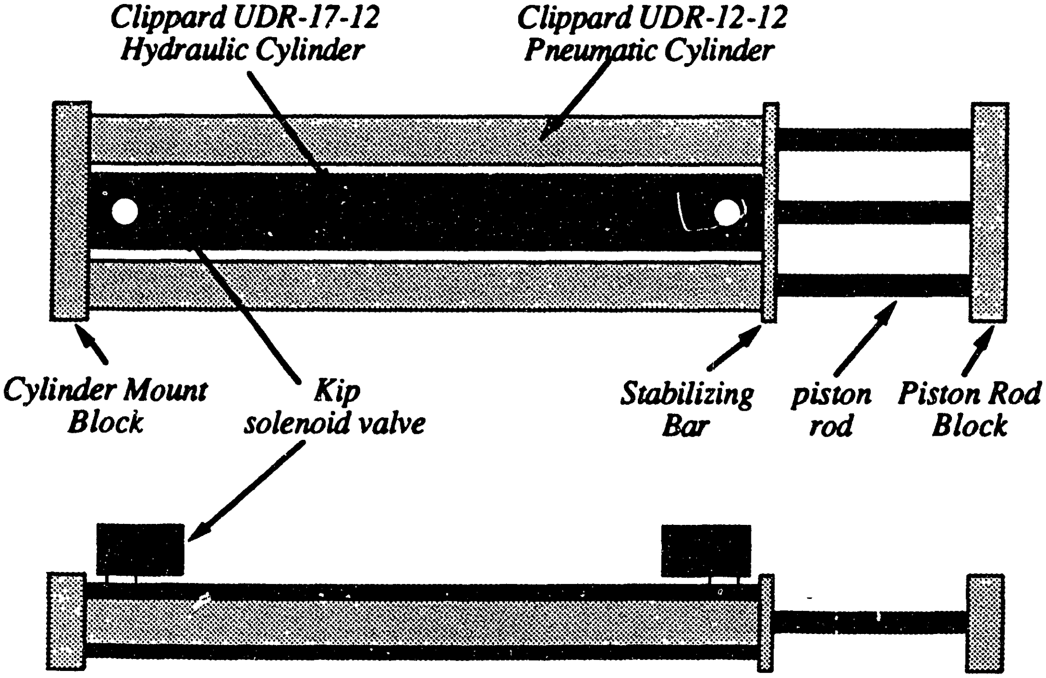


Figure 4.3.1: L-PHAD Design Schematic

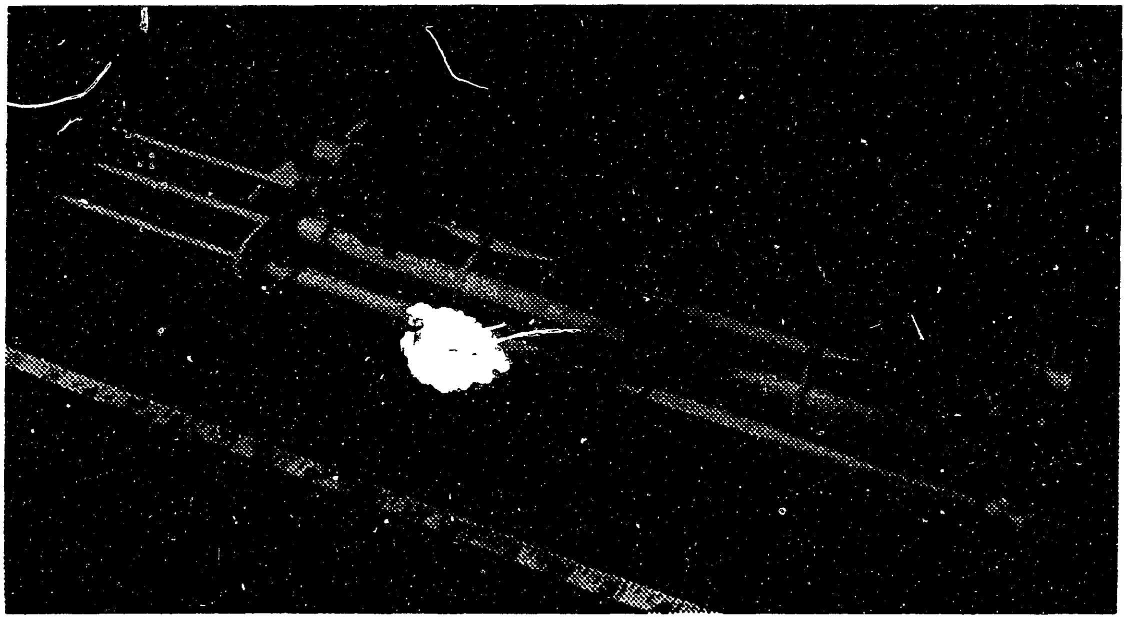


Figure 4.3.2: Assembled L-PHAD

4.3.2 Universal Joint

To provide the two degrees-of-freedom required by the Stewart Platform link base plate mounts, a simple universal joint was developed. This joint, shown schematically in Figure 4.3.3, incorporates a gimbal pivot and off-the-shelf hardware from PIC, Inc.

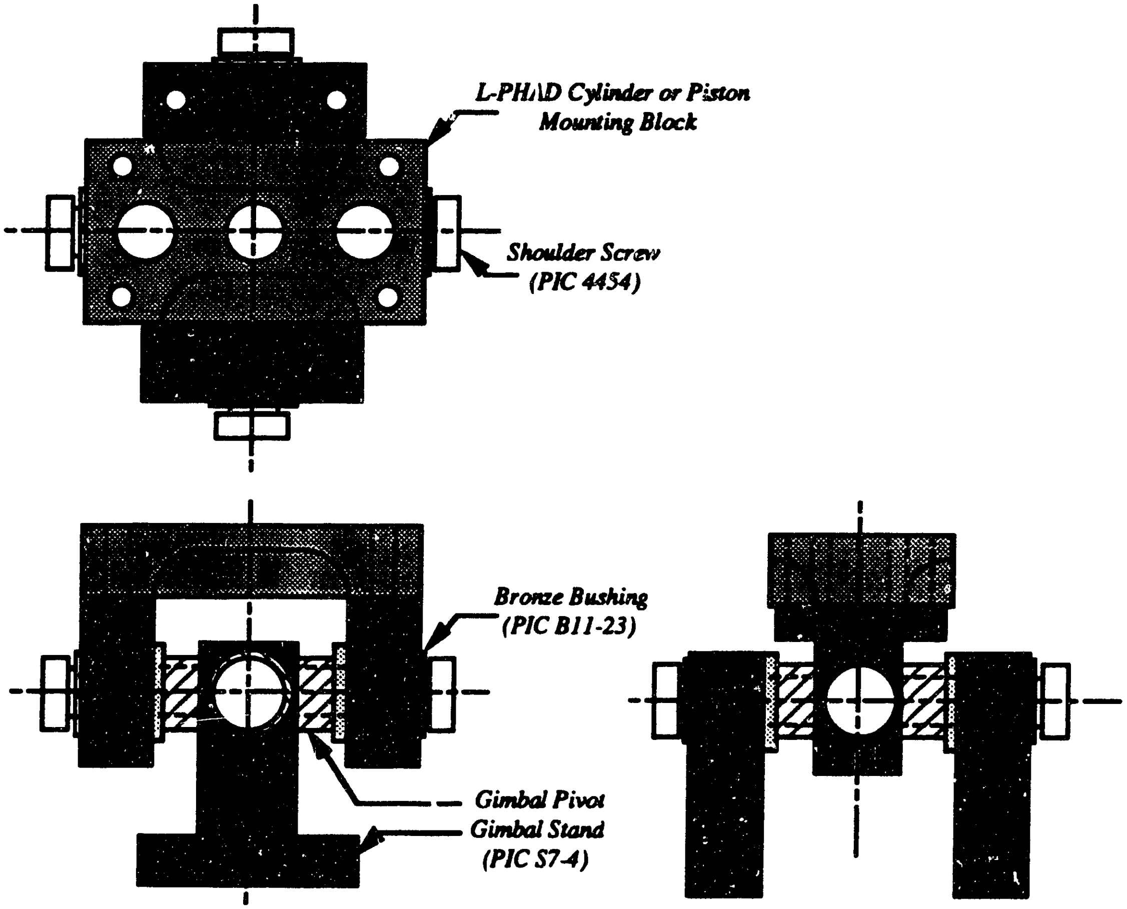


Figure 4.3.3: Universal Joint Design
schematic

The assembled Universal joint is pictured in Figure 4.3.4. Testing of this joint reveals that it has a uniform angular workspace, shown in Figure 4.3.5, with a maximum rolling angle of 55±2 degrees and maximum pitching of 40±2 degrees.

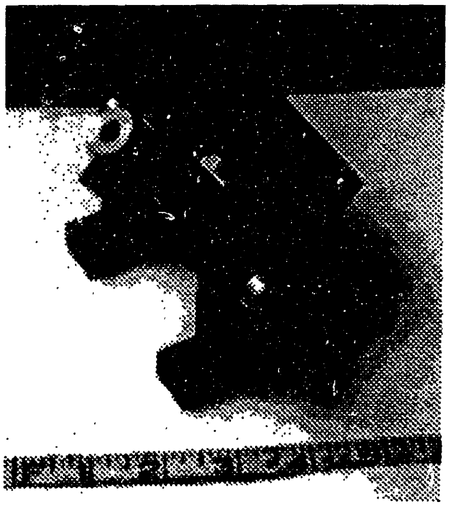


Figure 4.3.4: Assembled Universal Joint Design

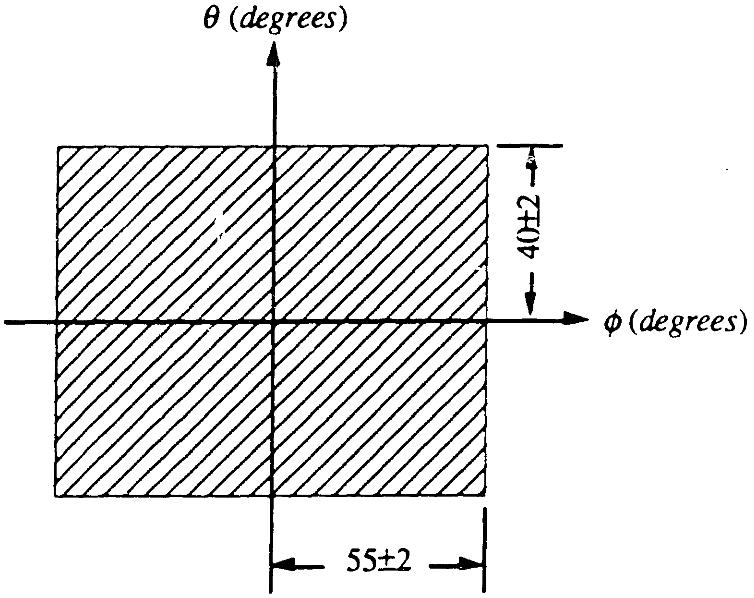


Figure 4.3.5: Universal Joint Workspace

4.3.3 Spherical Joint

The platform plate link mounting requires a three degree-of-freedom joint. True spherical joints, however, cannot realistically be implemented so that approximations are used in practice. Among the common spherical approximations, the ball-and-socket joint most closely mimics spherical joint workspace. Greater range-of-motion and better load capabilities may be achieved, however, with a properly designed gimbal arrangement. As a result, it was decided to approximate the spherical joint with a modified universal joint.

The final joint design, shown in Figure 4.3.6, augments the universal joint described previously with the addition of a swivel bearing. The assembled "spherical" joint is pictured in Figure 4.3.7.

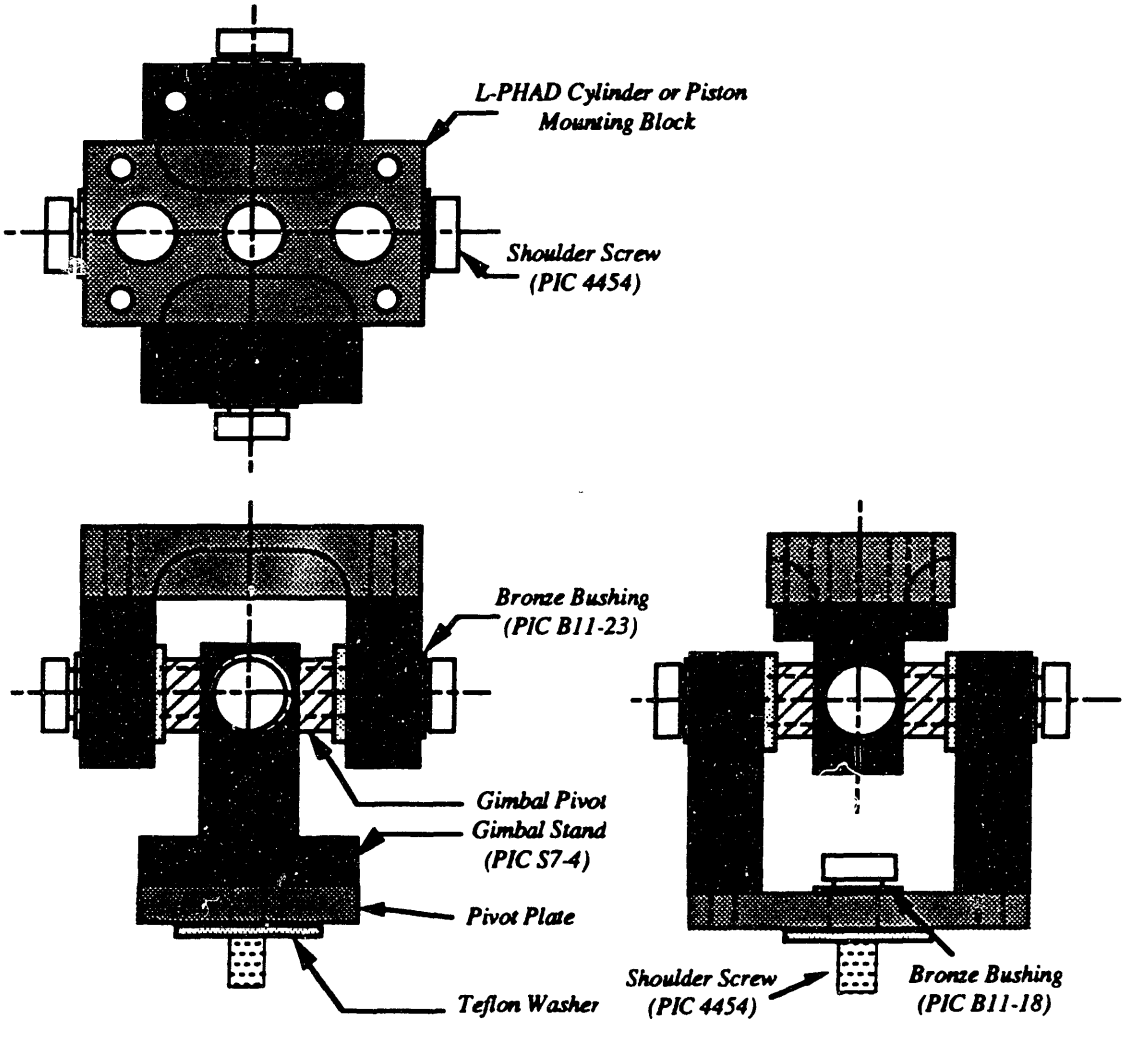


Figure 4.3.6: "Spherical" (U-Joint-plus-Swivel) Design
schematic

Figure 4.3.7: Assembled U-Joint-plus-Swivel Joint Design

Since this joint is a simple modification of the Universal joint, it has the same rolling and pitching angular workspace. The addition of the swivel, however, provides a full revolution of yawing. The combined workspace is shown below in Figure 4.3.8

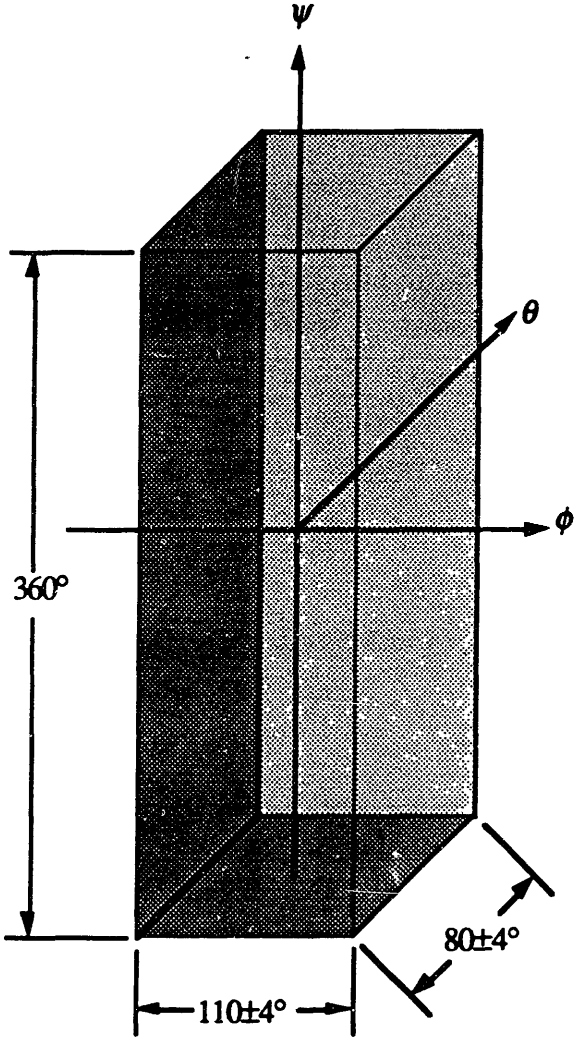


Figure 4.3.8: U-Joint-plus-Swivel Workspace

4.3.4 JELLO Sensor

To provide a measurement of L-PHAD prismatic joint extension, a position sensor was needed. Unfortunately, most off-the-shelf linear position sensors are unsuitable for underwater operation due to the difficulty of waterproofing linear electronic assemblies. Since rotary optical encoders can be modified for underwater operation, a number of designs attempted to use these sensors by converting linear to rotary motion. Unfortunately, these designs proved to be unsuccessful since the resulting sensors were too bulky, did not provide high precision or accuracy, or were too mechanically complex.

In the end, it was necessary to abandon rotary enconders and to focus efforts on developing a custom linear sensor. The *Joint Encoder with Localized Linear Optics* (JELLO) is the result of these efforts and is shown schematically in Figure 4.3.9 The assembled JELLO is shown in Figure 4.3.10.

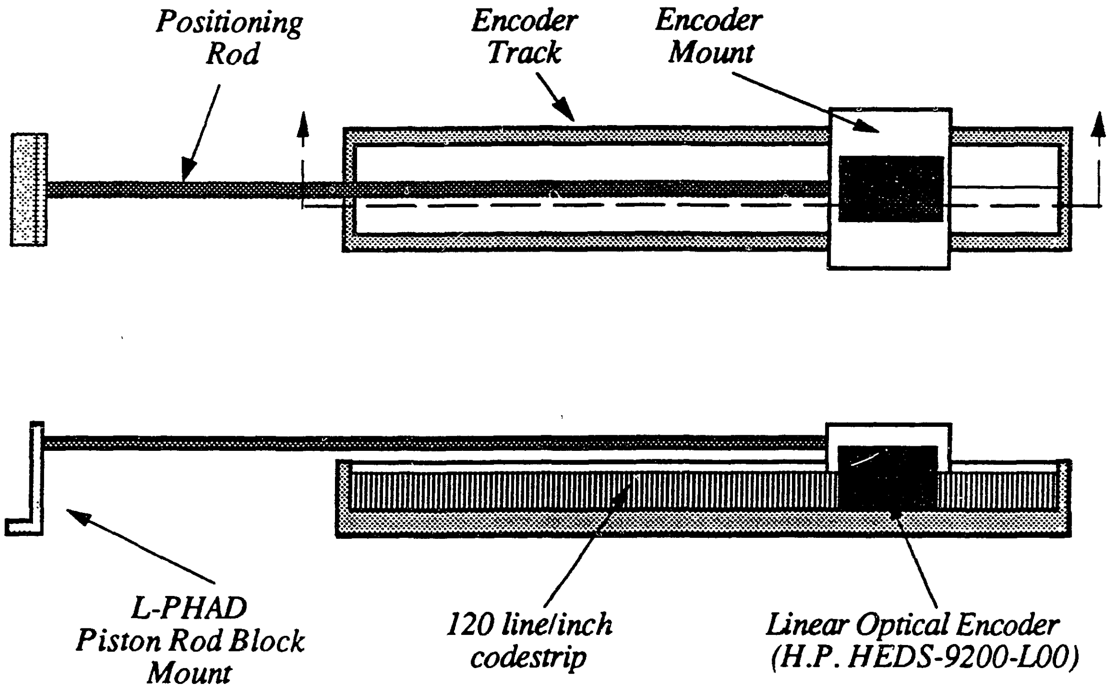

Positioning Rod
Encoder Track
Encoder Mount
L-PHAD Piston Rod Block Mount
120 line/inch codestrip
Linear Optical Encoder (H.P. HEDS-9200-L00)

Figure 4.3.9: Joint Encoder with Localized Linear Optics (JELLO) schematic

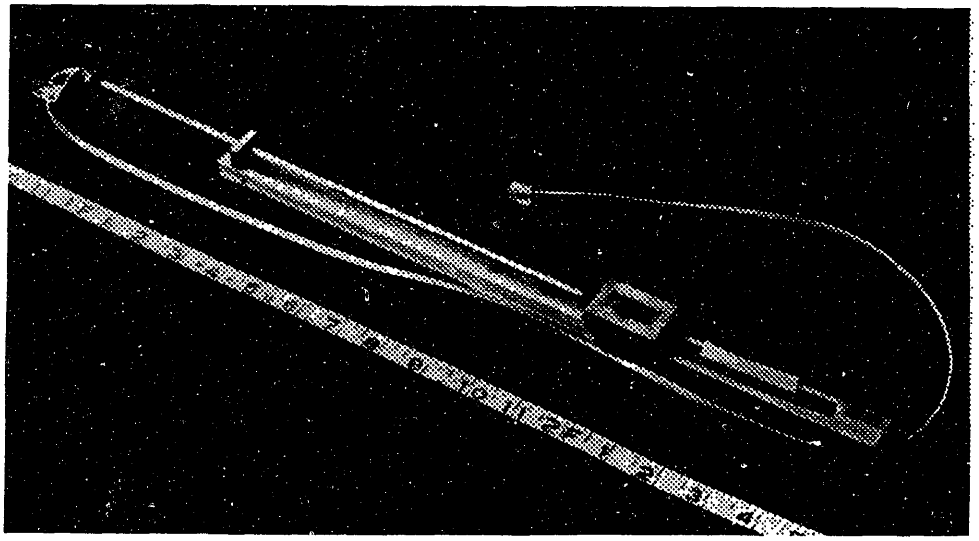


Figure 4.3.10: Joint Encoder with Localized Linear Optics (JELLO)

The fundamental component of the JELLO is the Hewlett-Packard HEDS-9200-L00 incremental linear optical encoder. The HEDS-9200-L00, when used in conjunction with a codestrip, generates linear position information in the form of quadrature signals. Since the mylar codestrip used contains 120 photo-etched lines per inch, quadrature decoding gives the JELLO sensor has an inherent resolution of $\frac{1}{480}$ or approximately 2 thousandths of an inch. Over the 12-inch stroke length of the L-PHAD this gives 5,760 counts of linear position:

$$\begin{aligned}\left(120\,\frac{\text{lines}}{\text{inch}}\right)\left(4\,\frac{\text{counts}}{\text{line}}\right)(12\text{ inches}) &= 5760\text{ counts} \\ &= 2^{13}\text{ bits}\end{aligned} \tag{4.19}$$

4.3.5 L-PHAD Servo Control

Since only Stewart Platform position control was considered in section 4.2, only a L-PHAD servo position servo was designed and tested. Specifically, a proportional-plus-derivative (P-D) control structure was implemented in software on an IBM microcomputer. Joint position was determined using the JELLO sensor and velocity was inferred from three step backwards differencing.

All testing of the L-PHAD P-D position servo was performed using the electronic circuitry and software designed for R-PHAD joint testing (see section 3.3.5). As such, the L-PHAD PWM control signal was restricted to a coarse 4-bits. Additionally, the servo control loops were run at 9 Hz, due to solenoid hardware limits. Since this results in very low controller bandwidth, aliasing and sampling noise were significant concerns.

The L-PHAD P-D position servo has the same form as the R-PHAD position servo presented in section 3.3.5.1 except that linear instead of angular position is the controlled quantity. As a result, the control law is:

$$u_k = K_1(x_d - x)_k - K_2(x_d - x)_{k-1} \tag{4.20}$$

where x indicates L-PHAD position. In block diagram form, the P-D position servo is:

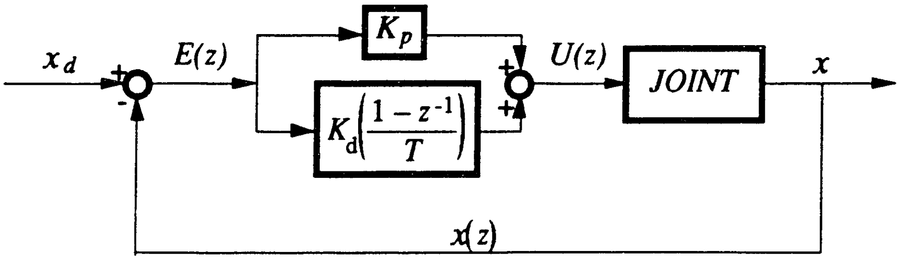


Figure 4.3.11: P-D Position Servo

Testing of the position servo was performed in neutral buoyancy using an unloaded L-PHAD joint. The results of step command testing are shown in Figures 4.3.12 (a)-(c):

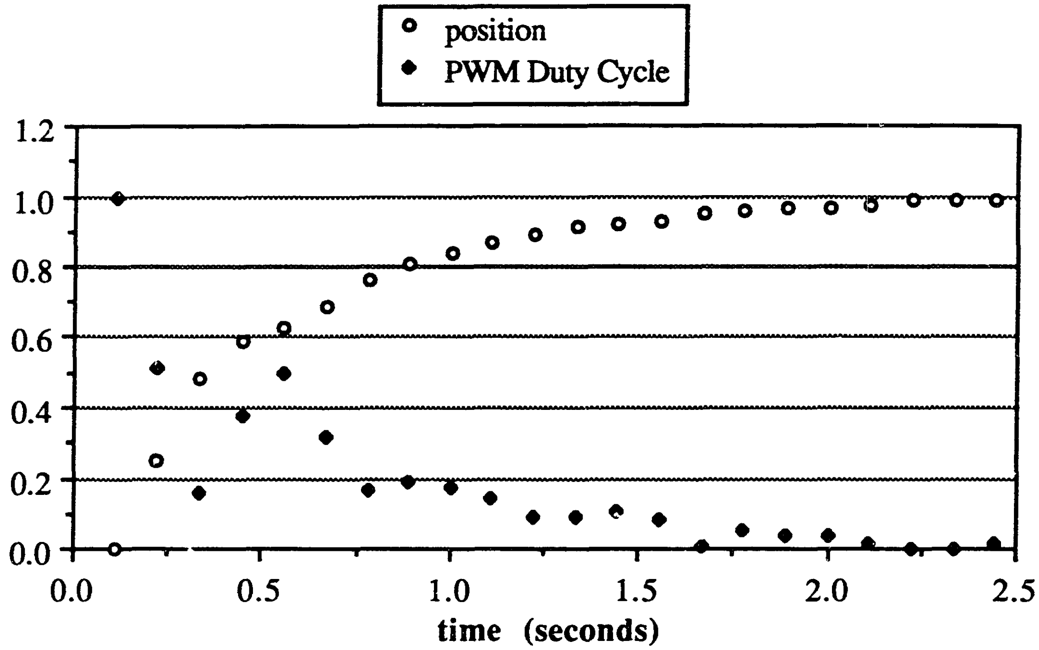


Figure 4.3.12 (a): P-D Position Servo ($K_p = 0.05$, $K_d = 0.01$)
1" step response (normalized)

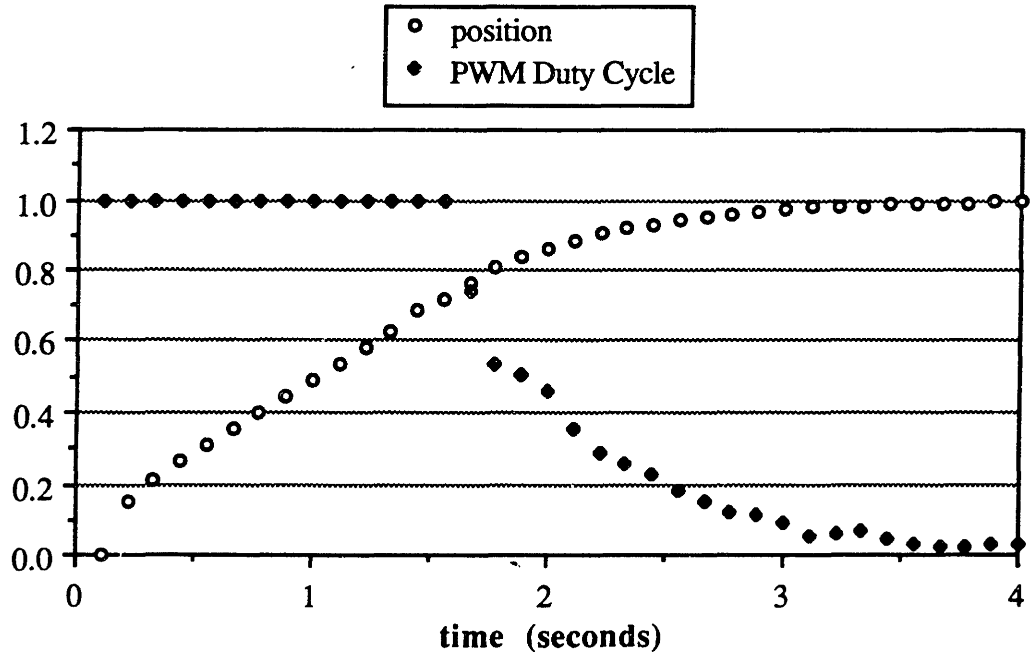


Figure 4.3.12 (b): P-D Position Servo ($K_p = 0.05$, $K_d = 0.01$)
3" step response (normalized)

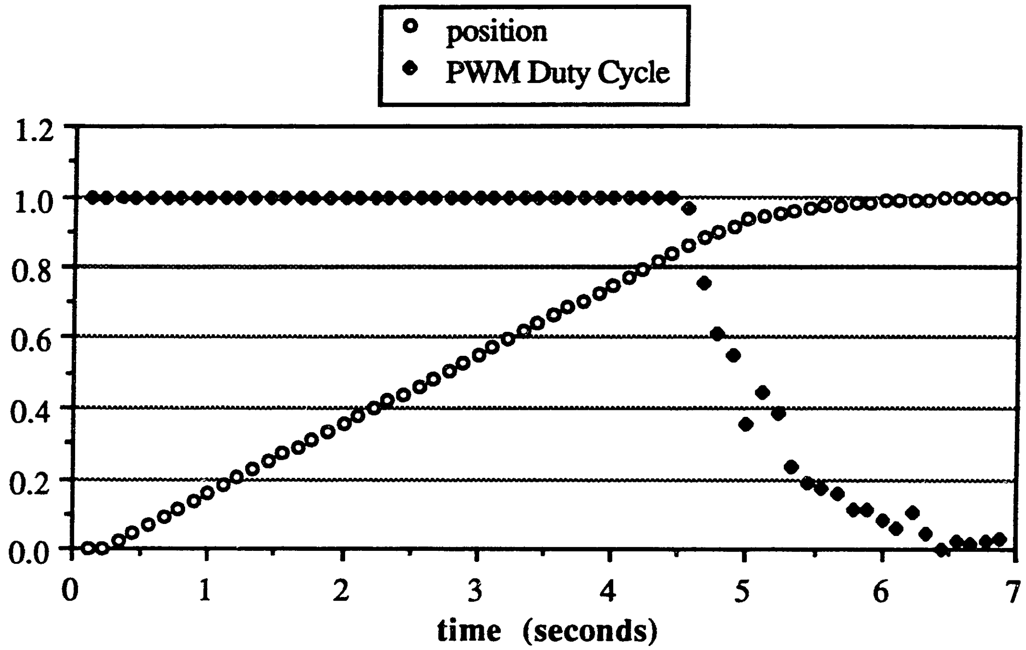


Figure 4.3.12 (c): P-D Position Servo ($K_p = 0.05$, $K_d = 0.01$)
6" step response (normalized)

As the figures show, the P-D servo exhibits a smooth, overdamped response to step command inputs. For large inputs, the L-PHAD actuator is saturated for much of the step, resulting in constant L-PHAD velocity during the saturated period. For example, a six inch step, see Figure 4.3.12 (c), exhibits this behavior during the first 4.5 seconds. As the L-PHAD nears the target position, the P-D servo commands smaller and smaller actuation, resulting in the overdamped response characteristic.

To determine if aliasing was occuring due to the low (9 Hz) sampling rate, fast quadrature sampling circuitry was constructed and operated on another IBM microcomputer. This *Quadrature Observer* ciruitry utilizes an Intel 8253 to generate hardware interrupts on fixed time intervals and is shown in Appendix A. Installed in an IBM PC/XT microcomputer, the circuitry is capable of sampling and decoding quadrature signals over a 16-3,000 Hz. frequency range.

Quadrature signals were taken from the JELLO sensor position output lines and connected to the fast sampling circuit. A step test was then performed with the sampling circuitry operating in parallel. The normalized step response for a 6" step command is shown below in Figure 4.3.13 (a) and (b).

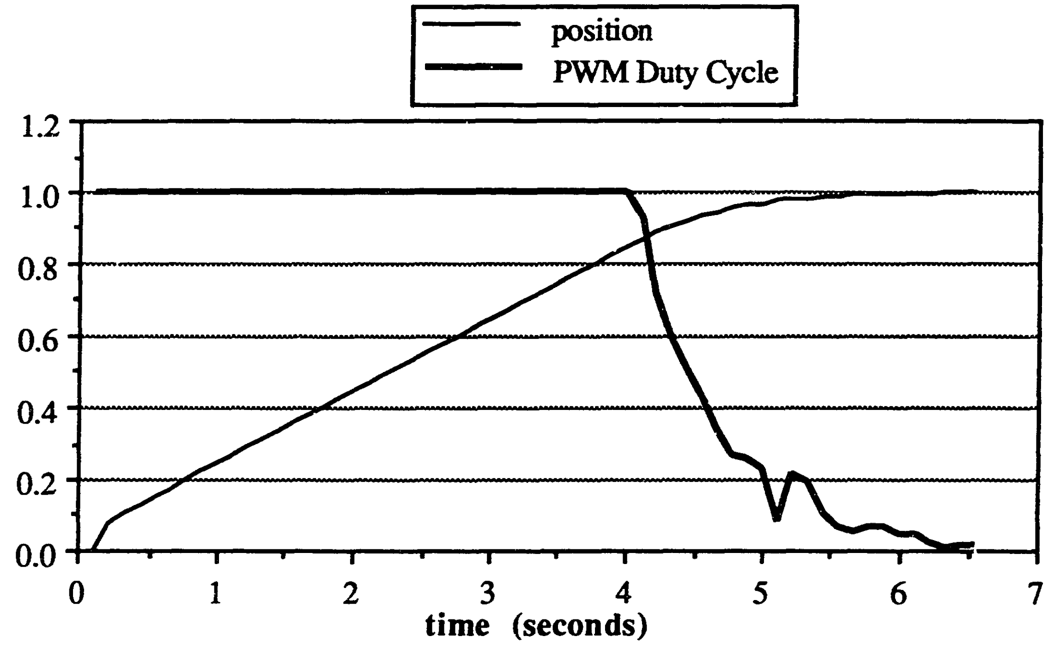


Figure 4.3.13 (a): P-D Position Servo ($K_p = 0.05$, $K_d = 0.01$)
6" step response (normalized) with 9 Hz. sampling

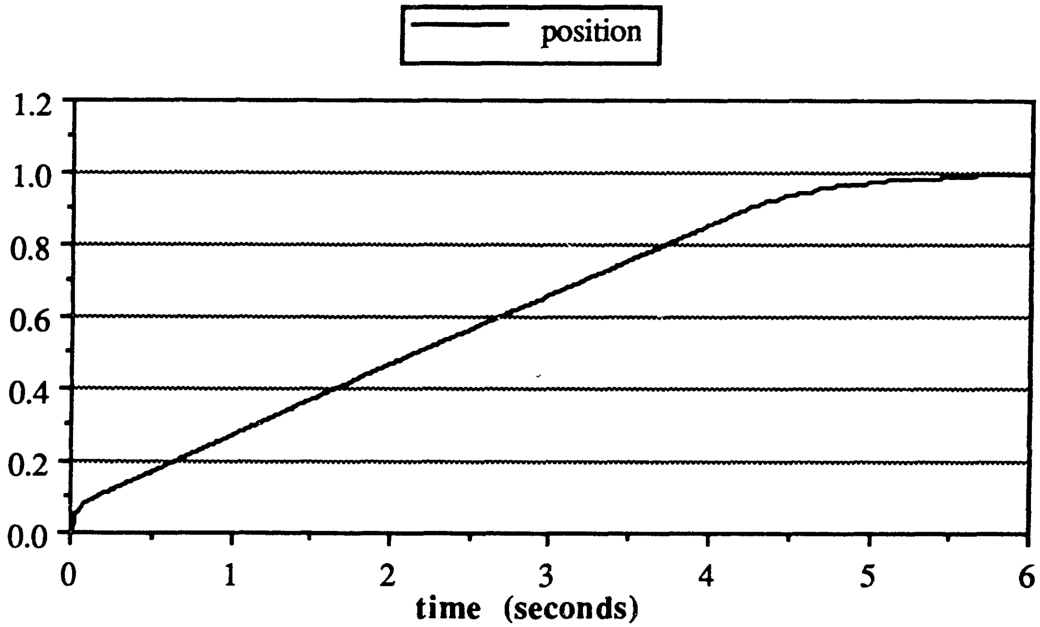


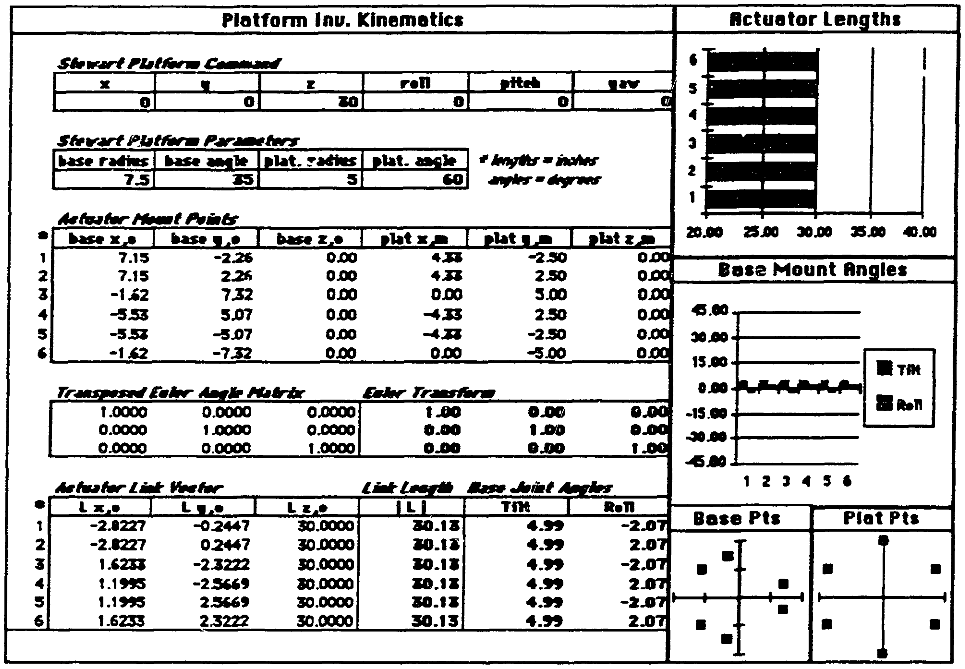
Figure 4.3.13 (b): P-D Position Servo ($K_p = 0.05$, $K_d = 0.01$)
6" step response (normalized), 100 Hz. sampled position

The two figures show the L-PHAD extension at two radically different sampling rates. The top figure (a) shows the position signal used by the 9 Hz. servo control loop. This step response shows a smooth, overdamped system response. The bottom figure (b) shows the L-PHAD extension, but at a 100 Hz. sampling rate. Comparing the two figures reveals little difference between the position plots, indicating an absence of aliased high-frequency behavior. As a result, it may be concluded that the low (9 Hz) controller bandwidth is adequate for controlling the L-PHAD system.

4.4 SPAM Stewart Platform Design

The design of the SPAM Stewart Platform presented interesting challenges. Due to the complexity of the parallel linkage, layout of the mechanism could not easily be obtained given workspace specifications. As a result, a computer model of the Stewart Platform was used for analysis of varied linkage and plate geometries. There are a great number of methods for approaching CAD of a Stewart Platform. In (Stelman, 1988), for example, a complete kinematic model was implemented on a Silicon Graphics IRIS. Although this computer model is very useful for examining kinematic constraints and for mechanism design, the overall CAD system is quite complex; requiring a vast amount of FORTRAN code and IRIS graphic primitives. It is possible, however, to create a useful system for interactive layout without resorting to this amount of programming. In particular, Stewart Platform geometry and inverse kinematics can be studied by using spreadsheet software.

To this end, a computer model of a symmetrical Stewart Platform was constructed using Microsoft Excel v2.2 on an Apple Macintosh II. Geometrical paramters (see section 4.1.1) are transformed into base and platform link mount points using (4.7) and (4.8). Platform state, specified in 6-DOF Cartesian space, is mapped to link length and link/mount point angles. CAD of a Stewart Platform using this Excel model, therefore, involves simply specifying geometrical parameters along with desired platform states and observing the resulting joint lengths and angles. Since the platform translational and rotational specifications are given, and because physical actuator and joint limits are known, evaluation of theoretical Stewart Platforms is a straightforward task. Typical numerical and graphical displays provided by Excel are shown in Figure 4.4.1.



Platform Inv. Kinematics

Stewart Platform Command

x	y	z	roll	pitch	yaw
0	0	30	0	0	0

Stewart Platform Parameters

base radius	base angle	plat. radius	plat. angle
7.5	35	5	60

** lengths = inches*
angles = degrees

Actuator Mount Points

#	base x,o	base y,o	base z,o	plat x,m	plat y,m	plat z,m
1	7.15	-2.26	0.00	4.33	-2.50	0.00
2	7.15	2.26	0.00	4.33	2.50	0.00
3	-1.62	7.32	0.00	0.00	5.00	0.00
4	-5.53	5.07	0.00	-4.33	2.50	0.00
5	-5.53	-5.07	0.00	-4.33	-2.50	0.00
6	-1.62	-7.32	0.00	0.00	-5.00	0.00

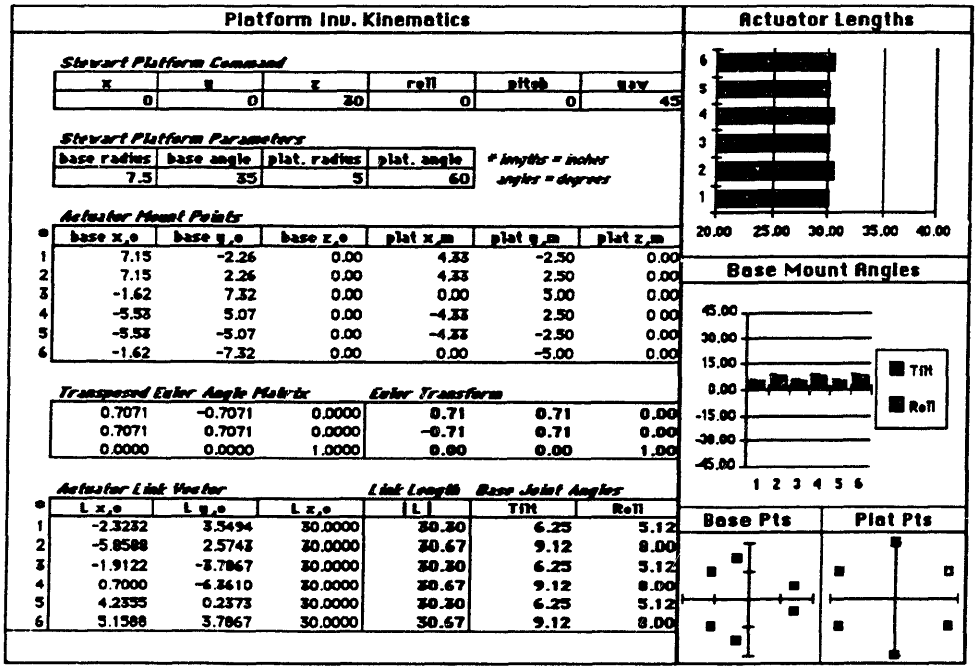
Transposed Euler Angle Matrix *Euler Transform*

1.0000	0.0000	0.0000	1.00	0.00	0.00
0.0000	1.0000	0.0000	0.00	1.00	0.00
0.0000	0.0000	1.0000	0.00	0.00	1.00

Actuator Link Vector *Link Length* *Base Joint Angles*

#	L x,o	L y,o	L z,o	|L|	Tilt	Roll
1	-2.8227	-0.2447	30.0000	30.13	4.99	-2.07
2	-2.8227	0.2447	30.0000	30.13	4.99	2.07
3	1.6233	-2.3222	30.0000	30.13	4.99	-2.07
4	1.1995	-2.5669	30.0000	30.13	4.99	2.07
5	1.1995	2.5669	30.0000	30.13	4.99	-2.07
6	1.6233	2.3222	30.0000	30.13	4.99	2.07

(a)



Platform Inv. Kinematics

Stewart Platform Command

x	y	z	roll	pitch	yaw
0	0	30	0	0	45

Stewart Platform Parameters

base radius	base angle	plat. radius	plat. angle
7.5	35	5	60

** lengths = inches*
angles = degrees

Actuator Mount Points

#	base x,o	base y,o	base z,o	plat x,m	plat y,m	plat z,m
1	7.15	-2.26	0.00	4.33	-2.50	0.00
2	7.15	2.26	0.00	4.33	2.50	0.00
3	-1.62	7.32	0.00	0.00	5.00	0.00
4	-5.53	5.07	0.00	-4.33	2.50	0 00
5	-5.53	-5.07	0.00	-4.33	-2.50	0.00
6	-1.62	-7.32	0.00	0.00	-5.00	0.00

Transposed Euler Angle Matrix *Euler Transform*

0.7071	-0.7071	0.0000	0.71	0.71	0.00
0.7071	0.7071	0.0000	-0.71	0.71	0.00
0.0000	0.0000	1.0000	0.00	0.00	1.00

Actuator Link Vector *Link Length* *Base Joint Angles*

#	L x,o	L y,o	L z,o	|L|	Tilt	Roll
1	-2.3232	3.5494	30.0000	30.30	6.25	5.12
2	-5.8588	2.5743	30.0000	30.67	9.12	8.00
3	-1.9122	-3.7067	30.0000	30.30	6.25	5.12
4	0.7000	-6.3610	30.0000	30.67	9.12	8.00
5	4.2355	0.2373	30.0000	30.30	6.25	5.12
6	5.1588	3.7067	30.0000	30.67	9.12	8.00

(b)

Figure 4.4.1: Excel CAD model of Stewart Platform
(a) z-axis translation only (b) z-axis plus 45° yaw

Initially, Stewart Platform designs were studied on a trial-and-error basis. After several attempts, a mechanism was specified that marginally fulfilled the design specifications. Iteration of this design was performed by varying the geometrical parameters. The resulting "optimal" Stewart Platform has the following parameters:

Table 4.2: SPAM Stewart Platform Parameters

Base Plate Radius (r_b)	7.5 inches
Base Mounting Angle (α_b)	35 degrees
Platform Plate Radius (r_p)	5 inches
Platform Mounting Angle (α_p)	60 degrees

which results in the base and platform mount points shown in Figure 4.4.2.

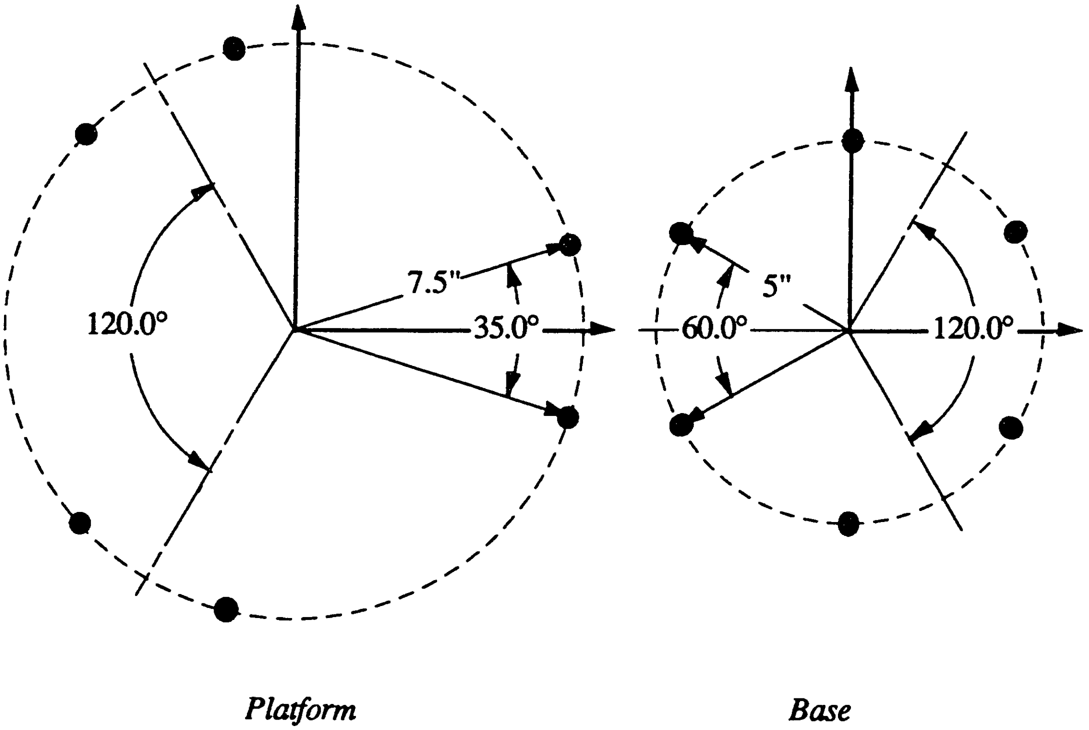


Figure 4.4.2: SPAM Stewart Platform Mounting Points

4.5 SPAM Stewart Platform Hardware

In this section, the physical hardware of the SPAM Stewart Platform has been fully described. At the time of this writing, all of the L-PHAD, Universal, and spherical joints have been constructed. The complete mechanical Stewart Platform has been assembled with these joints mounted to two circular plates. Operational testing of the mechanism, however, has not performed due to delays in JELLO sensor, pneumatic power system, and electronic control system development.

4.5.1 Assembled Hardware

The assembled SPAM Stewart Platform is shown schematically in Figure 4.5.1 and pictured in Figure 4.5.2 below. As the figures show, the Stewart Platform has six SPS-links attached in the design configuration previously described. Both the base and platform plates are 18" diameter aluminum discs. Mounted on the platform plate is a circular box surrounded by a flotation ring. The box is pressurized and contains the twelve Koganei solenoids required by the six L-PHAD joints. The ring is constructed with Rohacell high-density foam covered with fiberglass cloth and is used to help make the Stewart Platform neutrally buoyant.

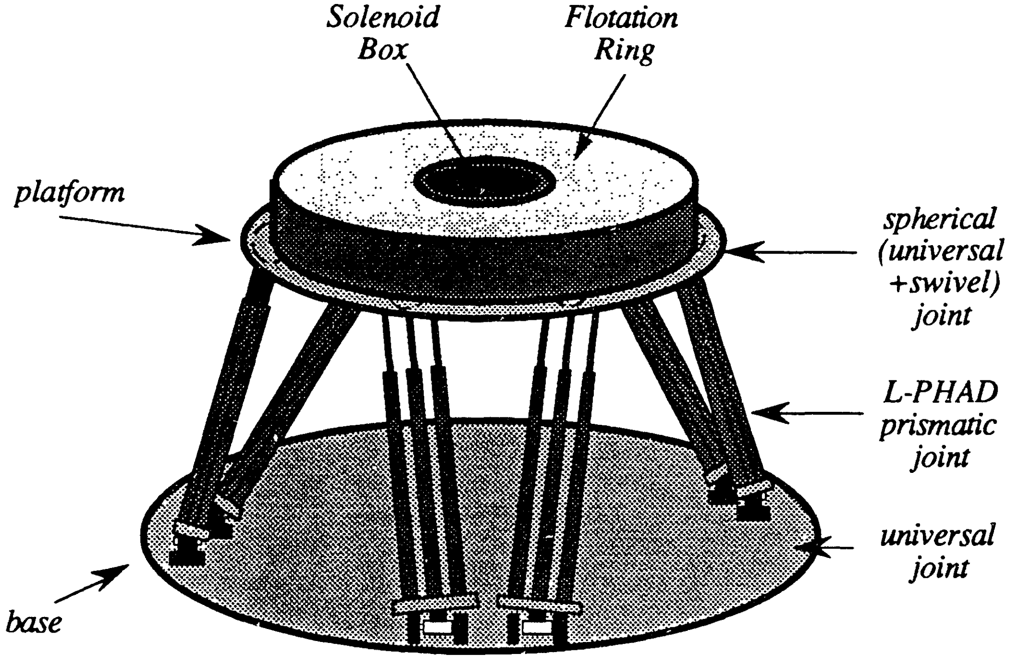


Figure 4.5.1: SPAM Stewart Platform schematic

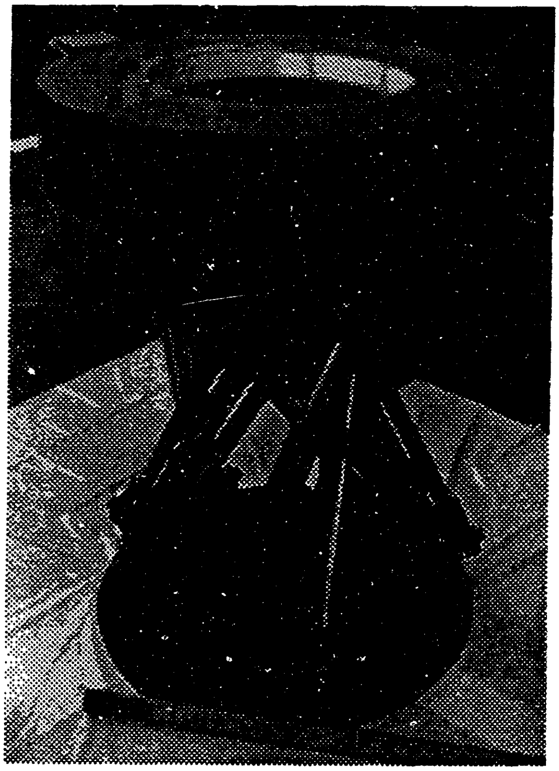


Figure 4.5.2: Assembled SPAM Stewart Platform

4.5.2 Pneumatic System

The SPAM Stewart Platform utilizes a pneumatic supply similar to the manipulator arm system described in section 3.4.3. Compressed air from 72 cu. ft. scuba tanks is regulated to 125-140 p.s.i. using a first-stage scuba regulator and fed to the Stewart Platform solenoid box through 1/8" (I.D.) pressure tubing. A secondary line, tapped off the supply line, is used to pressurize the solenoid box after regulation via a Go, Inc. regulator. Inside the solenoid box, this high-pressure supply is attached to a manifold connected to the twelve Koganei solenoids. High pressure Koganei output, commanded by the L-PHAD servocontroller, is routed to the L-PHAD pneumatic cylinders using CPC quick disconnect fittings.

A schematic of the complete Stewart Platform pneumatic system is provided in Figure 4.5.3.

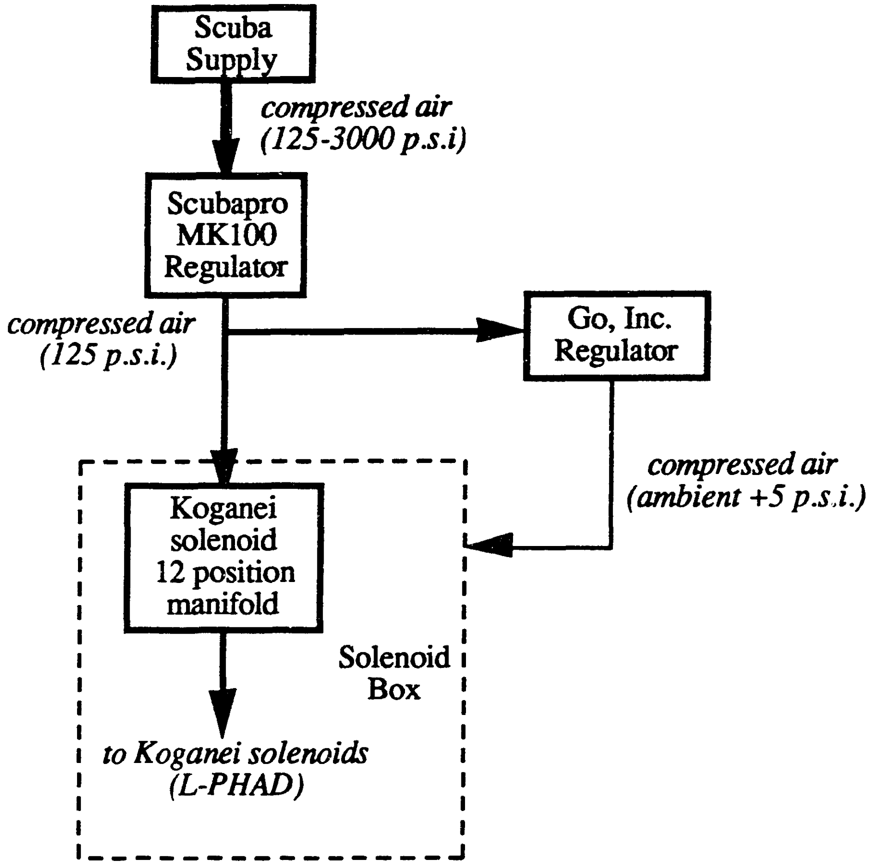

Scuba Supply
compressed air (125-3000 p.s.i)
Scubapro MK100 Regulator
compressed air (125 p.s.i.)
Go, Inc. Regulator
compressed air (ambient +5 p.s.i.)
Koganei solenoid 12 position manifold
Solenoid Box
to Koganei solenoids (L-PHAD)

Figure 4.5.3: SPAM Stewart Platform pneumatic system *(schematic)*

4.5.3 Electronic Control System

Although testing of a single L-PHAD joint successfully used the single R-PHAD joint electronic control system (ECS), described in section 3.4.4, this arrangement was deemed unsuitable for Stewart Platform operation for three reasons. First, the complete Stewart Platform implementation includes six L-PHAD joints, meaning that the R-PHAD ECS would have to be duplicated six times. Secondly, since the R-PHAD ECS solenoid signals are generated on the surface, control of six L-PHAD's would necessitate a bulky set of long electrical lines. Finally, the inherent coarseness of the R-PHAD PWM signal (4-bits) limits L-PHAD performance and precision.

To remedy these shortcomings, the *Stewart Platform Electronic Control System* (SPECS) was designed using microprocessors. The resulting distributed system is extremely small in size and is capable of generating high-fidelity (i.e. 16-bit) PWM signals. Moreover, because of the distributed systems approach, it was possible to reduce Stewart

Platform computation demands by locating the L-PHAD position and velocity servocontrollers in the microprocessors.

The microprocessors selected for SPECS are New Micros, Inc. “2x4” series (NMIS-0021) microcomputers. The NMIS-0021 units, based on the Motorola F68HC11 MCU, are physically compact, incorporate a full implementation of the FORTH language, and are ideally suited for direct hardware control. Moreover, direct MCU address and data bus access is easily obtained by the addition of a New Micros’ NMIS-0001 expansion/prototyping board. The New Micros’ microcomputer hardware is shown in Figure 4.5.4.

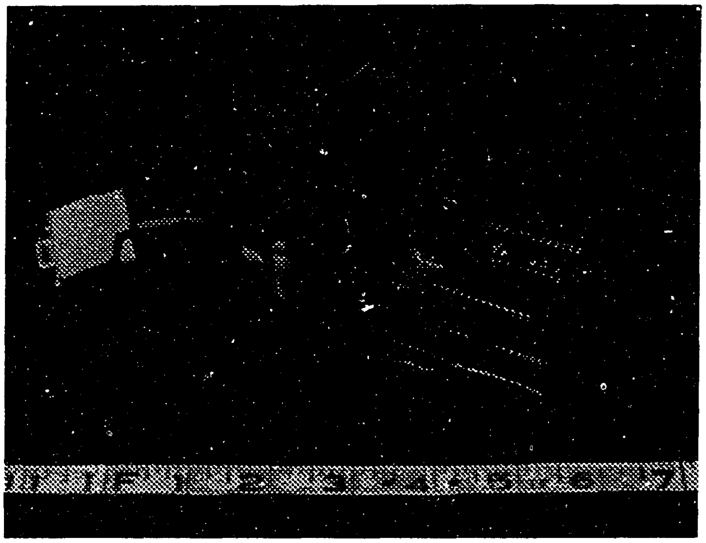


Figure 4.5.4: NMIS-0001 and NMIS-0021

In order to implement servocontrollers on the NMIS-0021, SPECS was required to use JELLO sensor feedback. Similar to the R-PHAD ECS design, Hewlett Packard HCTL-2000 chips were selected to decode the JELLO quadrature signal to a 12-bit position word. Due to physical limitations, however, only three of these chips could be placed on a single NMIS-0001 board. As a result, it was necessary to split the control task between two NMIS-0021’s with each computer controlling three L-PHAD joints. The final quadrature decoding circuitry design implemented on each NMIS-0001 is given in Appendix A.

During operation of the Stewart Platform, a surface computer sends position commands to the L-PHAD servos. Command transmission is accomplished using RS-422 serial communications between the surface computer and the two NMIS-0021’s (see section 5.1). L-PHAD control signals, generated by servos written in FORTH, are output through NMIS-0021 I/O ports and are transformed to solenoid signals with a transistor based power switching box. A block diagram of the SPECS is shown below:

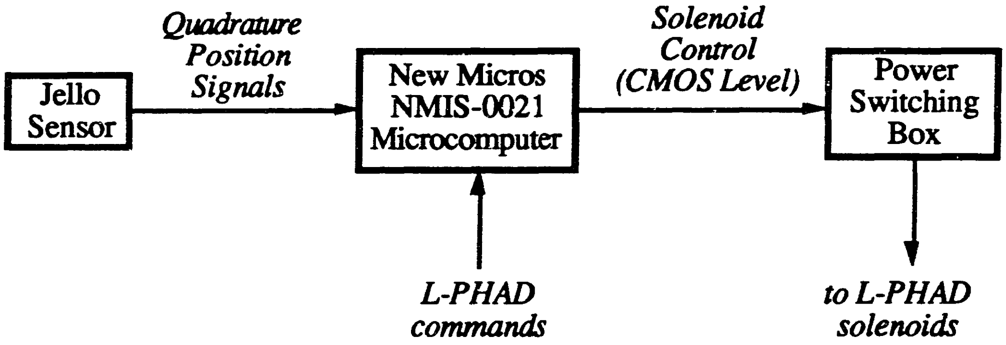

Jello Sensor
Quadrature Position Signals
New Micros NMIS-0021 Microcomputer
Solenoid Control (CMOS Level)
Power Switching Box
L-PHAD commands
to L-PHAD solenoids

Figure 4.5.5: Stewart Platform Electronics Control System

It should be noted, however, that this configuration of SPECS does not completely eliminate the need for surface to Stewart Platform signal lines. Since commands are sent from a controlling computer to the two NMIS-0021's, a serial communications line must be included. Moreover, the two NMIS-0021's and the L-PHAD sensors require a +5 VDC supply. Although it is possible to collocate a battery and the Stewart Platform, it is simpler to utilize a power supply line. Finally, the Power Switching Box requires a +12 VDC supply to transform the NMIS-0021 control signals from CMOS level to the +12 VDC needed to operate L-PHAD solenoids. Since these solenoids draw considerable power, an additional power supply line allows sustained Stewart Platform operation.

Chapter 5
SPAM Development & Control

The task of implementing the complete SPAM system is not an easy one. Although anthropomorphic manipulator operation is straightforward, the addition of a Stewart Platform micromanipulator greatly alters the situation. The integration of manipulator and Stewart Platform hardware systems (i.e. electrical, mechanical, computational, etc.), for example, requires careful management of resources such as computational facilities. Additionally, the control problem is increased in complexity from three to nine degrees-of-freedom. Finally, logistical concerns such as user interfacing, trajectory generation, and task planning are significantly complicated. To address these problems an object-oriented philosophy has been adopted for operation of SPAM. This approach, originally intended to facilitate representation of complex data in computer systems, is a useful abstraction mechanism and lends itself extremely well to operation of distributed systems.

In the following sections, the integration of the anthropomorphic manipulator (described in Chapter 3) with the symmetric Stewart Platform (described in Chapter 4) is discussed. Section 5.1 discusses physical issues concerning the integration of electrical and mechanical systems. Section 5.2 develops the object-oriented methodology, describes the SPAM software architecture and details the nine degree-of-freedom control scheme. Finally, Section 5.3 addresses user interfacing and explores operational concerns. It should be noted that at the time of this writing, the total integration of manipulator and Stewart Platform has not been completed. Hence, some of the following subjects are theoretical in nature, and should be viewed merely as proposals for future system development.

5.1 Hardware Integration

The integration of the SPAM manipulator arm and Stewart Platform hardware was separated into three phases. First, the Stewart Platform was attached to the tip of the manipulator arm. Secondly, a pneumatic supply system was developed. Finally, computer systems were linked using network communications protocols.

The connection of the manipulator arm and Stewart Platform was accomplished by installing an mechanical coupling. This coupling, shown in Figures 5.1.1, is an adjustable, locking hinge with a ninety degree range. Adjustment/locking of the hinge angle is performed by the removal/insertion of a "pip-pin". The coupling's manipulator arm interface is identical to the joint/arm boom connection described in section 3.4.2. During typical SPAM operations, the Stewart Platform base plate is fixed normal to the manipulator arm axis. The adjustable coupling, however, provides additional flexibility for tasks which require an offset Stewart Platform.

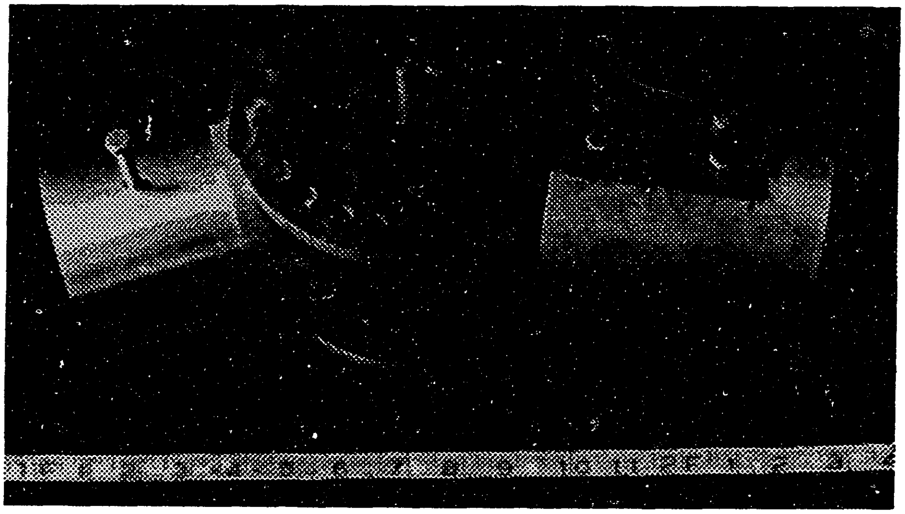


Figure 5.1.1: Adjustable Hinge Coupling

To provide compressed air to the three manipulator arm joints (R-PHAD) and six Stewart Platform joints (L-PHAD), a pneumatic supply system was designed. Since both types of joints were designed to operate with 125-140 p.s.i. air, it is theoretically possible to power all the joints from a single supply line. To provide the required flow rate and volume, however, would necessitate the use of large diameter pressure tubing. Unfortunately, flexible tubing of the required size is not available. Additionally, an individual, single-state scuba regulator would be unable to provide the desired output of compressed air.

A more desirable approach to pneumatics is to implement a dual supply system. In this arrangement, the outputs from two first-stage scuba regulators are used to supply the SPAM joints. Although it is possible to use one supply line for the manipulator arm and one supply for the Stewart Platform, this arrangement fails to consider operational requirements. Since normal SPAM operation involves coarse positioning with the manipulator arm followed by fine position tuning with the Stewart Platform, a configuration utilizing two supply lines in parallel would be a more efficient allocation of resources. As a result, the SPAM pneumatic supply system uses one line for the manipulator shoulder joints and the other for the manipulator elbow and Stewart Platform joints. Though both lines are not continually used (i.e. the shoulder line is not active during Stewart Platform operation), the higher pneumatic flow capability allows for better R-PHAD performance.

In order to achieve good system response, given limited computational power, it was necessary to implement a distributed processing system for real-time operation. This system utilizes several microcomputers, described in the following section, which need to communicate with each other. To provide the necessary communications capability, two network protocols were implemented. First, the Pilot-Vehicle-Communications-System (PiVeCS) is used for serial communications between IBM microcomputers. The PiVeCS protocol, developed for SSL use by R. Sanner, is a robust communications system intended to provide reliable hardware interfacing and efficient information exchange (Sanner, 1990). Secondly, the Microprocessor-Microcomputer-Communications-System (MiMiCS) was designed to provide communications between the New Micro's 68HC11 microprocessor systems and an IBM microcomputer. The MiMiCS protocol was designed to be similar to PiVeCS, utilizing the object-oriented paradigm of "message-passing" to handle information exchange.

In terms of physical hardware, the computer network is comprised of serial communication lines connecting microcomputer nodes. PiVeCS operates between two IBM microcomputers over a short RS-232 line running at 19,200 baud. MiMiCS connects two New Micro's "2x4"'s with one of the IBM microcomputers via RS-422 lines operating at 9,600 baud.

5.2 Distributed Systems

As discussed previously, to obtain the highest performance system possible given limited computational resources, a distributed processing system was implemented. This system was designed using an object-oriented philosophy to abstract the SPAM system into various components. By this approach, various *objects* with internal, local *states* operate in specific *environments* with other objects. Each object is able to perform certain tasks and may interact with other objects (i.e. ask for information, request an action). No object, however, can ever directly modify the internal state of another. This localization of state is often referred to as "information-hiding" or "data-encapsulation" and this method of interaction is known as "message passing". Taken together, these techniques offer a powerful framework for organizing computational models and developing complex systems (Abelson, 1985).

To begin, the SPAM system is divided into eight objects based on subsystem function. These objects and their respective functions are described in Table 5.1:

Table 5.1: SPAM Object Definitions

Object	Functions

User Interface
(see section 5.3)
	Receives user commands and produces displays showing current:
	manipulator 3-DOF tip state in {base} and platform 6-DOF state in {tip}
	OR
	platform 6-DOF state in {base}

SPAM Controller (see section 5.3)	If necessary, transforms commands from the *User Interface* to manipulator tip commands in {base} and Stewart Platform commands in {tip} Produces current manipulator tip state in {base}, platform state in {tip} and {base}
Stewart Platform Controller (see section 4.2)	Generates L-PHAD position commands via platform controller and calculates platform state in {tip} via FWDKIN.
Manipulator Arm Controller (see section 3.2)	Generates R-PHAD commands via arm controller and calculates arm state in {base} via FWDKIN.
L-PHAD Controller (see section 4.3.5)	Generates L-PHAD control signals via position servo.
R-PHAD Controller (see section 3.3.5)	Generates R-PHAD control signals via position and rate servos.
L-PHAD Joints	Controls Stewart Platform 6-DOF platform location. L-PHAD state (lengths) output from JELLO sensor.
R-PHAD Joints	Controls Manipulator Arm 3-DOF tip position. R-PHAD state (angles) output from optical encoder.

Each object maintains a set of internal (local) data which it shares with other objects upon request. This structure is advantageous because it offers a clean, efficient structure for information exchange and eliminates the need for maintaining redundant sets of data. The *Manipulator Arm Controller*, for example, calculates the current tip position with manipulator forward kinematics and provides the state information to the *SPAM Controller*.

Similarly, the *SPAM Controller* provides this acquired state information to the *User Interface* for output to the user. From a hardware perspective, moreover, this structure is quite natural; the *L-PHAD Joints* and the *R-PHAD Joints* maintain real data (e.g. joint state) and provide the information to other objects via sensor readings.

Implementation of these objects is straightforward. Obviously the *R-PHAD Joints* and the *L-PHAD Joints* are components of the manipulator arm (Chapter 3) and the Stewart Platform (Chapter 4). The *User Interface* may be implemented in any number of ways, but keyboard entry and computer display (section 5.3) does not requires additional hardware. The allocation of controllers was determined by considering computational resources. Four microcomputers were available for SPAM operation. A summary of these systems is provided in Table 5.2:

Table 5.2: SPAM Microcomputer Systems

Computer	Description
Pryor	IBM PC/AT compatible 80286 CPU running at 12 MHz 80287 floating point coprocessor
R2-D2	IBM PC/XT 8088 CPU running at 4.77 MHz 8087 floating point coprocessor
Bill	New Micro's "2x4" F68HC11 CPU running at 2 MHz On board FORTH interpreter
Ted	New Micro's "2x4" F68HC11 CPU running at 2 MHz On board FORTH interpreter

First, it was determined that the *SPAM Controller*, the *Manipulator Arm Controller*, and the *R-PHAD Controller* have the highest computational demands. Consequently, these objects were placed on *Pryor* along with the keyboard-driven *User Interface*. Secondly, the *Stewart Platform Controller* was assigned to *R2-D2*. Finally, the *L-PHAD Controller* was divided between the two New Micro's "2x4"s, *Bill* and *Ted.*

A summary of SPAM object locations is given below in Table 5.3:

Table 5.3: SPAM Object Distribution

Object	Location
User Interface *SPAM Controller* *Manipulator Arm Controller* *R-PHAD Controller*	*Pryor*
Stewart Platform Controller	*R2-D2*
L-PHAD Controller	*Bill & Ted*
L-PHAD Joints	*Stewart Platform*
R-PHAD Joints	*Manipulator Arm*

Objects communicate with each other using a message-passing scheme. Messages are sent from one object to another to request data or to ask that an action be performed. For objects coexisting on a common microcomputer, messages are passed directly between software modules. If objects are located on different machines, however, messages are sent over a physical link using the appropriate serial communications protocol, PiVeCS or MiMiCS. To facilitate streamlined message-passing, the eight objects have been configured in a tree structured network. The distributed processing system architecture, with communication channels, is given in Figure 5.2.1.

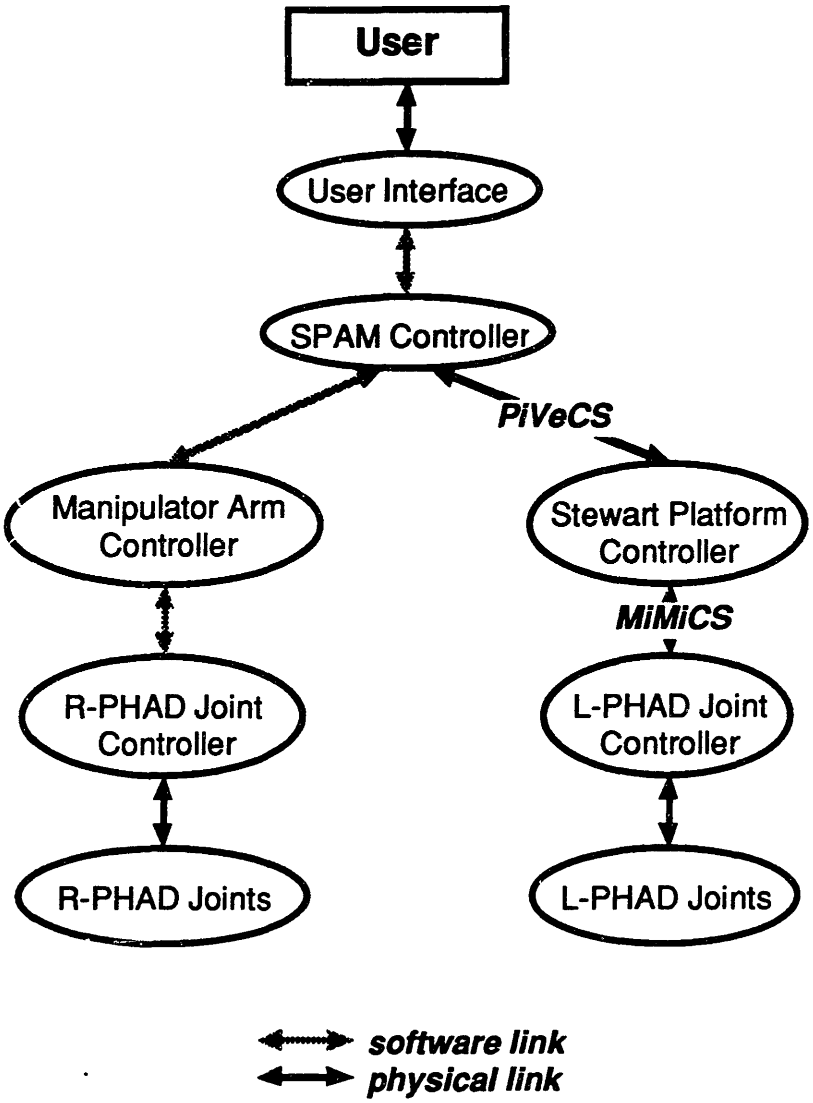


Figure 5.2.1: SPAM Distributed Processing Network

The benefits of using a distributed processing system are readily apparent. Since computations have been split between several microprocessors, parallel processing of data may be performed. Once the user has specified desired system state (i.e. end-effector position), the *SPAM Controller* divides the task of controlling nine degrees-of-freedom into manipulator arm and Stewart Platform operations. High-level controllers, working in parallel, generate commands for the joint servos. These servos, in turn, produce signals to control the actual SPAM hardware. From a performance perspective, this system structure is extremely efficient because computational resources are utilized to the greatest extent possible.

5.3 Operation of SPAM

5.3.1 User Interface and SPAM Control

To ease the operation of a complex mechanism, careful consideration must be given to the user interface. In particular, the input of desired actions should not place excessive workload or burdens on the operator. For the SPAM system, this implies that specification of manipulator and Stewart Platform movement be easy for the user to express. Since commands are most intuitively expressed as Cartesian position and orientation, the design of the SPAM user interface should reflect Cartesian thought.

There are many method of implementing a Cartesian-based interface. Rotational and translational hand-controllers, for example, may be used to "fly" the desired hardware frame (e.g. manipulator tip, platform plate) in six degrees-of-freedom. The simplest implementation, however, is for the user to specify desired actions directly to a microcomputer through keyboard entry. This method is advantageous since it does not require the integration of additional hardware to SPAM and because it allows great flexibility through software design.

Assuming that all input is obtained with keyboard entry, the task of executing user commands is performed by the SPAM Controller object. This controller differs from the high-level controllers and joint servos discussed in previous chapters in that it does not actually perform control actions. Rather, the SPAM Controller acts as a supervisor by translating user commands into desired states for the manipulator arm and Stewart Platform. For example, consider that an arbitrary trajectory has been specified. This trajectory contains a set of Cartesian frames that the user desires SPAM to achieve. The SPAM Controller processes these frames and produces a set of commands, which may in fact be another Cartesian trajectory, for the manipulator and Stewart Platform controllers to follow.

5.3.2 Operational Issues

Besides user interfacing, there are a number of other important operational issues. First, there is the problem of trajectory specification. Given that SPAM has nine degrees-of-freedom, one question is whether it is necessary to completely specify all nine degrees at each desired location. Since the manipulator arm was intended to provide 3-DOF coarse positioning and the Stewart Platform 6-DOF fine tip positioning, it is reasonable for the user to give 3-DOF specifications for trajectory way-points and 6-DOF at the end points. A processor then processes these incomplete specifications to produce smooth 9-DOF trajectories.

Secondly, there is the issue of obstacle avoidance to consider. For any robotic device, there is always a potential for collision with objects. If the operating system has *a priori* knowledge of obstacles entering (or already existing in) the robot's reachable workspace, then trajectory modifications can be planned. However, if obstacles are not known to the system, contingency planning is problematic.

Finally, the SPAM system utilizes and produces large forces. If not controlled properly, these forces are potentially hazardous to humans working with SPAM or in the reachable workspace. As a result, the issue of human safety requires consideration. In terms of logistics, this means that SPAM must be operated in a manner conducive to safety at all levels. Fault handling and recovery must be implemented to deal with events such as control systems failure, mechanical breakage, and loss of power.

Chapter 6
Conclusion

6.1 Current SPAM Hardware

At this time, the entire SPAM system is not operational. The manipulator arm has only been tested in a planar 2-link planar configuration. To complete the manipulator, it is necessary to add a revolute shoulder yaw joint. Although such a joint has been designed, the R-PHAD-B of chapter 3, fabrication and operational testing need to be completed.

In terms of the Stewart Platform, all the joints (prismatic, universal, and spherical) have been completed. Installation of the electronic control and pneumatics systems, however, needs to be finished. Testing of the JELLO sensor has revealed that photoetched, mylar codestrips cannot be submerged for extended time periods, indicating that further research and development is necessary. Finally, microprocessor servo control and MiMiCS network interfacing have only been investigated to a limited extent.

A portion of the mechanically integrated SPAM system is shown in the following figures. The first, Figure 6.1.1, shows the elbow-to-Stewart Platform section with a 25 foot arm boom. The two smaller arm booms, 6 and 12 feet, can be seen alongside. In figure 6.1.2 SPAM is seen from a tip perspective, with a close-up of the Stewart Platform.

Figure 6.1.1: SPAM Elbow-to-Stewart Platform (elbow view)

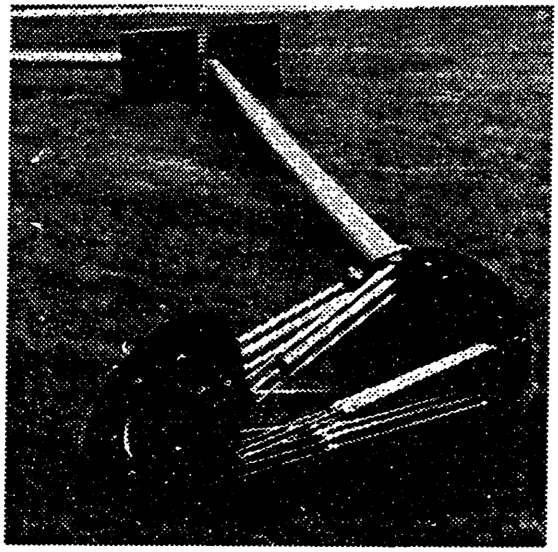


Figure 6.1.2: SPAM Elbow-to-Stewart Platform (tip view)

6.2 SPAM Development Observations

The development of SPAM, from concept inception to the present time, has spanned a period of two years. The first three months, July–September 1988, were spent investigating neutral buoyancy actuators and studying the viability of pneumatics for high-precision systems. The next year, October 1988–September 1989, saw the development of manipulator joint designs and the fabrication of Stewart Platform mechanisms. Finally, during the last eight months, October 1989–May 1990, actuator testing, sensor development, and implementation of joint servocontrol techniques have been performed.

The original SPAM development plan proposed that all hardware systems (electrical, mechanical and pneumatic) be constructed within the first six months. The actual construction of SPAM, however, has taken much longer. In fact, after two years of work only a 2–link manipulator is operational and considerable Stewart Platform implementation work remains to be done. Looking back, it is clear that the substantial construction delay may be attributed to a single factor; design of the *Pneumatic–Hydraulic Actuating Device* for neutral buoyancy operation.

Several problems impeded smooth development of PHAD. First, obtaining pneumatic and hydraulic cylinders for neutral buoyancy operation was difficult. Besides satisfying force production specifications, these cylinders were required to be corrosion-resistant, reliable at depth underwater, and cost-effective. After several months, PHAD was finally implemented with aluminum Clippard pneumatic cylinders and stainless-steel Aurora hydraulic cylinders. Unfortunately, the Clippard cylinders originally used contained a defective piston seal design, causing a delay in PHAD testing until improved cylinders could be obtained.

Secondly, guaranteeing smooth water flow through PHAD was a problem due to difficulties in acquiring robust solenoid valves. Initially, Kip solenoids were used in all PHAD designs for water service. These valves were chosen due to compactness, low cost, and availability. Testing of the R-PHAD joint, however, revealed that the Kip solenoids were unable to handle the water pressure and water flow produced by 2" bore cylinders. After contacting several distributors, it was decided that replacement of the Kip valves with another solenoid line would be required. The ensuing search resulted in custom manufactured high-performance (pressure rating vs. size) solenoid valves from the Laketown Co. of Waconia, Minnesota.

Finally, and perhaps most significantly, the mechanical R-PHAD design proved to have inherent difficulties. Since all the cylinders are mounted by threading directly into aluminum blocks, joint friction is strongly dependent on thread alignment. In other words, very small tolerances must be maintained during fabrication or the joint will not move smoothly. Additionally, the threaded mounting made alignment of cylinder ports difficult, especially if snugly mounted cylinders was desired. Finally, the asymmetrical cylinder arrangement (required for symmetrical joint forces) resulted in potential for uneven piston loading.

6.3 Recommendations for Future Studies

At this stage of SPAM concept development, there are several implementation issues which remain unresolved and require investigation. Additionally, a great number of interesting manipulator arm and Stewart Platform topics warrant study and experimentation. Finally, there are very exciting neutral buoyancy space research areas for which SPAM is an excellent candidate.

6.3.1 Implementation Specific Issues

In terms of unfinished work, two obvious concerns are how well the design of the R-PHAD-B manipulator shoulder yaw joint and the Stewart Platform will perform in neutral buoyancy operation. As discussed previously, the R-PHAD-B has been fully blueprinted, though not completely fabricated. Immediate attention should be given to completion of this joint and to submerged testing. For the Stewart Platform, final subsystem assembly and integration remain to be performed. Efforts should additionally, therefore, be given to this task.

Although the current SPAM hardware systems perform adequately, it may be possible to achieve better performance by refinement of the PHAD design. To this end, it will be necessary to conduct a detailed study of actuator operational characteristics. Especially important would be a characterization of water dynamics (flow properties, cavitation, etc.) through the PHAD cylinders and solenoids since this would allow better hydraulic design. Additionally, refinement of the PHAD mechanical linkage and connections could reduce mechanical slop and produce higher force and torque outputs.

Other issues of concern are computational structure implementation and user interfacing. The use of a parallel processing system opens a wide range of topics for study. Optimality of the current distributed architecture for SPAM computations is one issue. Investigation of alternative computational structures is another. As far as user interfacing, there are many unanswered questions since limited research has been performed thus far. Research in this area may delve into areas such as input devices, information display, and human factors.

6.3.2 Manipulator Arm

Control engineering of serial-link manipulators is fascinating because the field is relatively young and extremely dynamic. Although the manipulator control systems discussed in this thesis would have been considered sophisticated a few years ago, by current standards they are quite unremarkable. Moreover, the design approach utilized for the SPAM manipulator arm, individual joint control, places inherent limitations on performance and robustness.

Consequently, it is appropriate to research and implement more advanced and sophisticated controllers. Some current manipulator schemes include adaptive control, non-linear control, and optimal control. Of these, non-linear sliding-surface control seems especially promising since this technique can provide extreme robustness to non-linear systems. Additionally, there is some interest in novel strategies such as neural network and intelligent (e.g. fuzzy-logic) controllers.

Finally, there is an abundance of manipulator operational and logistical issues suitable for investigation. Present day path planning, for example, is largely heuristic in nature and almost exclusively utilizes trajectory curve splining. This approach is hardly optimal and any research, especially involving mathematical rigor, could lead to vast improvements. Another interesting topic is obstacle detection and avoidance. Previous research in this area has been focused on sensor fusion and vision processing. Applying such concepts to SPAM would be beneficial for tasks in crowed workspaces. Lastly, there is a need to determine efficient methods for handing of kinematic singularities. In particular, techniques that maintain smooth movement near singularities would greatly aid manipulator operation.

6.3.3 Stewart Platform

The approach to Stewart Platform control presented in this thesis is extremely limited since system dynamics have not been considered. The rationale for ignoring these dynamics was that neutral-buoyancy linear actuators could not provide high performance. Development of the L-PHAD, however, has shown in fact this assumption was incorrect; that it is possible to create a robust actuator. Consequently, the door is now open for study of more sophisticated Stewart Platform control strategies. In particular, it is possible to direct efforts towards control of Stewart Platform forces, both internal and applied. To this end, it may be desirable to study impedance and admittance control techniques to create controllers for a dynamic Stewart Platform.

Additionally, it has been stated that the forward kinematics mapping of a Stewart Platform is extremely difficult to solve in closed-form, and that a Newton-Raphson numerical method is typically used instead. This is not to imply that a closed-form solution of the forward kinematics is not possible, or that other numerical solutions are less efficient, simply that N-R is used because it can produce a solution. However, since the NR method has variable convergence properties and may be unstable in the presence of singularities, it would be prudent to investigate alternative means of performing forward kinematics. Since the SPAM Stewart Platform is a symmetrical implementation, it may in fact be possible to determine a closed-form solution.

Another area of interest is workspace analysis of a Stewart Platform. In general, formal analysis is difficult due to the inherent geometrical complexity and the multiloop linkage structure. Since the SPAM Stewart Platform is a symmetrical arrangement, however, it is possible to obtain approximate spatial limits by considering the workspace to be conical in shape (Yang, D.C.H and Lee, T.W., 1984). Although actual workspace analysis was not performed for the SPAM Stewart Platform, the basic method is to investigate translational limits along cross-sectional planes. For example, the platform center could be moved in the XZ and $X'Z$ planes, of {base} shown in Figure 4.4.3. The locus of reachable (i.e. achievable within joint limits) points would then be recorded.

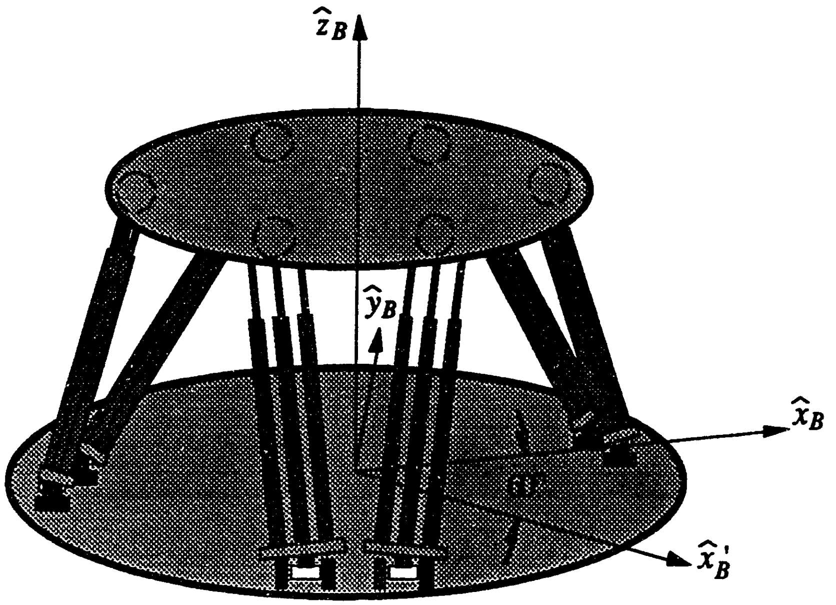


Figure 6.3.1: SPAM Stewart Platform Base Frames

Finally, the current SPAM Stewart Platform implementation does not attempt to address several important issues. Link interference avoidance, joint workspace constraints, stability near singularities and fault detection/handling are certainly valid concerns when operating a Stewart Platform. Handling of these concerns, however, may be difficult since current, known methods require substantial amount of computational power. If, however, alternative (i.e. less computationally demanding) routines can be found then it certainly is viable to attempt implementation.

6.3.4 Space Research Topics

Although the study recommendations given in the preceding sections are certainly valid topics, the underlying motivational factor for developing SPAM is neutral buoyancy space research. Since neutral buoyancy offers many advantages (e.g. fidelity, scale) over other ground-based space simulators, investigation in this environment is especially worthwhile. In fact, the neutral buoyancy tank for space research has been compared in importance to the wind-tunnel for aircraft development (Akin, 1988).

A great many issues await investigation in neutral buoyancy. Of prime importance for efficient human expansion into space is research focusing on enhancing man/machine productivity. Study on determining optimal methods of integrating humans and robots (autonomous, semi-autonomous, and teleoperated) will certainly have an impact on near term space activities, such as the construction of the Space Station. Additionally, research qualifying and quantifying the use of a manipulator arm for positioning tasks will influence design of future man/machine systems, such as maintenance and structural assembly vehicles. Finally, the successful use of SPAM may spur further research into novel mechanism designs and space system concepts which transcend the imagination.

6.4 Research Summary

The research described in this thesis has created a firm foundation for SPAM in terms of conceptual development, mechanical design, and implementation methodology. First, the SPAM concept was developed as a solution to neutral buoyancy SRMS simulation problems. By decoupling the system design into a three degree-of-freedom, revolute joint, manipulator arm augmented by a six degree-of-freedom, parallel-link, micromanipulator, the SPAM system is able to provide fine tip positioning as well as producing large forces during neutral buoyancy operation.

Second, extensive mechanical engineering has resulted in useful, efficient mechanisms. The development of PHAD has resulted in a new neutral buoyancy actuator with desirable performance characteristics at low cost. The manipulator arm revolute joints, in spite of some inherent design problems, have proven to be extremely robust and reliable during extensive testing. Using relatively simple servo controllers, the R-PHAD-A joints are able to produce high torque output and to achieve precise positioning. Additionally, the Stewart Platform prismatic joint (L-PHAD) is a very compact linear actuator capable of high precision and performance. Furthermore, the JELLO sensor offers a true linear sensor for underwater use with extremely fine position resolution.

Third, the design of advanced, high-level controllers and of an object-oriented, distributed processing system offers SPAM the potential for sophisticated and robust operations. Cartesian and joint-based manipulator controllers provide SPAM with SRMS simulation capabilities superior to other simulators and provide the capability for high-fidelity neutral-buoyancy research. Distributed processing through a parallel computer architecture allow SPAM, with nine physical degrees-of-freedom and substantial mathematical complexity, to achieve high system performance given limited computational resources.

The SPAM concept offers much potential for future SSL neutral buoyancy space research. By providing a robust and reliable manipulator, SPAM will be extremely useful in studies focusing on enhancing man/machine system productivity. In particular, tasks requiring fine positioning of humans and robots could benefit greatly from the use of SPAM. These include such exciting topics as investigation of human/telerobot task coordination, satellite servicing, and structural assembly. At this point, though, work remains to complete and successfully implement the *Stewart Platform Augmented Manipulator*. It is hoped, however, that the foundation created by this thesis will provide a solid base on which to build.

References

Abelson, H., and Sussman, G. J., *Structure and Interpretation of Computer Programs*, The MIT Press, Cambridge, Massachusetts, 1985.

Akin, D. L., "A Design Methodology for Neutral Buoyancy Simulation of Space Operations", *AIAA* paper 88-4628, 1988.

Asada, H., and Slotine, J.-J., *Robot Analysis and Control*, Wiley and Sons, New York, 1986.

Åstrom, K. J., and Wittenmark, B., *Computer Controlled Systems*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984.

Craig, John J., *Introduction to Robotics*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1986.

Dieudonne, J. E., Parrish, R.V., and Bardusch, R.E., *An Actuator Extension Transformation for a Motion Simulator and an Inverse Transformation Applying Newton-Raphson's Method*, NASA TN D-7067, Nov. 1972.

Fichter, E. F., "A Stewart Platform-Based Manipulator: General Theory and Practical Construction", *International Journal of Robotics Research*, Vol. 5, No. 2, Summer 1986.

Ismail, A. N., "The Design and Construction of a 6 Degree-of-Freedom Parallel-Link Platform Type Manipulator", SM Thesis, MIT Dept. of Mechanical Engineering, 1988.

Miller, R. H., Minsky, M. L., and Smith, D., *Space Applications of Automation, Robotics, and Machine Intelligence Systems (ARAMIS)*, NASA CR-162079, Aug. 1982.

Moore, J. H., *Building Scientific Apparatus*, Addison-Wesley Publishing Co., Redwood City, California, 1989.

Ogata, K., *Discrete-Time Control Systems*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1987.

Sanner, R. M., "The MIT SSL Pilot-Vehicle-Control Station Communications Protocol", (*draft*), Feb. 1990.

Stelman, N. M., "Design and Control of a 6 Degree-of-Freedom Platform with Variable Admittance", SM Thesis, MIT Dept. of Mechanical Engineering, 1988.

Stewart, D., "A Platform with Six Degrees of Freedom", *Proceedings of the Institution of Mechanical Engineers*, Vol. 180, 1965-66.

Ussher, T. H., and Doetsch, K. H., *An Overview of the Shuttle Remote Manipulator System*, NASA N85-16964, 1985.

Yang, D. C. H., and Lee, T. W., "Feasibility Study of a Platform Type of Robotic Manipulator from a Kinematic Viewpoint", *Journal of Mechanisms, Transmissions and Automation in Design*, Vol. 106, June 1984.

Appendix A
SPAM Electronics

This appendix contains schematics for the electronics used by the *Stewart Platform Augmented Manipulator*. All of the circuitry shown on these schematics has been breadboarded by wirewrapping at this time. Descriptions of each circuit follow:

Observer Board

The Observer Board was designed to asynchronously sample quadrature position signals and to generate IBM hardware interrupt requests. The design of the circuit centers on the Hewlett-Packard HCTL-2000 chip. The HCTL-2000 is a compact device (16-pin I.C.) which provides a low-cost, efficient means of decoding quadrature signals to a 12-bit counter value. Since this chip continually samples the two input channels, resolving quadrature phase shifts to counters changes, it is extremely useful for monitoring a position encoder.

IBM interrupt signals are generated with an Intel 8253 Programmable Interval Timer. The 8253, operating in mode #2 (Rate Generator), generates rising edges on the IBM PC-BUS IRQ2 line at regular intervals. It should be noted that the 8253 Rate Generator mode produces low pulses only one clock period wide, meaning the IRQ2 line is high for most of the time. This is not a problem since the 8259 interrupt controller used by the PC is only triggered by rising edges. Additionally, since the interrupt signals are wired to IRQ2, this board probably will NOT run on an PC/AT-bus since the AT uses IRQ2 to cascade 8259's.

To utilize the Observer Board, do the following:

1) Install the board in an IBM PC-BUS slot.
2) Connect the two quadrature input and ground lines to a quadrature generating device (e.g. optical encoder).
3) Run the QUADOBS software described in Appendix C.

Since the 8253 is driven by a 1 MHz oscillator and has a 16-bit internal countdown register, the slowest sampling rate that may be requested is 15.23 Hz. The upper limit on sampling rate for a PC/XT running at 4.77 MHz is approximately 2 KHz. If quadrature position is displayed on the screen, sampling rates should be limited to 1 KHz, otherwise the system will operate VERY SLOWLY (due to tremendous overloading of the DOS interrupt handler).

2-Joint Controller

This circuit is intended to control 1 or 2 SPAM R-PHAD or L-PHAD type joints and to read quadrature signals from 1 or 2 optical encoders. To operate a SPAM joint, control signals for 4 solenoid valves, two air (Koganei 110-4E1-12VDC) and two water (Laketown or Kip), must be generated. The air solenoids require a binary signal, commanding ON/OFF state, whereas the water solenoids need PWM signals. To produce the necessary signals, two Intel 8255's and a 4-bit PWM generator (based on an '85 and '163) are used. Decoded of quadrature input signals is performed with two HCTL-2000 chips.

To utilize the 2-Joint Controller, do the following:

1) Install the board in an IBM PC or PC/AT bus slot.
2) Connect a 26 line ribbon cable from the board's ribbon cable header to the Manipulator Hookup Board (MHB).
3) Hookup power to the MHB.
4) Run the SPAM, PHAD or SPAM2 software described in Appendix C.

There are three important things to note. First, the addressing logic is setup such that the board appears at IBM I/O port addresses 0x3E0 to 0x3E7. Secondly, the PWM generator is driven by the output of a '555 timer circuit (555-2 on schematic). Because of this, the PWM duty cycle frequency can be varied from 0 to approximately 100 Hz. simply by changing resistors (or utilizing a poteniometer). Finally, the software listed in (4) above (as currently written) look for a PWM synch signal on 8255-2 PC4 line (pin 13) to determine control loop frequency. As a result, this line cannot be disconnected without rewriting part of the software (specifically, the routine "SYNCH").

Manipulator Hookup Board (MHB)

This circuit's primary function is to provide transformation of solenoid control signals from TTL to 12 VDC levels. The actual signal transformation is performed with transistor (TIP 31 and MJ11028) switching. LED's are used to indicate active (12 VDC) solenoid lines. The secondary function of the board is to provide a regulated 5 VDC supply to three optical encoder lines. This is accomplished with a 7805 voltage regulator connected to the 12 VDC supplied by battery #1.

To utilize the MHB, do the following:

1) Connect a 26 line ribbon cable between the MHB and controller board (e.g. 2-Link Controller).
2) Connect one to three 12 VDC batteries to the board via the banana plugs. (Note: Battery #1 MUST be connected to generate optical encoder power).

Solenoid End Plug

The two Koganei solenoids used by each manipulator arm joint (R-PHAD-A) are mounted on a PVC plug. This plug is then used to seal the flotation tube of an arm boom. The schematic shows the wiring of between the end plug 5-pin Amphenol (AMP) connector, the two Koganei 110-4E1-12VDC solenoids, and the 4-pin SureSeal.

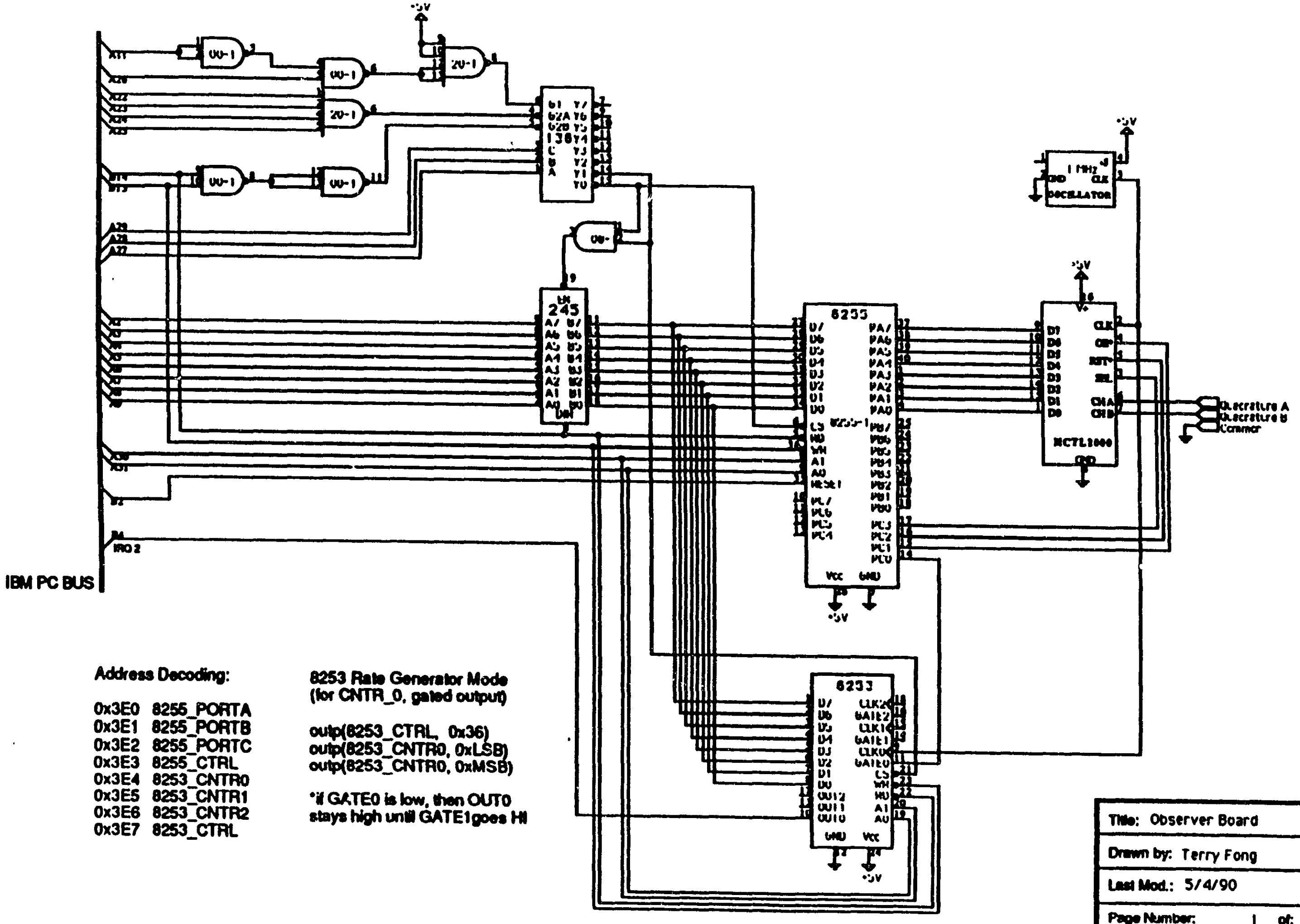
NMIS-0021 RS-422 XCVR

Although this circuitry appears in the New Micro's "100 Squared" system documentation, and is intended for use with the "100 Squared" microcomputer, it works perfectly with the "2x4" microcomputer as well. The function of the circuit, as might be inferred from the name, is to provide transmission/reception of RS-422 serial communications signals. To utilize the RS-422 XCVR, connect the *SO* line to the "2x4" *SI* line and the *SI* line to the "2x4" *SO* line. RS-422 receive and transmit lines should then be connected to the appropriate lines. For more details, consult the "100 Squared" system documentation or the TI databook for the 75176 chip.

NMIS-0021 Quadrature Decoder

This circuit is designed to be constructed on a New Micro's NMIS-0001 prototype expansion board and to provide a NMIS-0021 (New Micro's "2x4" microcomputer) with the capability to decode three quadrature input signals. The design utilizes two '139 decoders and three HCTL-2000 chips. Though the circuitry is straightforward, verification of successful function has not been accomplished at this time.

Observer Board



Address Decoding:

0x3E0	8255_PORTA
0x3E1	8255_PORTB
0x3E2	8255_PORTC
0x3E3	8255_CTRL
0x3E4	8253_CNTR0
0x3E5	8253_CNTR1
0x3E6	8253_CNTR2
0x3E7	8253_CTRL

8253 Rate Generator Mode
(for CNTR_0, gated output)

outp(8253_CTRL, 0x36)
outp(8253_CNTR0, 0xLSB)
outp(8253_CNTR0, 0xMSB)

*if GATE0 is low, then OUT0
stays high until GATE1goes HI

Title: Observer Board

Drawn by: Terry Fong

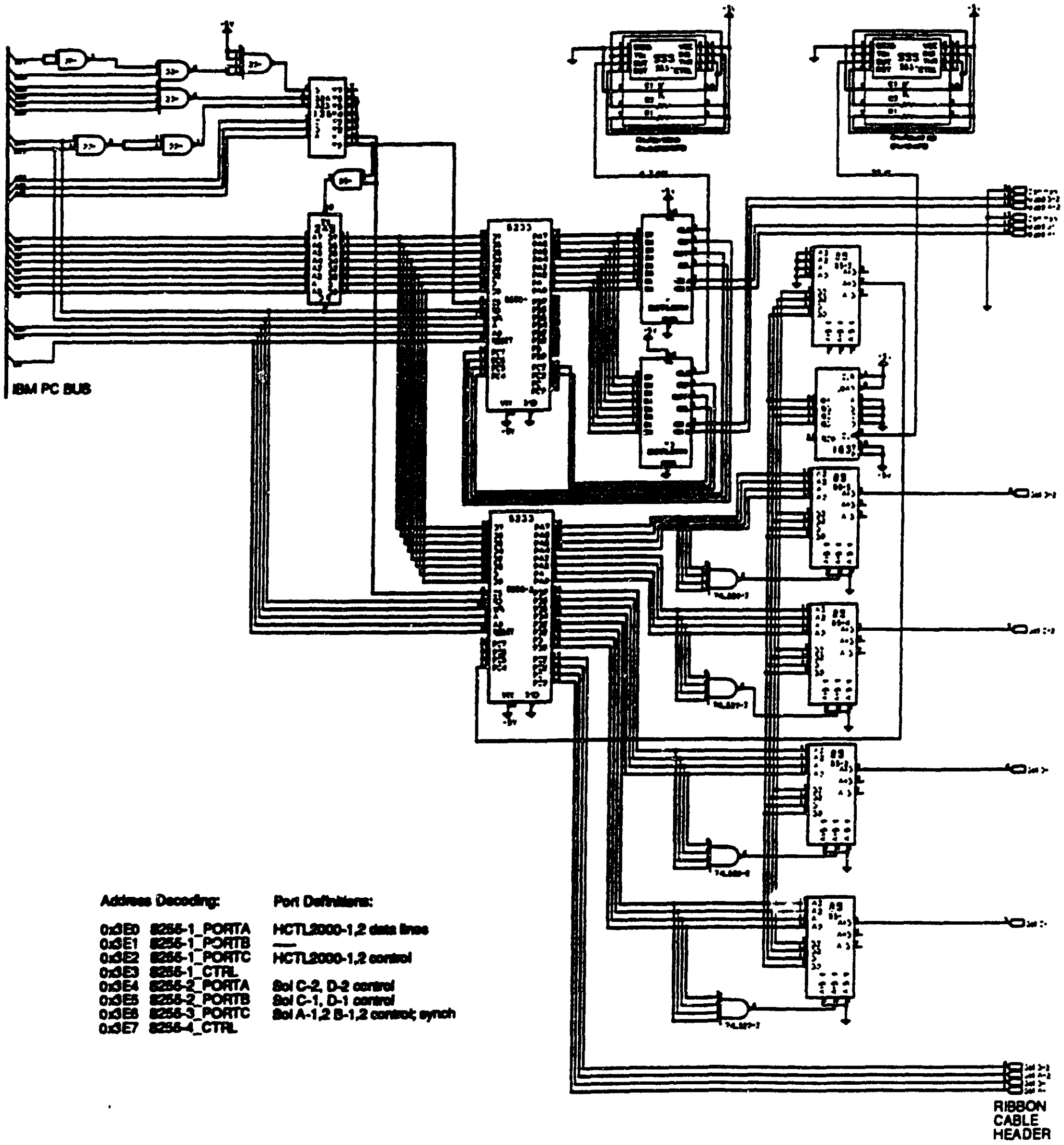
Last Mod.: 5/4/90

Page Number: 1 of: 1

Project SPAM

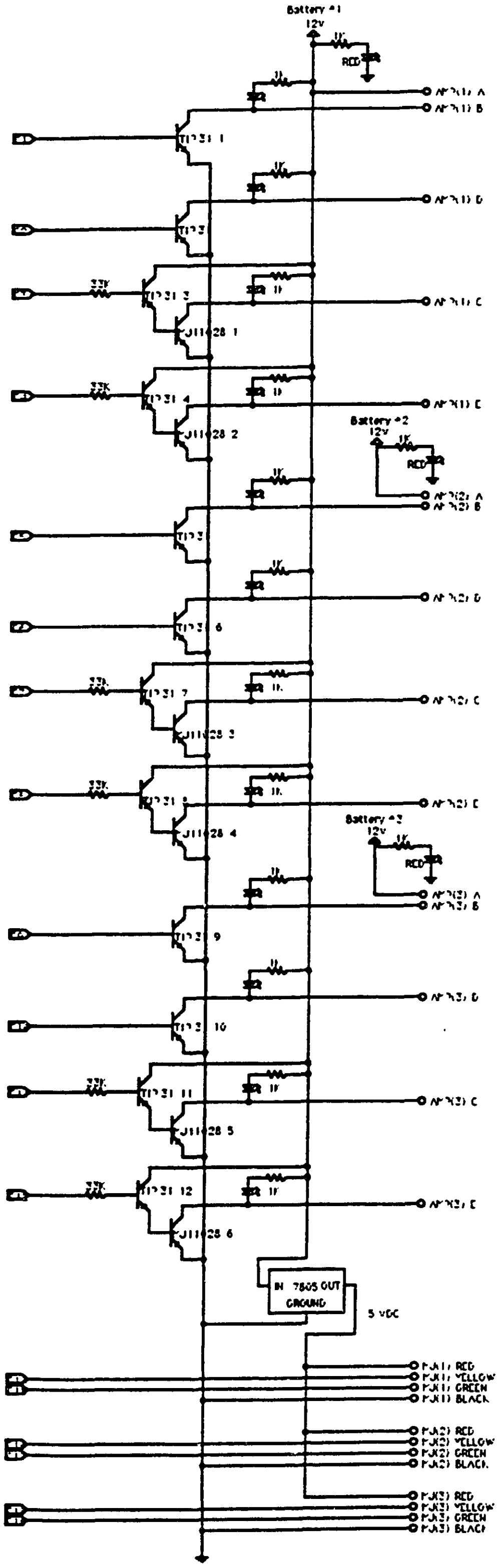
MIT Space Systems Laboratory

Title: 2-Joint Controller	
Drawn by: Terry Fong	
Last Mod.: 5/4/90	
Page Number:	1 of 1
Project SPAM MIT Space Systems Laboratory	



Address Decoding:		Port Definitions:
0x3E0	8255-1_PORTA	HCTL2000-1,2 data lines
0x3E1	8255-1_PORTB	——
0x3E2	8255-1_PORTC	HCTL2000-1,2 control
0x3E3	8255-1_CTRL	
0x3E4	8255-2_PORTA	Sol C-2, D-2 control
0x3E5	8255-2_PORTB	Sol C-1, D-1 control
0x3E6	8255-3_PORTC	Sol A-1,2 B-1,2 control; synch
0x3E7	8255-4_CTRL	

Manipulator Hookup Board



Title. Manipulator Hookup Board
Drawn by: Terry Fong
Last Mod : 5/4/90
Page Number: 1 of: 1
Project SPAM ***MIT Space Systems Laboratory***

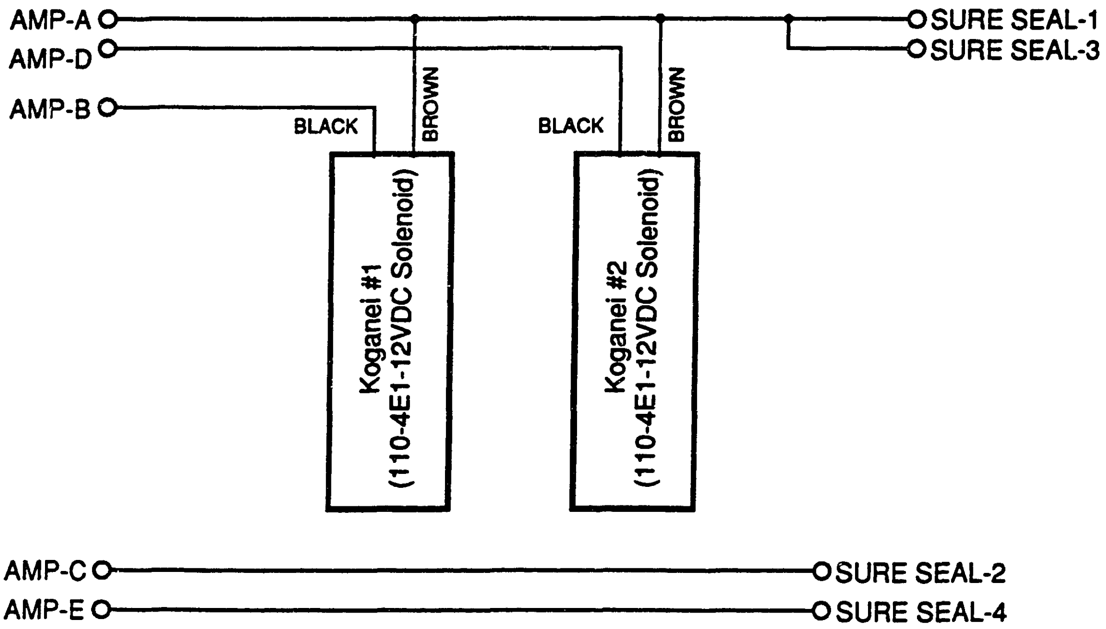
CONNECTOR LEGEND

AMP(1)-A= Amphenol #1, pin A
MJ(1)-RED = Modular phone jack, #1, RED line
RC1 = Ribbon cable connector, pin #1

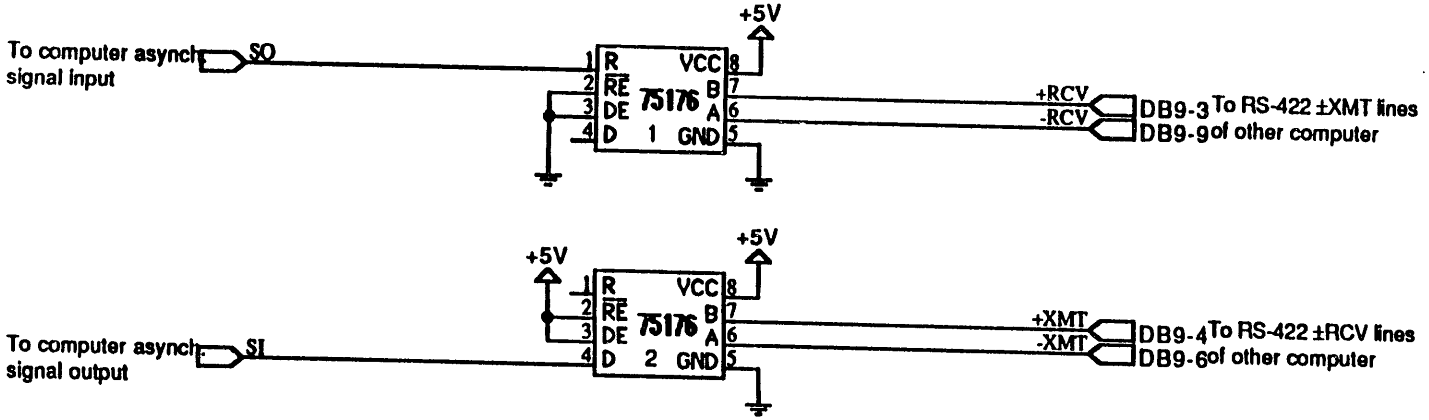
*All LED's are green unless otherwise marked

Solenoid Block End Plug

Title: Solenoid End Plug	
Drawn by: Terry Fong	
Last Mod.: 5/4/90	
Page Number:	1 of: 1
Project SPAM ***MIT Space Systems Laboratory***	



NMIS-0021 RS-422 XCVR


To computer asynch signal input
SO
To computer asynch signal output
SI
+5V
+5V
+5V
R
RE
DE
D
VCC
B
A
GND
75176
1
2
1
2
3
4
5
6
7
8
+RCV
-RCV
+XMT
-XMT
DB9-3 To RS-422 ±XMT lines
DB9-9 of other computer
DB9-4 To RS-422 ±RCV lines
DB9-6 of other computer

SI = Asynch. Signal In
SO=Asynch. Singal Out

±RCV = RS-422 Receiver Pair
±XMT = RS-422 Transmit Pair

Note: labels are given w.r.t. the transceiver circuit

Title: NMIS-0021 RS-422 XCVR

Drawn by: Terry Fong

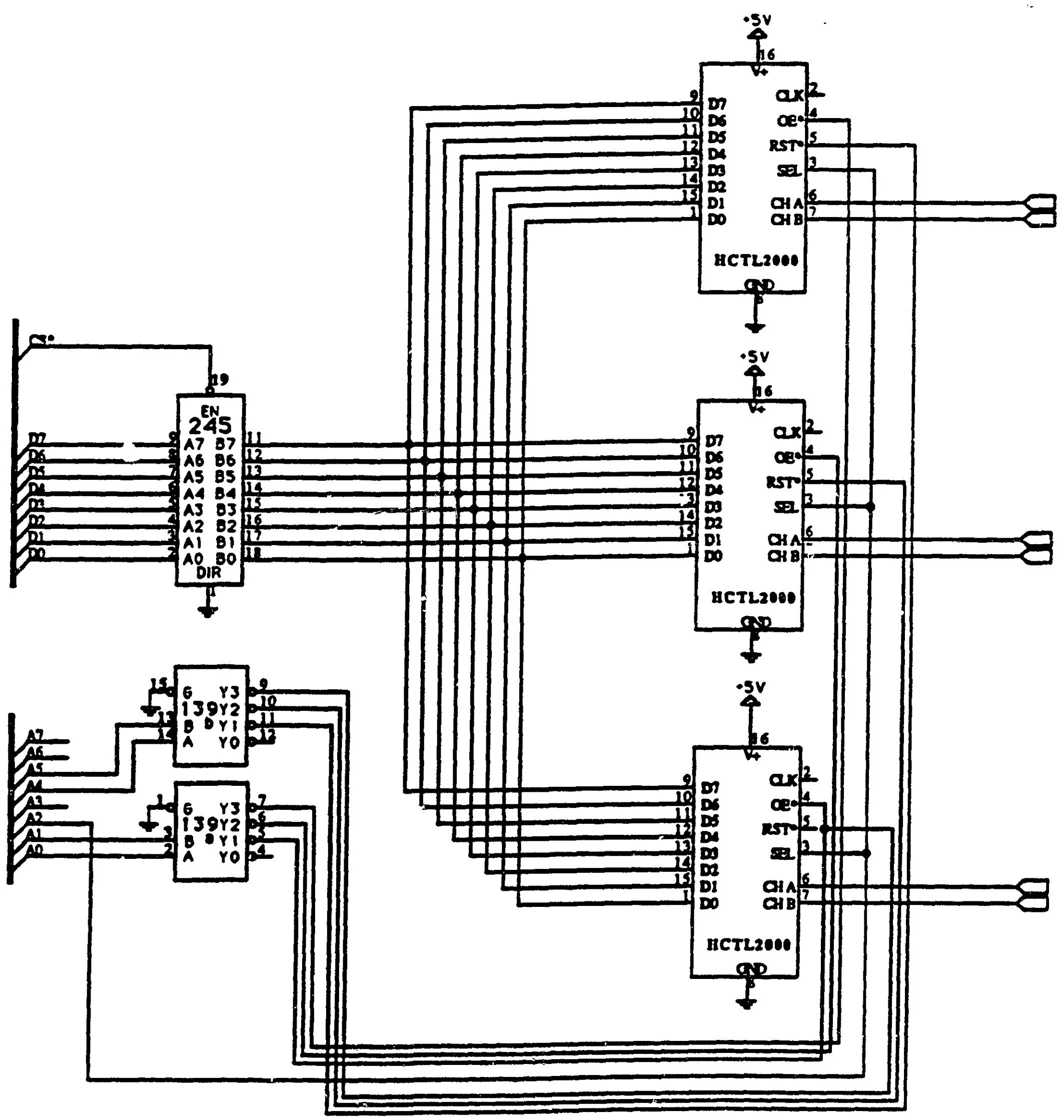
Last Mod.: 5/4/90

Page Number: 1 of: 1

Project SPAM
MIT Space Systems Laboratory

NMIS-0021 Quadrature Decoder

Title: NMIS-0021 Quad. Decoder
Drawn by: Terry Fong
Last Mod.: 5/4/90
Page Number: 1 of: 1
Project SPAM *MIT Space Systems Laboratory*



Appendix B
SPAM Hardware

This appendix contains design drawings for the *Stewart Platform Augmented Manipulator*. All dimensions are given in English Standard decimal and fractional units. The drawings are categorized into the following six groups:

R-PHAD-A Schematics and detailed drawings for the R-PHAD-A used for the SPAM Manipulator Arm shoulder and elbow joints.

R-PHAD-B Schematics and detailed drawings for the R-PHAD-B intended for the SPAM Manipulator Arm waist joint.

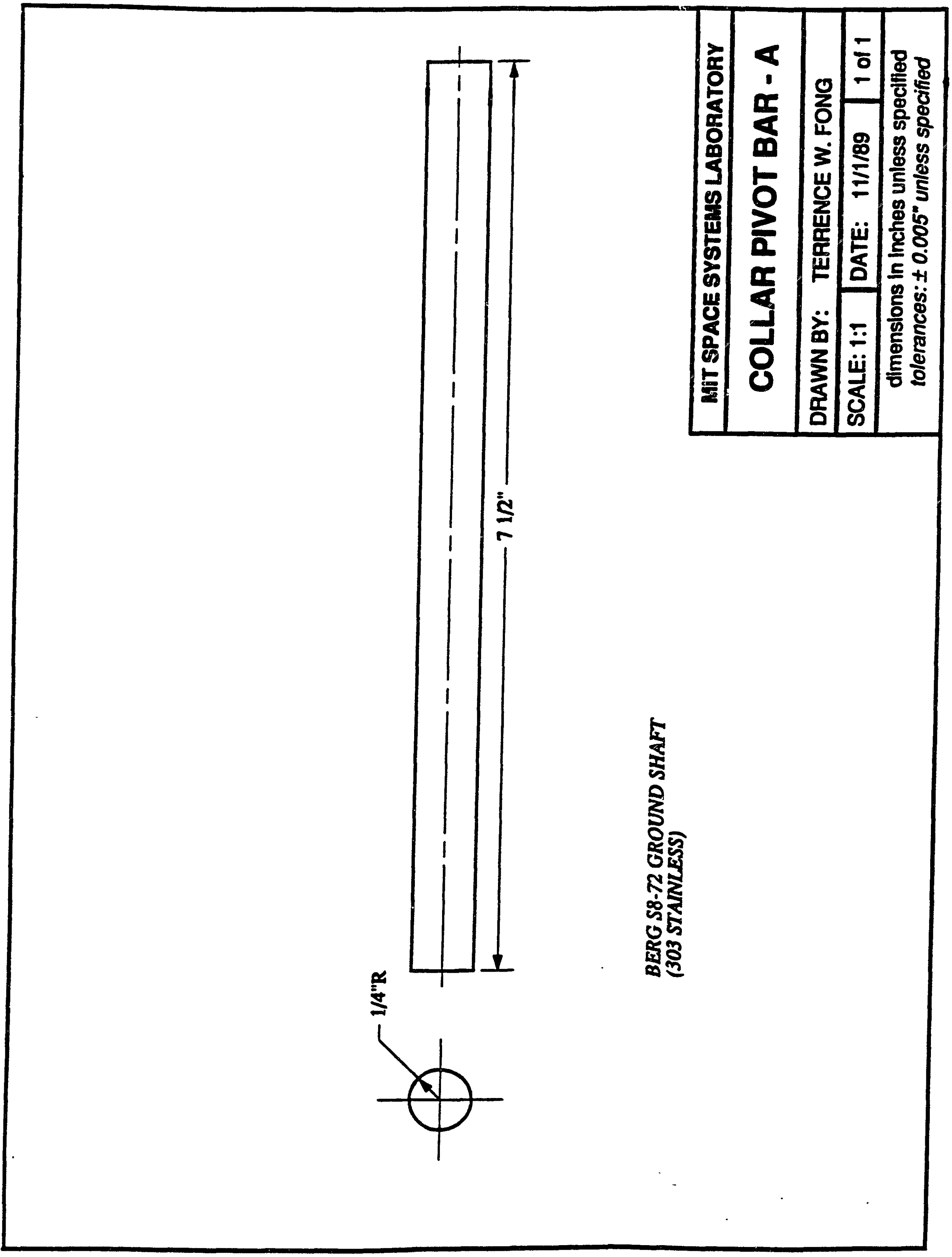
Arm Booms Schematics and detailed drawings for the arm boom tubes used in the SPAM Manipulator Arm.

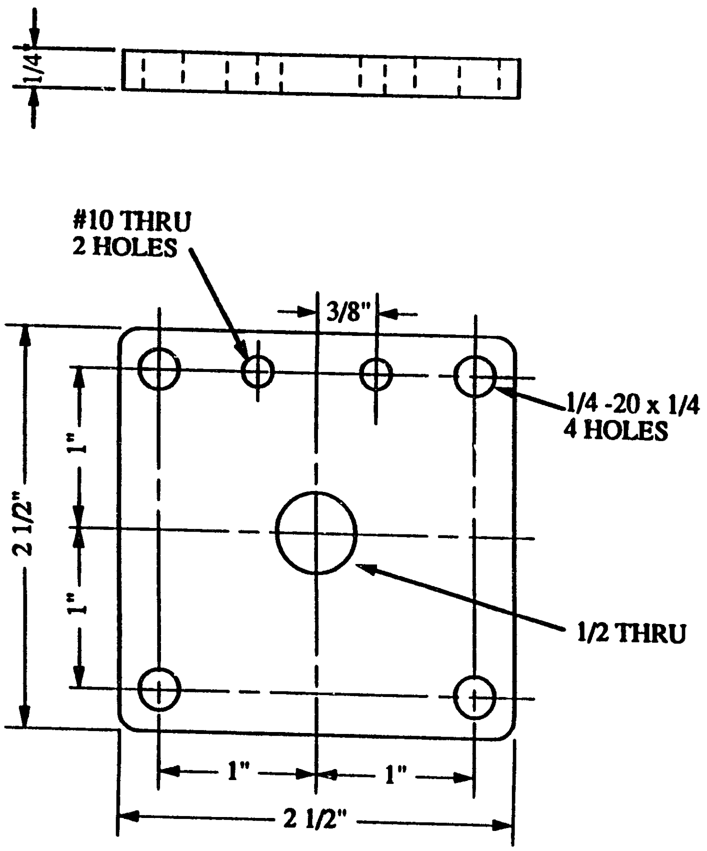
Manual Joint Detailed drawings for the adjustable hinge used as the SPAM Stewart Platform/Manipulator Arm coupling

Universal Joint Schematics and detailed drawings for the Universal Joint used in the SPAM Stewart Platform.

Stewart Platform Schematics for the layout of the SPAM Stewart Platform.

All drawings were current as of 6 May 1990.

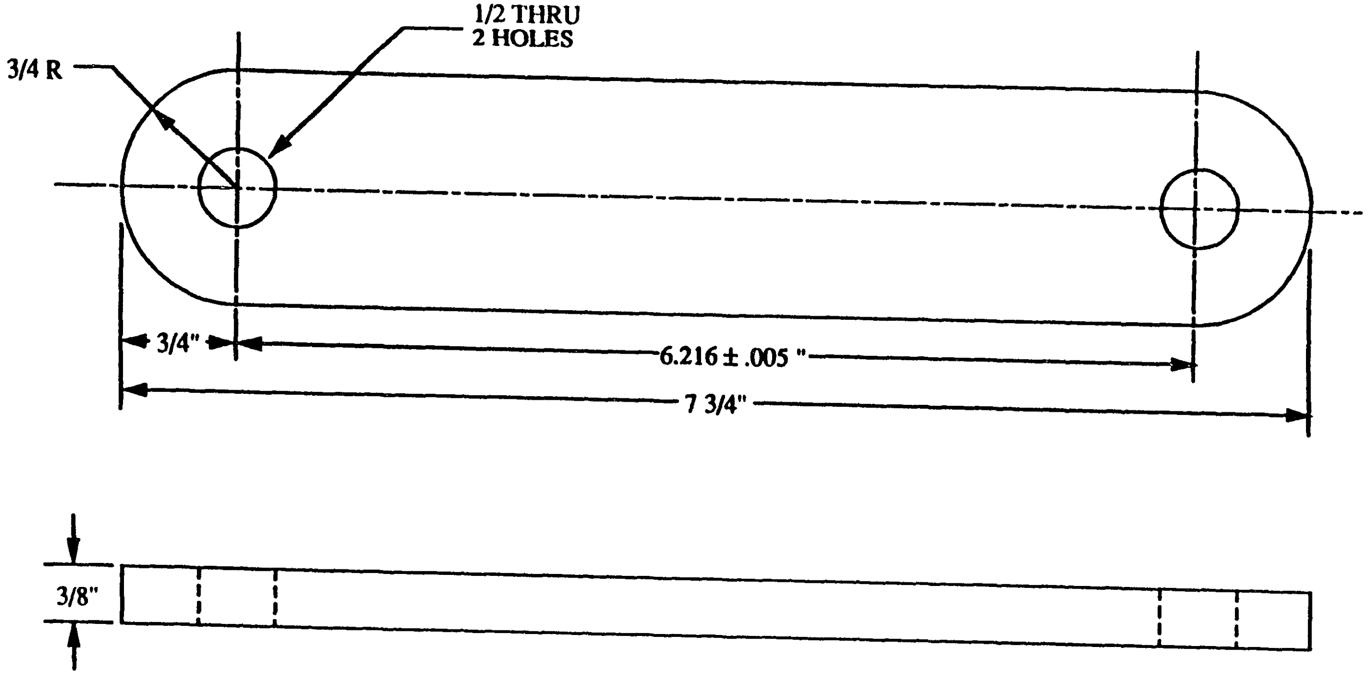

MIT SPACE SYSTEMS LABORATORY
COLLAR PIVOT BAR - A
DRAWN BY: TERRENCE W. FONG
SCALE: 1:1
DATE: 11/1/89
1 of 1
dimensions in inches unless specified
tolerances: ± 0.005" unless specified
7 1/2"
1/4"R
BERG S8-72 GROUND SHAFT
(303 STAINLESS)



FABRICATE USING AL2023-T4
BLACK ANODIZE

ROUND ALL EDGES 1/8 R

MIT SPACE SYSTEMS LABORATORY		
ENCODER SUPPORT FLANGE		
DRAWN BY: TERRENCE W. FONG		
SCALE: 1:1	DATE: 11/1/89	1 of 1
dimensions in inches unless specified *tolerances: ± 0.005" unless specified*		



FABRICATE USING AL 2023 - T4
BLACK ANODIZE

ROUND ALL EDGES 1/8 R

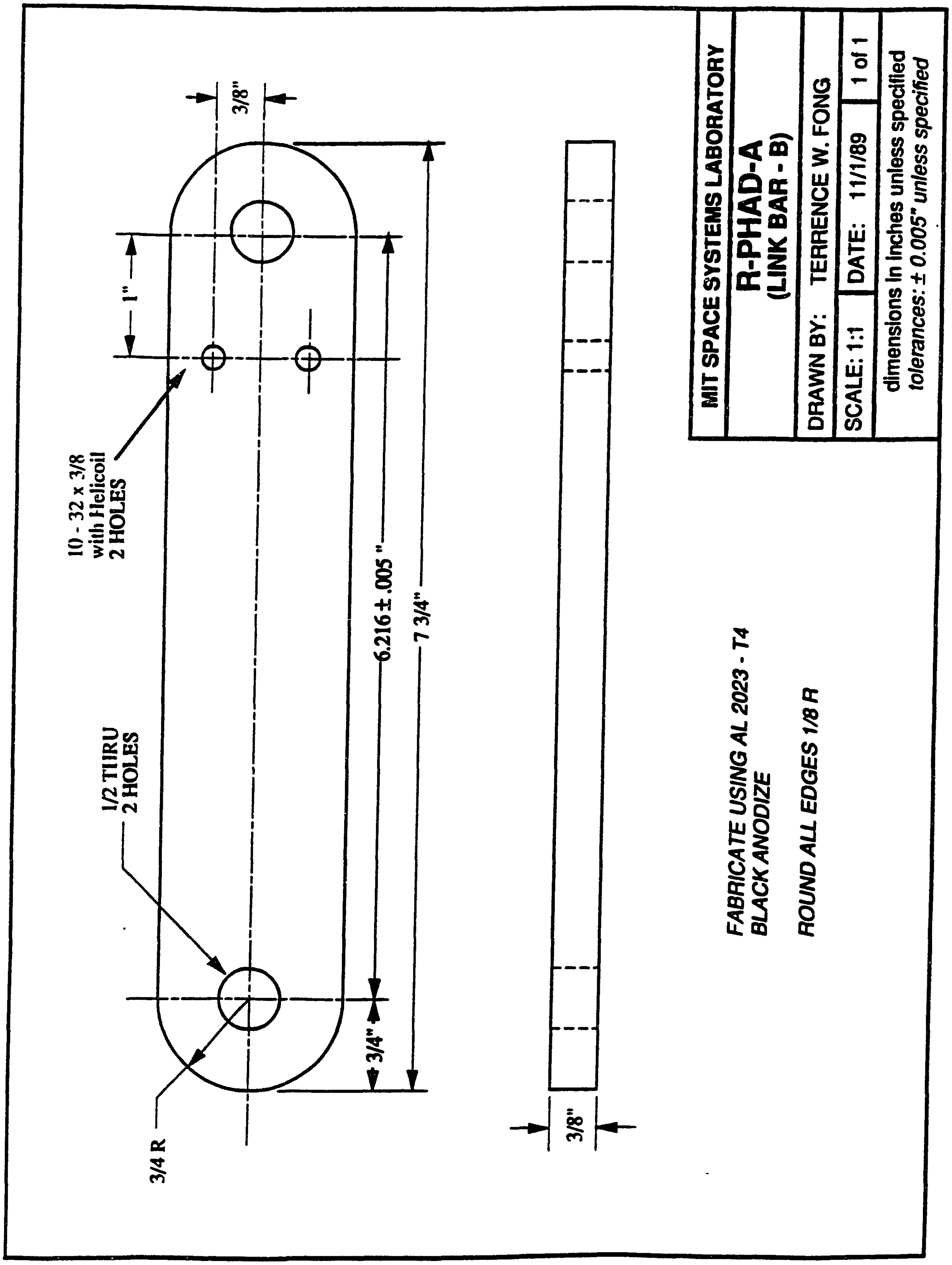
MIT SPACE SYSTEMS LABORATORY

R-PHAD-A
(LINK BAR - A)

DRAWN BY: TERRENCE W. FONG

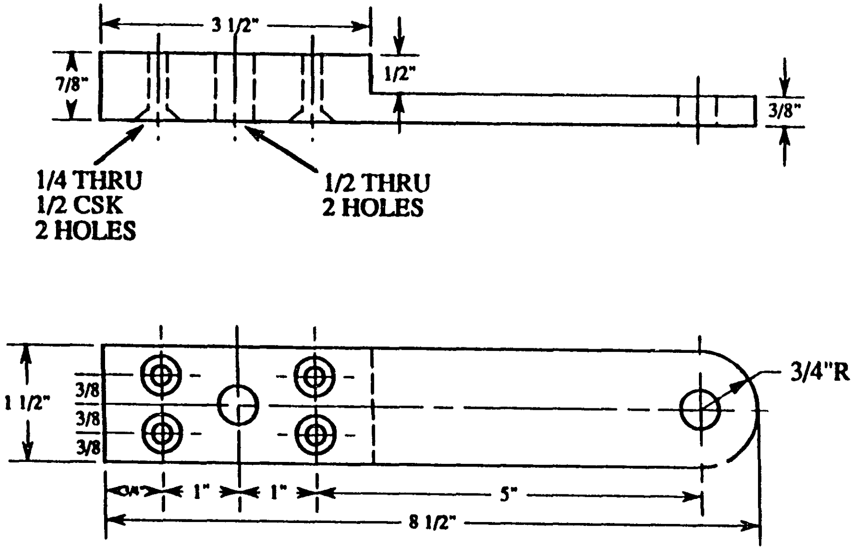
SCALE: 1:1 DATE: 11/1/89 1 of 1

dimensions in inches unless specified
tolerances: ± 0.005" unless specified



FABRICATE USING AL 2023 - T4
BLACK ANODIZE

ROUND ALL EDGES 1/8 R



FABRICATE USING AL2023-T4
BLACK ANODIZE

ROUND ALL EDGES 1/8 R

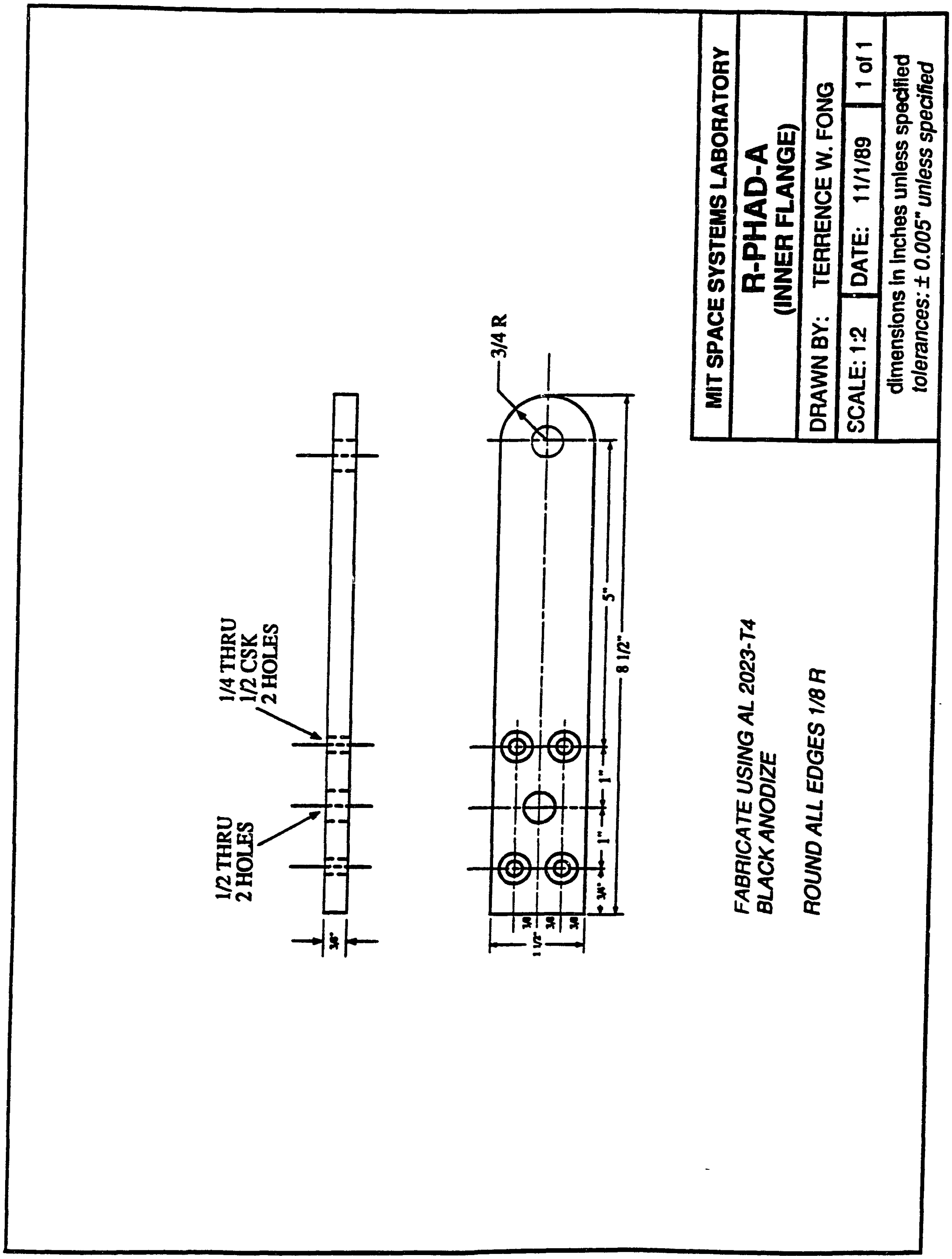
MIT SPACE SYSTEMS LABORATORY

R-PHAD-A
(OUTER FLANGE)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:2 | DATE: 11/2/89 | 1 of 1

dimensions in inches unless specified
tolerances: ± 0.005" unless specified



FABRICATE USING AL 2023-T4
BLACK ANODIZE

ROUND ALL EDGES 1/8 R

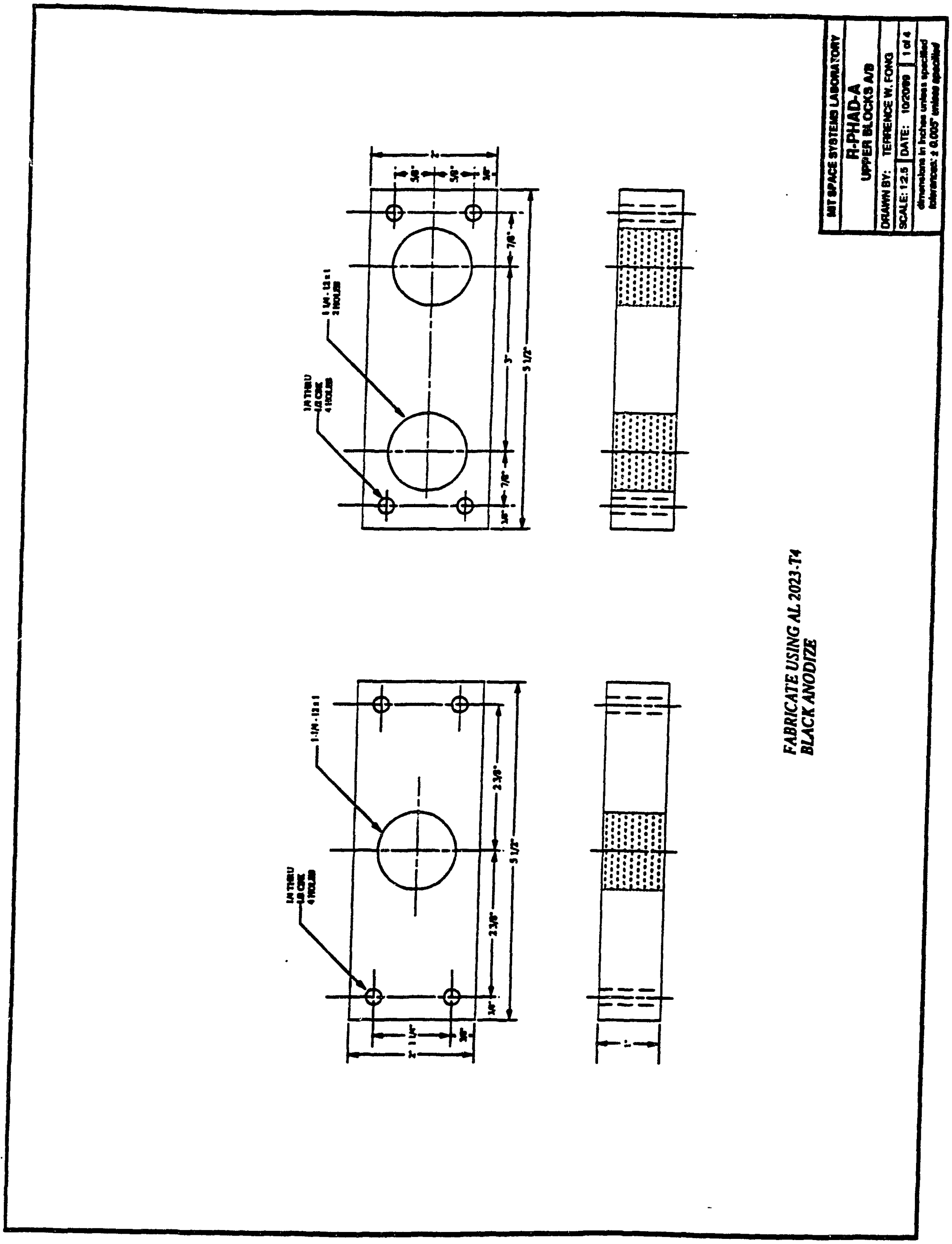
MIT SPACE SYSTEMS LABORATORY

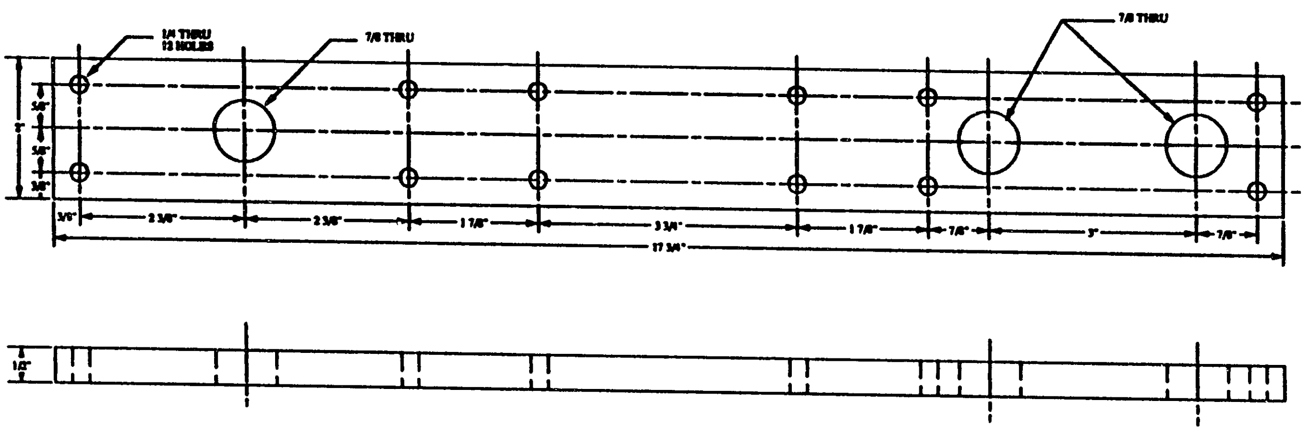
R-PHAD-A
(INNER FLANGE)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:2 | DATE: 11/1/89 | 1 of 1

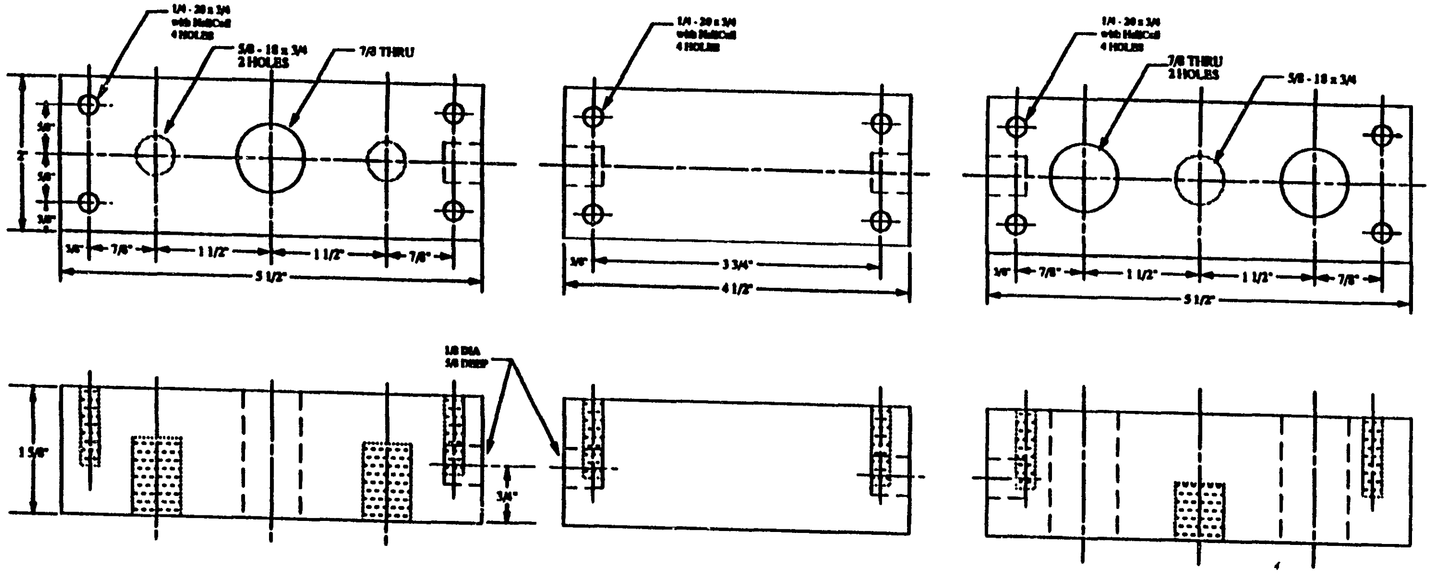
dimensions in inches unless specified
tolerances: ± 0.005" unless specified


MIT SPACE SYSTEMS LABORATORY
R-PHAD-A
UPPER BLOCKS A/B
DRAWN BY: TERRENCE W. FONG
SCALE: 1:2.5
DATE: 10/20/89
1 of 4
dimensions in inches unless specified
tolerances: ± 0.005" unless specified
FABRICATE USING AL 2023-T4
BLACK ANODIZE
1/4 THRU
1/2 CBE
4 HOLES
1 1/4 - 12 x 1
2 HOLES
2"
5/8"
5/8"
3/8"
7/8"
3"
7/8"
3/8"
5 1/2"
1 1/4 - 12 x 1
2 3/8"
2 3/8"
5 1/2"
2"
1 1/4"
3/8"
1"



NOTE: USE 314 STAINLESS STEEL

MIT SPACE SYSTEMS LABORATORY
R-PHAD-A
UPPER STEELSUPPORT
DRAWN BY: TERRENCE W. FONG
SCALE 1:2.5 | DATE: 10/26/88 | 2 of 4
dimensions in inches unless specified
tolerances: ± 0.005" unless specified



FABRICATE USING AL 2023-T4
BLACK ANODIZE

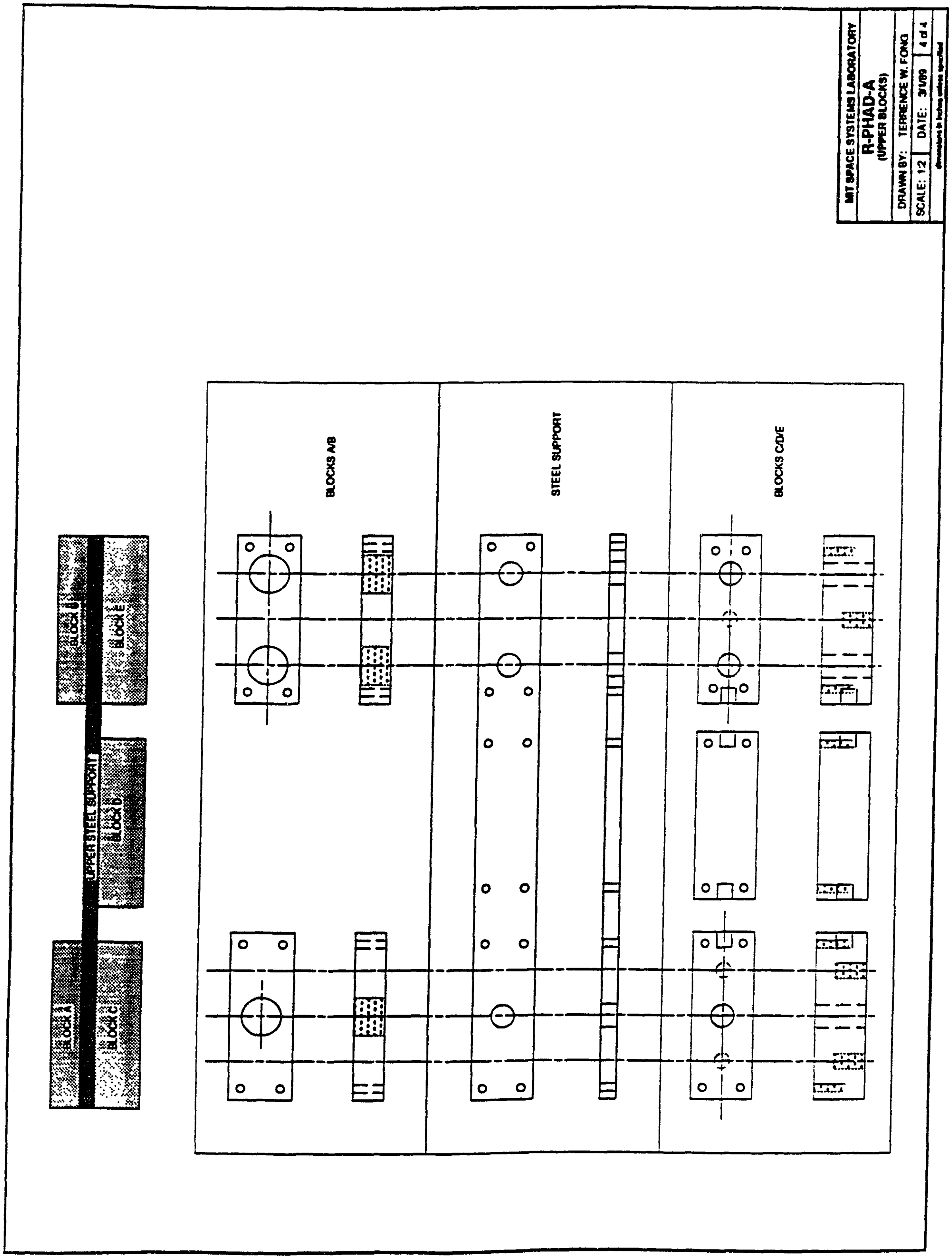
MIT SPACE SYSTEMS LABORATORY

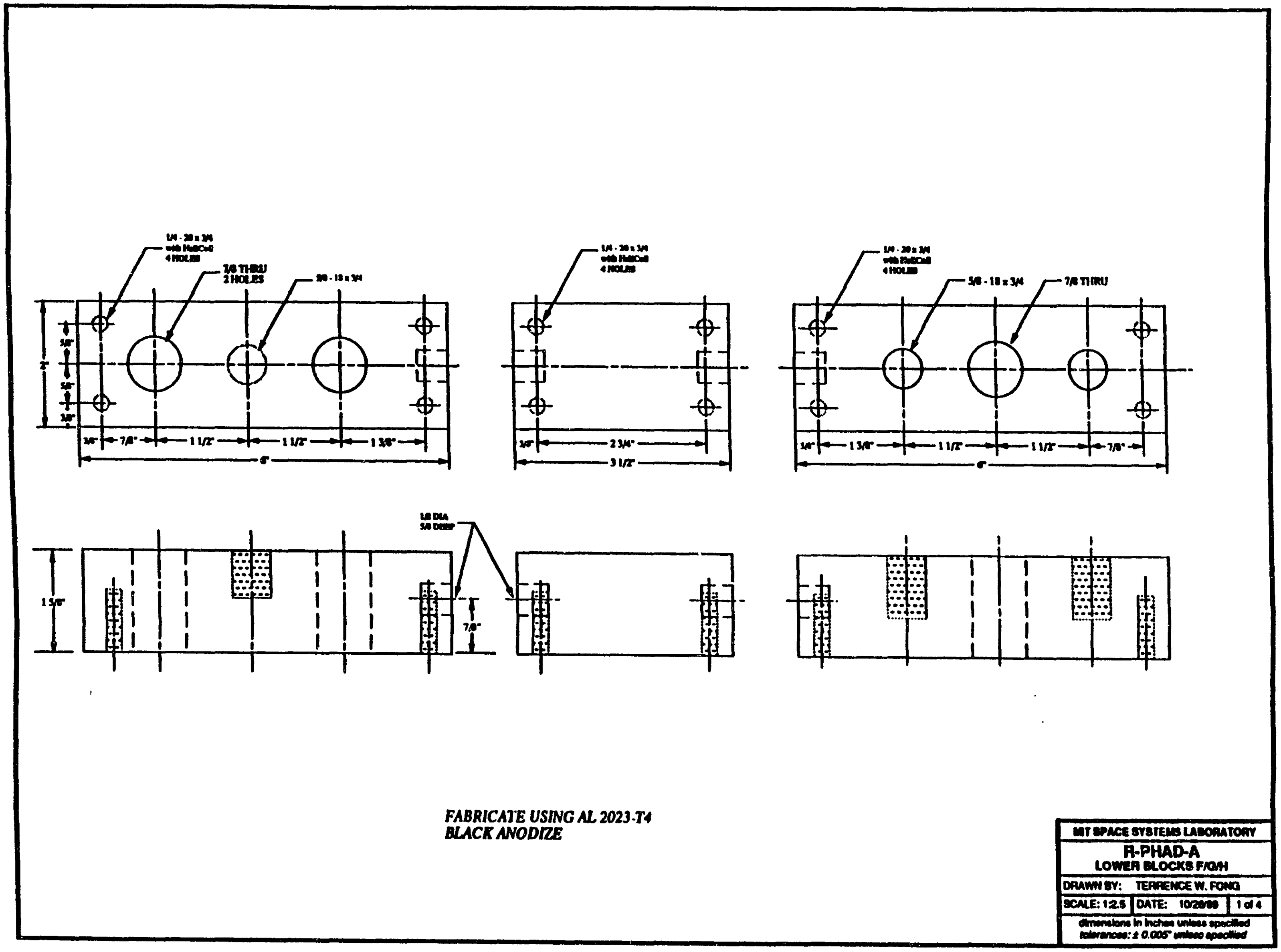
R-PHAD-A
UPPER BLOCKS C/D/E

DRAWN BY: TERRENCE W. FONG

SCALE: 1:2.5 | DATE: 10/26/89 | 3 of 4

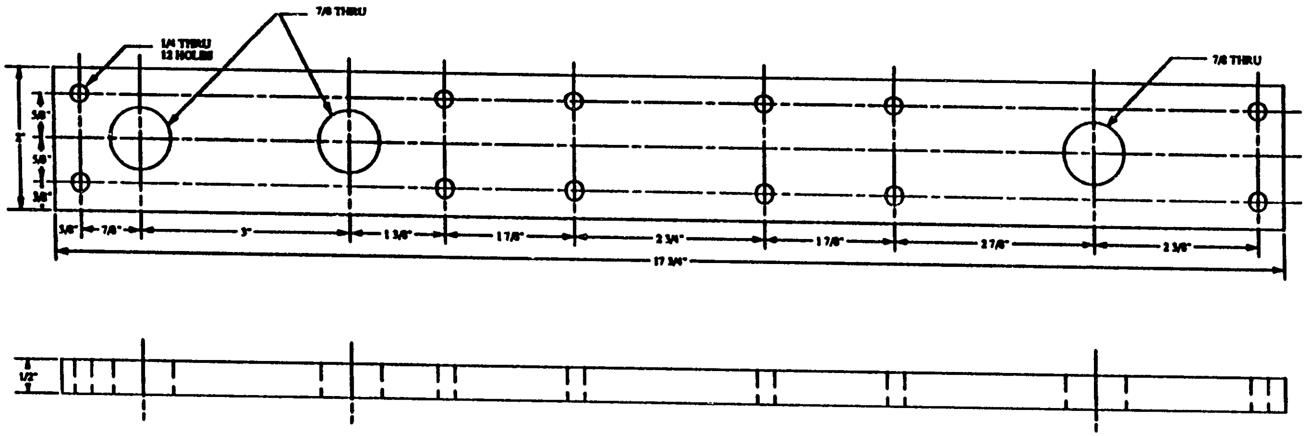
dimensions in inches unless specified
tolerances: ± 0.005" unless specified


MIT SPACE SYSTEMS LABORATORY
R-PHAD-A
(UPPER BLOCKS)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:2
DATE: 3/1/89
4 of 4
dimensions in inches unless specified
BLOCKS A/B
STEEL SUPPORT
BLOCKS C/D/E
BLOCK A
BLOCK B
UPPER STEEL SUPPORT
BLOCK C
BLOCK D
BLOCK E



FABRICATE USING AL 2023-T4
BLACK ANODIZE

MIT SPACE SYSTEMS LABORATORY
R-PHAD-A
LOWER BLOCKS F/G/H
DRAWN BY: TERRENCE W. FONG
SCALE: 1:2.5 DATE: 10/26/89 1 of 4
dimensions in inches unless specified
tolerances: ± 0.005" unless specified



NOTE: USE 314 STAINLESS STEEL

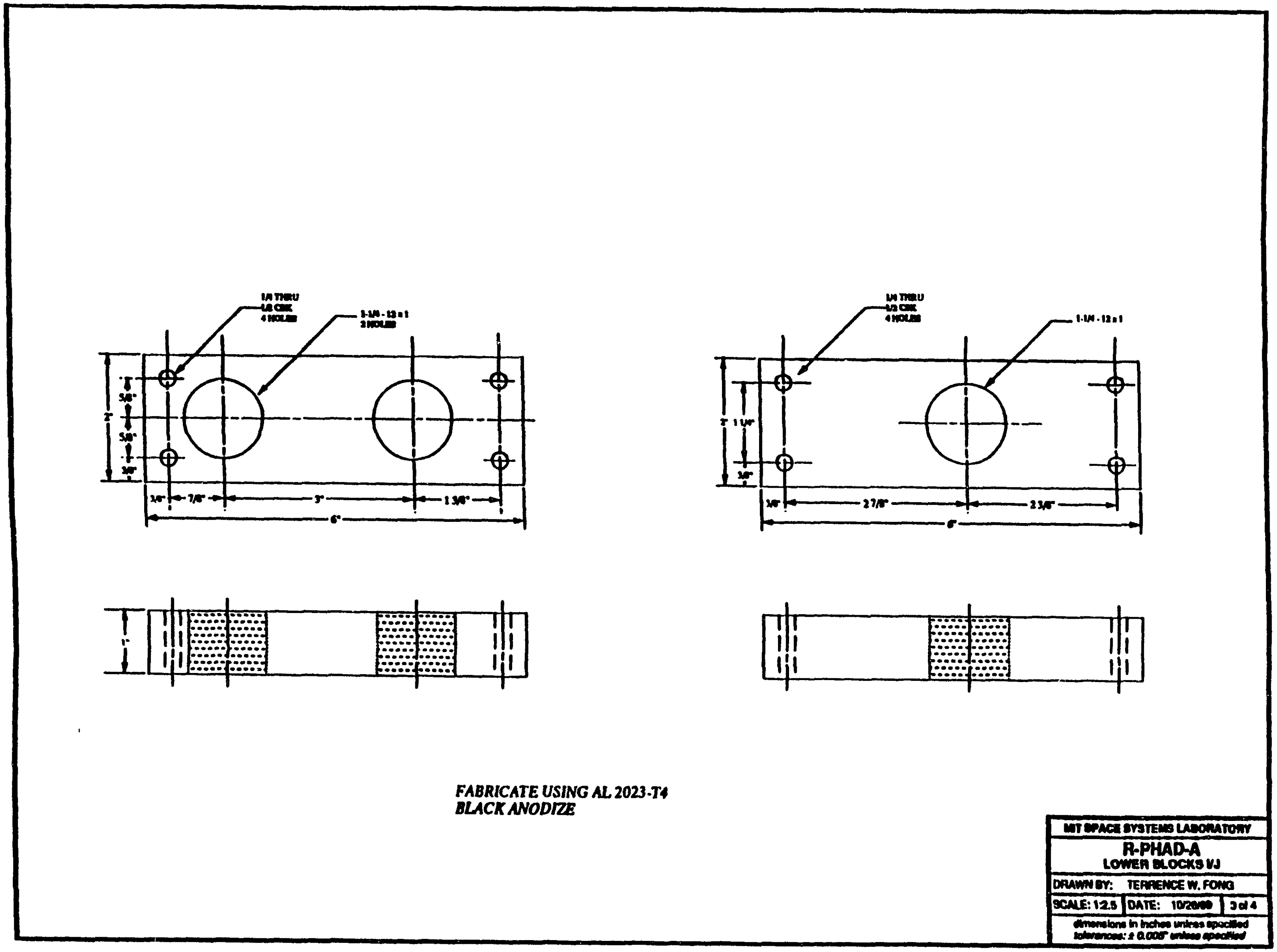
MIT SPACE SYSTEMS LABORATORY

R-PHAD-A
LOWER STEELSUPPORT

DRAWN BY. TERRENCE W FONG

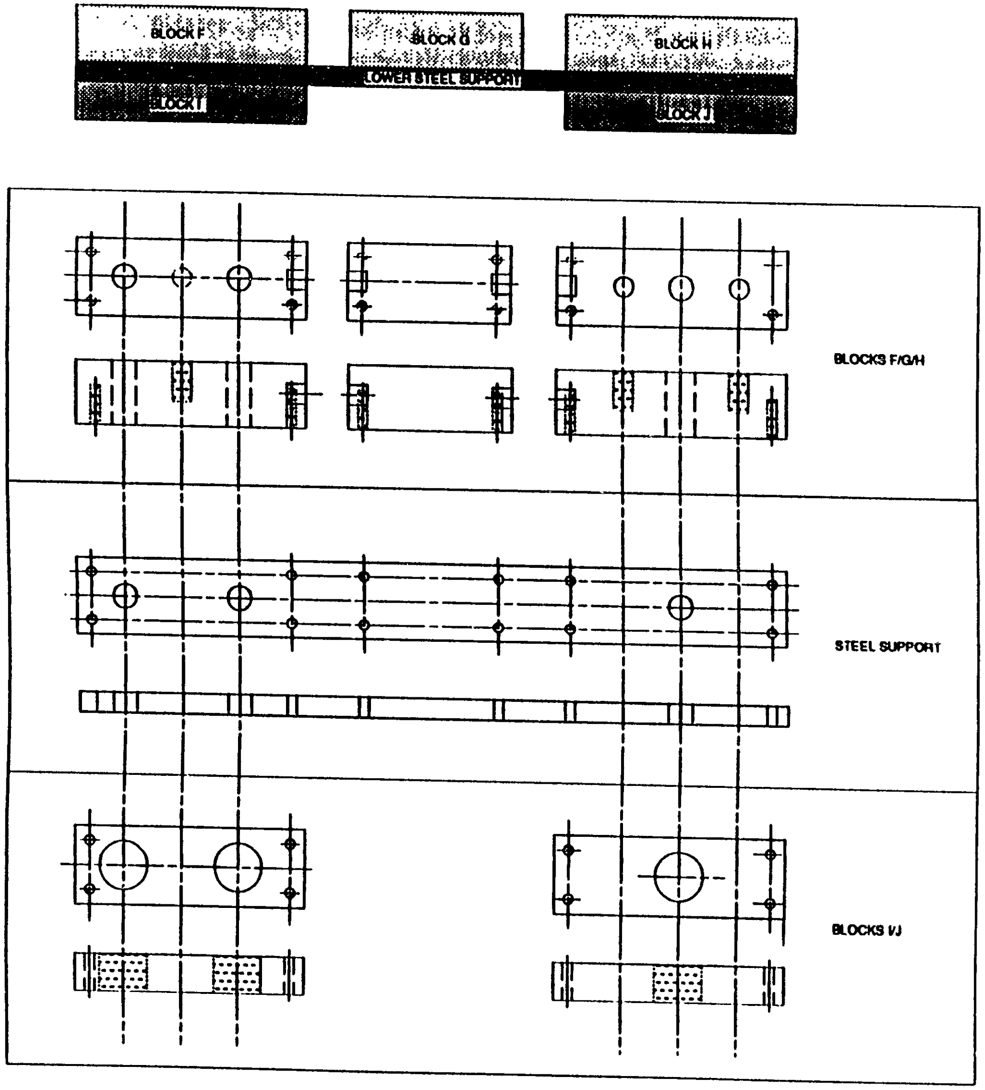
SCALE. 1 2 5 | DATE 10/26/88 | 2 of 4

dimensions in inches unless specified
tolerances: ± 0.005" unless specified



FABRICATE USING AL 2023-T4
BLACK ANODIZE

MIT SPACE SYSTEMS LABORATORY
R-PHAD-A
LOWER BLOCKS I/J
DRAWN BY: TERRENCE W. FONG
SCALE: 1:2.5 | DATE: 10/26/89 | 3 of 4
dimensions in inches unless specified
tolerances: ± 0.005" unless specified

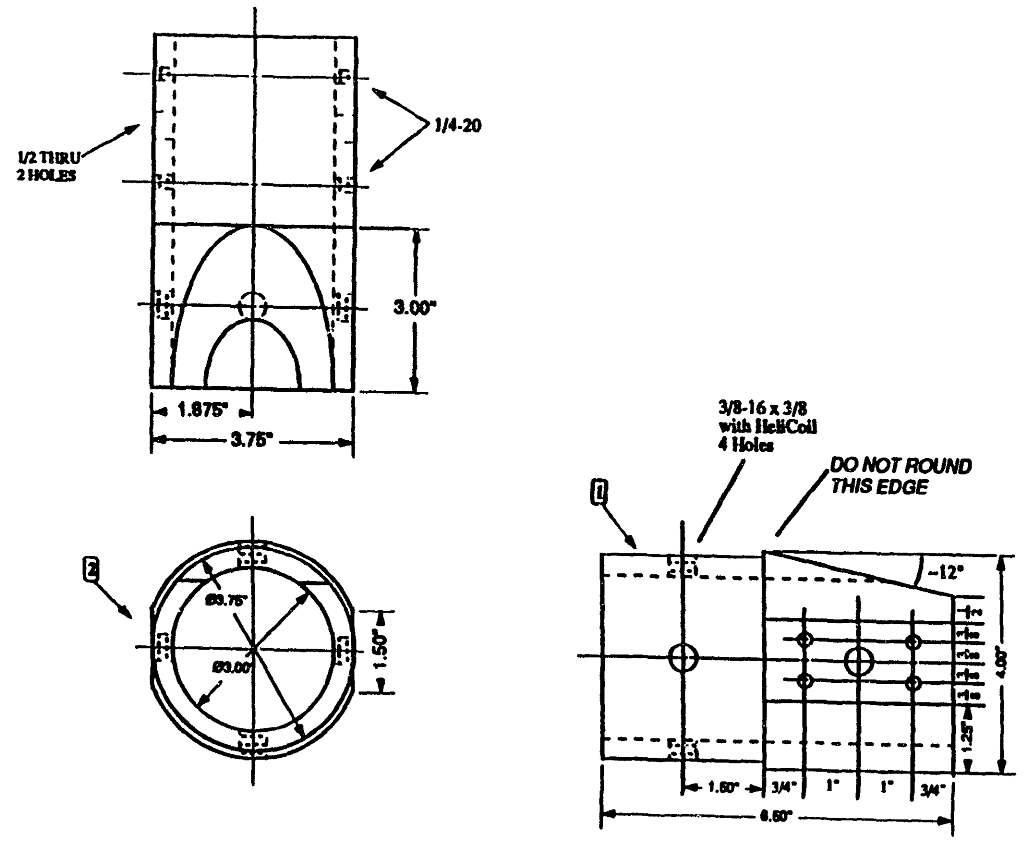


MIT SPACE SYSTEMS LABORATORY

R-PHAD-A

(LOWER BLOCKS)

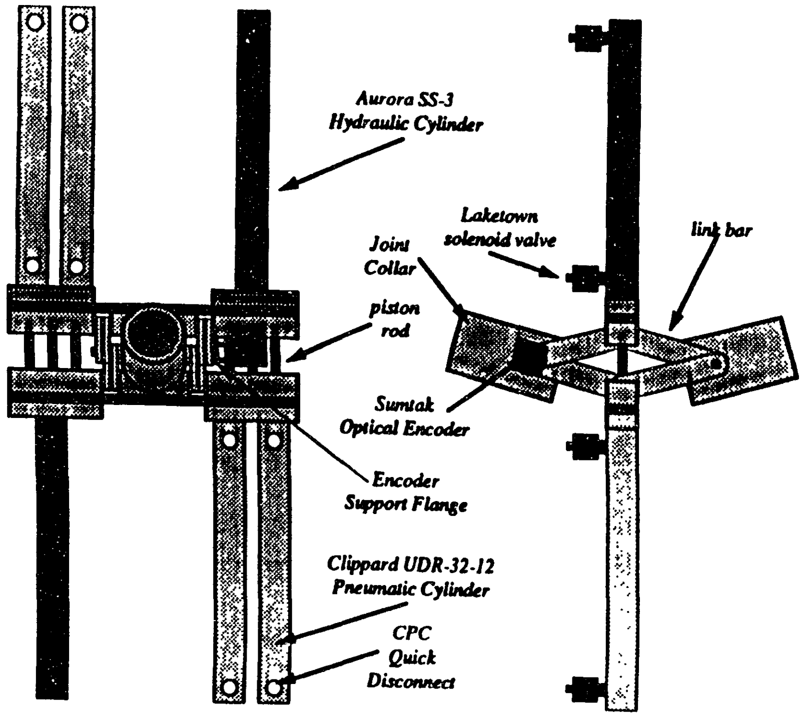
DRAWN BY: TERRENCE W. FONG		
SCALE: 1:2	DATE: 3/1/89	4 of 4
dimensions in inches unless specified		



MACHINING ORDER:
1 - LATHE TO 3.740 ± .005
2 - MILL SIDE FLATS
3 - BEVEL ~12° FACES
4 - USE HELICOILS FOR ALL THREADED HOLES
(4) 3/8-16, (8) 1/4-20

DO NOT INSERT HELICOIL BEFORE ANODIZING PIECE
ROUND ALL EDGES 1/8 R EXCEPT WHERE NOTED

MIT SPACE SYSTEMS LABORATORY		
R-PHAD-A (TUBE COLLAR)		
DRAWN BY:	TERRENCE W. FONG	
SCALE: 1:2	DATE: 5/4/90	1 of 1
dimensions in inches unless specified		



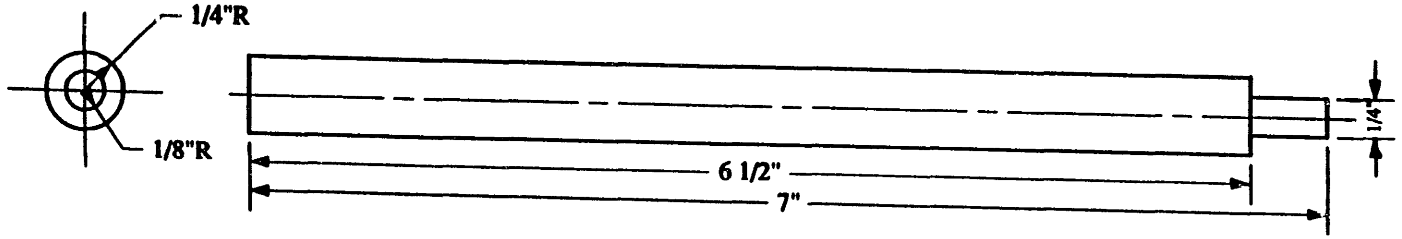
MIT SPACE SYSTEMS LABORATORY

R-PHAD-A
(SCHEMATIC)

DRAWN BY: TERRENCE W. FONG

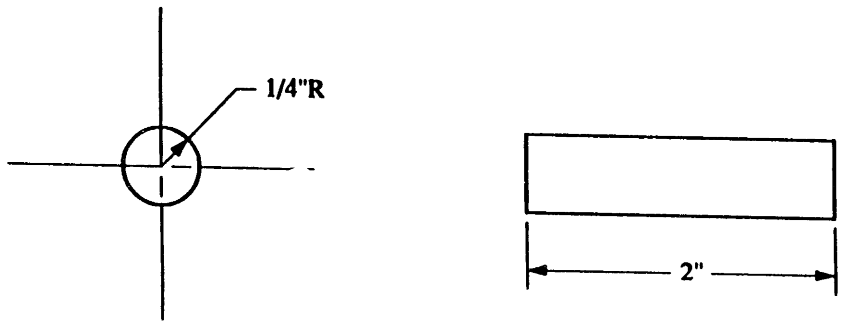
SCALE: 1:10 | DATE: 5/6/90 | 1 of 1

dimensions in inches unless specified
tolerances: ± 0.005" unless specified



FABRICATE FROM
BERG S8-70 GROUND SHAFT
(303 STAINLESS)

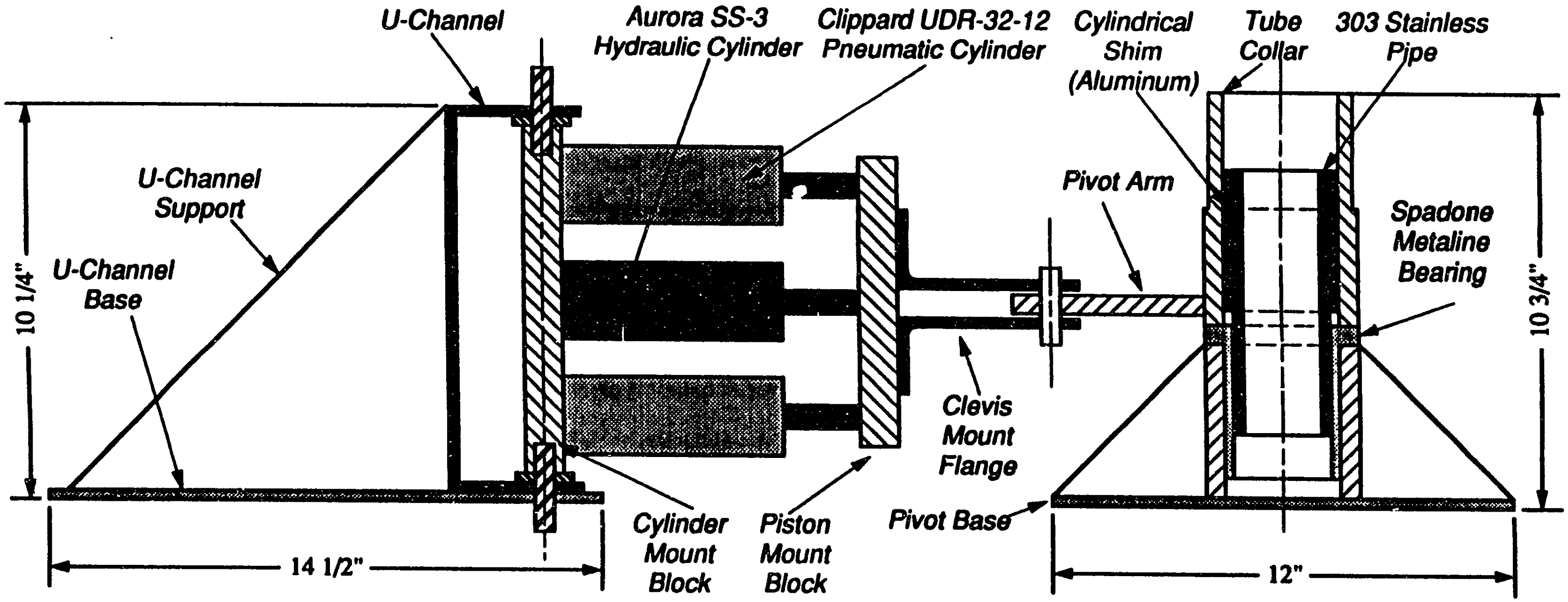
MIT SPACE SYSTEMS LABORATORY		
COLLAR PIVOT BAR - B		
DRAWN BY: TERRENCE W. FONG		
SCALE: 1:1	DATE: 11/1/89	1 of 1
dimensions in inches unless specified *tolerances: ± 0.005" unless specified*		



BERG D22-18 DOWEL PIN
(303 STAINLESS)

MIT SPACE SYSTEMS LABORATORY		
R-PHAD-A **(PIVOT BAR)**		
DRAWN BY: TERRENCE W. FONG		
SCALE: 1:1	DATE: 11/1/89	1 of 1
dimensions in inches unless specified *tolerances: ± 0.005" unless specified*		

FRONT VIEW



NOTE: Cylinder and piston lengths are not to scale

MIT SPACE SYSTEMS LABORATORY

R-PHAD-B

(ASSEMBLED SCHEMATIC)

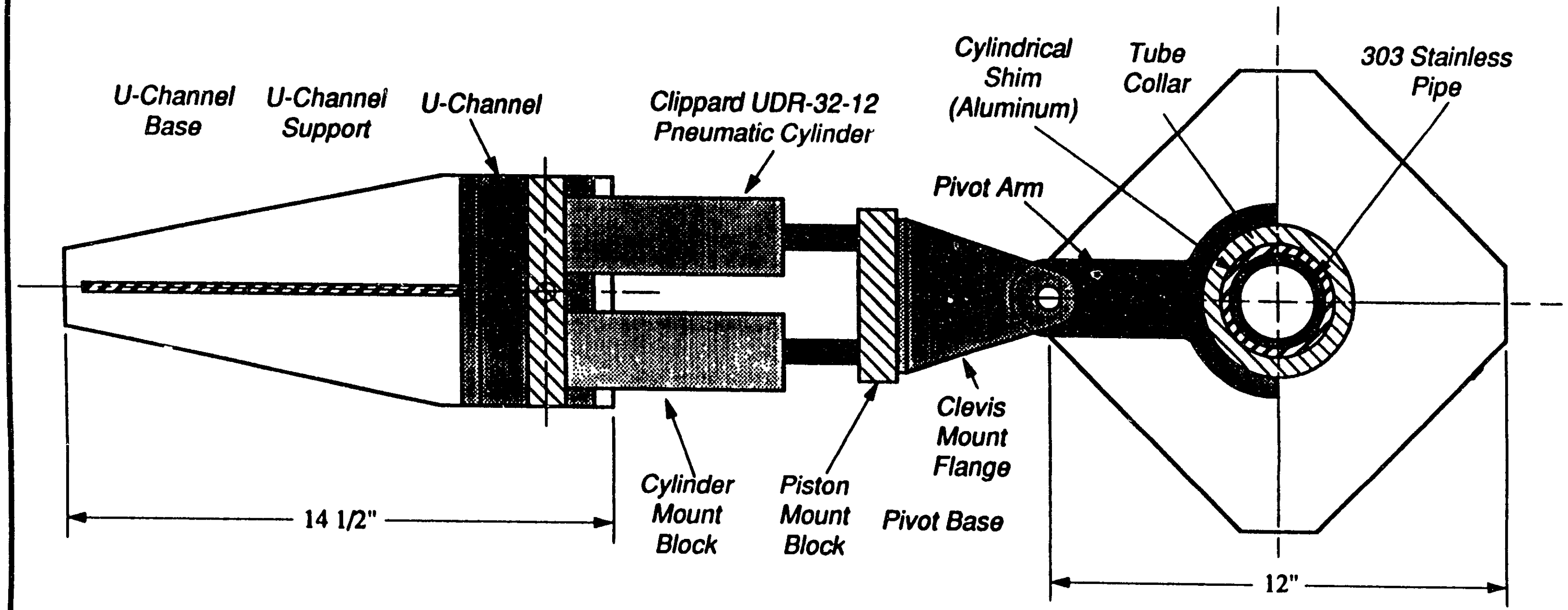
DRAWN BY: TERRENCE W. FONG

SCALE: 1:4 | DATE· 12/4/89 | 1 of 2

dimensions in inches unless specified
standard tolerance = ± 0.005"

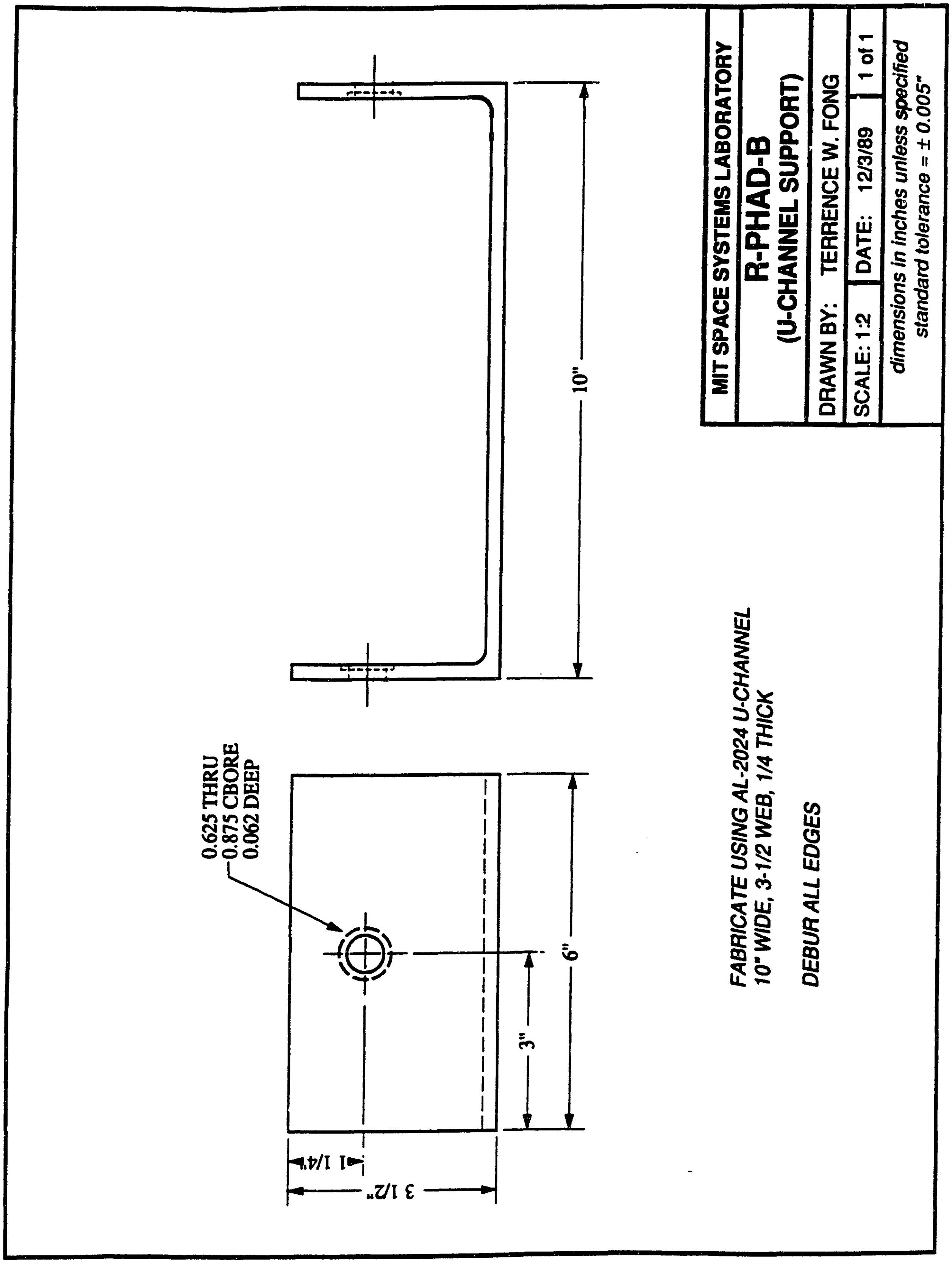
TOP VIEW

a.k.a. "the SPAM Manta Ray"



NOTE: Cylinder and piston lengths are not to scale

MIT SPACE SYSTEMS LABORATORY		
R-PHAD-B (ASSEMBLED SCHEMATIC)		
DRAWN BY: TERRENCE W. FONG		
SCALE: 1:4	DATE: 12/4/89	2 of 2
dimensions in inches unless specified standard tolerance = ± 0.005"		



MIT SPACE SYSTEMS LABORATORY

R-PHAD-B

(U-CHANNEL SUPPORT)

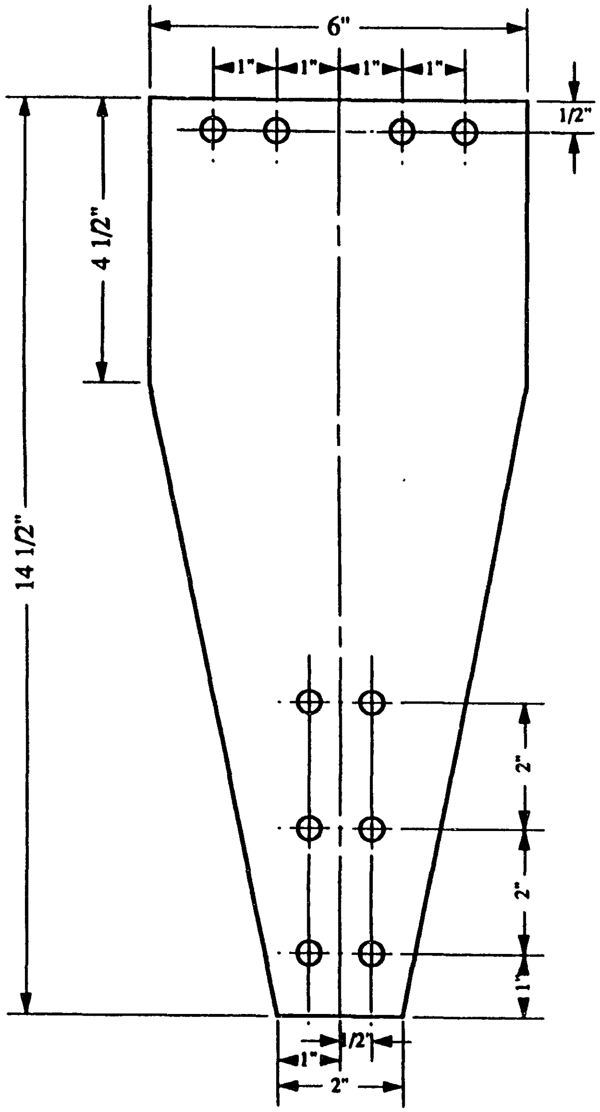
DRAWN BY: TERRENCE W. FONG

SCALE: 1:2 | DATE: 12/3/89 | 1 of 1

dimensions in inches unless specified standard tolerance = ± 0.005"

FABRICATE USING AL-2024 U-CHANNEL 10" WIDE, 3-1/2 WEB, 1/4 THICK

DEBUR ALL EDGES



FABRICATE USING 1/4" AL 2024 PLATE

DEBUR ALL EDGES

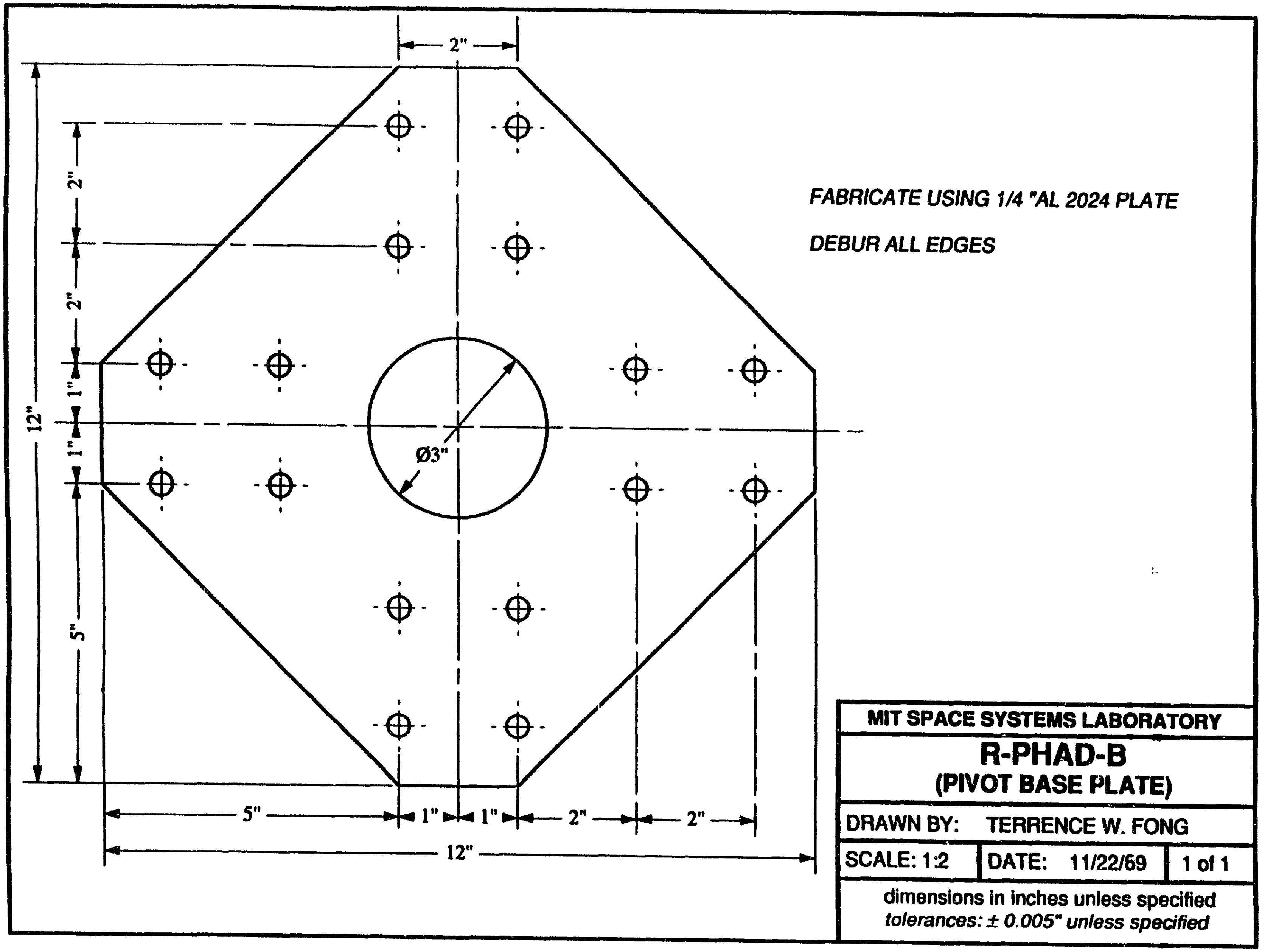
MIT SPACE SYSTEMS LABORATORY

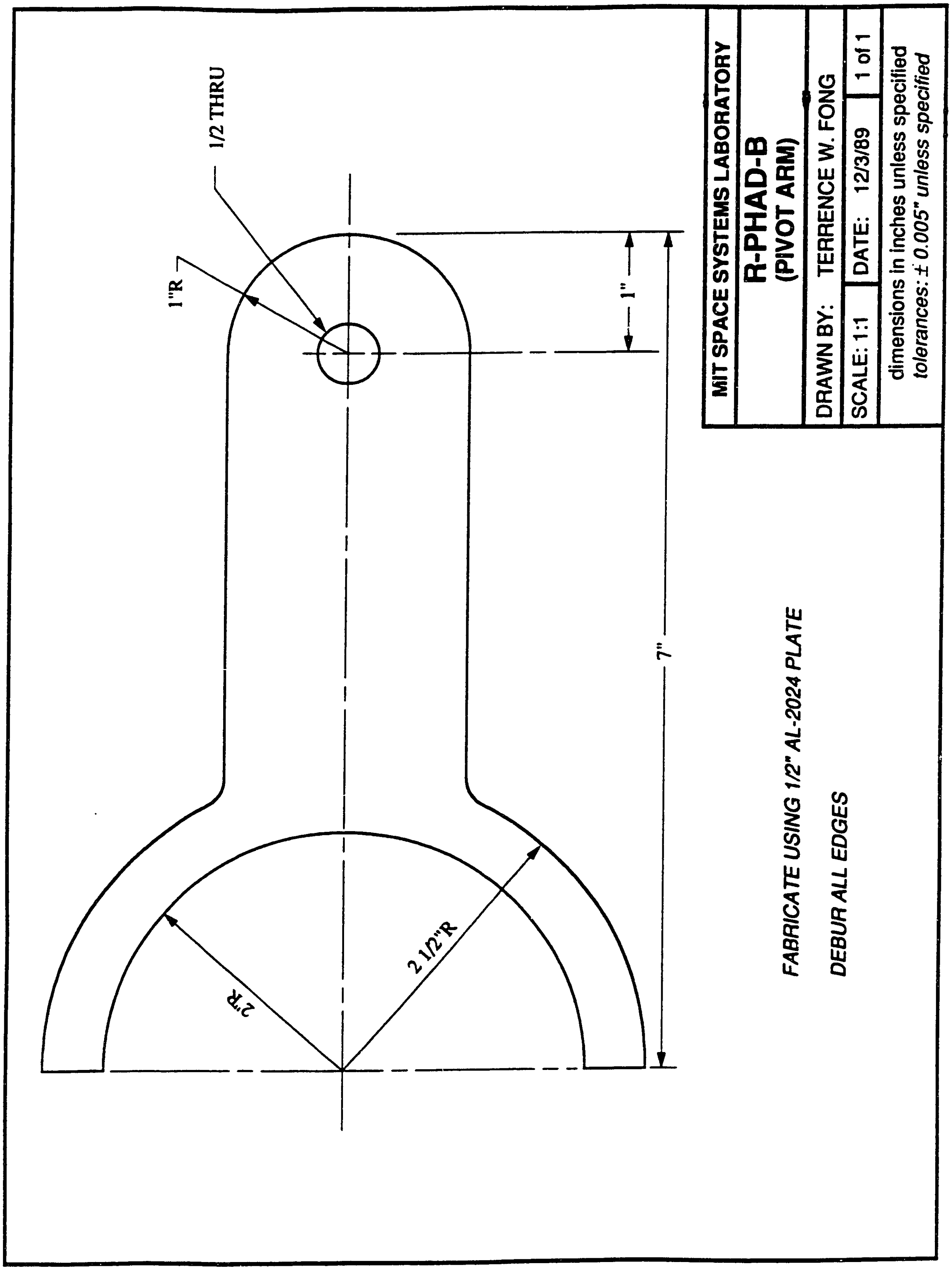
R-PHAD-B
(U-CHANNEL BASE PLATE)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:2.5 | DATE: 11/2/89 | 1 of 1

dimensions in inches unless specified
tolerances: ± 0.005" unless specified


FABRICATE USING 1/4 "AL 2024 PLATE
DEBUR ALL EDGES
2"
2"
2"
1"
1"
12"
5"
Ø3"
5"
1"
1"
2"
2"
12"
MIT SPACE SYSTEMS LABORATORY
R-PHAD-B
(PIVOT BASE PLATE)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:2
DATE: 11/22/69
1 of 1
dimensions in inches unless specified
tolerances: ± 0.005" unless specified



MIT SPACE SYSTEMS LABORATORY

R-PHAD-B
(PIVOT ARM)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:1 | DATE: 12/3/89 | 1 of 1

dimensions in inches unless specified
tolerances: ± 0.005" unless specified

FABRICATE USING 1/2" AL-2024 PLATE

DEBUR ALL EDGES

MIT SPACE SYSTEMS LABORATORY

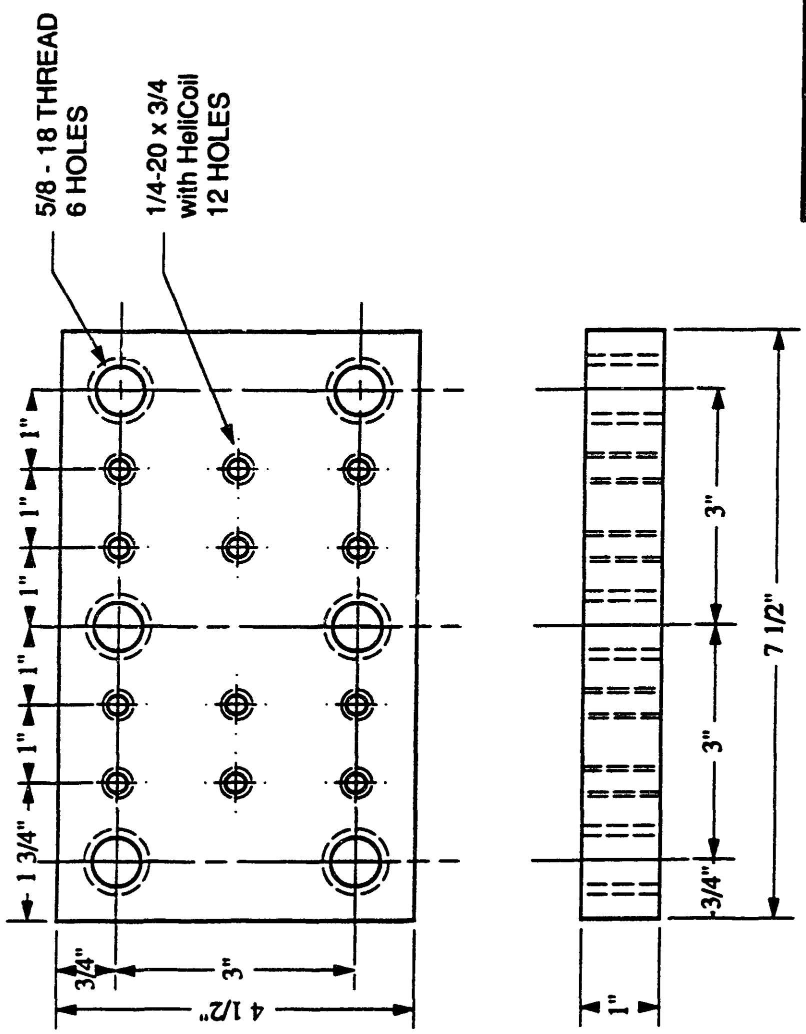
R-PHAD-B

(PISTON MOUNT BLOCK)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:2 | DATE: 12/3/89 | 1 of 1

dimensions in inches unless specified
standard tolerance = ±0.005"



FABRICATE USING AL2024

DEBUR ALL EDGES

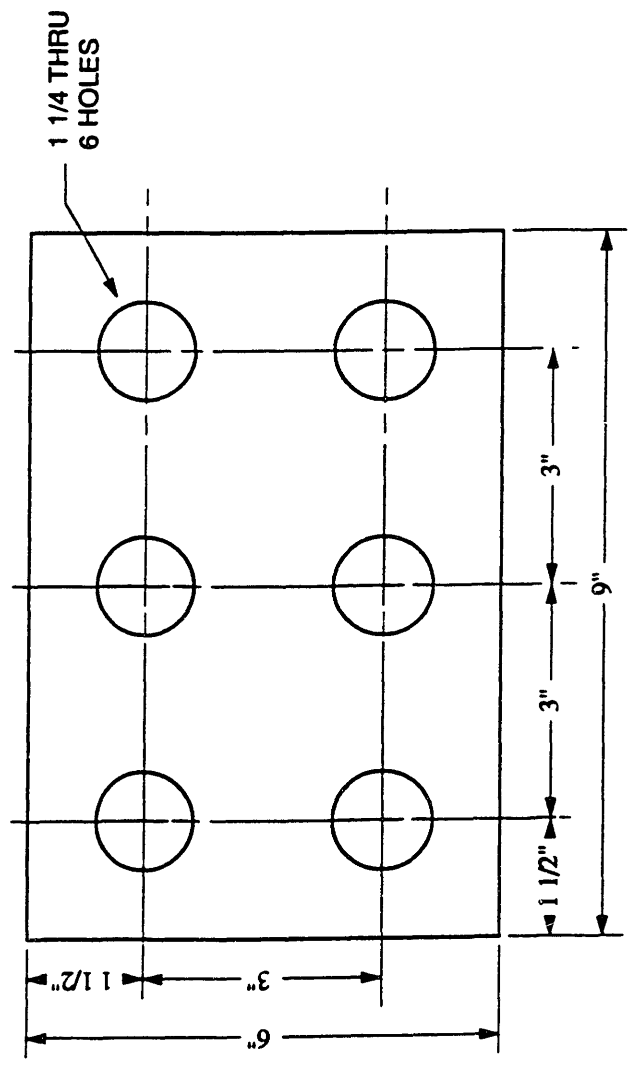
MIT SPACE SYSTEMS LABORATORY

R-PHAD-B
(CYLINDER SUPPORT PLATE)

DRAWN BY: TERRENCE W. FONG

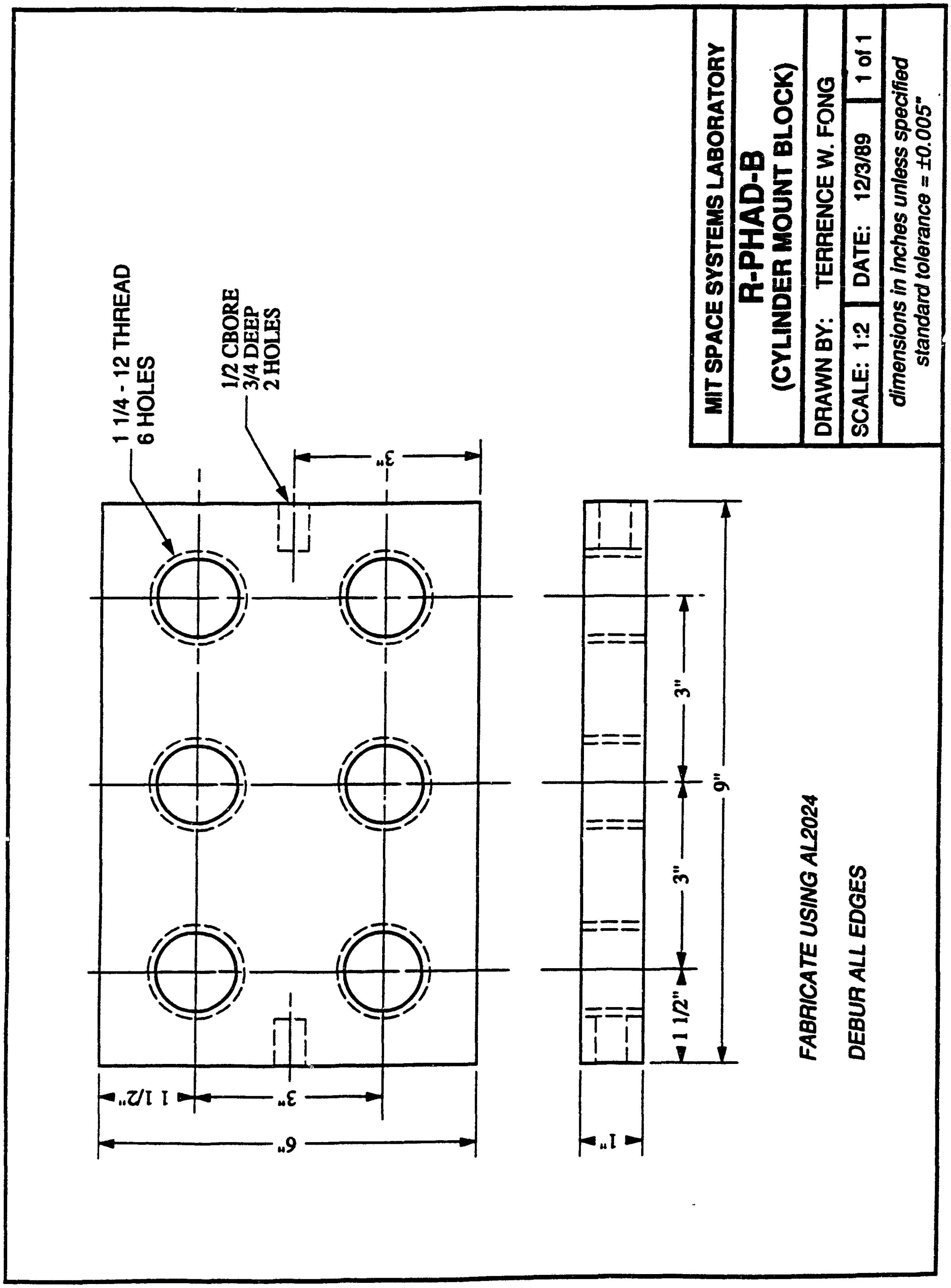
SCALE: 1:2	DATE: 12/2/89	1 of 1

dimensions in inches unless specified
standard tolerance = ±0.005"



FABRICATE USING 1/4" AL2024 PLATE

DEBUR ALL EDGES



MIT SPACE SYSTEMS LABORATORY

R-PHAD-B

(CYLINDER MOUNT BLOCK)

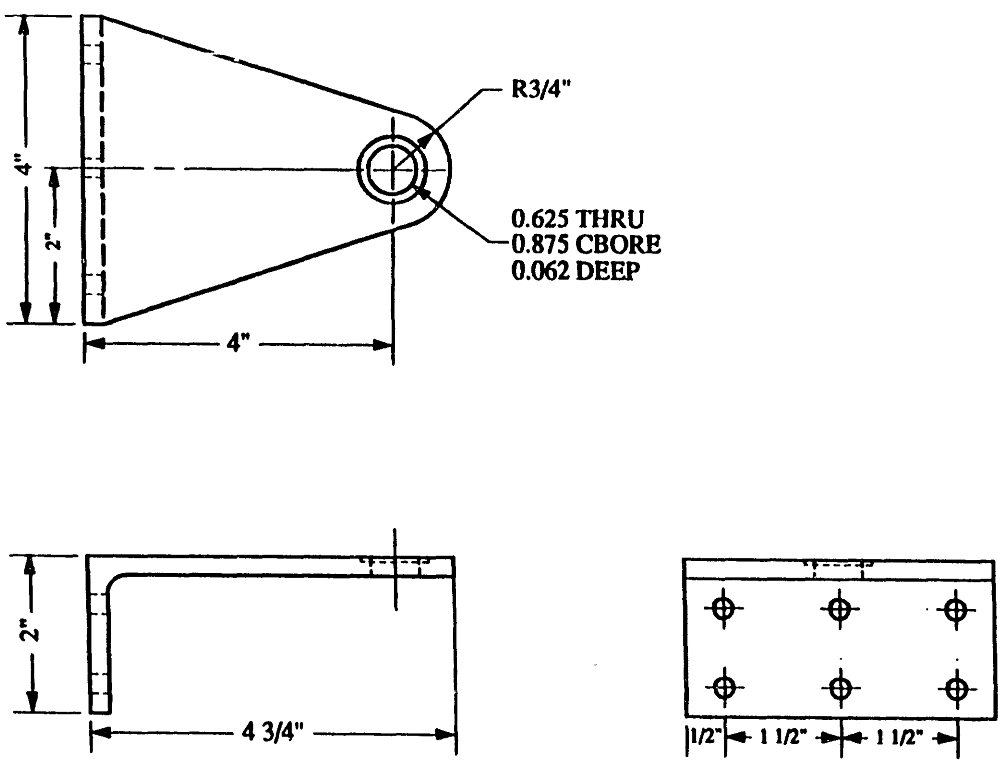
DRAWN BY: TERRENCE W. FONG

SCALE: 1:2 | DATE: 12/3/89 | 1 of 1

dimensions in inches unless specified standard tolerance = ±0.005"

FABRICATE USING AL2024

DEBUR ALL EDGES



FABRICATE USING AL-2024

DEBUR ALL EDGES

MIT SPACE SYSTEMS LABORATORY

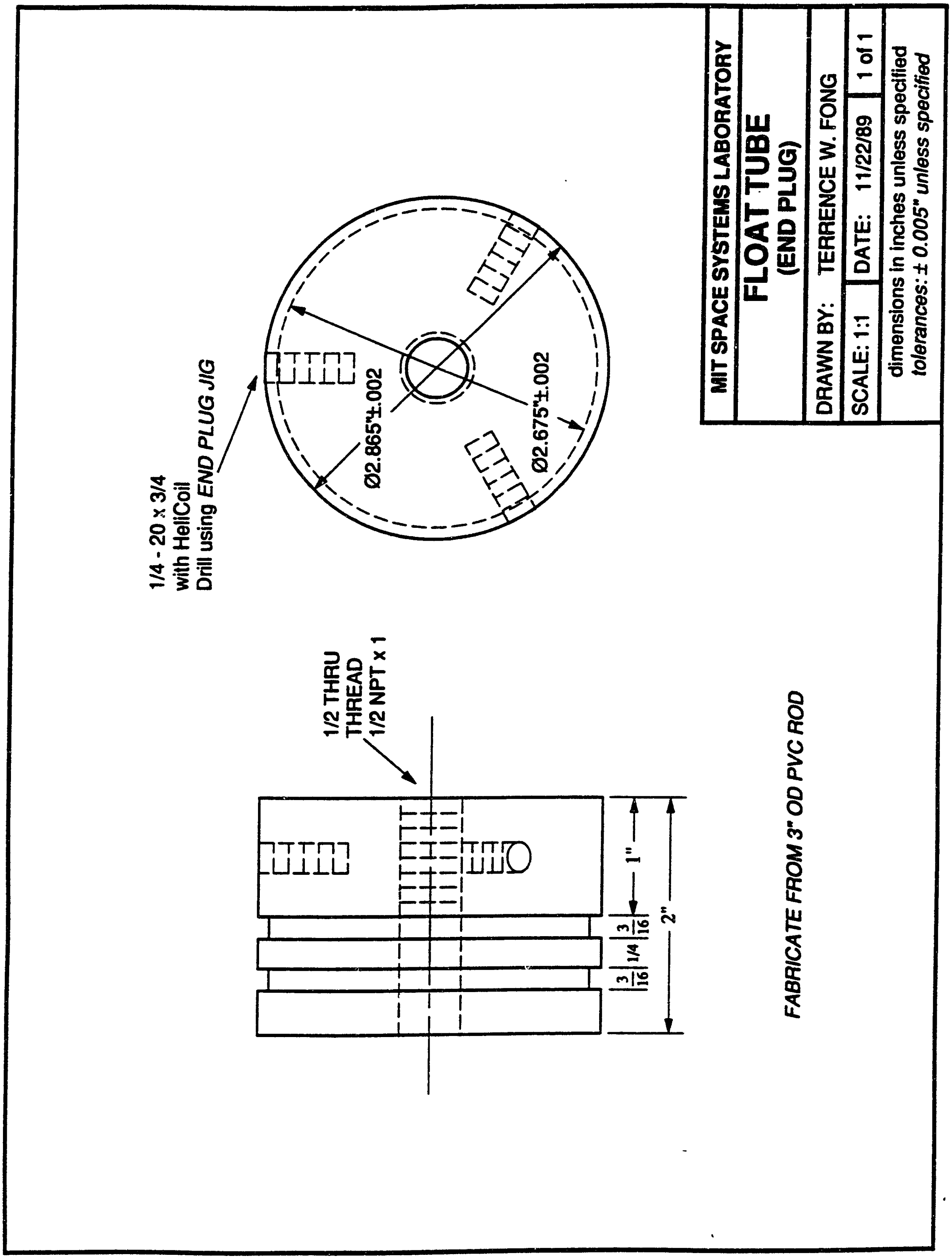
R-PHAD-B

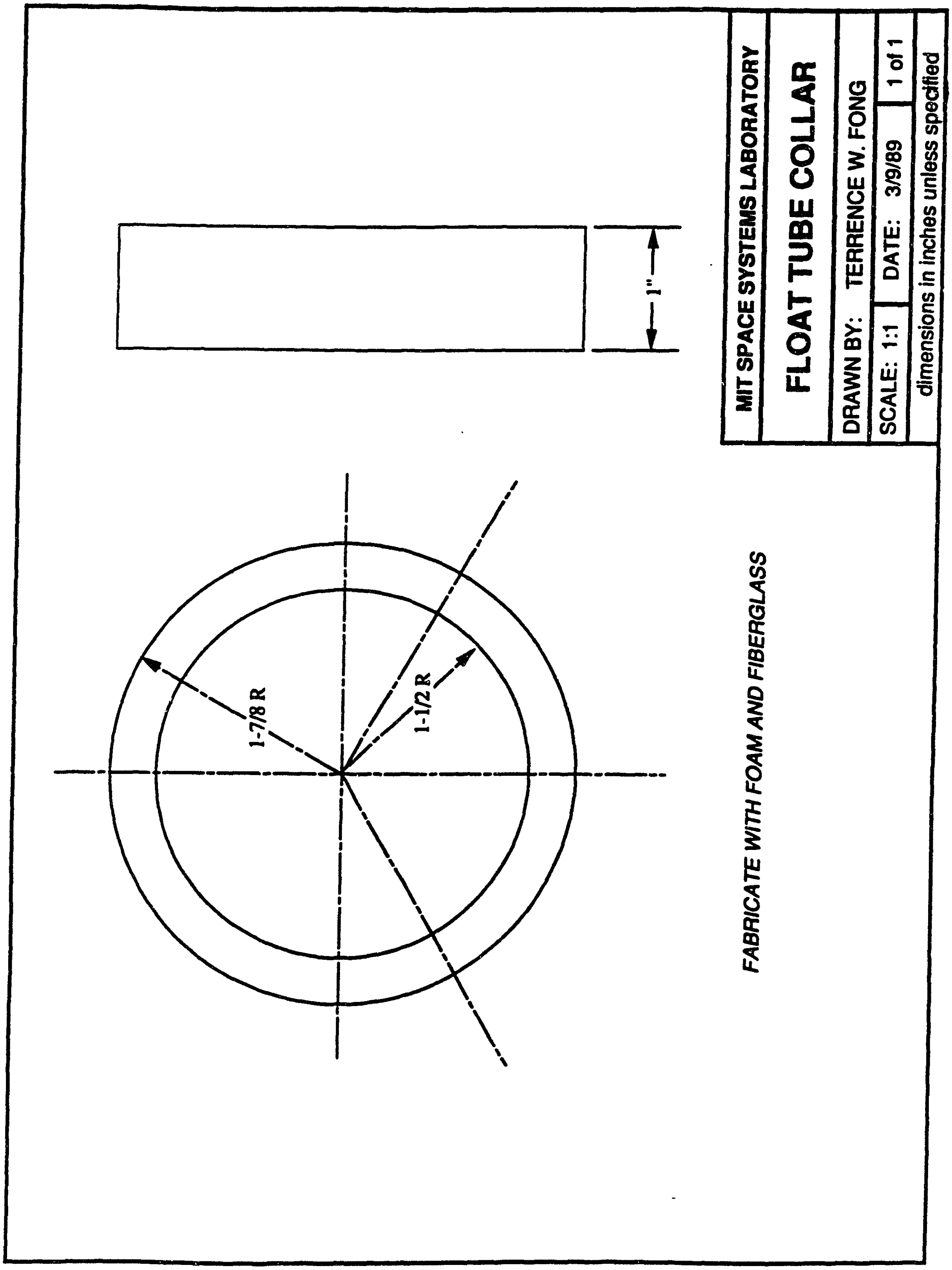
(CLEVIS MOUNT FLANGE)

DRAWN BY: TERRENCE W. FONG

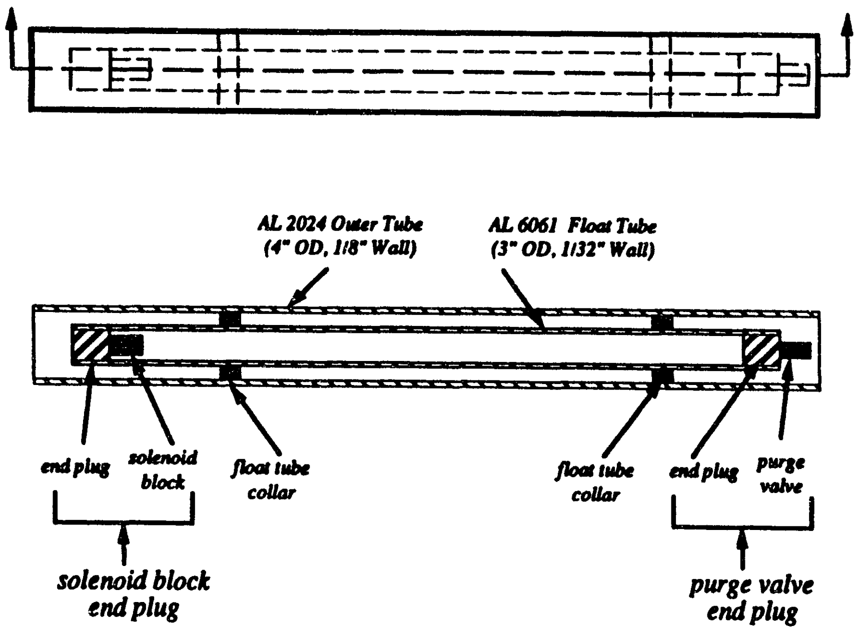
SCALE: 1:2 DATE: 12/3/89 1 of 1

dimensions in inches unless specified
standard tolerance =± 0.005"


MIT SPACE SYSTEMS LABORATORY
FLOAT TUBE
(END PLUG)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:1
DATE: 11/22/89
1 of 1
dimensions in inches unless specified
tolerances: ± 0.005" unless specified
Ø2.865"±.002
Ø2.675"±.002
1/4 - 20 x 3/4
with HeliCoil
Drill using END PLUG JIG
1/2 THRU
THREAD
1/2 NPT x 1
1"
2"
3/16
1/4
3/16
FABRICATE FROM 3" OD PVC ROD


MIT SPACE SYSTEMS LABORATORY
FLOAT TUBE COLLAR
DRAWN BY: TERRENCE W. FONG
SCALE: 1:1
DATE: 3/9/89
1 of 1
dimensions in inches unless specified
1"
1-7/8 R
1-1/2 R

FABRICATE WITH FOAM AND FIBERGLASS



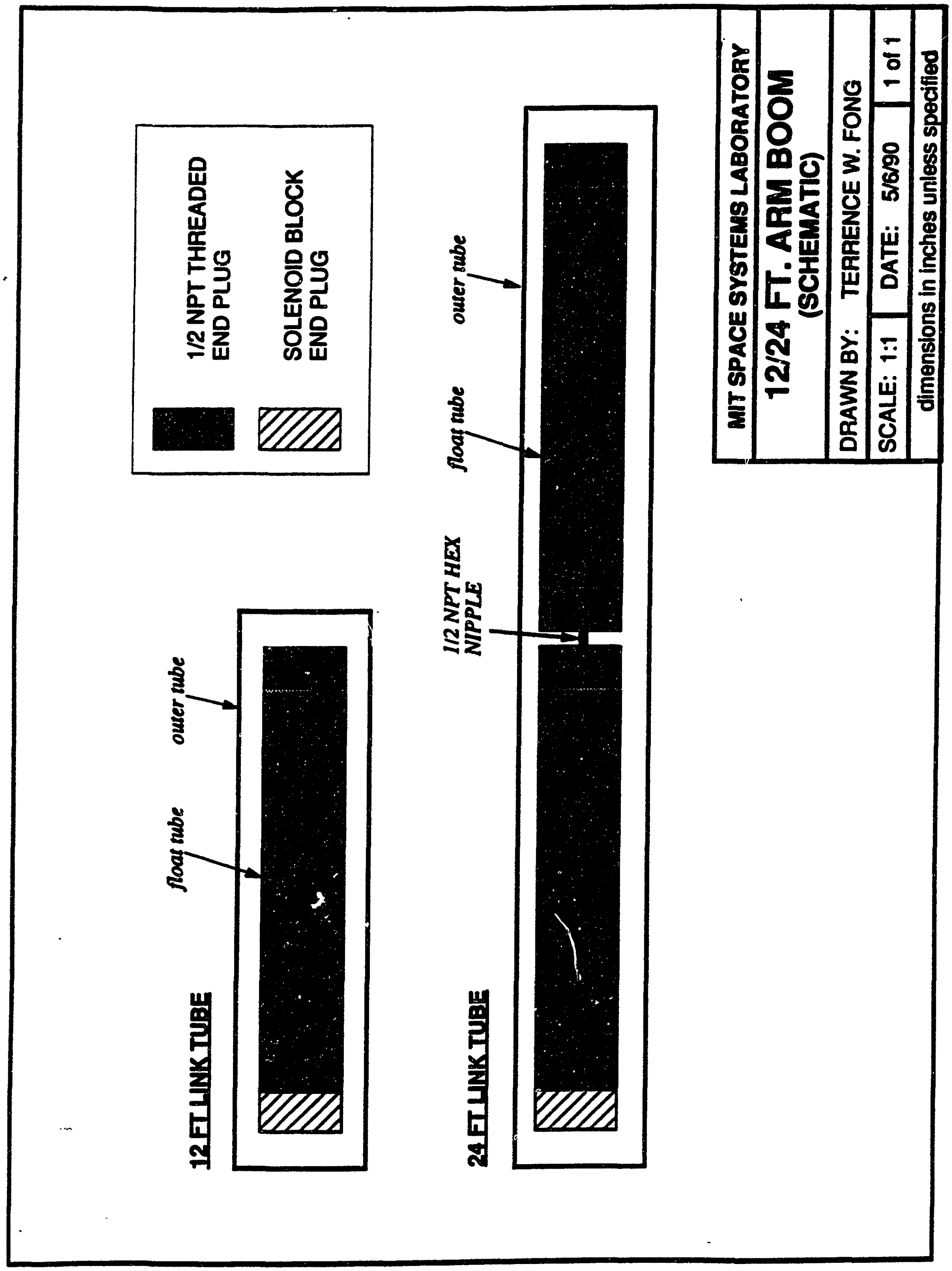
MIT SPACE SYSTEMS LABORATORY

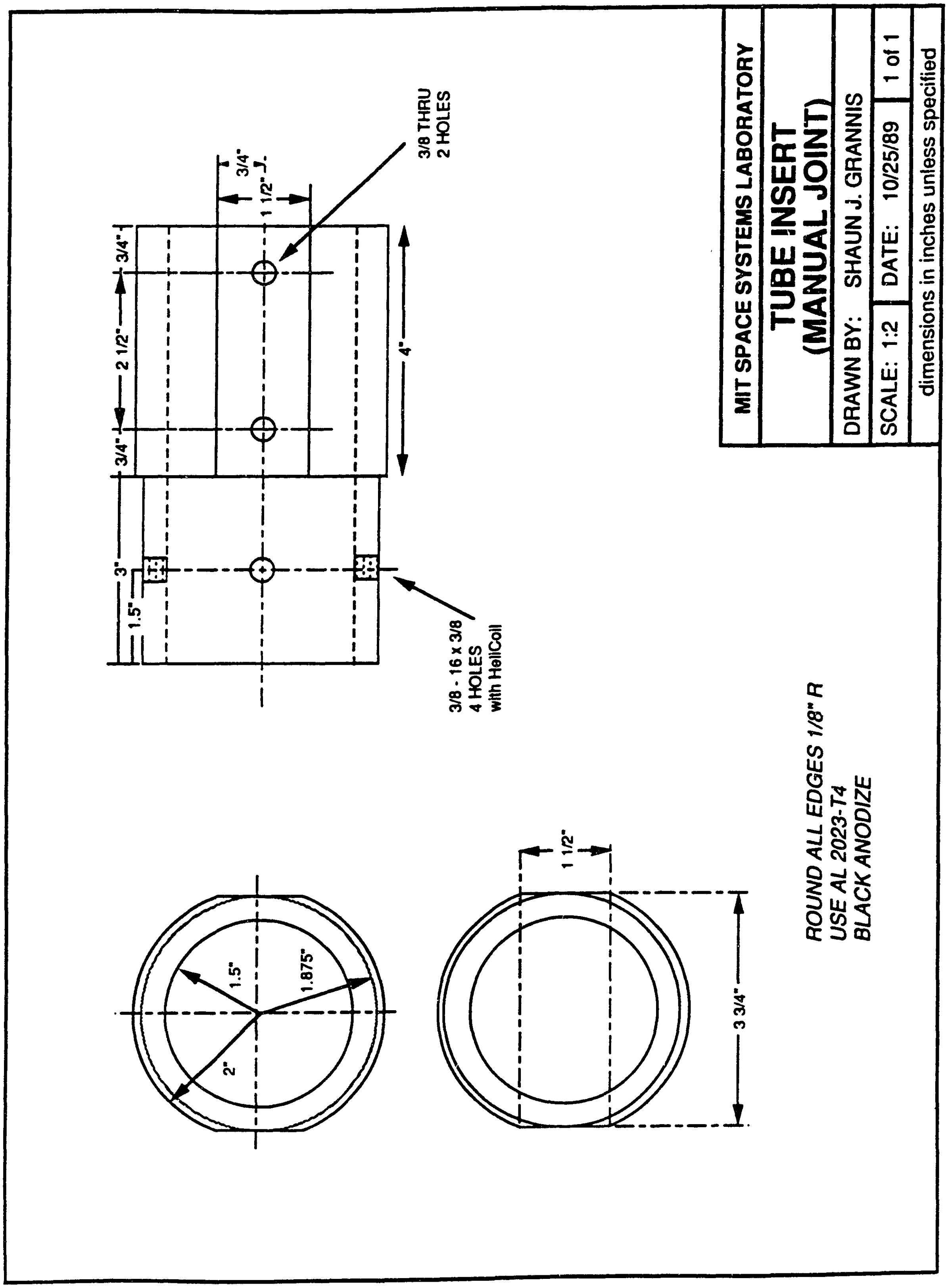
ARM BOOM
(XSEC SCHEMATIC)

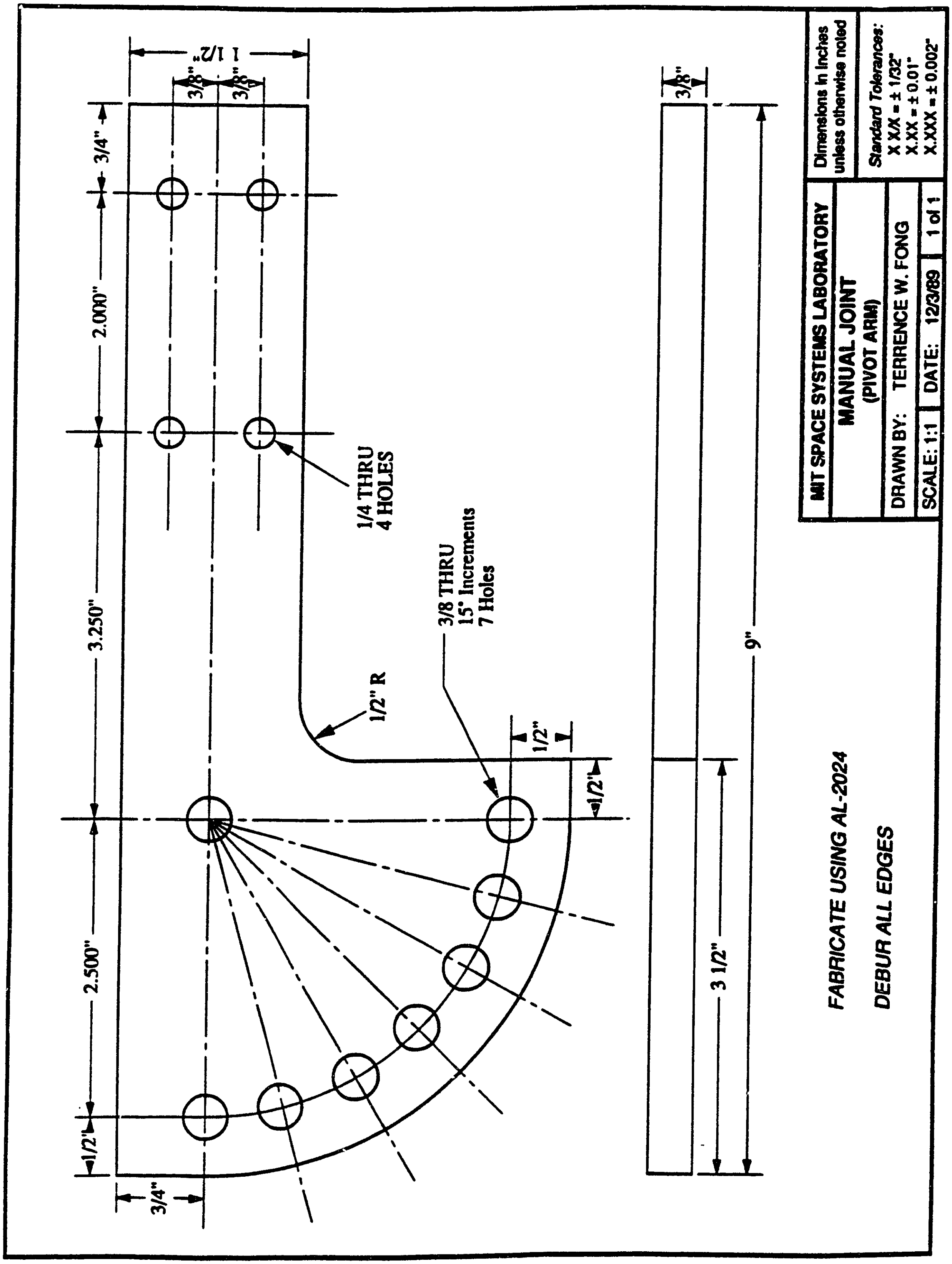
DRAWN BY: TERRENCE W. FONG

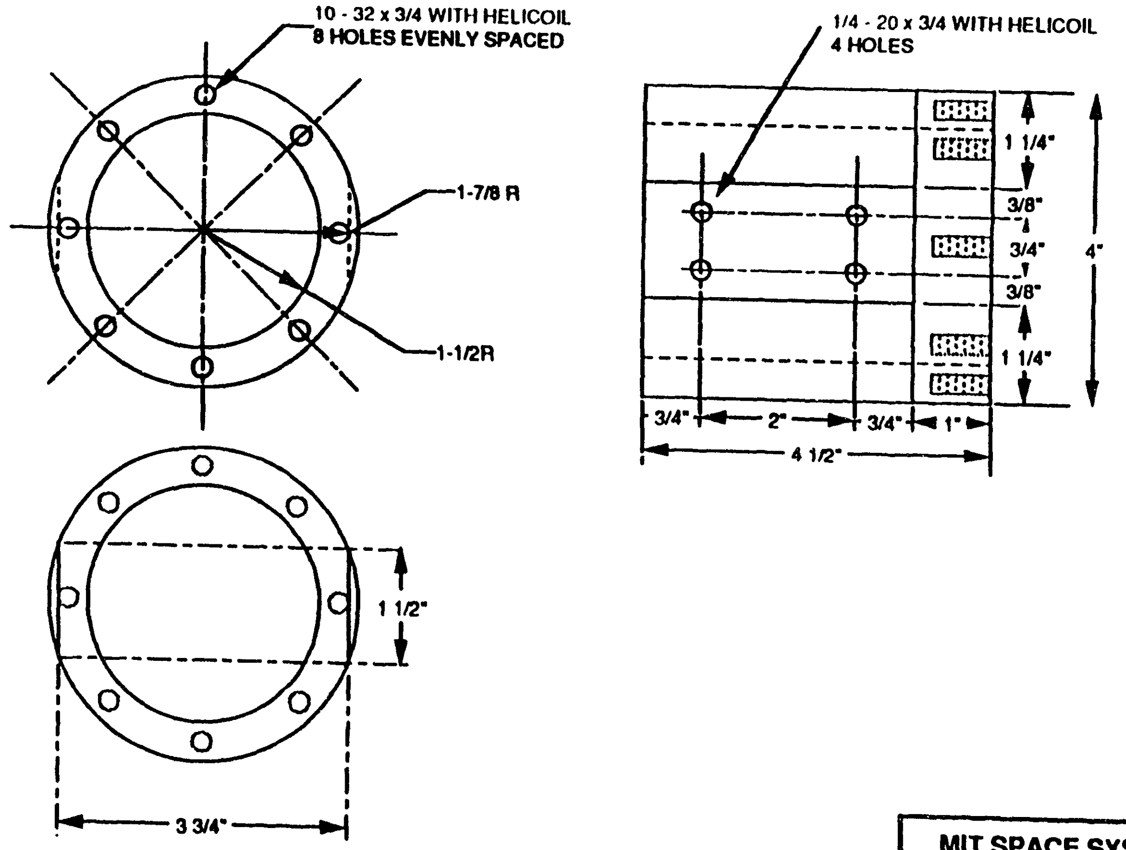
SCALE: 1:1 DATE: 5/6/90 1 of 1

dimensions in inches unless specified

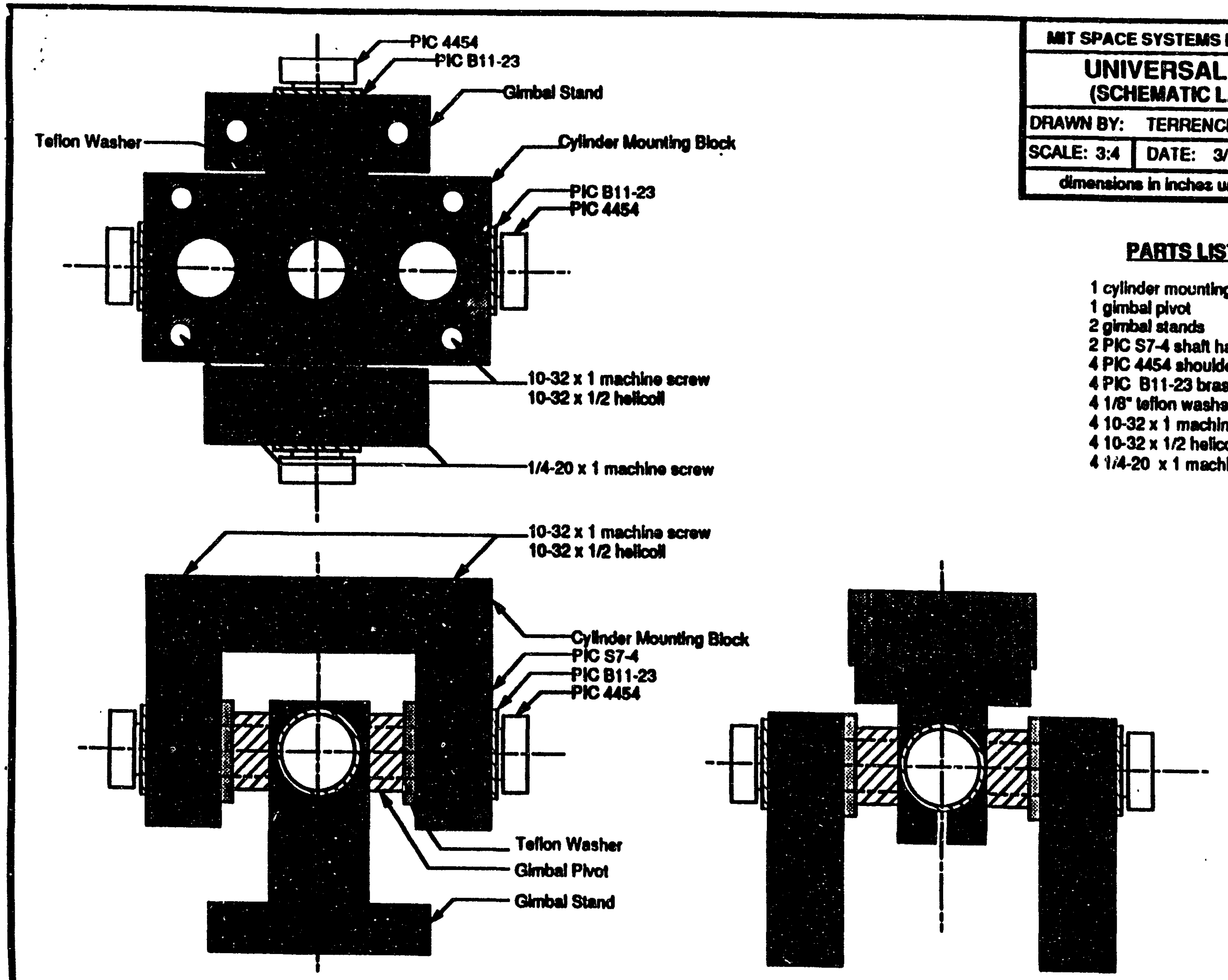

MIT SPACE SYSTEMS LABORATORY
12/24 FT. ARM BOOM
(SCHEMATIC)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:1
DATE: 5/6/90
1 of 1
dimensions in inches unless specified
■ 1/2 NPT THREADED END PLUG
▨ SOLENOID BLOCK END PLUG
12 FT LINK TUBE
float tube
outer tube
24 FT LINK TUBE
1/2 NPT HEX NIPPLE
float tube
outer tube


MIT SPACE SYSTEMS LABORATORY
TUBE INSERT
(MANUAL JOINT)
DRAWN BY: SHAUN J. GRANNIS
SCALE: 1:2
DATE: 10/25/89
1 of 1
dimensions in inches unless specified
3/8 THRU
2 HOLES
3/4"
1 1/2"
3/4"
2 1/2"
3/4"
4"
3"
1.5"
3/8 - 16 x 3/8
4 HOLES
with HeliCoil
ROUND ALL EDGES 1/8" R
USE AL 2023-T4
BLACK ANODIZE
1 1/2"
3 3/4"
1.5"
1.875"
2"


1 1/2"
3/8"
3/8"
3/4"
2.000"
3.250"
2.500"
1/2"
3/4"
1/4 THRU
4 HOLES
3/8 THRU
15° Increments
7 Holes
1/2" R
1/2"
1/2"
3/8"
9"
3 1/2"
FABRICATE USING AL-2024
DEBUR ALL EDGES
MIT SPACE SYSTEMS LABORATORY
MANUAL JOINT
(PIVOT ARM)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:1
DATE: 12/3/89
1 of 1
Dimensions in Inches
unless otherwise noted
Standard Tolerances:
X X/X = ± 1/32"
X.XX = ± 0.01"
X.XXX = ± 0.002"



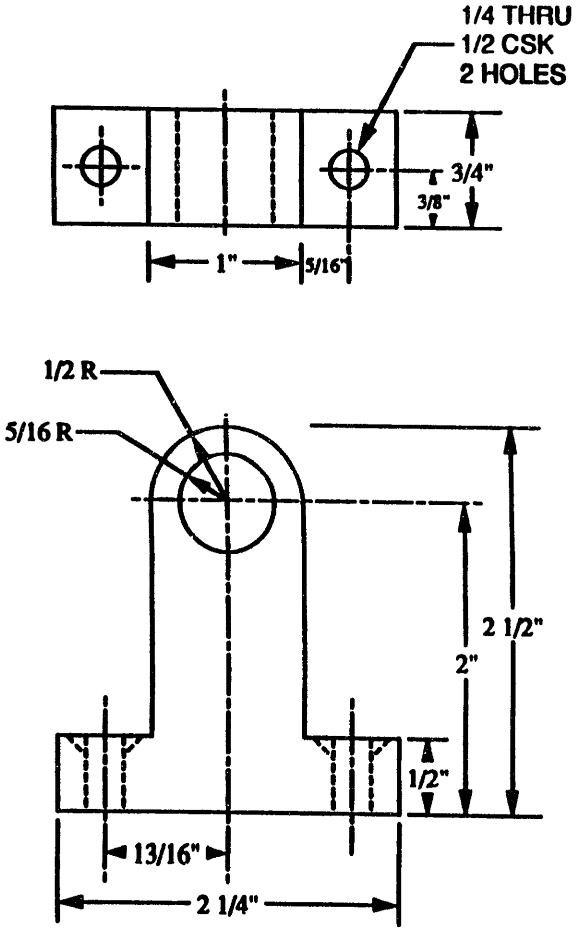
MIT SPACE SYSTEMS LABORATORY	
PLATFORM COLLAR **(TUBE/PLATFORM MOUNT)**	
DRAWN BY: SHAUN J. GRANNIS	
SCALE: 1:2	DATE: 10/23/89 | 1 of 1
dimensions in inches unless specified	



MIT SPACE SYSTEMS LABORATORY		
UNIVERSAL JOINT (SCHEMATIC LAYOUT)		
DRAWN BY: TERRENCE W. FONG		
SCALE: 3:4	DATE: 3/7/89	1 of 5
dimensions in inches unless specified		

PARTS LIST

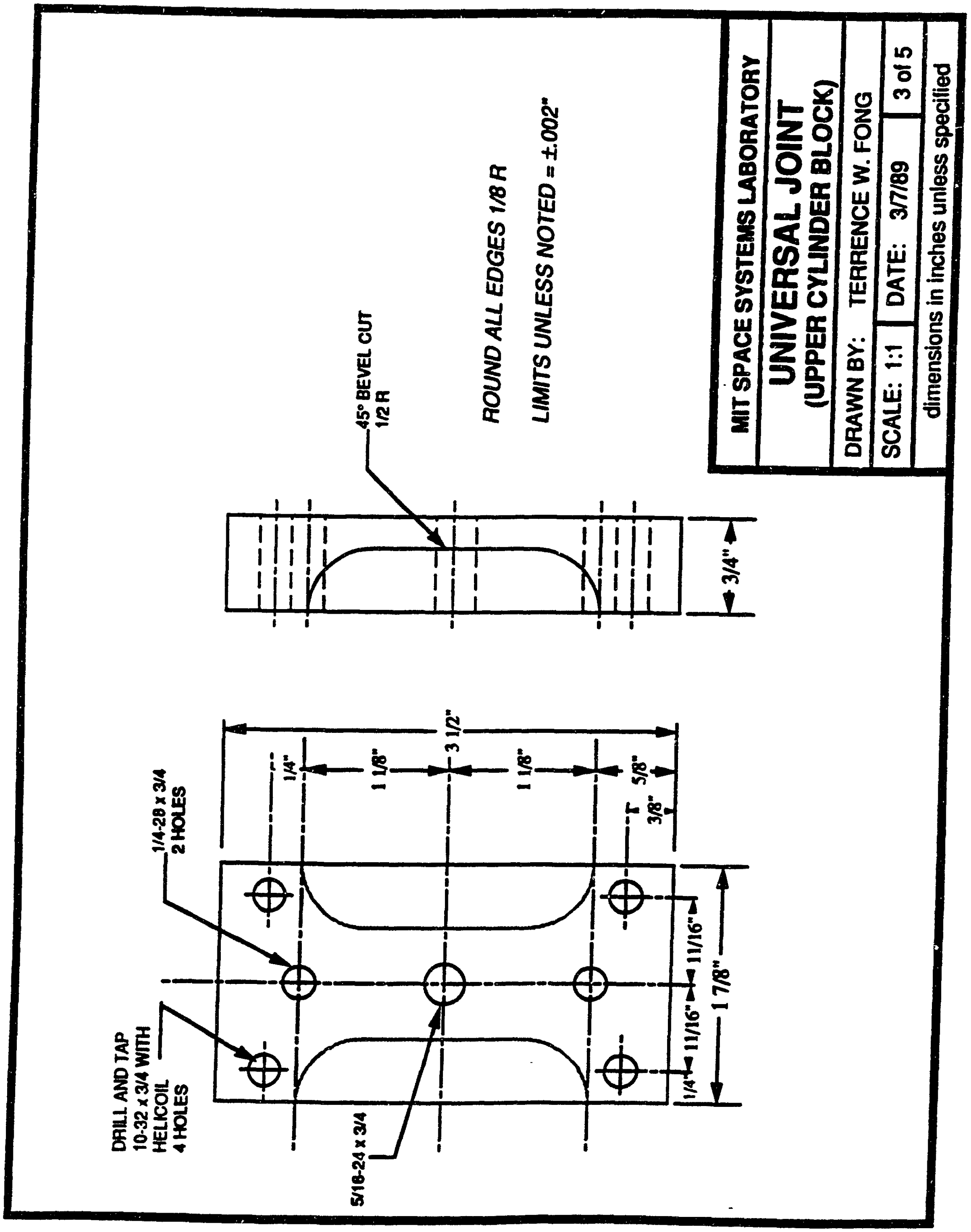
1 cylinder mounting block
1 gimbal pivot
2 gimbal stands
2 PIC S7-4 shaft hangers
4 PIC 4454 shoulder screws
4 PIC B11-23 brass bushings
4 1/8" teflon washers
4 10-32 x 1 machine screws
4 10-32 x 1/2 helicoils
4 1/4-20 x 1 machine screws

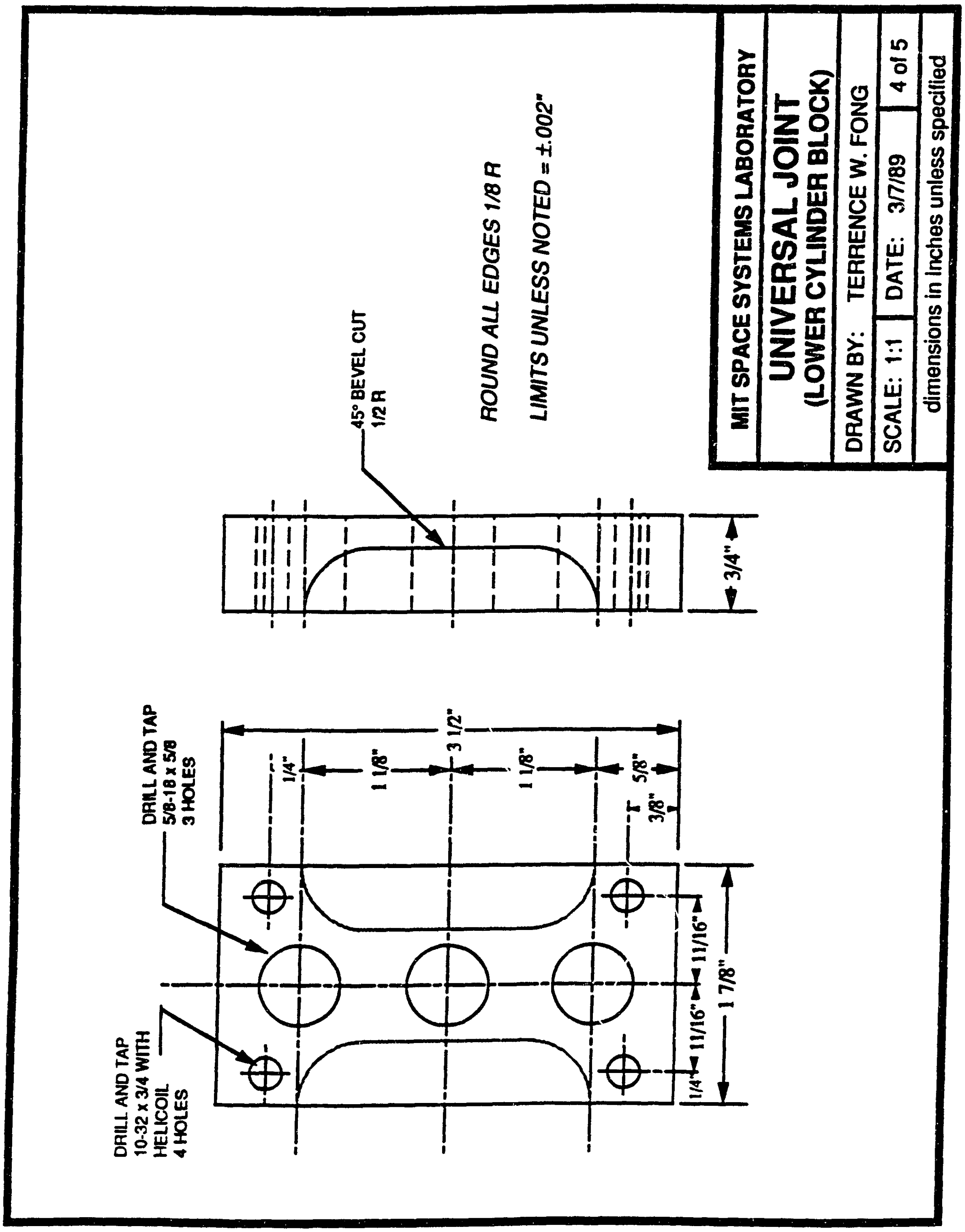


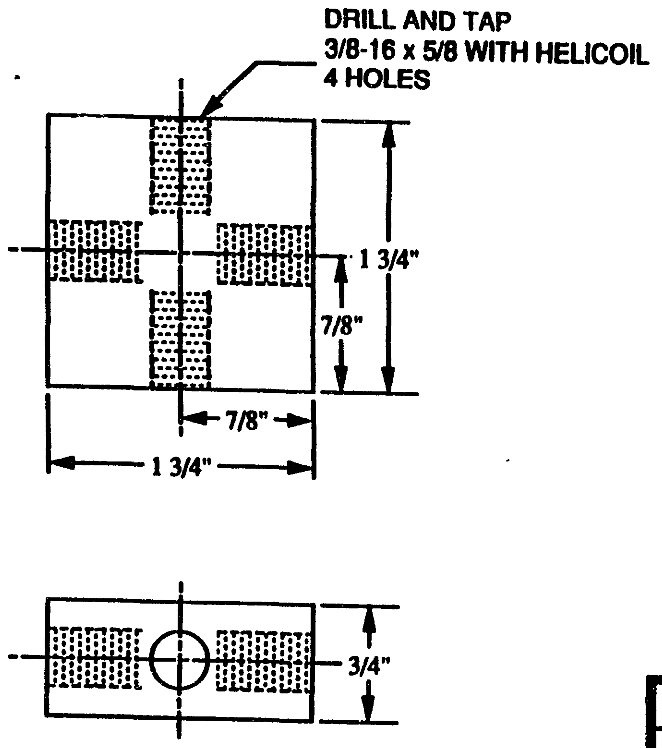
ROUND ALL EDGES 1/8 R

LIMITS UNLESS NOTED = ±.002"

MIT SPACE SYSTEMS LABORATORY		
UNIVERSAL JOINT (GIMBAL STAND)		
DRAWN BY: TERRENCE W. FONG		
SCALE: 1:1	DATE: 3/7/89	2 of 5
dimensions in inches unless specified		


MIT SPACE SYSTEMS LABORATORY
UNIVERSAL JOINT
(UPPER CYLINDER BLOCK)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:1
DATE: 3/7/89
3 of 5
dimensions in inches unless specified
ROUND ALL EDGES 1/8 R
LIMITS UNLESS NOTED = ±.002"
45° BEVEL CUT 1/2 R
3/4"
3 1/2"
1/4"
1 1/8"
1 1/8"
5/8"
3/8"
1/4-28 x 3/4 2 HOLES
DRILL AND TAP 10-32 x 3/4 WITH HELICOIL 4 HOLES
5/16-24 x 3/4
1/4"
11/16"
11/16"
1 7/8"


MIT SPACE SYSTEMS LABORATORY
UNIVERSAL JOINT
(LOWER CYLINDER BLOCK)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:1
DATE: 3/7/89
4 of 5
dimensions in inches unless specified
ROUND ALL EDGES 1/8 R
LIMITS UNLESS NOTED = ±.002"
45° BEVEL CUT
1/2 R
3/4"
DRILL AND TAP
5/8-18 x 5/8
3 HOLES
DRILL AND TAP
10-32 x 3/4 WITH
HELICOIL
4 HOLES
3 1/2"
1/4"
1 1/8"
1 1/8"
5/8"
3/8"
11/16"
11/16"
1 7/8"
1/4"



ROUND ALL EDGES 1/8 R

LIMITS UNLESS NOTED = ±.002"

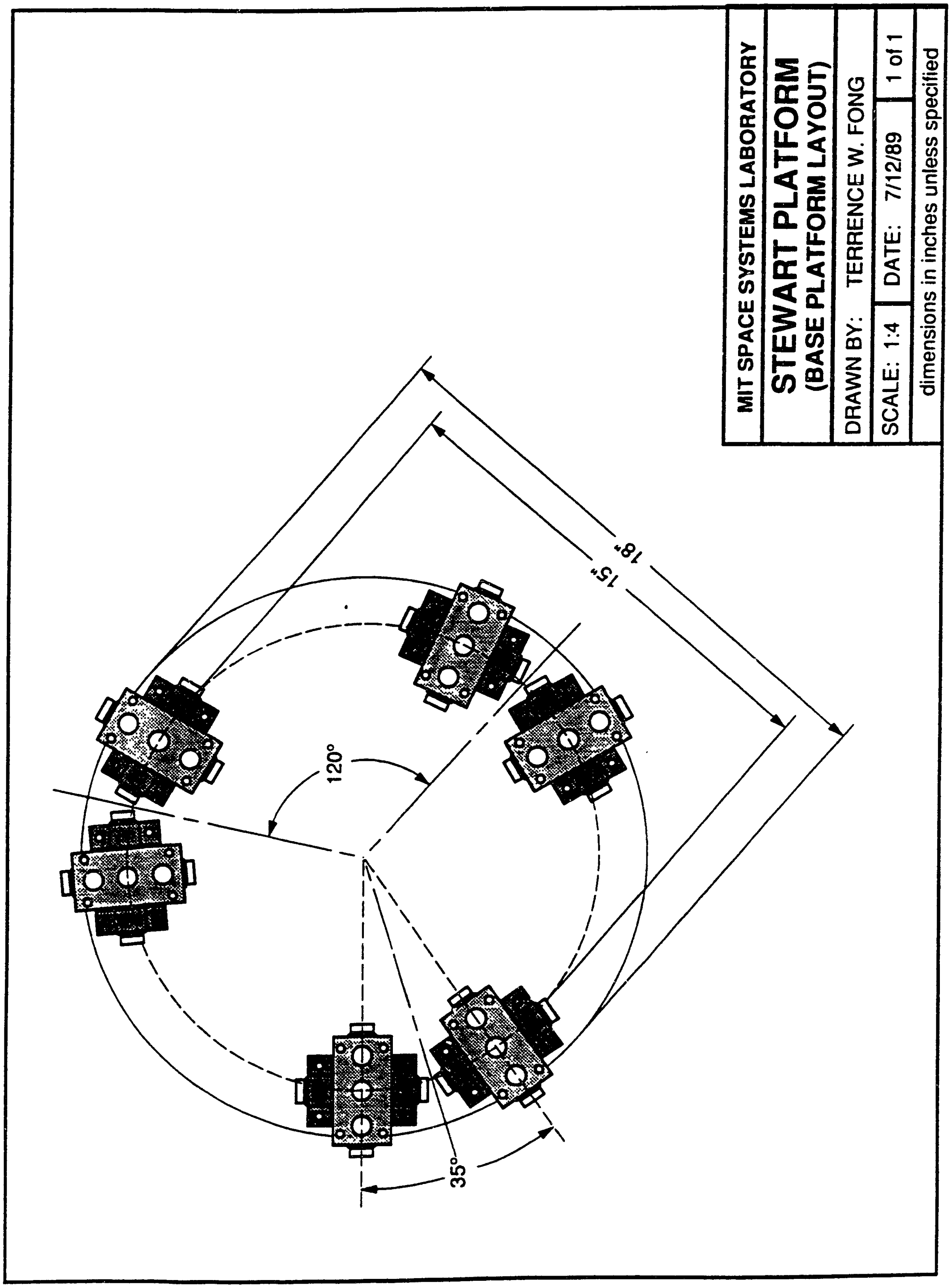
MIT SPACE SYSTEMS LABORATORY

UNIVERSAL JOINT
(GIMBAL PIVOT)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:1 | DATE: 3/7/89 | 5 of 5

dimensions in inches unless specified


MIT SPACE SYSTEMS LABORATORY
STEWART PLATFORM
(BASE PLATFORM LAYOUT)
DRAWN BY: TERRENCE W. FONG
SCALE: 1:4
DATE: 7/12/89
1 of 1
dimensions in inches unless specified
18"
15"
120°
35°

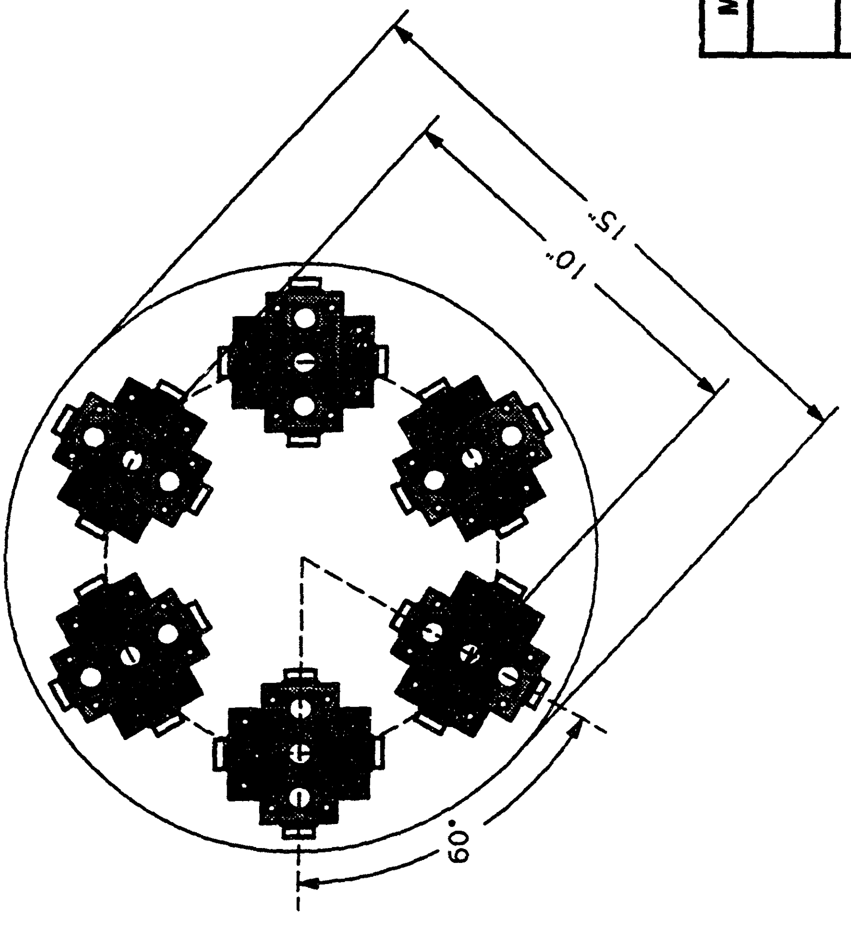
MIT SPACE SYSTEMS LABORATORY

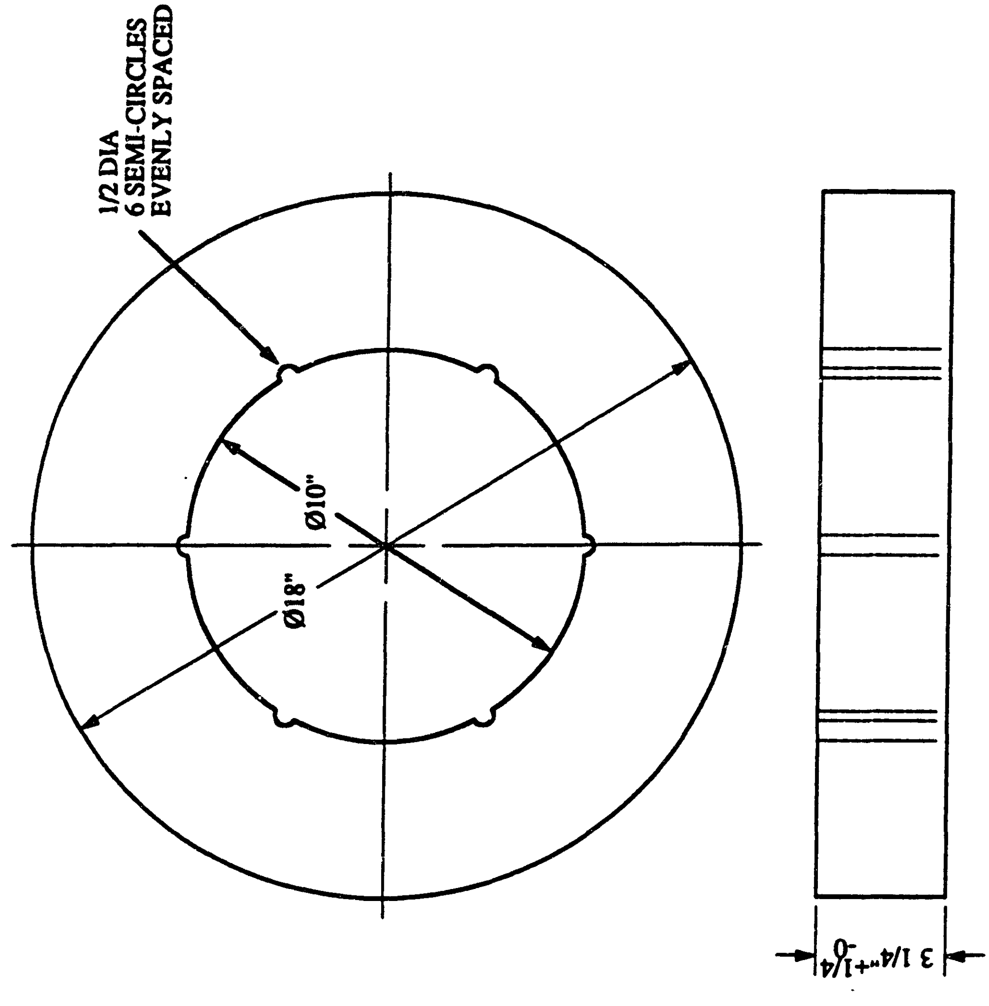
STEWART PLATFORM
(TOP PLATFORM LAYOUT)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:4 | DATE: 7/3/89 | 1 of 1

dimensions in inches unless specified





FABRICATE USING ROHACELL FOAM

EPOXY AND FIBERGLASS

MIT SPACE SYSTEMS LABORATORY

STEWART PLATFORM

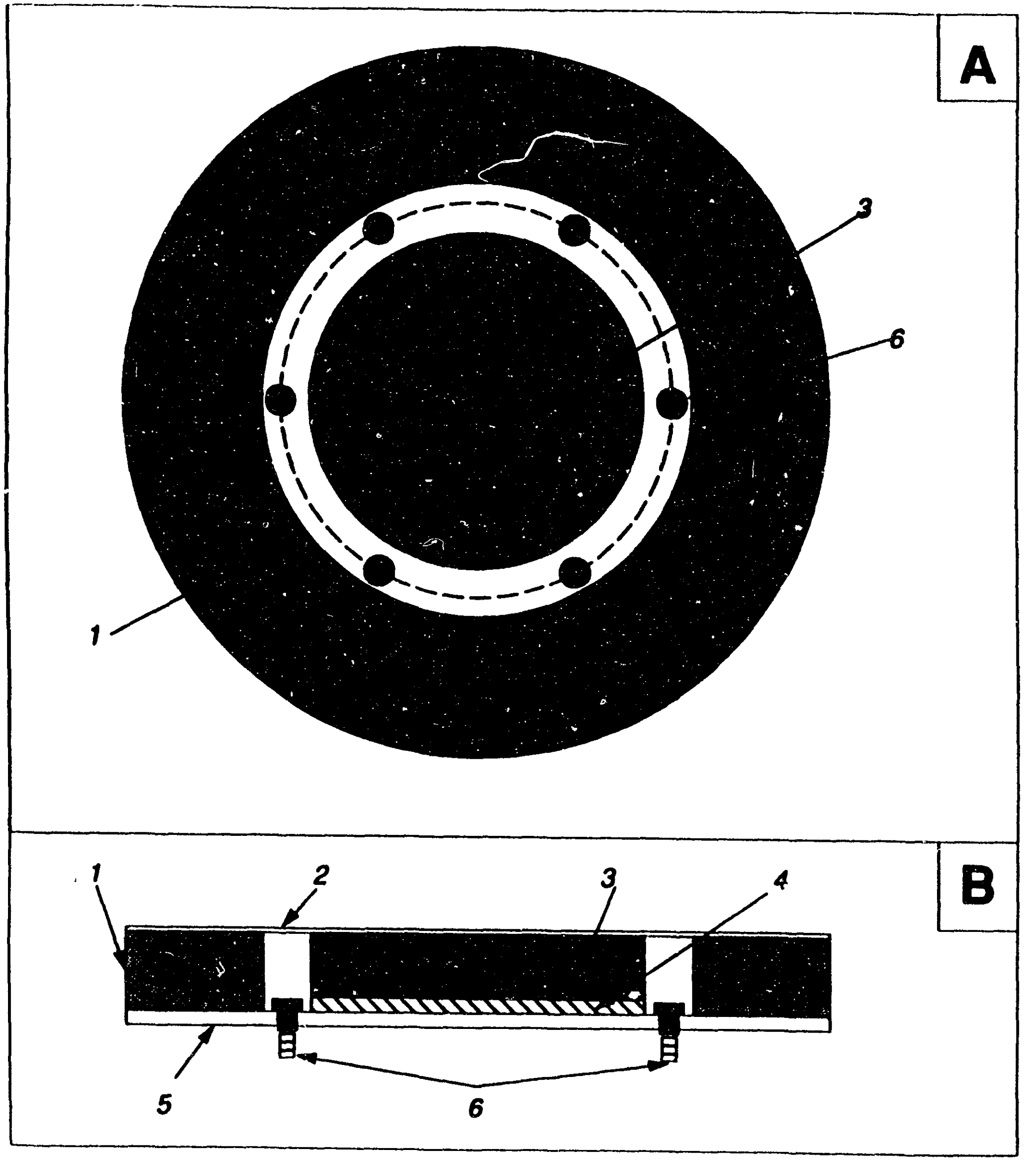
(FLOTATION FOAM "DOUGHNUT")

DRAWN BY: TERRENCE W. FONG

SCALE: 1:4 | DATE: 2/15/90 | 1 of 1

Dimensions in inches unless otherwise noted

Standard Tolerances:
X X/X = ± 1/16"
X.XX = ± 0.01"
X.XXX = ± 0.002"



#	DESCRIPTION
1	FOAM RING
2	1/8" AL6063 PLATE
3	SOLENOID BOX
4	SOLENOID BOX LID
5	TOP PLATFORM PLATE
6	PIC 4451 SCREW

MIT SPACE SYSTEMS LABORATORY

STEWART PLATFORM
(FLOTATION SANDWICH SECT.)

DRAWN BY: TERRENCE W. FONG

SCALE: 1:4 | DATE: 8/29/89 | 1 of 1

dimensions in inches unless specified

Appendix C
SPAM Software

This appendix contains a complete listing of the software used to operate and test the *Stewart Platform Augmented Manipulator*. All software code was written for the Microsoft C v5.1 Optimizing Compiler and is intended to be run on an IBM PC/XT, PC/AT or compatible. All of the software generates color video display and assumes that an IBM CGA or CGA-emulating video card is present. Additionally, the manipulator arm control software (noted below) requires that the *Manipulator Controller* board (see Appendix A) be installed due to hardware signals used for control loop timing.

The following software is described in this appendix:

SPAM Open-loop and closed-loop control of a single R-PHAD-A joint. Requires the *Manipulator Controller.*

PHAD Open-loop and closed-loop control of a single L-PHAD joint. Code is identical to SPAM software except for small modifications. Details of differences are given in the listing. Requires the *Manipulator Controller.*

SPAM2 Open-loop and closed-loop control of a 2-link planar manipulator with R-PHAD-A joints. Requires the *Manipulator Controller.*

SPLAT Stewart Platform kinematic testing. Intended to control the Stewart Platform, but hardware interfacing and coding was not completed.

QUADOBS Quadrature position sampling. Requires the *Quadrature Observer* (see Appendix A).

All code listings were generated on 4 May 1990.

```
#-----------------------------------------------------------------------------
# SPAM - Microsoft C v5.1 MAKE file for SPAM.EXE
#
#   Author   :  Terry Fong
#   Created  :  11-1-89
#
#   NOTES: Exercise caution when compiling with /Oa (relax alias checking) !!!
#          (remember:  /Ox = /Oailt /Gs )
#-----------------------------------------------------------------------------
#
# Macro definitions
#
model=/AS /FPi87                     # small memory model, inline 8087 support
dbgcomp=/c /Od /Zi $(model)          # create code for CodeView debugger
stdcomp=/c /Oilt /Gs $(model)        # optimize loops & speed, no stack probes
stdlink=/NOI /NOD                    # case-sensitive, no default library search
dbglink=/NOI /NOD /CO                # for CodeView debugger
stdlibs=slibc7+graphics+gfs
objects=spam+support+spamio+control
progdep=spam.obj support.obj spamio.obj control.obj
program=spam.exe

#
# Standard .C -> .OBJ inference rule
#
.C.OBJ :
      cl $(stdcomp) $*.c

#
# Dependency block definitions for compile with above rule
#
control.obj  : control.c portdefs.h

spam.obj     : spam.c portdefs.h spam.h

spamio.obj   : spamio.c portdefs.h

support.obj  : support.c

#
# Create SPAM.EXE !!!
#
$(program) : $(progdep)
      link $(stdlink) $(objects),$*,NUL,$(stdlibs)
```

```
/*-----------------------------------------------------------------------------

PORTDEFS.H - Prototype Expansion Board I/O Port definitions

    Author   :  Terry Fong
    Created  :  05-04-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-----------------------------------------------------------------------------*/

#define CTRL_PORT1      0x3E3 /* 8255-1 port definitions */
#define PORT_C1         0x3E2
#define PORT_B1         0x3E1
#define PORT_A1         0x3E0

#define CTRL_PORT2      0X3E7 /* 8255-2 port definitions */
#define PORT_C2         0x3E6
#define PORT_B2         0x3E5
#define PORT_A2         0x3E4
```

```
/*--------------------------------------------------------------------------

CONTROL.H - Function declarations for CONTROL.C

    Author   :  Terry Fong
    Created  :  01-23-90
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
--------------------------------------------------------------------------*/
#define BD_STEPS  3

typedef struct pos_node {
      double angle;
      struct pos_node *next;
      struct pos_node *prev;
} pos_node;

void BLAH_vel_servo (double des_pos, double rate, FILE *fp, char DATASAVE);
void P_vel_servo (double des_pos, double rate, FILE *fp, char DATASAVE);
void PD_vel_servo (double des_pos, double rate, FILE *fp, char DATASAVE);
void kickit_vel_servo (double des_pos, double rate, FILE *fp, char DATASAVE);
void PD_pos_servo (double des_pos, double rate, FILE *fp, char DATASAVE);
void calib_T (void);
void init_pos_ring (void);

#define round(x)  ((int) floor(x+0.5))
```

```
/*--------------------------------------------------------------------------

  CONTROL.C - SPAM Closed loop joint control algorithms

  Author   :  Terry Fong
  Created  :  01-21-90
  Modified :  04-13-90

  MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY

--------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <sys\timeb.h>
#include "spam.h"
#include "portdefs.h"

#define CAJ_ITERS 30    /* number of loops for calib_T */

extern double T;
pos_node pos_list[BD_STEPS];
double deadband;          /* set by init_system in SPAMIO.C */

/*
** BLAH veloc servo - if too slow, pw++. if too fast, pw--
*/
void BLAH_vel_servo (double des_pos, double des_rate, FILE *fp, char DATASAVE)
{
      int pulse_width = 0;
      int direction;

      double da;
      double cur_rate;

      char buffer[80];

      pos_node *cur_pos;

      init_pos_ring ();
      if (DATASAVE)
            fprintf (fp, "Start of BLAH Control\n");

      sprintf (buffer, "BLAH Control");
      message (buffer);

      cur_pos = &pos_list[0];
      cur_pos->angle = read_position ();

      if (cur_pos->angle < des_pos) {      /* need to close joint */
            open_sol (D);
            open_sol (B);
            direction = CLOSE;
      }
      else {                          /* need to open joint */
            open_sol (C);
            open_sol (A);
            direction = OPEN;
      }

      cur_pos = cur_pos->next;
```

```
    do {
        /* backward difference two time steps to get velocity */
        cur_pos->angle = read_position();
        da = fabs (cur_pos->angle - cur_pos->prev->angle);
        cur_rate = da / (T*BD_STEPS);

        /* if positive error, then we need to increase joint rate.
            negative error, then we need to decrease joint rate. */
        if ((des_rate - cur_rate) > 0) pulse_width++;
        else pulse_width--;

        /* actuators can be saturated !!! */
        if (pulse_width < 0) pulse_width = 0;
        else if (pulse_width > 15) pulse_width = 15;

        synch ();   /* always synch to PWM clock */

        if (direction == CLOSE) /* close joint */
            pulse_sol (C, pulse_width);
        else        /* open  joint */
            pulse_sol (D, pulse_width);

        /* display stuff and/or save stuff */
        show_state (cur_pos->angle, cur_rate);
        if (DATASAVE) {
          fprintf (fp, "%lf,%lf,%d\n",
          cur_pos->angle, cur_rate, pulse_width);
        }

        /* update loop variables */
        cur_pos = cur_pos->next;

    } while ((fabs (cur_pos->angle-des_pos) > deadband)
          && (!kbhit()));              /* or a key press */

    close_all (); /* shut off all solenoids */

    message ("CL Control Terminated ...");
    beep ();
    if (kbhit ()) getch ();       /* clear keyboard buffer */
}

/*
**  Proportional velocity error servo
*/
void P_vel_servo (double des_pos, double des_rate, FILE *fp, char DATASAVE)
{
    double pulse_width = 0;
    int direction;

    double da;
    double cur_rate;
    double Kp, error, command;

    char buffer[80];

    pos_node *cur_pos;

    init_pos_ring ();
    if (DATASAVE)
        fprintf (fp, "Start of P Control\n");

    get_double ("Kp: ", &Kp);
    sprintf (buffer, "ProVelocity Control with Kp = %lf", Kp);
    message (buffer);

    cur_pos = &pos_list[0];
    cur_pos->angle = read_position ();

    if (cur_pos->angle < des_pos) {     /* need to close joint */
        open_sol (D);
        open_sol (B);
        direction = CLOSE;
    }
    else {                       /* need to open joint */
        open_sol (C);
        open_sol (A);
        direction = OPEN;
    }

    cur_pos = cur_pos->next;

    do {
        /* backward difference two time steps to get velocity */
        cur_pos->angle = read_position();
        da = fabs (cur_pos->angle - cur_pos->prev->angle);
        cur_rate = da / (T*BD_STEPS);

        /* if positive error, then we need to increase joint rate.
            negative error, then we need to decrease joint rate. */
        error = des_rate - cur_rate;

        /* calculate command signal */
        command = Kp * error;

        /* actuators can be saturated !!! */
        pulse_width += command;
        if (pulse_width < 0) pulse_width = 0;
        else if (pulse_width > 15) pulse_width = 15;

        synch ();   /* always synch to PWM clock */

        if (direction == CLOSE) /* close joint */
            pulse_sol (C, round(pulse_width));
        else        /* open  joint */
            pulse_sol (D, round(pulse_width));

        /* display stuff and/or save stuff */
        show_state (cur_pos->angle, cur_rate);
        if (DATASAVE) {
          fprintf (fp, "%lf,%lf,%lf,%lf\n",
          cur_pos->angle, cur_rate, command, pulse_width);
        }

        /* update loop variables */
        cur_pos = cur_pos->next;

    } while ((fabs (cur_pos->angle-des_pos) > deadband)
          && (!kbhit()));              /* or a key press */

    close_all (); /* shut off all solenoids */
```

```
        message ("CL Control Terminated ...");
        beep ();
        if (kbhit ()) getch ();        /* clear keyboard buffer */
}


/*
**  Proportional-Derivative velocity error control, positional form
*/
void PD_vel_servo (double des_pos, double des_rate, FILE *fp, char DATASAVE)
{
        double pulse_width = 0;
        int direction;

        double da;
        double cur_rate;
        double cur_error, last_error=0;
        double Kp, Kd, K1, K2, command;

        char buffer[80];

        pos_node *cur_pos;

        init_pos_ring ();

        get_double ("Kp: ", &Kp);
        get_double ("Kd: ", &Kd);
        K1 = Kp + Kd/T;
        K2 = Kd/T;
        sprintf (buffer, "PD Control: Kp=%5.3lf, Kd=%5.3lf", Kp, Kd);
        message (buffer);
        if (DATASAVE) {
                fprintf (fp, "Start of P-D Veloc Servo Control\n");
                fprintf (fp, "Kp=%5.3lf, Kd=%5.3lf, K1=%5.3lf, K2=%5.3lf\n",
                        Kp, Kd, K1, K2);
        }

        cur_pos = &pos_list[0];
        cur_pos->angle = read_position ();

        if (cur_pos->angle < des_pos) {       /* need to close joint */
                open_sol (D);
                open_sol (B);
                direction = CLOSE;
        }
        else {                          /* need to open joint */
                open_sol (C);
                open_sol (A);
                direction = OPEN;
        }

        cur_pos = cur_pos->next;

        do {

                /* backward difference to get velocity */
                cur_pos->angle = read_position();
                da = fabs (cur_pos->angle - cur_pos->prev->angle);

                cur_rate = da / (T*BD_STEPS);
```

```
                /* calculate error, e(kT) */
                cur_error = des_rate - cur_rate;

                /* calculate command u(kT) */
                command = K1*cur_error - K2*last_error;

                /* actuators can be saturated !!! */
                pulse_width += command;
                if (pulse_width < 0) pulse_width = 0;
                else if (pulse_width > 15) pulse_width = 15;

                synch ();   /* always synch to PWM clock */

                if (direction == CLOSE) /* close joint */
                        pulse_sol (C, round(pulse_width));
                else            /* open  joint */
                        pulse_sol (D, round(pulse_width));

                /* display stuff and/or save stuff */
                show_state (cur_pos->angle, cur_rate);
                if (DATASAVE) {
                  fprintf (fp, "%lf,%lf,%lf,%lf\n",
                  cur_pos->angle, cur_rate, command, pulse_width);
                }

                /* update loop variables */
                cur_pos = cur_pos->next;
                last_error = cur_error;

        } while ((fabs (cur_pos->angle-des_pos) > deadband)
              && (!kbhit()));                  /* or a key press */

        close_all (); /* shut off all solenoids */

        message ("CL Control Terminated ...");
        beep ();
        if (kbhit ()) getch (); /* clear keyboard buffer */
}

/*
** KickIt - Let's give it a little kick before doing BLAH
*/
void kickit_vel_servo (double des_pos,double des_rate,FILE *fp,char DATASAVE)
{
/* how about either: (a) calculating #of full PWM kick cycles OR
   (b) calculating starting PWM level (since BLAH starts at pulse_width=0) */
}

/*
** PD position servo control. Note: the RATE argument is ignored !!!
*/
void PD_pos_servo (double des_pos, double rate, FILE *fp, char DATASAVE)
{
        double pulse_width = 0;
        int direction;

        double da, cur_rate;
        double cur_error, last_error=0;
        double Kp, Kd, K1, K2;
```

```
char buffer[80];

pos_node *cur_pos;

init_pos_ring ();

get_double ("Kp: ", &Kp);
get_double ("Kd: ", &Kd);
K1 = Kp + Kd/T;
K2 = Kd/T;
sprintf (buffer, "PD Control: Kp=%5.3lf, Kd=%5.3lf", Kp, Kd);
message (buffer);
if (DATASAVE) {
      fprintf (fp, "Start of P-D Pos Servo Control\n");
      fprintf (fp, "Kp=%5.3lf, Kd=%5.3lf, K1=%5.3lf, K2=%5.3lf\n",
            Kp, Kd, K1, K2);
}

cur_pos = &pos_list[0];
cur_pos->angle = read_position ();

if (cur_pos->angle < des_pos) {      /* need to close joint */
      open_sol (D);
      open_sol (B);
      direction = CLOSE;
}
else {                          /* need to open joint */
      open_sol (C);
      open_sol (A);
      direction = OPEN;
}

cur_pos = cur_pos->next;

do {

      /* backward difference to get velocity */
      cur_pos->angle = read_position();

      da = fabs (cur_pos->angle - cur_pos->prev->angle);
      cur_rate = da / (T*BD_STEPS);

      /* calculate error, e(kT) */
      cur_error = des_pos - cur_pos->angle;

      /* calculate command u(kT) */
      pulse_width = K1*cur_error - K2*last_error;

      /* actuators can be saturated !!! */
      if (pulse_width < 0) pulse_width = 0;
      else if (pulse_width > 15) pulse_width = 15;

      synch ();    /* always synch to PWM clock */

      if (direction == CLOSE) /* close joint */
            pulse_sol (C, round(pulse_width));
      else        /* open  joint */
            pulse_sol (D, round(pulse_width));

      /* display stuff and/or save stuff */
      show_state (cur_pos->angle, cur_rate);

      if (DATASAVE) {
        fprintf (fp, "%lf,%lf,%lf\n",
        cur_pos->angle, cur_rate, pulse_width);
      }

      /* update loop variables */
      cur_pos = cur_pos->next;
      last_error = cur_error;

   } while ((fabs (cur_pos->angle-des_pos) > deadband)
         && (!kbhit()));                   /* or a key press */

   close_all (); /* shut off all solenoids */

   message ("CL Control Terminated ...");
   beep ();
   if (kbhit ()) getch (); /* clear keyboard buffer */
}

/*
** Calculate Sampling Period (T) by comparing to the system clock.
** The result is stored in the module global variable "T".
*/
void calib_T (void)
{
      struct timeb start, end;
      double start_time, end_time;
      register int i;

      printf ("\n\n Please wait...testing sampling interval!!!\n");
      printf (" Press any key to stop calibration...\n\t");
      for (i=0; i<CAL_ITERS; i++) printf(".");
      for (i=0; i<CAL_ITERS; i++) printf("\b");

      ftime (&start);   /* grab start time from system clock */
      for (i=0; i<CAL_ITERS; i++) { /* average over CAL_ITERS synchs */
            synch ();
            printf (" ");
            if (kbhit()) {
                  getch ();
                  break;
            }
      }
      ftime (&end); /* grab end time from system clock */

      /* convert structure time to decimal seconds */
      start_time= (double) start.time + (double) start.millitm/1000;
      end_time= (double) end.time + (double) end.millitm/1000;

      T = (end_time-start_time) / i;

      printf ("\n\n\t Sampling Interval (T) = %5.3lf seconds\n", T);
      printf ("\t Pulse Width Mod. Freq = %5.3lf Hertz\n", 1/T);
      printf ("\n Press any key to continue.\n\n\n");
      while (!kbhit());
      getch ();
}

/*
** Setup backwards difference circular linked list.
*/
```

```
void init_pos_ring (void)
{
	register i;

	for (i=0; i<BD_STEPS; i++) pos_list[i].angle=0;

	for (i=0; i<BD_STEPS-1; i++) {
		pos_list[i].next = &pos_list[i+1];
	}
	pos_list[BD_STEPS-1].next=&pos_list[0];

	for (i=BD_STEPS-1; i>0; i--) {
		pos_list[i].prev = &pos_list[i-1];
	}
	pos_list[0].prev = &pos_list[BD_STEPS-1];
}
```

```
/*---------------------------------------------------------------------------

SPAM.H - Contains the definition of a solenoid object, useful labels and
       function declarations for all routines.

    Author   :  Terry Fong
    Created  :  05-09-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

/* Solenoid stuff */
#define A           1
#define B           2
#define C           3
#define D           4
#define MAX_PULSE 15    /* PWM circuit max pulse = 15 = full ON */
#define MIN_PULSE 1     /* since a pulse of 0 is not useful */
#define CLOSE_DELAY     3     /* in seconds */

/* PC keyboard key definitions */

/* Misc. labels */
#define ON        1
#define CLOSE       '
#define OPEN        (
#define OFF       0
#define TRUE        1
#define FALSE       0

/*------------------ Declaration of function prototypes ----------------------*/

/* CONTROL.C */
#include "control.h"

/* SPAM.C */
void user_loop (void);
void choose_control (void);

/* SUPPORT.C */
#include "support.h"

/* SPAMIO.C */
#include "spamio.h"

/*------------------------ End of function declarations ----------------------*/
```

```
/*
**
**  SPAM.C - Run the elbow joint with closed loop control
**
**  Author   :  Terry Fong
**  Created  :  01-20-90
**  Modified :
**
**  MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
**
*/

#include <stdio.h>
#include <math.h>
#include <graph.h>
#include <gnleaf\gf.h>
#include <gnleaf\ibmkeys.h>

#include "portdefs.h"   /* 8255 PIA port definitions */
#include "spam.h" /* solenoid, misc labels and function prototypes */

/* Globals */
void (*control) ();     /* function pointer to CL control routine */
double T;          /* sampling period (seconds) */

/*
**
**  Program main control.
**
*/
main ()
{
      choose_control ();      /* pick a control algorithm */
      init_system ();   /* Initialize system */
      draw_screen ();         /* Layout the interface display, cursor off */
      user_loop ();           /* Main program loop */
      init_system ();   /* Initialize system */
      spam_exit ();           /* turn the cursor on and clean up stuff */
}


/*
**
**  Read the keyboard and perform required actions (incredible, huh?)
**
*/
void user_loop ()
{
      extern pos_node pos_list[BD_STEPS]; /* CONTROL.C */

      char DATASAVE=FALSE;
      char RUN=TRUE;
      char buffer [80];
      FILE *fp; /* file pointer for data save file */

      /* solenoid state vars */
      static struct {
            int a, b, c, d; /* ON or OFF */
            int c_pw, d_pw;   /* pulse width */
      } state = {OFF, OFF, OFF, OFF, 7, 7};

      /* commanded position and rate */
      static struct {
            double pos, rate;
      } command = {0.0, 0.0};

      /* the following are for back diff veloc calculations */
      double da, cur_rate;
      pos_node *cur_pos;

      /***** End of declarations *****/

      init_pos_ring ();
      cur_pos = &pos_list[0];
      cur_pos->angle = read_position ();
      cur_pos = cur_pos->next;

      while (RUN) {

            synch ();   /* always synch to pwm clock */

            cur_pos->angle = read_position();
            da = fabs (cur_pos->angle - cur_pos->prev->angle);

            cur_rate = da / (T*BD_STEPS);

            show_state (cur_pos->angle, cur_rate);

            if (DATASAVE) {
              fprintf (fp, "%.2lf,%.2lf,%d,%d\n",
              cur_pos->angle, cur_rate, state.c_pw, state.d_pw);
            }

            cur_pos = cur_pos->next;

            /* keyboard scanning loop */
            if (gfkbhit ()) {
                  switch (getkey()) {

                        case ' ' : /* close all solenoids */
                              close_all ();
                              state.a = OFF:
                              state.b = OFF;
                              state.c = OFF;
                              state.d = OFF;
                              break;

                        case F1 : /* toggle A */
                              if (state.a) {
                                    close_sol (A);
                                    state.a = OFF;
                              }
                              else {
                                    open_sol (A);
                                    state.a = ON;
                              }
                              break;

                        case F2 : /* toggle B */
                              if (state.b) {
                                    close_sol (B);
                                    state.b = OFF;
```

```
                }
                else {
                        open_sol (B);
                        state.b = ON;
                }
                break;

        case F3 : /* toggle C */
                if (state.c) {
                        close_sol (C);
                        state.c = OFF;
                }
                else {
                        open_sol (C);
                        state.c = ON;
                }
                break;

        case F4 : /* toggle D */
                if (state.d) {
                        close_sol (D);
                        state.d = OFF;
                }
                else {
                        open_sol (D);
                        atate.d = ON;
                }
                break;

        case F5 :
                state.c = ON;
                ++state.c_pw;
                if (state.c_pw > 15) state.c_pw = 0;
                pulse_sol (C, state.c_pw);
                sprintf (buffer, "C_pw = %d  D_pw = %d",
                        state.c_pw, state.d_pw);
                message (buffer);
                break;

        case F6 :
                state.d = ON;
                ++state.d_pw;
                if (state.d_pw > 15) state.d_pw = 0;
                pulse_sol (D, state.d_pw);
                sprintf (buffer, "C_pw = %d  D_pw = %d",
                        state.c_pw, state.d_pw);
                message (buffer);
                break;

        case F7 : /* set C and D pulse widths */
                get_pw (&state.c_pw, &state.d_pw);
                sprintf (buffer, "C_pw = %d  D_pw = %d",
                        state.c_pw, state.d_pw);
                message (buffer);
                break;

        case CURDN :
        case 'c' : /* close joint */
                open_sol (D);
                pulse_sol (C, state.c_pw);
                open_sol (B);
                state.b = ON;
                state.c = ON;
                state.d = ON;
        sprintf (buffer, "DOWN: C_pw = %d  D_pw = %d",
                        state.c_pw, state.d_pw);
                message (buffer);
                break;

        case 'e' : /* do it, to it !!! */
                prompt ("Hit any key during CL to stop!");
                (*control) (command.pos, command.rate,
                        fp, DATASAVE);
                break;

        case CURUP :
        case 'o' : /* open joint */
                open_sol (C);
                pulse_sol (D, state.d_pw);
                open_sol (A);
                state.a = ON;
                state.c = ON;
                state.d = ON;
        sprintf (buffer, "UP: C_pw = %d  D_pw = %d",
                        state.c_pw, state.d_pw);
                message (buffer);
                break;

        case 'r' : /* reset encoder zero */
                zero_position ();
                message ("Encoder zero reset !");
                break;

          case 's' : /* toggle data save */
                if (DATASAVE == OFF) {
                        if (start_save (&fp))
                                DATASAVE = ON;
                }
                else {
                        if (close_save (fp))
                                DATASAVE = OFF;
                }
                break;

        case 't' : /* handle joint command */
                get_command (&command.pos, &command.rate);
                show_command (command.pos, command.rate);
                break;

          case ESC : /* terminate program on user command */
                if (DATASAVE) close_save (fp);
                RUN = OFF;
                break;
      }
    }
  }
}

/*
**
** Choose a control algorithm, calibrate sampling period
```

```
**
*/
void choose_control (void)
{
      char ch=0;

      while ((ch < '1') || (ch > '5')) {
            _clearscreen (_GCLEARSCREEN);
            printf ("\n\n Choose your control system blah!\n");
            printf ("-----------------------------------\n");
            printf (" 1 - BLAH (PWM++/PWM--) veloc servo\n");
            printf (" 2 - P veloc servo\n");
            printf (" 3 - PD veloc servo\n");
            printf (" 4 - Kick it veloc servo\n");
            printf (" 5 - PD position servo\n");
            printf ("\n\n Enter number of desired control: ");
            scanf ("%c", &ch);
      }

      switch (ch) {
            case '1' :
                  control = BLAH_vel_servo;
                  break;

            case '2' :
                  control = P_vel_servo;
                  break;

            case '3' :
                  control = PD_vel_servo;
                  break;

            case '4' :
                  control = kickit_vel_servo;
                  break;

            case '5' :
                  control = PD_pos_servo;
                  break;
      }

      _clearscreen (_GCLEARSCREEN);
      calib_T ();
}
```

```
/*---------------------------------------------------------------------------

SPAMIO.H - Function declarations for SPAMIO.C

    Author   :  Terry Fong
    Created  :  05-04-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

void close_all (void);
void close_sol (int sol);
void open_sol (int sol);
void pulse_sol (int sol, int pulse_width);
void init_system (void);
double read_position (void);
void zero_position (void);
void synch (void);
```

```
/*---------------------------------------------------------------------------

SPAMIO.C - Hardware I/O routines. Control solenoids, read encoder.

    Author   :  Terry Fong
    Created  :  12-7-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <stdio.h>
#include <conio.h>

#include "portdefs.h"
#include "spam.h"

#define INPUT 1
#define OUTPUT 0

static unsigned char kip_word;      /* control word for the kip solenoids */


/*---------------------------------------------------------------------------
  Close all the solenoids.
---------------------------------------------------------------------------*/
void close_all (void)
{
      close_sol (A);
      close_sol (B);
      close_sol (C);
      close_sol (D);
}


/*---------------------------------------------------------------------------
  Close a solenoid.
---------------------------------------------------------------------------*/
void close_sol (int sol)
{
      switch (sol)
      {
            case A :
                  outp (CTRL_PORT2, 0x4);
                  break;

            case B :
                  outp (CTRL_PORT2, 0x6);
                  break;

            case C :
                  kip_word = kip_word & 0xf0;   /* zero lower nib */
                  outp (PORT_B2, kip_word);
                  break;

            case D :
                  kip_word = kip_word & 0x0f;   /* zero upper nib */
                  outp (PORT_B2, kip_word);
                  break;

      }
```

```
}

/*---------------------------------------------------------------------------
  Open a solenoid.
---------------------------------------------------------------------------*/
void open_sol (int sol)
{
      switch (sol)
      {
            case A :
                  outp (CTRL_PORT2, 0x5);
                  break;

            case B :
                  outp (CTRL_PORT2, 0x7);
                  break;

            case C :
                  kip_word = kip_word | 0x0f;   /* set lower nib */
                  outp (PORT_B2, kip_word);
                  break;

            case D :
                  kip_word = kip_word | 0xf0;   /* set upper nib */
                  outp (PORT_B2, kip_word);
                  break;

      }
}


/*---------------------------------------------------------------------------
  Start one of the KIP solenoids pulse modulating.
---------------------------------------------------------------------------*/
void pulse_sol (int sol, int pulse_width)
{
      switch (sol)
      {
            case C : /* clear lower nib and insert pulse_width */
                  kip_word = (kip_word & 0xf0) | pulse_width;
                  break;

            case D : /* clear upper nib and insert pulse_width */
                  kip_word = (kip_word & 0x0f) | (pulse_width << 4);
                  break;
      }
      outp (PORT_B2, kip_word);
}


/*---------------------------------------------------------------------------
  Initialize the 8255's, zero the HCTL-2000 counter, and close all solenoids.
---------------------------------------------------------------------------*/
void init_system ()
{
      extern double deadband;

      deadband = 1.0;   /* controller deadband */
      set8255 (CTRL_PORT1, INPUT, OUTPUT, OUTPUT, OUTPUT);
      set8255 (CTRL_PORT2, OUTPUT, OUTPUT, OUTPUT, INPUT);
```

```
     zero_position ();  /* set optical encoder zero position */
     close_all ();             /* close all the solenoids */
}


/*---------------------------------------------------------------------------
  Read the 12 bit optical encoder position output through the HCTL-2000
  The position MSB/LSB are selected by a reset/set of PC6 on 8255-2
--------------------------------------------------------------------------*/
double read_position ()
{
     register int lb, hb;
     double phi_e, phi, theta;

     outp (CTRL_PORT1, 0xE); /* pc7 --> OE* = reset */
     outp (CTRL_PORT1, 0xC); /* pc6 --> SEL = reset */
     hb = inp (PORT_A1);     /* PORT_A1 --> data lines */
     outp (CTRL_PORT1, 0xD); /* pc6 --> SEL = set */
     lb = inp (PORT_A1);     /* PORT_A1 --> data lines */
     outp (CTRL_PORT1, 0xF); /* pc7 --> OE* = set */

     /* calculate position 12-bit position (stored in a 16-bit word) */
     phi_e = (double) (hb*256 + lb) / 22.7555556;
     phi = (phi_e + 15.15) * 0.017453;   /* in radians */

     /* 180/pi*2 = 114.592 */
     theta = 180 - 114.592*atan(6.216*sin(phi)/(6-6.216*cos(phi)));

     return (theta);
}


/*---------------------------------------------------------------------------
Set the zero position for the optical encoder by sending a low/high pulse to
  the HCTL RST* (pin 5)
--------------------------------------------------------------------------*/
void zero_position (void)
{
     read_position ();
               outp (CTRL_PORT1, 0x8); /* pc4 --> OE* = reset */
               outp (CTRL_PORT1, 0x6); /* pc3 --> SEL = reset */
               outp (CTRL_PORT1, 0x7); /* pc3 --> SEL = high */
               outp (CTRL_PORT1, 0x9); /* pc7 --> OE* = set */

     outp (CTRL_PORT1, 0xA);       /* reset 8255-1 PC5 */
     outp (CTRL_PORT1, 0xB);       /* set   8255-1 PC5 */
     outp (CTRL_PORT1, 0x4); /* reset 8255-1 PC2 */
     outp (CTKL_PORT1, 0x5); /* set   8255-1 PC2 */

}

/*
** Wait for a PWM synch pulse at start of duty cycle. The PWM driver clock
** (a 555 circuit with two 47K resistors and a 10 mFD capacitor) synchs every
** 110 millisecs (approx)., so that we have a 10 Hz synch rate.
*/
void synch (void)
{
     /* wait for synch line to be set */
     while (!(inp(PORT_C2) & 0x10)); /* mask to check only bit 0 */

     /* The following waits for the falling edge */
     while (!(inp(PORT_C2) & 0x10));
}

/*
** This routine calculates the 8-bit control word to setup the ports of
** an 8255 for Mode 0 operation (Basic Input/Output). The control word
** is output to the specified I/O port.
**
** The flags (a, b, cl, ch) should be 0 for output, 1 for input.
*/
set8255 (unsigned int port, int a, int b, int cl, int ch)
{
     int word = 0x80;  /* Mode set flag active */

     if (a)  word |= 0x10;
     if (b)  word |= 0x02;
     if (cl) word |= 0x01;
     if (ch) word |= 0x08;

     outp (port, word);
}

/*------------------------------End of SPAMIO.C------------------------------*/
```

```
void draw_screen (void);
void draw_box (int start_row, int start_col, int end_row, int end_col,
         long back_color, short border_color);
void printsc (int row, int col, char *string);
void prompt (char *string);
void message (char *string);
void cursor_on (void);
void cursor_off (void);
void show_command (double position, double rate);
void show_state (double angle, double rate);
void get_command (double *position, double *rate);
void get_pw (int *c_pw, int *d_pw);
void get_double (char *string, double *number);
void delay (int seconds);
int start_save (FILE **fp);
int close_save (FILE *fp);
void spam_exit (void);
void beep (void);
```

```
/*---------------------------------------------------------------------------

SUPPORT.C - Misc. support routines including video stuff, user i/o

    Author   :  Terry Fong
    Created  :  05-04-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <dos.h>
#include <graph.h>
#include <stdio.h>
#include <string.h>
#! clude <time.h>
#include <gnfuns\color.h>

#include "spam.h"

#define BLINK     16

/*---------------------------------------------------------------------------
  Layout the user interface in vivid color !!!
---------------------------------------------------------------------------*/
void draw_screen (void)
{
      /* Beep !!! */
      beep ();

      /* Init the video system for color */
      _setvideomode (_TEXTC80);
      _setbkcolor ((long) BLUE);
      _clearscreen (_GCLEARSCREEN);

      /* The Title & Info Box ... */
      draw_box (2, 16, 9, 63, (long) WHITE, BLACK);
      _setbkcolor ((long) WHITE);
      _settextcolor (BLUE);
      printsc (3, 18, "SPAM Closed-Loop Controller v2.1  (01/20/90)");
      printsc (4, 18, "        MIT Space Systems / LOOP Group        ");
      _settextcolor (BLACK);
      p r i n t s c                    ( 5 ,                    1 6 ,
"");
      printsc (6, 18, "Press [t] to spec. target, [e] to execute it");
      printsc (7, 18, "      [g] set CL_GAIN, [s] data save on/off");
      printsc (8, 18, "      [ESC] to quit");

      /* The Commanded state box */
      draw_box (11, 59, 14, 79, (long) CYAN, BLACK);
      _setbkcolor ((long) CYAN);
      _settextcolor (BLACK);
      printsc (11, 66, " Target ");
      printsc (12, 61, "Target pos:");
      printsc (13, 61, "Target vel:");

      /* The Joint State Box ... */
      draw_box (15, 59, 18, 79, (long) CYAN, BLACK);
      _setbkcolor ((long) CYAN);
      _settextcolor (BLACK);
```

```
        printsc (15, 64, "Joint State ");
        printsc (16, 61, "Position:");
        printsc (17, 61, "Velocity:");

        /* The Messages Box ... */
        draw_box (20, 2, 24, 79, (long) RED, BLACK);
        _setbkcolor ((long) RED);
        _settextcolor (BLACK);
        printsc (20, 35, " Messages ");

        /* The User Interface box */
        draw_box (11, 2, 18, 57, (long) LTGREEN, BLACK);
        _setbkcolor ((long) LTGREEN);
        _settextcolor (BLACK);
        printsc (11, 22, " Blah, blah! ");
        printsc (11, 22, " Terry's box ");

        show_state (0.0, 0.0);
        show_command (0.0, 0.0);

        _settextcolor (WHITE);
        _setbkcolor ((long) BLUE);
        printsc (2, 2, "F1-toggle A");
        printsc (3, 2, "F2-toggle B");
        printsc (4, 2, "F3-toggle C");
        printsc (5, 2, "F4-toggle D");
        printsc (6, 2, "F5-inc C_pw");
        printsc (7, 2, "F6-inc D_pw");
        printsc (8, 2, "F7-set pw's");
        printsc (2, 68, "Close Joint");
        printsc (3, 68, "  c / CURDN");
        printsc (4, 68, "Open Joint");
        printsc (5, 68, "  o / CURUP");

        cursor_off ();
}


/*---------------------------------------------------------------------------
  Draw a nice looking colored box. The coordinate origin is the upper left
  corner of the screen (1, 1). The bottom right corner is (25, 80).
---------------------------------------------------------------------------*/
void draw_box (int start_row, int start_col, int end_row, int end_col,
          long back_color, short border_color)
{
        char  top[81], middle[81], bottom[81];    /* Max box is 80 char wide */
        register int   i;
        long  old_back;
        short old_text;

        /* Save the current colors before messing with them */
        old_back = _getbkcolor ();
        old_text = _gettextcolor ();

        /* Compile the strings for the box */
        top [0] = '';          /* char 201 */
        middle [0] = '';       /* char 186 */
        bottom [0] = '';       /* char 200 */
        for (i = 1; i < (end_col - start_col); i++) {
              top [i] = '';             /* char 205 */
              middle [i] = ' '; /* char 32  */
              bottom [i] = '';        /* char 205 */
        }
        top [i] = '';                /* char 187 */
        middle [i] = '';             /* char 186 */
        bottom [i] = '';             /* char 188 */

        /* terminate all the strings with a null char. */
        i++;
        top [i] = '\0';
        middle [i] = '\0';
        bottom [i] = '\0';

        _setbkcolor (back_color);
        _settextcolor (border_color);

        /* Draw the top line */
          _settextposition (start_row, start_col);
        _outtext (top);

        /* Draw the middle lines */
        for (i=1; i < (end_row - start_row); i++) {
              _settextposition (start_row + i, start_col);
              _outtext (middle);
        }

        /* Draw the bottom of the box */
        _settextposition (end_row, start_col);
        _outtext (bottom);

        /* Restore the original colors */
        _setbkcolor (old_back);
        _settextcolor (old_text);
}


/*---------------------------------------------------------------------------
  Print a string at a specified row and column in a specific.
---------------------------------------------------------------------------*/
void printsc (int row, int col, char *string)
{
        _settextposition (row, col);
        _outtext (string);
}


/*---------------------------------------------------------------------------
  Print an error message, beep, and wait for a key to be pressed.
---------------------------------------------------------------------------*/
void prompt (char *string)
{
        long  old_back;
        short old_text;

        /* Save the current colors before messing with them */
        old_back = _getbkcolor ();
        old_text = _gettextcolor ();

        _setbkcolor ((long) RED);
        _settextcolor (WHITE);
        _settextposition (21, 5);
        _outtext ("                                                          ");
        _settextposition (21,5);
        _outtext (string);
        _settextposition (23, 5);
        _settextcolor (WHITE+BLINK);
        _outtext ("Press any key to continue . . .");
        beep ();

        while (!kbhit()); /* Wait for a keypress, discard char */
        getch ();

        _settextposition (23, 5);
        _outtext ("                                                  ");

        /* Restore the original colors */
        _setbkcolor (old_back);
        _settextcolor (old_text);
}


/*---------------------------------------------------------------------------
  Print a message.
---------------------------------------------------------------------------*/
void message (char *string)
{
        long  old_back;
        short old_text;

        /* Save the current colors before messing with them */
        old_back = _getbkcolor ();
        old_text = _gettextcolor ();

        _setbkcolor ((long) RED);
        _settextcolor (WHITE);
        _settextposition (21, 5);
        _outtext ("                                                  ");
        _settextposition (21,5);
        _outtext (string);

        /* Restore the original colors */
        _setbkcolor (old_back);
        _settextcolor (old_text);
}


/*---------------------------------------------------------------------------
  Turn off the cursor by playing with INT 10H.
---------------------------------------------------------------------------*/
void cursor_off (void)
{
        union REGS regs;

        regs.h.ah = 1;
        regs.h.ch = 0x20; /* set bit 5 to disable cursor */

        int86 (0x10, &regs, &regs);
}


/*---------------------------------------------------------------------------
  Turn on the cursor with INT 10H
---------------------------------------------------------------------------*/
void cursor_on (void)
{
        union REGS regs;

        regs.h.ah = 1;
        regs.h.ch = 6;
        regs.h.cl = 7;

        int86 (0x10, &regs, &regs);
}


/*---------------------------------------------------------------------------
  Display the command in the "Command box" (incredible isn't it?)
---------------------------------------------------------------------------*/
void show_command (double position, double rate)
{
        char buffer [10];

        _setbkcolor ((long) CYAN);
        _settextcolor (BLACK);
        sprintf (buffer, "%6.2lf", position);
        printsc (12, 72, buffer);
        sprintf (buffer, "%6.2lf", rate);
        printsc (13, 72, buffer);
}


/*---------------------------------------------------------------------------
  Display the current state in the "State box"
---------------------------------------------------------------------------*/
void show_state (double angle, double rate)
{
        char buffer [10];

        _setbkcolor ((long) CYAN);
        _settextcolor (BLACK);
        sprintf (buffer, "%6.2lf", angle);
        printsc (16, 72, buffer);
        if (rate > 100) return;
        sprintf (buffer, "%6.2lf", rate);
        printsc (17, 72, buffer);
}


/*---------------------------------------------------------------------------
  Query the user for position and rate commands.
---------------------------------------------------------------------------*/
void get_command (double *position, double *rate)
{
    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (13, 5, "Desired position (DEG): ");
    scanf ("%lf", position);
    printsc (14, 5, "Desired rate (DEG/SEC): ");
    scanf ("%lf", rate);
    printsc (13, 5, "                                                  ");
    printsc (14, 5, "                                                  ");
    cursor_off ();
}
```

```
/*---------------------------------------------------------------------------
  Query the user for pulse_widths
----------------------------------------------------------------------------*/
void get_pw (int *c_pw, int *d_pw)
{
    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (13, 5, "Solenoid C Pulse Width: ");
    scanf ("%d", c_pw);
    printsc (14, 5, "Solenoid D Pulse Width: ");
    scanf ("%d", d_pw);
    printsc (13, 5, "                                  ");
    printsc (14, 5, "                                  ");
    cursor_off ();
}


/*---------------------------------------------------------------------------
  Query the user for pulse_widths
----------------------------------------------------------------------------*/
void get_double (char *string, double *number)
{
    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (13, 5, string);
    scanf ("%lf", number);
    printsc (13, 5, "                                  ");
    cursor_off ();
}


/*----------------------------------------------------------------------------
  Just wait around for a few seconds
/*--------------------------------------------------------------------------*/
void delay (int seconds)
{
      long start, cur;

      time (&start);
      do {
            time (&cur);
      } while ((cur-start) < seconds);
}


/*----------------------------------------------------------------------------
  Open a file for data output. Returns a TRUE if file is opened successfully.
----------------------------------------------------------------------------*/
int start_save (FILE **fp)
{
    char filename [60];
    char buffer [80];

    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (16, 5, "                                  ");
    printsc (16, 5, "Enter filename [.PRN]: ");
    scanf ("%s", filename);
    strcat (filename, ".prn");
    cursor_off ();

    if ((*fp=fopen(filename, "w"))==NULL)
    {
      prompt ("ERROR: Cannot open file !!!!!");
      return (FALSE);
    }
    else
    {
      sprintf (buffer, "Position and rate data saved to: %s",
            strupr(filename));
      printsc (16, 5, buffer);
/*
      fprintf (*fp, "SPAM Data File\n");
      fprintf (*fp, "CurAngle CurRate cp dp\n\n");
*/
      return (TRUE);
    }
}


/*----------------------------------------------------------------------------
  Close the data file. Returns a TRUE if closed correctly.
----------------------------------------------------------------------------*/
int close_save (FILE *fp)
{
    message ("Data file closed.");
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (16, 5, "                                                    ");
    if ((fclose (fp)) == EOF) {
          prompt ("ERROR: Unable to close file !!!!!");
          return (FALSE);
    }
    else
          return (TRUE);
}


/*----------------------------------------------------------------------------
  Exit gracefully (of course!)
----------------------------------------------------------------------------*/
void spam_exit (void)
{
      _settextposition (25, 1);
      printf ("\n SPAM CL Control Done.\n\a");
      cursor_on ();
}

void beep (void)
{
      printf ("\a");
}
/*------------------------ End of SUPPORT.C -------------------------------*/
```

PHAD Listing

```
#------------------------------------------------------------------------------
#
# PHAD - Makefile for PHAD.EXE
#
#  Author   :  Terry Fong
#  Created  :  04-18-90
#  Modified :
#
# NOTE: PHAD.EXE uses the same functions as SPAM.EXE with these mods...
#
# SPAM file   PHAD file     routine         modification
# ---------   -----------   -------------   ------------------------------
# SPAMIO.C  -> PSPAMIO.C    read_position   12-bit linear position instead
#                                           of revolute joint angle
# SUPPORT.C -> PSUPPORT.C   show_state      displayed position not float
#                           show_command    displayed position not float
#                           draw_screen     prints PHAD instead of SPAM
#
#------------------------------------------------------------------------------
#
# Macro definitions
#
model=/AS /FPi87          # small memory model, inline 8087 support
dbgcomp=/c /Od /Zi $(model)   # create code for use with CodeView debugger
stdcomp=/c /Oilt /Gs $(model) # optimize loops & speed, no stack probes
stdlink=/NOI /NOD         # case-sensitive, no default library search
dbglink=/NOI /NOD /CO         # same as stdlink, but for CodeView debugger
stdlibs=slibc7+graphics+gfs
objects=spam+psupport+pspamio+control
progdep=spam.obj psupport.obj pspamio.obj control.obj
program=phad.exe

#
# Standard .C -> .OBJ inference rule
#
.C.OBJ :
      cl $(stdcomp) $*.c

#
# Dependency block definitions for compile with above rule
#
control.obj  : control.c portdefs.h

spam.obj     : spam.c portdefs.h spam.h

pspamio.obj   : pspamio.c portdefs.h

psupport.obj  : psupport.c

#
# Create PHAD.EXE !!!
#
$(program) : $(progdep)
      link $(stdlink) $(objects),$*,NUL,$(stdlibs)
```

```
/*-----------------------------------------------------------------------------

PSPAMIO.C - Hardware I/O routines. Control solenoids, read encoder.

    Author   :  Terry Fong
    Created  :  12-7-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
----------------------------------------------------------------------------*/

#include <stdio.h>
#include <conio.h>

#include "portdefs.h"
#include "spam.h"

#define INPUT 1
#define OUTPUT 0

static unsigned char kip_word;      /* control word for the kip solenoids */

/*-----------------------------------------------------------------------------
  Close all the solenoids.
----------------------------------------------------------------------------*/
void close_all (void)
{
      close_sol (A);
      close_sol (B);
      close_sol (C);
      close_sol (D);
}

/*-----------------------------------------------------------------------------
  Close a solenoid.
----------------------------------------------------------------------------*/
void close_sol (int sol)
{
      switch (sol)
      {
            case A :
                  outp (CTRL_PORT2, 0x4);
                  break;

            case B :
                  outp (CTRL_PORT2, 0x6);
                  break;

            case C :
                  kip_word = kip_word & 0xf0;   /* zero lower nib */
                  outp (PORT_B2, kip_word);
                  break;

            case D :
                  kip_word = kip_word & 0x0f;   /* zero upper nib */
                  outp (PORT_B2, kip_word);
                  break;
```

```
        }
}

/*----------------------------------------------------------------------------
  Open a solenoid.
----------------------------------------------------------------------------*/
void open_sol (int sol)
{
        switch (sol)
        {
                case A :
                        outp (CTRL_PORT2, 0x5);
                        break;

                case B :
                        outp (CTRL_PORT2, 0x7);
                        break;

                case C :
                        kip_word = kip_word | 0x0f;    /* set lower nib */
                        outp (PORT_B2, kip_word);
                        break;

                case D :
                        kip_word = kip_word | 0xf0;    /* set upper nib */
                        outp (PORT_B2, kip_word);
                        break;

        }
}

/*----------------------------------------------------------------------------
  Start one of the KIP solenoids pulse modulating.
----------------------------------------------------------------------------*/
void pulse_sol (int sol, int pulse_width)
{
        switch (sol)
        {
                case C : /* clear lower nib and insert pulse_width */
                        kip_word = (kip_word & 0xf0) | pulse_width;
                        break;

                case D : /* clear upper nib and insert pulse_width */
                        kip_word = (kip_word & 0x0f) | (pulse_width << 4);
                        break;
        }
        outp (PORT_B2, kip_word);
}

/*---------------------------------------------------------------------------
  Initialize the 8255's, zero the HCTL-2000 counter, and close all solenoids.
----------------------------------------------------------------------------*/
void init_system ()
{
        extern double deadband;

        deadband = 10.0;  /* controller deadband */
        set8255 (CTRL_PORT1, INPUT, OUTPUT, OUTPUT, OUTPUT);
        set8255 (CTRL_PORT2, OUTPUT, OUTPUT, OUTPUT, INPUT);
        zero_position ();  /* set optical encoder zero position */
        close_all ();            /* close all the solenoids */
}

/*----------------------------------------------------------------------------
  Read the 12 bit optical encoder position output through the HCTL-2000
  The position MSB/LSB are selected by a reset/set of PC6 on 8255-2
----------------------------------------------------------------------------*/
double read_position ()
{
        register int lb, hb, pos;

        outp (CTRL_PORT1, 0xE); /* pc7 --> OE* = reset */
        outp (CTRL_PORT1, 0xC); /* pc6 --> SEL = reset */
        hb = inp (PORT_A1);     /* PORT_A1 --> data lines */
        outp (CTRL_PORT1, 0xD); /* pc6 --> SEL = set */
        lb = inp (PORT_A1);     /* PORT_A1 --> data lines */
        outp (CTRL_PORT1, 0xF); /* pc7 --> OE* = set */

        /* calculate position 12-bit position (stored in a 16-bit word) */
        pos = hb*256 + lb;

        return ((double) pos);
}

/*----------------------------------------------------------------------------
Set the zero position for the optical encoder by sending a low/high pulse to
  the HCTL RST* (pin 5)
----------------------------------------------------------------------------*/
void zero_position (void)
{
        read_position ();
                    outp (CTRL_PORT1, 0x8); /* pc4 --> OE* = reset */
                    outp (CTRL_PORT1, 0x6); /* pc3 --> SEL = reset */
                    outp (CTRL_PORT1, 0x7); /* pc3 --> SEL = high */
                    outp (CTRL_PORT1, 0x9); /* pc7 --> OE* = set */

        outp (CTRL_PORT1, 0xA);       /* reset 8255-1 PC5 */
        outp (CTRL_PORT1, 0xB);       /* set   8255-1 PC5 */
        outp (CTRL_PORT1, 0x4); /* reset 8255-1 PC2 */
        outp (CTRL_PORT1, 0x5); /* set   8255-1 PC2 */

}

/*
** Wait for a PWM synch pulse at start of duty cycle. The PWM driver clock
** (a 555 circuit with two 47K resistors and a 10 mFD capacitor) synchs every
** 110 millisecs (approx)., so that we have a 10 Hz synch rate.
*/
void synch (void)
{
        /* wait for synch line to be set */
        while (!(inp(PORT_C2) & 0x10)); /* mask to check only bit 0 */

        /* The following waits for the falling edge */
        while ((inp(PORT_C2) & 0x10));
}
```

```
/*
** This routine calculates the 8-bit control word to setup the ports of
** an 8255 for Mode 0 operation (Basic Input/Output). The control word
** is output to the specified I/O port.
**
** The flags (a, b, cl, ch) should be 0 for output, 1 for input.
*/
set8255 (unsigned int port, int a, int b, int cl, int ch)
{
      int word = 0x80;  /* Mode set flag active */

      if (a)  word |= 0x10;
      if (b)  word |= 0x02;
      if (cl) word |= 0x01;
      if (ch) word |= 0x08;

      outp (port, word);
}

/*------------------------------End of SPAMIO.C------------------------------*/
```

```
/*--------------------------------------------------------------------------------

PSUPPORT.C - Misc. support routines including video stuff, user i/o

    Author   :  Terry Fong
    Created  :  05-04-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
------------------------------------------------------------------------------*/

#include <dos.h>
#include <graph.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <gnfuns\color.h>

#include "spam.h"

#define BLINK     16

/*--------------------------------------------------------------------------------
  Layout the user interface in vivid color !!!
------------------------------------------------------------------------------*/
void draw_screen (void)
{
      /* Beep !!! */
      beep ();

      /* Init the video system for color */
      _setvideomode (_TEXTC80);
      _setbkcolor ((long) BLUE);
      _clearscreen (_GCLEARSCREEN);

      /* The Title & Info Box ... */
      draw_box (2, 16, 9, 63, (long) WHITE, BLACK);
      _setbkcolor ((long) WHITE);
      _settextcolor (BLUE);
      printsc (3, 18, "PHAD Closed-Loop Controller v2.0  (04/08/90)");
      printsc (4, 18, "        MIT Space Systems / LOOP Group       ");
      _settextcolor (BLACK);
      p r i n t s c                         ( 5 ,                    1 6 ,
"éééééééééééééééééééééééééééééééééééééééééééééé");
      printsc (6, 18, "Press [t] to spec. target, [e] to execute it");
      printsc (7, 18, "      [g] set CL_GAIN, [s] data save on/off");
      printsc (8, 18, "      {ESC] to quit");

      /* The Commanded state box */
      draw_box (11, 59, 14, 79, (long) CYAN, BLACK);
      _setbkcolor ((long) CYAN);
      _settextcolor (BLACK);
      printsc (11, 66, " Target ");
      printsc (12, 61, "Target pos:");
      printsc (13, 61, "Target vel:");

      /* The Joint State Box ... */
      draw_box (15, 59, 18, 79, (long) CYAN, BLACK);
      _setbkcolor ((long) CYAN);
      _settextcolor (BLACK);
```

```
	printsc (15, 64, "Joint State ");
	printsc (16, 61, "Position:");
	printsc (17, 61, "Velocity:");

	/* The Messages Box ... */
	draw_box (20, 2, 24, 79, (long) RED, BLACK);
	_setbkcolor ((long) RED);
	_settextcolor (BLACK);
	printsc (20, 35, " Messages ");

	/* The User Interface box */
	draw_box (11, 2, 18, 57, (long) LTGREEN, BLACK);
	_setbkcolor ((long) LTGREEN);
	_settextcolor (BLACK);
	printsc (11, 22, " Blah, blah! ");
	printsc (11, 22, " Terry's box ");

	show_state (0.0, 0.0);
	show_command (0.0, 0.0);

	_settextcolor (WHITE);
	_setbkcolor ((long) BLUE);
	printsc (2, 2, "F1-toggle A");
	printsc (3, 2, "F2-toggle B");
	printsc (4, 2, "F3-toggle C");
	printsc (5, 2, "F4-toggle D");
	printsc (6, 2, "F5-inc C_pw");
	printsc (7, 2, "F6-inc D_pw");
	printsc (8, 2, "F7-set pw's");
	printsc (2, 68, "Close Joint");
	printsc (3, 68, "  c / CURDN");
	printsc (4, 68, "Open Joint");
	printsc (5, 68, "  o / CURUP");

	cursor_off ();
}


/*---------------------------------------------------------------------------
  Draw a nice looking colored box. The coordinate origin is the upper left
  corner of the screen (1, 1). The bottom right corner is (25, 80).
---------------------------------------------------------------------------*/
void draw_box (int start_row, int start_col, int end_row, int end_col,
		long back_color, short border_color)
{
	char  top[81], middle[81], bottom[81];	/* Max box is 80 char wide */
	register int	i;
	long  old_back;
	short old_text;

	/* Save the current colors before messing with them */
	old_back = _getbkcolor ();
	old_text = _gettextcolor ();

	/* Compile the strings for the box */
	top [0] = '●';		/* char 201 */
	middle [0] = '●';	/* char 186 */
	bottom [0] = '●';	/* char 200 */
	for (i = 1; i < (end_col - start_col); i++) {
		top [i] = '●';		/* char 205 */
		middle [i] = ' '; /* char 32  */
		bottom [i] = '●';	/* char 205 */
	}
	top [i] = '●';		/* char 187 */
	middle [i] = '●';	/* char 186 */
	bottom [i] = '●';	/* char 188 */

	/* terminate all the strings with a null char. */
	i++;
	top [i] = '\0';
	middle [i] = '\0';
	bottom [i] = '\0';

	_setbkcolor (back_color);
	_settextcolor (border_color);

	/* Draw the top line */
	  _settextposition (start_row, start_col);
	_outtext (top);

	/* Draw the middle lines */
	for (i=1; i < (end_row - start_row); i++) {
		_settextposition (start_row + i, start_col);
		_outtext (middle);
	}

	/* Draw the bottom of the box */
	_settextposition (end_row, start_col);
	_outtext (bottom);

	/* Restore the original colors */
	_setbkcolor (old_back);
	_settextcolor (old_text);
}


/*---------------------------------------------------------------------------
  Print a string at a specified row and column in a specific.
---------------------------------------------------------------------------*/
void printsc (int row, int col, char *string)
{
	_settextposition (row, col);
	_outtext (string);
}


/*---------------------------------------------------------------------------
  Print an error message, beep, and wait for a key to be pressed.
---------------------------------------------------------------------------*/
void prompt (char *string)
{
	long  old_back;
	short old_text;

	/* Save the current colors before messing with them */
	old_back = _getbkcolor ();
	old_text = _gettextcolor ();

	_setbkcolor ((long) RED);
	_settextcolor (WHITE);
	_settextposition (21, 5);
	_outtext ("                                                              ");
```

```
      _settextposition (21,5);
      _outtext (string);
      _settextposition (23, 5);
      _settextcolor (WHITE+BLINK);
      _outtext ("Press any key to continue . . .");
      beep ();

      while (!kbhit()); /* Wait for a keypress, discard char */
      getch ();

      _settextposition (23, 5);
      _outtext ("                                        ");

      /* Restore the original colors */
      _setbkcolor (old_back);
      _settextcolor (old_text);
}


/*---------------------------------------------------------------------------
  Print a message.
---------------------------------------------------------------------------*/
void message (char *string)
{
      long  old_back;
      short old_text;

      /* Save the current colors before messing with them */
      old_back = _getbkcolor ();
      old_text = _gettextcolor ();

      _setbkcolor ((long) RED);
      _settextcolor (WHITE);
      _settextposition (21, 5);
      _outtext ("                                        ");
      _settextposition (21,5);
      _outtext (string);

      /* Restore the original colors */
      _setbkcolor (old_back);
      _settextcolor (old_text);
}


/*---------------------------------------------------------------------------
  Turn off the cursor by playing with INT 10H.
---------------------------------------------------------------------------*/
void cursor_off (void)
{
      union REGS regs;

      regs.h.ah = 1;
      regs.h.ch = 0x20; /* set bit 5 to disable cursor */

      int86 (0x10, &regs, &regs);
}


/*---------------------------------------------------------------------------
  Turn on the cursor with INT 10H
---------------------------------------------------------------------------*/
void cursor_on (void)
{
      union REGS regs;

      regs.h.ah = 1;
      regs.h.ch = 6;
      regs.h.cl = 7;

      int86 (0x10, &regs, &regs);
}


/*---------------------------------------------------------------------------
  Display the command in the "Command box" (incredible isn't it?)
---------------------------------------------------------------------------*/
void show_command (double position, double rate)
{
      char buffer [10];

      _setbkcolor ((long) CYAN);
      _settextcolor (BLACK);
      sprintf (buffer, "%6.01f", position);
      printsc (12, 72, buffer);
      sprintf (buffer, "%6.11f", rate);
      printsc (13, 72, buffer);
}


/*---------------------------------------------------------------------------
  Display the current state in the "State box"
---------------------------------------------------------------------------*/
void show_state (double angle, double rate)
{
      char buffer [10];

      _setbkcolor ((long) CYAN);
      _settextcolor (BLACK);
      sprintf (buffer, "%6.01f", angle);
      printsc (16, 72, buffer);
      if (rate > 100) return;
      sprintf (buffer, "%6.11f", rate);
      printsc (17, 72, buffer);
}


/*---------------------------------------------------------------------------
  Query the user for position and rate commands.
---------------------------------------------------------------------------*/
void get_command (double *position, double *rate)
{
    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (13, 5, "Desired position (ENC. COUNT): ");
    scanf ("%lf", position);
    printsc (14, 5, "Desired rate (COUNTS/SEC): ");
    scanf ("%lf", rate);
    printsc (13, 5, "                                        ");
    printsc (14, 5, "                                        ");
    cursor_off ();
}
```

```
/*---------------------------------------------------------------------------
  Query the user for pulse_widths
----------------------------------------------------------------------------*/
void get_pw (int *c_pw, int *d_pw)
{
    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (13, 5, "Solenoid C Pulse Width: ");
    scanf ("%d", c_pw);
    printsc (14, 5, "Solenoid D Pulse Width: ");
    scanf ("%d", d_pw);
    printsc (13, 5, "                                   ");
    printsc (14, 5, "                                   ");
    cursor_off ();
}


/*---------------------------------------------------------------------------
  Query the user for pulse_widths
----------------------------------------------------------------------------*/
void get_double (char *string, double *number)
{
    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (13, 5, string);
    scanf ("%lf", number);
    printsc (13, 5, "                                   ");
    cursor_off ();
}


/*----------------------------------------------------------------------------
  Just wait around for a few seconds
/*---------------------------------------------------------------------------*/
void delay (int seconds)
{
      long start, cur;

      time (&start);
      do {
            time (&cur);
      } while ((cur-start) < seconds);
}


/*----------------------------------------------------------------------------
  Open a file for data output. Returns a TRUE if file is opened successfully.
----------------------------------------------------------------------------*/
int start_save (FILE **fp)
{
    char filename [60];
    char buffer [80];

    cursor_on ();
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (16, 5, "                                   ");
    printsc (16, 5, "Enter filename [.PRN]: ");
    scanf ("%s", filename);
    strcat (filename, ".prn");
    cursor_off ();

    if ((*fp=fopen(filename, "w"))==NULL)
    {
      prompt ("ERROR: Cannot open file !!!!!");
      return (FALSE);
    }
    else
    {
      sprintf (buffer, "Position and rate data saved to: %s",
            strupr(filename));
      printsc (16, 5, buffer);
/*
      fprintf (*fp, "SPAM Data File\n");
      fprintf (*fp, "CurAngle CurRate cp dp\n\n");
*/
      return (TRUE);
    }
}


/*---------------------------------------------------------------------------
  Close the data file. Returns a TRUE if closed correctly.
----------------------------------------------------------------------------*/
int close_save (FILE *fp)
{
    message ("Data file closed.");
    _settextcolor (BLACK);
    _setbkcolor ((long) LTGREEN);
    printsc (16, 5, "                                                  ");
    if ((fclose (fp)) == EOF) {
          prompt ("ERROR: Unable to close file !!!!!");
          return (FALSE);
    }
    else
          return (TRUE);
}


/*---------------------------------------------------------------------------
  Exit gracefully (of course!)
----------------------------------------------------------------------------*/
void spam_exit (void)
{
      _settextposition (25, 1);
      printf ("\n SPAM CL Control Done.\n\a");
      cursor_on ();
}

void beep (void)
{
      printf ("\a");
}
/*------------------------ End of SUPPORT.C -------------------------------*/
```

```
#----------------------------------------------------------------------------
#
# SPLAT - Makefile for SPLAT.EXE
#
#   Author   :  Terry Fong
#   Created  :  08-17-89
#   Modified :  10-30-89
#
#   NOTES: use the /Oa (relax alias checking) with EXTREME CAUTION !!!
#          (remember:  /Ox = /Oailt /Gs )
#
#          $* = outfile basename, $@ = outfile filename, $** = all infiles
#
#          make sure to "set LINK=/NOI /NOD" in AUTOEXEC.BAT
#
#----------------------------------------------------------------------------

#
# Some useful macro definitions
#
model=/AS /FPi87          # small memory model, inline 8087 support
noalias=/c /Od $(model)         # no alias relaxing
stdcomp=/c /Od $(model)         # no alias relaxing, no default optimizations
stdlibs=slibc7+graphics+gfs+splat+terrylib
objects=control+fwdkin+invkin+menu+misc+pathplan+splat
progdep=control.obj fwdkin.obj menu.obj misc.obj pathplan.obj splat.obj
program=splat.exe

#
# Standard .C -> .OBJ inference rule: compile with full optimization
#
.C.OBJ :
      cl $(stdcomp) $*.c

#
# Dependency block definitions
#
control.obj : control.c solenoid.h

fwdkin.obj : fwdkin.c fwdkin.h

invkin.obj : invkin.c

menu.obj : menu.c menu.h

misc.obj : misc.c
      cl $(noalias) $*.c

pathplan.obj : pathplan.c
      cl $(noalias) $*.c

splat.obj : splat.c splat.h menu.h windows.h
      cl $(noalias) $*.c

#
# The following object modules are stored in SPLAT.LIB
#
display.obj : display.c
      cl $(stdcomp) $*.c
      lib splat -+$*;
```

```
makeplat.obj : makeplat.c
      cl $(stdcomp) $*.c
      lib splat -+$*;

sensor.obj   : sensor.c portdefs.h
      cl $(stdcomp) $*.c
      lib splat -+$*;

solenoid.obj : solenoid.c solenoid.h portdefs.h
      cl $(stdcomp) $*.c
      lib splat -+$*;

#
# The following are object modules are stored in TERRYLIB.LIB
#
newtraph.obj : newtraph.c newtraph.h
      cl $(noalias) $*.c
      lib terrylib -+$*;

vector.obj : vector.c
      cl $(stdcomp) $*.c
      lib terrylib -+$*;

video.obj : video.c video.h
      cl $(stdcomp) $*.c
      lib terrylib -+$*;

#
# Create SPLAT.EXE !!!
#
$(program) : $(progdep)
      link $(objects),$*,NUL,$(stdlibs)
      if exist *.bak del *.bak
```

```
/*---------------------------------------------------------------------------

CONTROL.C - The heart and soul of PWM CL control for SPLAT

   Author   :  Terry Fong
   Created  :  08-15-89
   Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <stdio.h>
#include "solenoid.h"

/*---------------------------------------------------------------------------
  Perform CL control movement from point to point.

  * as of 8/15/89, NO RESOLVED RATE !!!!!
---------------------------------------------------------------------------*/
void cl_control (float lengths[], byte steps)
{

      /* preliminary stab at control logic (pseudo-code), assuming that
         LENGTHS[] is a 1x6 array of desired acutuator lengths */

/*

      read initial lengths;
      compare initial lengths with desired lengths;
      if (error > deadband) {
            adjust actuator lengths;
            read current lengths;
            loop to compare;
      }

*/
}

/*
===========================================================================
Question: do we adjust one at a time or do we do all 6 actuators concurrently?
===========================================================================
*/

/*---------------------------------------------------------------------------
  Open Loop control (ick!)
---------------------------------------------------------------------------*/
void ol_control (window_ptr TheWindow, int selection)
{
      byte i, pw;
      string buffer;

      clear_window (TheWindow);

      switch (selection) {

            case 1 : /* toggle koganei */
                  i = (int) input (TheWindow, " Koganei to toggle: ");
                  toggle_kog (i);
                  sprintf (buffer, " KOGANEI #%d toggled.\n", i);
                  message (buffer);
                  break;

            case 2 : /* pulse kip */
                  i = (int) input (TheWindow, " Kip to pulse: ");
                  pw = (int) input (TheWindow, " PW (1-15=ON): ");
                  pulse_kip (i, pw);
                  sprintf (buffer, " KIP #%d pulsing at %d.\n", i, pw);
                  message (buffer);
                  break;

            case 3 : /* pulse all kips */
                  pw = (int) input (TheWindow, " PW (1-15=ON): ");
                  for (i=0; i<12; i++) pulse_kip (i, pw);
                  sprintf (buffer, " All KIPs pulsing at %d.\n", pw);
                  message (buffer);
                  break;

            case 4 : /* close all kips */
                  close_all_kips ();
                  message (" All KIPs closed.\n");
                  break;

            case 5 : /* close everything */
                  init_sols ();
                  message (" All KIPs and KOGANEIs closed.\n");
                  break;
      }
}

/*--------------------------- End of CONTROL.C -----------------------------*/
```

```
/*---------------------------------------------------------------------------

DISPLAY.C - Routines to set up the display screen and to display various
        things in various windows.

    Author   :  Terry Fong
    Created  :  08-11-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <graph.h>
#include <stdio.h>
#include <gnleaf\music.h>
#include <gnleaf\timedate.h>
#include "splat.h"

/* local global for controlling sound */
static char play = FALSE;

/*---------------------------------------------------------------------------
  Layout the user interface in vivid color !!!
---------------------------------------------------------------------------*/
void draw_screen (int steps, float accuracy, frame current, frame command)
{
      extern window_ptr title_win, error_win, actuator_win, platform_win,
                    menu_win, param_win;

      float lengths[6], tilt[6], roll[6];
      register i;
      char buffer[30];

      show_time (TRUE);
      create_window (title_win, "");
      create_window (error_win, " MESSAGES ");
      create_window (actuator_win, " ACTUATOR STATE ");
      create_window (platform_win, " PLATFORM STATE ");
      create_window (menu_win, "");
      create_window (param_win, "");

      wcprintf (title_win, 1, "STEWART PLATFORM CONTROLLER v1.0");
      wcprintf (title_win, 2, "MIT Space Systems / LOOP Group");
      wcprintf (actuator_win, 1, " # length   tilt   roll");
      for (i=0; i<6; i++) {
            sprintf (buffer, " %d ", i);
            wprintrc (actuator_win, i+2, 1, buffer);
      }

      calc_cur_state (current, lengths, tilt, roll);
      for (i=0; i<6; i++)
            show_actuator (actuator_win, i, lengths[i], tilt[i], roll[i]);
      wcprintf (platform_win, 1, "   current target");
      wprintrc (platform_win, 2, 2, "X ");
      wprintrc (platform_win, 3, 2, "Y ");
      wprintrc (platform_win, 4, 2, "Z ");
      wprintrc (platform_win, 5, 2, "¤ ");
      wprintrc (platform_win, 6, 2, "º ");
      wprintrc (platform_win, 7, 2, "¬ ");
      show_current (platform_win, current);
      show_target (platform_win, command);
      wprintrc (param_win, 1, 2, "# steps =");
      wprintrc (param_win, 2, 2, "NR acc. =");
      show_steps (param_win, steps);
      show_accuracy (param_win, accuracy);
      play = TRUE;
      sound (NA2, 5);
      sound (560, 5);    /* Middle C# */
      sound (NE2, 5);
      sound (NA3, 10);
}

/*---------------------------------------------------------------------------
  Display target platform frame
---------------------------------------------------------------------------*/
void show_target (window_ptr TheWindow, frame platform)
{
      char buffer[25];

      TheWindow->color.text = BLUE;
      sprintf (buffer, "%5.2f\"", platform.position.x);
      wprintrc (TheWindow, 2, 13, buffer);
      sprintf (buffer, "%5.2f\"", platform.position.y);
      wprintrc (TheWindow, 3, 13, buffer);
      sprintf (buffer, "%5.2f\"", platform.position.z);
      wprintrc (TheWindow, 4, 13, buffer);
      sprintf (buffer, "%5.1f°", platform.angle.roll);
      wprintrc (TheWindow, 5, 13, buffer);
      sprintf (buffer, "%5.1f°", platform.angle.pitch);
      wprintrc (TheWindow, 6, 13, buffer);
      sprintf (buffer, "%5.1f°", platform.angle.yaw);
      wprintrc (TheWindow, 7, 13, buffer);
      if (play) sound (NE2, 5);
}

/*---------------------------------------------------------------------------
  Display # of steps for moving the platform
---------------------------------------------------------------------------*/
void show_steps (window_ptr TheWindow, int steps)
{
      char buffer[10];

      sprintf (buffer, "%5d", steps);
      wprintrc (TheWindow, 1, 14, buffer);
      if (play) sound (NE2, 5);
}

/*---------------------------------------------------------------------------
  Display convergence accuracy of Newton-Raphson
---------------------------------------------------------------------------*/
void show_accuracy (window_ptr TheWindow, float accuracy)
{
      char buffer[10];

      sprintf (buffer, "%5.3f", accuracy);
      wprintrc (TheWindow, 2, 14, buffer);
      if (play) sound (NE2, 5);
}

/*---------------------------------------------------------------------------
  Display current platform frame
---------------------------------------------------------------------------*/
```

```
void show_current (window_ptr TheWindow, frame platform)
{
      char buffer[25];

      TheWindow->color.text = BLUE;
      sprintf (buffer, "%5.2f\"", platform.position.x);
      wprintrc (TheWindow, 2, 6, buffer);
      sprintf (buffer, "%5.2f\"", platform.position.y);
      wprintrc (TheWindow, 3, 6, buffer);
      sprintf (buffer, "%5.2f\"", platform.position.z);
      wprintrc (TheWindow, 4, 6, buffer);
      sprintf (buffer, "%5.1f°", platform.angle.roll);
      wprintrc (TheWindow, 5, 6, buffer);
      sprintf (buffer, "%5.1f°", platform.angle.pitch);
      wprintrc (TheWindow, 6, 6, buffer);
      sprintf (buffer, "%5.1f°", platform.angle.yaw);
      wprintrc (TheWindow, 7, 6, buffer);
      if (play) sound (NE2, 5);
}

/*---------------------------------------------------------------------------
  Display state of actuator n = {0..5}
---------------------------------------------------------------------------*/
void show_actuator (window_ptr TheWindow, int n, float length, float tilt,
              float roll)
{
      char buffer[25];

      TheWindow->color.text = BLUE;
      sprintf (buffer, "%5.2f\" %5.1f°%5.1f°", length, tilt, roll);
      wprintrc (TheWindow, n+2, 6, buffer);
}

/*---------------------------------------------------------------------------
  Time stamp the display. If start is TRUE, then print "started on", else
  print "exited at".
---------------------------------------------------------------------------*/
void show_time (int start)
{
      struct TIMEDATE *ptd, *sgettime();
      string buffer;

      ptd = sgettime (8);
      if (start) sprintf (buffer, "SPLAT started on: %s", ptd->dateline);
      else sprintf (buffer, "SPLAT exited at: %s", ptd->dateline);

      _settextwindow (1, 1, 25, 79);
      _setbkcolor ((long) BLUE);
      _settextcolor (LTGREY);
      _settextposition (25, 2);
      _outtext (buffer);
}

void show_filename (char *filename, int display)
{
      string buffer;

      _settextwindow (1, 1, 25, 79);
      _setbkcolor ((long) BLUE);
      _settextcolor (LTGREY);
      _settextposition (25, 50);
      if (display) {
            sprintf (buffer, "Data saved to: %s", strupr(filename));
            _outtext (buffer);
      }
      else
            _outtext("                              ");
}

/*---------------------------------------------------------------------------
  Print a message in the error window and beep. Wait for a user keypress.
---------------------------------------------------------------------------*/
void prompt (char *string)
{
      extern window_ptr error_win;

      clear_window (error_win);
      wprintf (error_win, string);
      error_win->color.text += 16; /* set blinking */
      wprintf (error_win, " Press any key to continue ...\n");

      sound (NA2, 10);
      while (!kbhit ());
      getch ();

      error_win->color.text -= 16;
      error_win->curpos.row--;
      wprintf (error_win, "                                        \n");
}

/*---------------------------------------------------------------------------
  Display a message in the error window
---------------------------------------------------------------------------*/
void message (char *string)
{
      extern window_ptr error_win;

      wprintf (error_win, string);
}

/*------------------------ End of DISPLAY.C ------------------------------*/
```

```
/*---------------------------------------------------------------------------

FWDKIN.H - Function declarations for FWDKIN.C

    Author   :  Terry Fong
    Created  :  08-22-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/
#define X          0
#define Y          1
#define Z          2
#define ROLL             3
#define PITCH            4
#define YAW              5
#define STATES     6
#define NR_MAX_TRIALS    10
#define XTOL             0.001

#define sq(x)      x*x

void plat_func (double guess[], double H[NMAX][NMAX], double h[NMAX]);
void calc_euler (double x[]);
void calc_vfunc (double guess[], double h[NMAX]);
void calc_vderiv (double guess[], double H[NMAX][NMAX]);
void calc_help (double guess[]);
```

```
/*---------------------------------------------------------------------------

FWDKIN.C - Perform the Forward Kinematic analysis of the Stewart Platform
         (6-DOF platform position --> 6 actuator lengths) by using iter-
         ative Newton-Raphson to determine the roots of a system of six
         simultaneous non-linear, second-order equations.

    Author   :  Terry Fong
    Created  :  08-22-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include "splat.h"
#include "newtraph.h"
#include "fwdkin.h"

/* #define DEBUG */      /* if defined, debug output sent to file "DEBUG" */

extern vector base_pts[6];    /* Refer to MAKEPLAT.C */
extern vector plat_pts[6];    /*    "    "    "      */

/*-------------------------------------------------------------------------*/
/* Module Globals */
static double t[4][4];        /* Euler transformation matrix */
static vector *a = plat_pts;  /* Top Platform mounting points */
static vector *b = base_pts;  /* Base Platform mounting points */
static double unknown[6];
static unsigned char iters;
static double *lengths;

/* The following are storage to speed up calculations */
static double cphi, ctheta, cpsi, sphi, stheta, spsi;
static double temp[6][4], xblah[6], yblah[6], zblah[6];
/*-------------------------------------------------------------------------*/

#ifdef DEBUG
FILE *fp;
#endif

/*---------------------------------------------------------------------------
  Perform forward kinematic analysis using input LENGTHS. If USE_GUESS is TRUE
  use GUESS as first guess, otherwise, use STD_GUESS. Function returns the
  number of NR iterations carried out. The calculated result is returned in
  GUESS.
---------------------------------------------------------------------------*/
unsigned char fwdkin (double act_lengths[], int use_guess, frame *guess,
                  double accuracy)
{
      register i;

      lengths = &act_lengths[0];
      iters = 0;

      if (use_guess) {
            unknown[0] = (double) guess->position.x;
            unknown[1] = (double) guess->position.y;
```

```
          unknown[2] = (double) guess->position.z;
          unknown[3] = (double) guess->angle.roll * DEG2RAD;
          unknown[4] = (double) guess->angle.pitch * DEG2RAD;
          unknown[5] = (double) guess->angle.yaw * DEG2RAD;
     }
     else {
          unknown[0] = 0.0; /* x */
          unknown[1] = 0.0; /* y */
          unknown[2] = 28.25;     /* z */
          unknown[3] = 0.0; /* roll */
          unknown[4] = 0.0; /* pitch */
          unknown[5] = 0.0; /* yaw */
     }

#ifdef DEBUG
     fp = fopen ("debug", "a");
     fprintf (fp, "NR ACC = %lf\n", accuracy);
     fprintf (fp, "Lengths are:\n");
     for (i=0; i<6; i++) fprintf (fp, "%6.21f ", lengths[i]);
     fprintf (fp, "\n");
     fprintf (fp, "\n*******************************************\n");
     fprintf (fp, "Starting guess frame is:\n");
     fprintf (fp, "  X      Y      Z     Phi    Theta   Psi\n");
     for (i=0; i<3; i++) fprintf (fp, "%6.2f ", unknown[i]);
     for (i=3; i<6; i++) fprintf (fp, "%6.2f ", unknown[i] * RAD2DEG);
     fprintf (fp, "\n*******************************************\n");
#endif

     NRIteration ((unsigned char) NR_MAX_TRIALS, unknown, STATES,
                XTOL, accuracy, plat_func);

#ifdef DEBUG
     fprintf (fp, "\n*******************************************\n");
     fprintf (fp, "Ending guess frame is:\n");
     fprintf (fp, "  X      Y      Z     Phi    Theta   Psi\n");
     for (i=0; i<3; i++) fprintf (fp, "%6.2f ", unknown[i]);
     for (i=3; i<6; i++) fprintf (fp, "%6.2f ", unknown[i] * RAD2DEG);
     fprintf (fp, "\n*******************************************\n");
     fclose (fp);
#endif

     guess->position.x = (float) unknown[0];
     guess->position.y = (float) unknown[1];
     guess->position.z = (float) unknown[2];
     guess->angle.roll = (float) unknown[3] * RAD2DEG;
     guess->angle.pitch = (float) unknown[4] * RAD2DEG;
     guess->angle.yaw = (float) unknown[5] * RAD2DEG;

     return (iters);
}

/*---------------------------------------------------------------------------
  Calculate vector function values and a 6x6 matrix of partial derivatives for
  the matrix function defined in NASA TN D-7067, pg. 6, eq. [6] :

        fi(alpha) = li'*li - |li|^2

  The function calculates values and derivatives (returned in h and H respect-
  ively) given an 6x1 input vector. The input is expected to be in the follow-
  ing order:
                      X, Y, Z, ROLL, PITCH, YAW
                                           (rad) (rad) (rad)
---------------------------------------------------------------------------*/
void plat_func (double guess[], double H[NMAX][NMAX], double h[NMAX])
{
     register i;

     iters++;

#ifdef DEBUG
     fprintf (fp, "\n------------------------------------------\n");
     fprintf (fp, "ITERATION #%d:  Guess frame is:\n", iters);
     fprintf (fp, "  X      Y      Z     Phi    Theta   Psi\n");
     for (i=0; i<3; i++) fprintf (fp, "%6.2f ", guess[i]);
     for (i=3; i<6; i++) fprintf (fp, "%6.2f ", guess[i] * RAD2DEG);
     fprintf (fp, "\n------------------------------------------\n");
#endif

     calc_euler (guess);
     calc_help (guess);
     calc_vfunc (guess, h);
     calc_vderiv (guess, H);
}

/*---------------------------------------------------------------------------
  Calculate the 3x3 Euler Roll-Pitch-Yaw transformation matrix
---------------------------------------------------------------------------*/
void calc_euler (double x[])
{

#ifdef DEBUG
     register i;
     fprintf (fp, "Iteration #%d\n", iters);
#endif

     /* compute sine and cosine of angles */
     cphi = cos ( x[ROLL] );
     sphi = sin ( x[ROLL] );
     ctheta = cos ( x[PITCH] );
     stheta = sin ( x[PITCH] );
     cpsi = cos ( x[YAW] );
     spsi = sin ( x[YAW] );

     /* Set up the coordinate transformation matrix */
     t[1][1] = cpsi*ctheta;
     t[1][2] = spsi*ctheta;
     t[1][3] = -stheta;
     t[2][1] = cpsi*stheta*sphi-spsi*cphi;
     t[2][2] = spsi*stheta*sphi+cpsi*cphi;
     t[2][3] = ctheta*sphi;
     t[3][1] = cpsi*stheta*cphi+spsi*sphi;
     t[3][2] = spsi*stheta*cphi-cpsi*sphi;
     t[3][3] = ctheta*cphi;

#ifdef DEBUG
     fprintf (fp, "Euler transform is:\n");
     for (i=1; i<4; i++)
          fprintf(fp, "%6.21f %6.21f %6.21f\n", t[i][1], t[i][2], t[i][3]);
#endif
}

/*---------------------------------------------------------------------------
```

```
  Calculate the vector function values for all six degrees of freedom
-------------------------------------------------------------------------------*/
void calc_vfunc (double guess[], double h[NMAX])
{
      register i;

#ifdef DEBUG
      fprintf (fp, "\nVector function values are:\n");
#endif

      for (i=0; i<6; i++) {
            h[i] = -(sq(a[i].x) + sq(a[i].y) + sq(a[i].z) +
                   sq(b[i].x) + sq(b[i].y) + sq(b[i].z) +
                   sq(guess[X]) + sq(guess[Y]) + sq(guess[Z]) -
                   sq(lengths[i]) + xblah[i]*temp[i][1] +
                   yblah[i]*temp[i][2] + zblah[i]*temp[i][3] -
                   2*(guess[X]*b[i].x+guess[Y]*b[i].y+guess[Z]*b[i].z));

#ifdef DEBUG
             fprintf (fp, "  h[%d] = %8.2lf\n", i, h[i]);
#endif

      }
}

/*-------------------------------------------------------------------------------
  Calculate the 6x6 vector derivative matrix
-------------------------------------------------------------------------------*/
void calc_vderiv (double guess[], double H[NMAX][NMAX])
{
      register i, j;

      for (i=0; i<6; i++) {
            H[i][X] = 2*(guess[X] + temp[i][1] - b[i].x);

            H[i][Y] = 2*(guess[Y] + temp[i][2] - b[i].y);

            H[i][Z] = 2*(guess[Z] + temp[i][3] - b[i].z);

            H[i][ROLL] = xblah[i]*(a[i].y*t[3][1] - a[i].z*t[2][1]) +
                        yblah[i]*(a[i].y*t[3][2] - a[i].z*t[2][2]) +
                        zblah[i]*(a[i].y*t[3][3] - a[i].z*t[2][3]);

            H[i][PITCH] = xblah[i] * (-a[i].x*stheta*cpsi +
                       a[i].y*sphi*ctheta*cpsi + a[i].z*cphi*ctheta*cpsi) +
                             yblah[i] * (-a[i].x*stheta*spsi +
                      a[i].y*sphi*ctheta*spsi + a[i].z*cphi*ctheta*spsi) -
                         zblah[i] * (a[i].x*ctheta +
                      a[i].y*sphi*stheta + a[i].z*cphi*stheta);

            H[i][YAW] = -xblah[i]*temp[i][2] +
                        yblah[i]*temp[i][1];
      }

#ifdef DEBUG
      fprintf (fp,"\nThe vector derivative matrix !!!\n");
      fprintf (fp,"     dx       dy       dz     dphi    dtheta    dpsi\n");
      for (i=0; i<6; i++) {
            for (j=0; j<6; j++)
                  fprintf (fp, "%8.2f ", H[i][j]);
            fprintf (fp, "\n");
```

```
      }
#endif
}

/*-------------------------------------------------------------------------------
  Compute partial sums and multiplies to speed up calculation of vector func-
  tion values and the derivative matrix. Results are stored in XBLAH, YBLAH
  ZBLAH, and TEMP.
-------------------------------------------------------------------------------*/
void calc_help (double guess[])
{
      register i, j;

      for (i=0; i<6; i++) {
            xblah[i] = 2 * (guess[X] - b[i].x);
            yblah[i] = 2 * (guess[Y] - b[i].y);
            zblah[i] = 2 * (guess[Z] - b[i].z);

            for (j=1; j<4; j++)
            temp[i][j] = a[i].x*t[1][j] + a[i].y*t[2][j] + a[i].z*t[3][j];

#ifdef DEBUG
      fprintf (fp, "xblah[%d]=%6.2lf yblah[%d]=%6.2lf zblah[%d]=%6.2lf\n",
            i, xblah[i], i, yblah[i], i, zblah[i]);
#endif

      }
}

/*--------------------------- End of FWDKIN.C -------------------------------*/
```

```
/*-----------------------------------------------------------------------------
INVKIN.C - Stewart Platform Inverse Kinematics

    Author   :  Terry Fong
    Created  :  07-20-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-----------------------------------------------------------------------------*/

#include <math.h>
#include "splat.h"

/*-----------------------------------------------------------------------------
  Calcuate Stewart Platform actuator vectors using the "Actuator Extension
  Transformation" from NASA TN D-7067 (essentially, inverse kinematics).
  Routine returns the calculated actuator vector for input platform position
  and orientation in "actuator[]" (which should be of size 6).

  Note: commanded angles must be in degrees, commanded positions in inches !!!
-----------------------------------------------------------------------------*/
void invkin (frame platform, vector actuator[])
{
      extern vector base_pts [6];   /* vector of base pts in (base) */
      extern vector plat_pts [6];   /* vector of plat pts in (plat) */

      float x, y, z;
      float cphi, ctheta, cpsi, sphi, stheta, spsi;
      float t11, t12, t13, t21, t22, t23, t31, t32, t33;
      register i;

      x = platform.position.x;
      y = platform.position.y;
      z = platform.position.z;

      /* compute sine and cosine of angles */
      cphi = cos (platform.angle.roll * DEG2RAD);
      sphi = sin (platform.angle.roll * DEG2RAD);
      ctheta = cos (platform.angle.pitch * DEG2RAD);
      stheta = sin (platform.angle.pitch * DEG2RAD);
      cpsi = cos (platform.angle.yaw * DEG2RAD);
      spsi = sin (platform.angle.yaw * DEG2RAD);

      /* Set up the coordinate transformation matrix */
      t11 = cpsi*ctheta;
      t12 = spsi*ctheta;
      t13 = -stheta;
      t21 = cpsi*stheta*sphi-spsi*ctheta;
      t22 = spsi*stheta*sphi+cpsi*cphi;
      t23 = ctheta*sphi;
      t31 = cpsi*stheta*cphi+spsi*sphi;
      t32 = spsi*stheta*cphi-cpsi*sphi;
      t33 = ctheta*cphi;

      /* Calculate actuator vector */
      for (i=0; i<6; i++){
            actuator[i].x = plat_pts[i].x*t11 + plat_pts[i].y*t21 +
                        plat_pts[i].z*t31 + x - base_pts[i].x;
            actuator[i].y = plat_pts[i].x*t12 + plat_pts[i].y*t22 +
                        plat_pts[i].z*t32 + y - base_pts[i].y;
            actuator[i].z = plat_pts[i].x*t13 + plat_pts[i].y*t23 +
                        plat_pts[i].z*t33 + z - base_pts[i].z;
      }
}

/*-----------------------------------------------------------------------------
  Calculate actuator lengths from current frame position
-----------------------------------------------------------------------------*/
void calc_cur_state (frame current, float length[], float tilt[],
                  float roll[])
{
      vector actuator[6];

      invkin (current, actuator);
      calc_len (length, actuator);
      calc_angles (tilt, roll, actuator);
}

/*-----------------------------------------------------------------------------
  Calculate actuator lengths from the actuator vector.
-----------------------------------------------------------------------------*/
void calc_len (float length[], vector actuator[])
{
      register i;

      for (i=0; i<6; i++)
            length[i] = magnitude (actuator[i]);
}

/*-----------------------------------------------------------------------------
  Calculate base joint tilt and roll angles (in degrees).
-----------------------------------------------------------------------------*/
void calc_angles (float tilt[], float roll[], vector actuator[])
{
      extern vector base_pts [6];   /* vector of base pts in (base) */
      extern vector plat_pts [6];   /* vector of plat pts in (plat) */

      register i;
      vector b, t;
      float mag_b;

      for (i = 0; i < 6; i++) {
            b = normalize (base_pts[i]);
            b.x = -b.x;
            b.y = -b.y;
            b.z = -b.z;
            t.x = b.y;
            t.y = -b.x;
            t.z = 0;

            tilt[i] = atan(dot_prod(actuator[i], b) / actuator[i].z) *
                  RAD2DEG;
            roll[i] = atan(dot_prod(actuator[i], t) / actuator[i].z) *
                  RAD2DEG;
      }
}

/*-------------------------- End of INVKIN.C ------------------------------*/
```

```
/*---------------------------------------------------------------------------

MAKEPLAT.C - Generate a Stewart Platform by reading parameter file "SPLAT.DAT"

    Author   :  Terry Fong
    Created  :  07-28-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-------------------------------------------------------------------------*/

#include <math.h>
#include <stdio.h>
#include <string.h>
#include "splat.h"

#define SETUP "setup.dat"

/* Global variables accessible by all modules */
vector base_pts[6];     /* vector of base pts in {base} */
vector plat_pts[6];     /* vector of plat pts in {plat} */

/*---------------------------------------------------------------------------
  Read Stewart Platform actuator mount point data from a file
-------------------------------------------------------------------------*/
void create_sp ()
{
      extern vector base_pts[6];    /* vector of base pts in {base} */
      extern vector plat_pts[6];    /* vector of plat pts in {plat} */

      FILE *fp;
      byte i;

      fp = fopen (SETUP, "r");
      fscanf (fp, "     Stewart Platform / Actuator Mount Points\n");
      fscanf (fp, "------------------------------------------------\n");
      fscanf (fp, "       {Base Frame}          {Platform Frame}\n");
      fscanf (fp, " #  base x  base y  base z  plat x  plat y  plat z\n");
      fscanf (fp, "------------------------------------------------\n");
      for (i=0; i<6; i++)
            fscanf (fp, "%*s%f%f%f%f%f%f\n",
                  &base_pts[i].x, &base_pts[i].y, &base_pts[i].z,
                  &plat_pts[i].x, &plat_pts[i].y,     &plat_pts[i].z);
      fclose (fp);
}

/*---------------------------------------------------------------------------
  Display current Stewart Platform layout
-------------------------------------------------------------------------*/
void display_sp ()
{
      byte i;

      printf ("\nStewart Platform / Actuator Mount Points\n");
      printf ("<Data read from %s>\n", strupr(SETUP));
      printf ("------------------------------------------------\n");
      printf ("       {Base Frame}          {Platform Frame}\n");
      printf (" # base x  base y  base z  plat x  plat y  plat z\n");
      printf ("------------------------------------------------\n");
      for (i=0; i<6; i++)
            printf("%2d%8.2f%8.2f%8.2f%8.2f%8.2f%8.2f\n",i,base_pts[i].x,
                  base_pts[i].y, base_pts[i].z, plat_pts[i].x, plat_pts[i].y,
                  plat_pts[i].z);
      printf ("\n--------------------------------------------------\n\n");
      printf ("Press a key to continue ...\n");
      while (!kbhit ());
      getch ();
}

/*------------------------- End of MAKEPLAT.C ---------------------------*/
```

```
/*---------------------------------------------------------------------------

MENU.H - Menu level definitions.

    Author   :  Terry Fong
    Created  :  08-11-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#define TOPLEVEL  0
#define PLATCOMMAND      1
#define OPENLOOP  2
```

```
/*---------------------------------------------------------------------------

MENU.C - Display menu and return user selection

    Author   :  Terry Fong
    Created  :  07-20-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <conio.h>

#include "splat.h"
#include "menu.h"

/*---------------------------------------------------------------------------
  Display menus to the user and scan for keypress. Returns menu selection
  number. ESC returns a zero.
---------------------------------------------------------------------------*/
int show_menu (window_ptr TheWindow, int menu)
{
      char ch;

      clear_window (TheWindow);

      switch (menu) {

            case TOPLEVEL :
                  wprintf (TheWindow, "   1 - Give Command\n");
                  wprintf (TheWindow, "   2 - Run Command\n");
                  wprintf (TheWindow, "   3 - Toggle Save\n");
                  wprintf (TheWindow, "   4 - Set NR accuracy\n");
                  wprintf (TheWindow, "   5 - Set # of steps\n");
                  wprintf (TheWindow, "   6 - Open Loop control\n\n");
                  wprintf (TheWindow, " ESC - Exit SPLAT\n");
                  break;

            case PLATCOMMAND :
                  wprintf (TheWindow, "   1 - Enter X\n");
                  wprintf (TheWindow, "   2 - Enter Y\n");
                  wprintf (TheWindow, "   3 - Enter Z\n");
                  wprintf (TheWindow, "   4 - Enter ¤\n");
                  wprintf (TheWindow, "   5 - Enter º\n");
                  wprintf (TheWindow, "   6 - Enter ¬\n");
                  wprintf (TheWindow, "   7 - Enter all\n\n");
                  wprintf (TheWindow, " ESC - Quit\n");
                  break;

            case OPENLOOP :
                  wprintf (TheWindow, "   1 - Toggle KOGANEI\n");
                  wprintf (TheWindow, "   2 - Pulse one KIP\n");
                  wprintf (TheWindow, "   3 - Pulse all KIPs\n");
                  wprintf (TheWindow, "   4 - Close all KIPs\n");
                  wprintf (TheWindow, "   5 - Stop everything!\n\n");
                  wprintf (TheWindow, " ESC - Quit\n");
                  break;
      }

      while (!kbhit ());
      ch = getch ();
```

```
     if (ch == ESC) return (0);
     else return ((int) ch - 48);
}

/*--------------------------- End of MENU.C ---------------------------------*/
```

```
/*-----------------------------------------------------------------------------

MISC.C - A bunch of misc. routines

   Author   :  Terry Fong
   Created  :  07-20-89
   Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-----------------------------------------------------------------------------*/

#include <time.h>
#include <graph.h>
#include <gnleaf\music.h>
#include <gnleaf\timedate.h>
#include "splat.h"

/*-----------------------------------------------------------------------------
  Just wait around for a few seconds
/*---------------------------------------------------------------------------*/
void delay (int seconds)
{
     long start, cur;

     time (&start);
     do {
          time (&cur);
     } while ((cur-start) < seconds);
}

/*-----------------------------------------------------------------------------
  Terminate SPLAT.EXE
-----------------------------------------------------------------------------*/
void exit_splat ()
{

     _setvideomode (_TEXTC80);
     _setbkcolor ((long) BLUE);
     _clearscreen (_GCLEARSCREEN);
     show_time (FALSE);
     cursor_on ();
}

/*-----------------------------------------------------------------------------
  Close a data save file
-----------------------------------------------------------------------------*/
int close_save (FILE *fp, char *filename)
{
     string buffer;

     if (fclose (fp)==NULL) {
          sprintf(buffer, " %s closed.\n", strupr(filename));
          prompt (buffer);
          show_filename ("", FALSE);
          return (TRUE);
     }
     else {
          sprintf (buffer, " ERROR: Cannot close %s !!!!!\n",
                 strupr(filename));
          prompt (buffer);
          return (FALSE);
```

```
        }
}

/*---------------------------------------------------------------------------
  Open a file for data output. Returns a TRUE if file is opened successfully.
---------------------------------------------------------------------------*/
int start_save (window_ptr TheWindow, FILE **fp, char *filename)
{
        string buffer;
        struct TIMEDATE *ptd, *sgettime();

        clear_window (TheWindow);
        wprintf (TheWindow, " Filename [.DAT]: ");
        TheWindow->curpos.row++;
        TheWindow->curpos.col = 1;
        cursor_on ();
        scanf ("%s", filename);
        cursor_off ();

        strcat (filename, ".dat");

        if ((*fp=fopen(filename, "w"))==NULL) {
                prompt (" ERROR: Cannot open file !!!!!\n");
                return (FALSE);
        }
        else {
                sprintf (buffer, " %s opened for data save.\n",
                        strupr(filename));
                message (buffer);
                show_filename (filename, TRUE);

                ptd = sgettime (8);
                fprintf (*fp, "%s\n\n", ptd->dateline);
                fprintf (*fp, "SPLAT v1.0 Data Save File\n");
                fprintf (*fp, "----------------------------\n");

                return (TRUE);
        }
}

/*---------------------------------------------------------------------------
  Print a message in the error window and beep. Wait for a user keypress.
---------------------------------------------------------------------------*/
void prompt (char *string)
{
        extern window_ptr error_win;

        clear_window (error_win);
        wprintf (error_win, string);
        error_win->color.text += 16; /* set blinking */
        wprintf (error_win, " Press any key to continue ...\n");

        sound (NA2, 10);
        while (!kbhit ());
        getch ();

        error_win->color.text -= 16;
        error_win->curpos.row--;
        wprintf (error_win, "                                    \n");
}

/*---------------------------------------------------------------------------
  Display a message in the error window
---------------------------------------------------------------------------*/
void message (char *string)
{
        extern window_ptr error_win;

        wprintf (error_win, string);
}

/*---------------------------------------------------------------------------
  Save the path of actuator lengths
---------------------------------------------------------------------------*/
void save_path (FILE *fp, byte steps, float lengths[][6], frame current,
                frame command)
{
        byte i, j;
        vector actuator[6];
        float blah[6];

        invkin (current, actuator);
        calc_len (blah, actuator);
        fprintf (fp, "\nStarting frame is:\n");
        dump_frame (fp, current);
        fprintf (fp, "Starting lengths are:\n");
        dump_lengths (fp, blah);

        invkin (command, actuator);
        calc_len (blah, actuator);
        fprintf (fp, "\nEnding frame is:\n");
        dump_frame (fp, command);
        fprintf (fp, "Ending lengths are:\n");
        dump_lengths (fp, blah);
        fprintf (fp, "\n");

        for (i=0; i<steps; i++) {
                fprintf (fp, "%3d: ", i);
                for (j=0; j<6; j++)
                        fprintf (fp, "<L%d=%6.2f> ", j, lengths[i][j]);
                fprintf (fp, "\n");
        }
}

/*---------------------------------------------------------------------------
  Dump 6 DOF frame to a file
---------------------------------------------------------------------------*/
void dump_frame (FILE *fp, frame x)
{
        fprintf (fp, "x=%8.2f,y=%8.2f,z=%8.2f,r=%8.2f,p=%8.2f,y=%8.2f\n",
                x.position.x, x.position.y, x.position.z, x.angle.roll,
                x.angle.pitch, x.angle.yaw);
}

/*---------------------------------------------------------------------------
  Dump 6 lengths to a file
---------------------------------------------------------------------------*/
void dump_lengths (FILE *fp, float lengths[])
{
        int i;

        for (i=0; i<6; i++)
```

```
          fprintf (fp, "<L%d=%6.2f> ", i, lengths[i]);
     fprintf (fp, "\n");
}

/*------------------------------------------------------------------------------
  Dump an array of six vectors to a file.
------------------------------------------------------------------------------*/
dump_vectors (FILE *fp, vector x[])
{
     int i;

     for (i=0; i<6; i++)
          fprintf (fp, "v.x=%8.2f,v.y=%8.2f,v.z=%8.2f\n",
               x[i].x, x[i].y, x[i].z);
}

/*------------------------- End of MISC.C ---------------------------------*/
```

```
/*------------------------------------------------------------------------------

NEWTRAPH.H - Function declarations for NEWTRAPH.C

          Thanks to Robert "Neural God" Sanner for the routines.

------------------------------------------------------------------------------*/
#define NMAX      6
#define TINY      1.0e-20
#define ODD 1

void LUFactor   ( double Matrix[NMAX][NMAX], unsigned char Dim,
                         unsigned char *Permutes, char *NPermutes );
void LUBackSub  ( double Matrix[NMAX][NMAX],unsigned char Dim,
                         unsigned char *Permutes, double RHS[NMAX] );
void NRIteration(unsigned char NTrials, double *X, unsigned char Dim,
          double XToler, double FToler, void (*UserFun)() );
```

```
/*---------------------------------------------------------------------------

NEWTRAPH.C - Iterative Newton-Raphson method for finding roots of a non-linear
           system of equations.

           Thanks to Robert "Neural God" Sanner for the routines.

---------------------------------------------------------------------------*/

#include <math.h>
#include "newtraph.h"

/* #define DEBUG */
#define MAXDIM 20

#ifdef DEBUG
#include <stdio.h>
FILE *fp;
#endif

void LUFactor(double Matrix[NMAX][NMAX], unsigned char Dim,
           unsigned char *Permutes, char *NPermutes)
{

     register    ii, jj, kk;
     double           Sum, TempElem;

     unsigned char    MaxRow;
     double           Elem, ScaleFac[NMAX], MaxElem;
     double           Merit;

     if (Dim > 32) {
          printf("LU ERROR -- Matrix too large!\n");
          return;
     }

     *NPermutes = ODD;

     for (ii = 0; ii < Dim; ii++) {
          MaxElem = 0.0;
          for (jj = 0; jj < Dim; jj++)
               if ((Elem = fabs(Matrix[ii][jj])) > MaxElem)
                    MaxElem = Elem;
          if (MaxElem == 0.0) {
               printf("LU ERROR -- Singular Matrix!\n");
               return;
          } else ScaleFac[ii] = 1.0/MaxElem;
     }

     for (jj = 0; jj < Dim; jj++) {

          if (jj > 0) {

               for (ii = 0; ii <= jj-1; ii++) {
                    Sum = Matrix[ii][jj];
                    if (ii > 0) {
                        for (kk = 0; kk <= ii-1; kk++)
                          Sum -= Matrix[ii][kk]*Matrix[kk][jj];
                        Matrix[ii][jj] = Sum;
                    }
               }
          }
          MaxElem = 0.0;

          for (ii = jj; ii < Dim; ii++) {

               Sum = Matrix[ii][jj];
               if (jj > 0) {
                    for (kk = 0; kk <= jj-1; kk++)
                         Sum -= Matrix[ii][kk]*Matrix[kk][jj];
                    Matrix[ii][jj] = Sum;
               }
               Merit = ScaleFac[ii]*fabs(Sum);
               if (Merit >= MaxElem) {
                    MaxRow = ii;
                    MaxElem = Merit;
               }

          }

          if (jj != MaxRow) {
               for (kk = 0; kk < Dim; kk++) {
                    TempElem = Matrix[MaxRow][kk];
                    Matrix[MaxRow][kk] = Matrix[jj][kk];
                    Matrix[jj][kk] = TempElem;
               }
               *NPermutes *= -1;
               ScaleFac[MaxRow] = ScaleFac[jj];
          }
          Permutes[jj] = MaxRow;

          if (jj != Dim-1) {
               if (Matrix[jj][jj] == 0.) Matrix[jj][jj] = TINY;
               TempElem = 1.0/Matrix[jj][jj];
               for (ii = jj+1; ii < Dim; ii++)
                    Matrix[ii][jj] *= TempElem;
          }
     }

     if (Matrix[Dim-1][Dim-1] == 0.0) Matrix[Dim-1][Dim-1] = TINY;
}


void LUBackSub(double Matrix[NMAX][NMAX], unsigned char Dim,
           unsigned char *Permutes, double RHS[NMAX])
{

     register    i,ii,j,ll;
     double           Sum;

     ii = -1;

     for (i = 0; i < Dim; i++) {
          ll = Permutes[i];
          Sum = RHS[ll];
          RHS[ll] = RHS[i];
          if (ii != -1) {
               for (j = ii; j <= i-1; j++)
                    Sum -= Matrix[i][j]*RHS[j];
          } else if (Sum != 0.0) {
               ii = i;
          }
          RHS[i] = Sum;
     }

     for (i = Dim-1; i >= 0; i--) {
          Sum = RHS[i];
          if (i < Dim-1) {
               for (j = i+1; j < Dim; j++)
                    Sum -= Matrix[i][j]*RHS[j];
          }
          RHS[i] = Sum/Matrix[i][i];
     }
}


void NRIteration(unsigned char NTrials, double *X, unsigned char Dim,
          double XToler, double FToler, void (*UserFun)())
{
     register ii,kk;
     double     ErrF, ErrX;

     double          H[NMAX][NMAX], h[NMAX];
     unsigned char   RowPermutes[NMAX];
     char       NPermutes;

#ifdef DEBUG
     fp = fopen ("nrdebug", "a");
     fprintf (fp, "FTOL = %lf, XTOL = %lf, NTRIALS = %d\n",
          FToler, XToler, NTrials);
#endif
     for (kk = 0; kk < NTrials; kk++) {

          (*UserFun)(X, H, h);
          ErrF = 0.0;
          for (ii = 0; ii < Dim; ii++) ErrF += fabs(h[ii]);
#ifdef DEBUG
     fprintf (fp, "NR: ErrF = %6.2lf\n", ErrF);
#endif
          if (ErrF <= FToler) {
#ifdef DEBUG
               fclose (fp);
#endif
               return;
          }

          LUFactor (H, Dim, RowPermutes, &NPermutes);
          LUBackSub(H, Dim, RowPermutes, h);

          ErrX = 0.0;
          for (ii = 0; ii < Dim; ii++) {
               ErrX += fabs(h[ii]);
               X[ii] += h[ii];
          }
#ifdef DEBUG
     fprintf (fp, "NR: ErrX = %6.2lf\n", ErrX);
#endif
          if (ErrX <= XToler) {
#ifdef DEBUG
               fclose (fp);
#endif
               return;
          }
     }
}

/*---------------------------- End of NEWTRAPH.C -----------------------------*/
```

```
/*---------------------------------------------------------------------------

PATHPLAN.C - Generate a trajectory from point to point in 6 DOF.

    Author   :  Terry Fong
    Created  :  07-27-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/
#ifdef DEBUG
#include <stdio.h>
#endif

#include "splat.h"

/*---------------------------------------------------------------------------
  Generate "points" number of intermediate points between start and end by
  calculating the actuator lengths at the starting and ending points and
  linear interpolating lengths. The calculated actuator lengths are returned
  in "lengths".
---------------------------------------------------------------------------*/
void pathplan (frame start, frame end, byte points, float lengths[][6])
{
      vector actuator[6];
      float start_len[6], end_len[6], delta[6];
      register i, j;

#ifdef DEBUG
      FILE *fp;
      fp=fopen("debug", "w");
      fprintf(fp, "\nBefore INVKIN, start frame is:\n");
      dump_frame (fp, start);
#endif

      invkin (start, actuator);

#ifdef DEBUG
      fprintf(fp, "After INVKIN, start frame is:\n");
      dump_frame (fp, start);
      fprintf(fp, "\nACTUATOR vectors are:\n");
      dump_vectors (fp, actuator);
#endif

      calc_len (start_len, actuator);

#ifdef DEBUG
      fprintf(fp, "\nSTART_LEN are:\n");
      dump_lengths (fp, start_len);
#endif

      invkin (end, actuator);
      calc_len (end_len, actuator);
      for (i=0; i<6; i++)
            delta[i] = (end_len[i] - start_len[i]) / (points - 1);
      for (i=0; i<points; i++)
            for (j=0; j<6; j++)
                  lengths[i][j] = start_len[j] + i*delta[j];
#ifdef DEBUG
      fclose (fp);
#endif
}

/*-------------------------- End of PATHPLAN.C ------------------------------*/
```

```
/*----------------------------------------------------------------------------

PORTDEFS.H - Prototype Expansion Board I/O Port definitions for SPLAT.C

    Author   :  Terry Fong
    Created  :  07-20-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
----------------------------------------------------------------------------*/

#define PORT_A1         0x3E0 /* 8255-1 port definitions */
#define PORT_B1         0x3E1
#define PORT_C1         0x3E2
#define CTRL_PORT1      0x3E3

#define PORT_A2         0x3E4 /* 8255-2 port definitions */
#define PORT_B2         0x3E5
#define PORT_C2         0x3E6
#define CTRL_PORT2      0x3E7

#define PORT_A3         0x3E8 /* 8255-3 port definitions */
#define PORT_B3         0x3E9
#define PORT_C3         0x3EA
#define CTRL_PORT3      0x3EB

#define PORT_A4         0x3EC /* 8255-4 port definitions */
#define PORT_B4         0x3ED
#define PORT_C4         0x3EE
#define CTRL_PORT4      0x3EF
```

```
/*----------------------------------------------------------------------------

SENSOR.C - Routines for reading quadrature signals with HCTL-2000's

    Author   :  Terry Fong
    Created  :  07-27-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
----------------------------------------------------------------------------*/

#include <conio.h>
#include "portdefs.h"
#include "splat.h"

/*----------------------------------------------------------------------------
  Read 12 bit optical encoder positions via the six HCTL-2000s wired to 8255-1
  Position is returned as a signed integer.  Valid sensor range is 0 to 5.
----------------------------------------------------------------------------*/
int read_position (byte sensor)
{
      int pos;
      byte lb, hb;
      byte sel_code = 0x01;

      /* reset sensor bit (OE*) to select HCTL to read */
      outp (PORT_B1, 0xff ^ (sel_code << sensor));

      outp (CTRL_PORT1, 0x2); /* reset 8255-1 PC1 to select position msb */
      hb = inp (PORT_A1);     /* grab msb from 8255-1 */
      outp (CTRL_PORT1, 0x3); /* set PC1 to select position lsb */
      lb = inp (PORT_A1);     /* grab lsb from 8255-1 */
      outp (PORT_B1, 0xff);   /* set (OE*) to force inhibit logic reset */

      pos = hb*256 + lb;      /* calculate unsigned 12-bit position */

      if (hb > 7) pos -= 4096;/* change position to signed 12-bit */

      return (pos);           /* return the position as a 16-bit number */
}

/*----------------------------------------------------------------------------
  Set the zero position for the optical encoder by sending a low/high pulse to
  the HCTL RST* (pin 5)
----------------------------------------------------------------------------*/
void zero_position ()
{
      outp (CTRL_PORT1, 0x0);       /* reset 8255-1 PC5 */
      outp (CTRL_PORT1, 0x1);       /* set   8255-1 PC5 */
}

/*------------------------ End of SENSOR.C --------------------------------*/
```

```
     Stewart Platform / Actuator Mount Points
---------------------------------------------------
        (Base Frame)            (Platform Frame)
#  base x  base y  base z  plat x  plat y  plat z
---------------------------------------------------
0   7.15   -2.26    0.00    4.33   -2.50    0.00
1   7.15    2.26    0.00    4.33    2,50    0.00
2  -1.62    7.32    0.00    0.00    5.00    0.00
3  -5.53    5.07    0.00   -4.33    2.50    0.00
4  -5.53   -5.07    0.00   -4.33   -2.50    0.00
5  -1.62   -7.32    0.00    0.00   -5.00    0.00
---------------------------------------------------
Notes:      1) All dimensions are in decimal inches.
      2) All dimensions are taken from the plate centroid.
      3) Do NOT alter the first 5 lines since they are read by MAKEPLAT.C

******************************************************************************
MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
         Stewart Platform Augmented Manipulator (SPAM) Project
            All Rights Reserved. Copyright (C) 1989
******************************************************************************

/*------------------------ End of SETUP.DAT--------------------------------*/
```

```
/*---------------------------------------------------------------------------

SOLENOID.H - Define the Super Sniffy Solenoid Object and various things about
           those wondefully wacky solenoids.

    Author   :  Terry Fong
    Created  :  07-23-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include "splat.h"

typedef struct solenoid {
      byte on_code;               /* Koganei's have a single ON code */
      byte off_code;              /*    "       "  "  "    OFF  "  */
      byte pw;          /* Kips' have 16 pwm levels */
      byte nybble;                /* Kips' are wired to either UPPER or LOWER */
      unsigned int port;          /* port to send output codes to */
      struct solenoid *buddy;     /* pointer to link up solenoids */
} solenoid;

typedef struct {
      struct {
            solenoid top;
            solenoid bottom;
      } air;
      struct {
            solenoid top;
            solenoid bottom;
      } water;
} sol_group;

#define NULL      0
#define SOL_ON  15
#define SOL_OFF   0

#define UPPER     1
#define LOWER     0

#define UPPER_MASK      0x0f
#define LOWER_MASK      0xf0

/* SOLENOID.C */
void link_kips (solenoid kips[]);
void control_sol (solenoid *sol, byte code);
void display_pw (solenoid kips[]);
void init_sols (void);
void close_all (solenoid sols[]);
void toggle_kog (byte number);
void pulse_kip (byte number, byte pw);
void close_all_kips (void);
```

```
/*---------------------------------------------------------------------------

SOLENOID.C - Solencid PWM operation routines.


    Author   :  Terry Fong
    Created  :  07-20-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <stdio.h>
#include <conio.h>
#include "portdefs.h"
#include "solenoid.h"

/*--------------------- Module global solenoids !!! ----------------------*/
static solenoid kips[12] =
      {0,0,0,LOWER,PORT_A2,0,  0,0,0,UPPER,PORT_A2,0,
       0,0,0,LOWER,PORT_B2,0,  0,0,0,UPPER,PORT_B2,0,
       0,0,0,LOWER,PORT_A3,0,  0,0,0,UPPER,PORT_A3,0,
       0,0,0,LOWER,PORT_B3,0,  0,0,0,UPPER,PORT_B3,0,
       0,0,0,LOWER,PORT_A4,0,  0,0,0,UPPER,PORT_A4,0,
       0,0,0,LOWER,PORT_B4,0,  0,0,0,UPPER,PORT_B4,0};
static solenoid kogs[12] =
      {0,1,0,0,CTRL_PORT2,0,   2,3,0,0,CTRL_PORT2,0,
       4,5,0,0,CTRL_PORT2,0,   6,7,0,0,CTRL_PORT2,0,
       0,1,0,0,CTRL_PORT3,0,   2,3,0,0,CTRL_PORT3,0,
       4,5,0,0,CTRL_PORT3,0,   6,7,0,0,CTRL_PORT3,0,
       0,1,0,0,CTRL_PORT4,0,   2,3,0,0,CTRL_PORT4,0,
       4,5,0,0,CTRL_PORT4,0,   6,7,0,0,CTRL_PORT4,0};

/*---------------------------------------------------------------------------
  Solenoid control: ON, OFF, or a pulse CODE. The Koganei solenoids are wired
  individually and have only 2 states (ON/OFF). The Kip solenoids are wired
  in "buddy" pairs and have 16 states (ON/OFF and 14 PWM levels). To determine
  what kind of solenoid has been passed, a check is made of sol->buddy. If
  this is NULL, then we are dealing with a Koganei, else we have a Kip.
---------------------------------------------------------------------------*/
void control_sol (solenoid *sol, byte code)
{
      if (sol->buddy == NULL)
            switch (code) { /* switch is used to ensure that only ON and
                               OFF are valid Koganei codes, all other
                               codes "fall off the end of the switch" */

                  case SOL_ON:
                        outp (sol->port, sol->on_code);
                        sol->pw = SOL_ON;
                        break;

                  case SOL_OFF:
                        outp (sol->port, sol->off_code);
                        sol->pw = SOL_OFF;
                        break;
            }
      else {      /* Here we deal with a Kip solenoid. If it is wired to the
               upper nybble of a 8255 port, we must calculate the output
               code by zeroing the upper nybble of pw (using UPPER_MASK
               and joining the desired pulse code (shifted into the upper
               nybble */

            if (sol->nybble == UPPER)
                  sol->pw = (sol->pw & UPPER_MASK) | (code << 4);
            else
                  sol->pw = (sol->pw & LOWER_MASK) | code;

            outp (sol->port, sol->pw);

            /* Since Kips are wired in pairs, we must save the current
               output code in the "buddy's" pw */
            sol->buddy->pw = sol->pw;
      }
}

/*---------------------------------------------------------------------------
  Initialize the solenoids by "linking" kips and closing all valves.
---------------------------------------------------------------------------*/
void init_sols ()
{
      link_kips (kips);
      close_all (kogs);
      close_all (kips);
}

/*---------------------------------------------------------------------------
  Close a set of solenoids.
---------------------------------------------------------------------------*/
void close_all (solenoid sols[12])
{
      register i;

      for (i=0; i<12; i++) control_sol (&sols[i], SOL_OFF);
}

/*---------------------------------------------------------------------------
  Connect the buddy pairs of kips.
---------------------------------------------------------------------------*/
void link_kips (solenoid kips[12])
{
      register i;

      for (i=0; i<11; i=i+2) {
            kips[i].buddy = &kips[i+1];
            kips[i+1].buddy = &kips[i];
      }
}

/*---------------------------------------------------------------------------
  Display the pulse width codes for all the kip solenoids.
---------------------------------------------------------------------------*/
void display_pw (solenoid kips[12])
{
      byte i;

      printf ("\n");
      for (i=0; i<12; i++) {
            printf ("%4x", kips[i].pw);
      }
      printf ("\n");
}
```

```
/*------------------------------------------------------------------------------
  Toggle a Koganei solenoid
------------------------------------------------------------------------------*/
void toggle_kog (byte number)
{
      if (kogs[number].pw)
            control_sol (&kogs[number], OFF);
      else
            control_sol (&kogs[number], ON);
}

/*------------------------------------------------------------------------------
  Pulse a KIP solenoid
------------------------------------------------------------------------------*/
void pulse_kip (byte number, byte pw)
{
      control_sol (&kips[number], pw);
}

/*------------------------------------------------------------------------------
  Close all KIPS completely
------------------------------------------------------------------------------*/
void close_all_kips ()
{
      close_all (kips);
}
/*-------------------------- End of SOLENOID.C ------------------------------*/
```

```
/*------------------------------------------------------------------------------

SPLAT.H - Constants, typedefs, and function prototype declarations

    Author   :  Terry Fong
    Created  :  07-20-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
------------------------------------------------------------------------------*/

#include <gnleaf\gf.h>
#include <stdio.h>
#include "video.h"

/*------------------------- Some cool constants -----------------------------*/
#define PI          3.1415926
#define DEG2RAD     PI/180
#define RAD2DEG     180/PI
#define MAX_STEPS 5
#define STD_ACC           0.001

/*--------------------- Define and typedef some vars ------------------------*/
typedef struct {        /* Here is a vector in position space */
      float x;
      float y;
      float z;
} pos_vector;

typedef struct {        /* Here is a vector in orientation space */
      float roll;
      float pitch;
      float yaw;
} ang_vector;

typedef pos_vector vector;    /* A "normal" vector describes position */

typedef struct {        /* A 6-DOF location is called a "frame" */
      pos_vector position;
      ang_vector angle;
} frame;

typedef char string[80];
typedef unsigned char byte;

/*-------------------------- Function declarations -------------------------*/

/*** CONTROL.C ***/
void cl_control (float lengths[], byte steps);
void ol_control (window_ptr TheWindow, int selection);

/*** FWDKIN.C ***/
unsigned char fwdkin (double act_lengths[], int use_guess, frame *guess,
                  double xtol);

/*** INVKIN.C ***/
void invkin (frame platform, vector actuator[]);
void calc_cur_state (frame current, float length[], float tilt[],
                  float roll[]);
void calc_len (float length[], vector actuator[]);
void calc_angles (float tilt[], float roll[], vector actuator[]);
```

```
/*** MAKEPLAT.C ***/
void create_sp (void);
void display_sp (void);

/*** PATHPLAN.C ***/
void pathplan (frame start, frame end, byte points, float lengths[][6]);

/*** SENSOR.C ***/
void zero_position (void);
int read_position (byte sensor);

/*** SPLAT.C ***/
void try_command (void);
int dispatch (void);
int get_command (window_ptr TheWindow, int selection);
void display_path (window_ptr actuator_win, byte steps, float lengths[][6]);

/*** VECTOR.C ***/
float dot_prod (vector x, vector y);
float magnitude (vector x);
vector normalize (vector x);

/*** DISPLAY.C ***/
void draw_screen (int steps, float accuracy, frame current, frame command);
void show_actuator (window_ptr TheWindow, int n, float length, float tilt,
          float roll);
void show_current (window_ptr TheWindow, frame platform);
void show_target (window_ptr TheWindow, frame platform);
void show_accuracy (window_ptr TheWindow, float accuracy);
void show_steps (window_ptr TheWindow, int steps);
void show_time (int start);
void show_filename (char *string, int display);

/*** MISC.C ***/
void delay (int seconds);
void exit_splat (void);
int close_save (FILE *fp, char *filename);
int start_save (window_ptr TheWindow, FILE **fp, char *filename);
void save_path (FILE *fp, byte steps, float lengths[][6], frame current,
          frame command);
void prompt (char *string);
void message (char *string);
void dump_frame (FILE *fp, frame x);
void dump_lengths (FILE *fp, float lengths[]);
void dump_vector (FILE *fp, vector x[]);

/*** MENU.C ***/
int show_menu (window_ptr TheWindow, int menu);
/*-------------------------- End of SPLAT.H ------------------------------*/
```

```
/*---------------------------------------------------------------------------

VECTOR.C - Assorted vector math routines.

    Author   :  Terry Fong
    Created  :  07-20-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <math.h>
#include "splat.h"

/*---------------------------------------------------------------------------
  Calculate the magnitude (2-norm) of some arbitrary vector.
---------------------------------------------------------------------------*/
float magnitude (vector x)
{
      double xsquare;

      xsquare = (double) (x.x*x.x + x.y*x.y + x.z*x.z);

      return((float) sqrt(xsquare));
}

/*---------------------------------------------------------------------------
  Calculate the dot (scalar) product of two arbitrary vectors.
---------------------------------------------------------------------------*/
float dot_prod (vector x, vector y)
{
      return((float) (x.x*y.x + x.y*y.y + x.z*y.z));
}

/*---------------------------------------------------------------------------
  Calculate a normalized unit vector.
---------------------------------------------------------------------------*/
vector normalize (vector x)
{
      float mag_x;
      vector y;

      mag_x = magnitude(x);
      y.x = x.x / mag_x;
      y.y = x.y / mag_x;
      y.z = x.z / mag_x;

      return (y);
}

/*-------------------------- End of VECTOR.C -----------------------------*/
```

```
/*----------------------------------------------------------------------------

VIDEO.H - Yo! It's FONGMASTER's graphics stuff !!!

    Author   :  Terry Fong
    Created  :  08-03-89
    Modified :  03-13-90

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
----------------------------------------------------------------------------*/

#include <gnleaf\color.h>
#include <graph.h>

/*------------------------ define a window structure --------------------*/
typedef struct {
      struct rccoord top;
      struct rccoord bottom;
      struct rccoord curpos;
      struct {
            short background;
            short border;
            short text;
      } color;
      char *name;

      void (*draw) ();
} window;

/*----------------------- Function Prototypes ---------------------------*/

void init_video ();
void create_window (window_ptr x, char *string);
void clear_window (window_ptr x);
void wprintrc (window_ptr x, short row, short col, char *string);
void wprintf (window_ptr x, char *string);
void wcprintf (window_ptr x, short row, char *string);
void cursor_off ();
void cursor_on ();
float input (window_ptr TheWindow, char *string);

/*-------------------------- End of VIDEO.H -----------------------------*/
```

```
/*----------------------------------------------------------------------------

VIDEO.C - IBM PC Video and color text screen graphic primitives

    Author   :  Terry Fong
    Created  :  07-31-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
----------------------------------------------------------------------------*/

#include <dos.h>
#include <stdio.h>
#include <string.h>
#include "video.h"

/*----------------------------------------------------------------------------
  Initialize the video to 80 column color, turn wrapping off.
----------------------------------------------------------------------------*/
void init_video ()
{
      _setvideomode (_TEXTC80);
      _setbkcolor ((long) BLUE);
      _clearscreen (_GCLEARSCREEN);
      _wrapon (_GWRAPOFF);
      cursor_off ();
}

/*----------------------------------------------------------------------------
  Input a floating point number
----------------------------------------------------------------------------*/
float input (window_ptr TheWindow, char *string)
{
      float result;

      wprintf (TheWindow, string);
      TheWindow->curpos.row++;
      TheWindow->curpos.col = 1;
      cursor_on ();
      scanf ("%f", &result);
      cursor_off ();

      return (result);
}

/*----------------------------------------------------------------------------
  Clear a specific window and fill with window background color. The window
  cursor is "homed" to (1, 1).
----------------------------------------------------------------------------*/
void clear_window (window_ptr x)
{
      _settextwindow (x->top.row+1, x->top.col+1,
                      x->bottom.row-1, x->bottom.col-1);
      _setbkcolor ((long) x->color.background);
      _clearscreen (_GWINDOW);
      x->curpos.row = 1;
      x->curpos.col = 1;
}

/*----------------------------------------------------------------------------
  Create a window on the screen by drawing a border and filling with color
  if NAME is not NULL, print a centered title on the upper border line.
--------------------------------------------------------------------------*/
void create_window (window_ptr x, char *name)
{
     char  top[81], middle[81], bottom[81];    /* Max box is 80 char wide */
     register i;
     long  old_back;
     short old_text;

     /* Save the current colors before messing with them */
     old_back = _getbkcolor ();
     old_text = _gettextcolor ();

     /* Compile the strings for the box */
     top [0] = '●';            /* char 201 */
     middle [0] = '●';         /* char 186 */
     bottom [0] = '●';         /* char 200 */
     for (i = 1; i < (x->bottom.col - x->top.col); i++) {
          top [i] = '●';            /* char 205 */
          middle [i] = ' '; /* char 32  */
          bottom [i] = '●';         /* char 205 */
     }
     top [i] = '●';            /* char 187 */
     middle [i] = '●';         /* char 186 */
     bottom [i] = '●';         /* char 188 */

     /* terminate all the strings with a null char. */
     i++;
     top [i] = '\0';
     middle [i] = '\0';
     bottom [i] = '\0';

     _setbkcolor ((long) x->color.background);
     _settextcolor (x->color.border);

     /* Draw the top line */
       _settextposition (x->top.row, x->top.col);
     _outtext (top);

     if (name != NULL) {
          _settextposition(x->top.row,
                (x->bottom.col + x->top.col - strlen(name))/2 + 1);
          _outtext (name);
     }

     /* Draw the middle lines */
     for (i=1; i < (x->bottom.row - x->top.row); i++) {
          _settextposition (x->top.row + i, x->top.col);
          _outtext (middle);
     }

     /* Draw the bottom of the box */
     _settextposition (x->bottom.row, x->top.col);
     _outtext (bottom);

     /* Restore the original colors */
     _setbkcolor (old_back);
     _settextcolor (old_text);
}

/*------------------------------------------------------------------------

  Print a text string at a specified row and column (in window coords)
  using the window text color. If string exceeds right window edge, it is
  truncated. Does not update the window CURPOS. Note: the origin is (1,1)
  and is located inside the upper left corner window border:

     ●●●
     ●*%                * = row 1, col 1     % = row 1, col 2
     ●+                 + = row 2, col 1     etc...

  Minimal escape char line control is provided (_OUTTEXT really only prints
  text strings well). '\n' works correctly though ...
--------------------------------------------------------------------------*/
void wprintrc (window_ptr x, short row, short col, char *string)
{
     _settextwindow (x->top.row+1, x->top.col+1,
                x->bottom.row-1, x->bottom.col-1);
     _settextcolor (x->color.text);
     _setbkcolor ((long) x->color.background);
     _settextposition (row, col);
     _outtext (string);
     x->curpos = _gettextposition();
}

/*--------------------------------------------------------------------------
  Print a text string starting at the current text position. If printing at
  last line, all text in window is scrolled up. Note: no word wrapping!!!
--------------------------------------------------------------------------*/
void wprintf (window_ptr x, char *string)
{
     _settextwindow (x->top.row+1, x->top.col+1,
                x->bottom.row-1, x->bottom.col-1);
     _settextcolor (x->color.text);
     _setbkcolor ((long) x->color.background);

     /* if past last row, scroll up */
     if (x->curpos.row == x->bottom.row-x->top.row) {
          x->curpos.row--;
          upscroll (1, x->top.row, x->bottom.row-2, x->top.col,
                x->bottom.col-2, x->color.background<<4);
     }

     _settextposition (x->curpos.row, x->curpos.col);
     _outtext (string);
     x->curpos = _gettextposition();
}

/*--------------------------------------------------------------------------
  Print centered text on a specified window row
--------------------------------------------------------------------------*/
void wcprintf (window_ptr x, short row, char *string)
{
     wprintrc (x, row,(x->bottom.col-x->top.col-strlen(string))/2, string);
}

/*--------------------------------------------------------------------------
  Turn off the cursor by playing with INT 10H.
--------------------------------------------------------------------------*/
void cursor_off ()
{
     union REGS regs;
     regs.h.ah = 1;
     regs.h.ch = 0x20; /* set bit 5 to disable cursor */

     int86 (0x10, &regs, &regs);
}

/*---------------------------------------------------------------------------
  Turn on the cursor with INT 10H
---------------------------------------------------------------------------*/
void cursor_on ()
{
     union REGS regs;

     regs.h.ah = 1;
     regs.h.ch = 6;
     regs.h.cl = 7;

     int86 (0x10, &regs, &regs);
}

/*-------------------------- End of VIDEO.C -------------------------------*/
```

```
/*---------------------------------------------------------------------------

WINDOW.H - Window definitions

    Author   :  Terry Fong
    Created  :  07-20-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

/* top row, col, bot row, col, start row, col, back, border, text */
static window title_w    = {2, 21, 5, 57, 1, 1, WHITE, BLACK, BLACK};
static window error_w    = {20, 2, 24, 79, 1, 1, RED, BLACK, YELLOW};
static window actuator_w = {10, 32, 18, 58, 1, 1, CYAN, BLACK, BLACK};
static window platform_w = {10, 59, 18, 79, 1, 1, CYAN, BLACK, BLACK};
static window menu_w     = {7, 2, 18, 30, 1, 1, LTGREEN, BLACK, BLUE};
static window param_w    = {5, 59, 8, 79, 1, 1, CYAN, BLACK, BLACK};

window_ptr title_win    = &title_w,
         error_win    = &error_w,
         actuator_win = &actuator_w,
         platform_win = &platform_w,
         menu_win     = &menu_w,
         param_win    = &param_w;
```

```
#--------------------------------------------------------------------------
#
#   Microsoft C MAKEFILE for SPAM2
#
#   Author    :  Terry Fong
#   Created   :  03-12-90
#   Modified  :
#
#   NOTES: use the /Oa (relax alias checking) with EXTREME CAUTION !!!
#          (remember:  /Ox = /Oailt /Gs )
#
#--------------------------------------------------------------------------
#
# Some useful macro definitions
#
model=/AS /FPi87          # small memory model, inline 8087 support
optimize=/c /Oilt $(model)    # optimize, do not relax alias, do stack probe
stdcomp=/c /Od $(model)       # no alias relaxing, no default optimizations
stdlibs=slibc7+graphics+gfs
stdlink=/NOI /NOD /NOE
objects=arm+control+display+spam2+spamio+video
progdep=arm.obj control.obj display.obj spam2.obj spamio.obj spam2
program=spam2.exe

#
# Standard .C -> .OBJ inference rule: compile with full optimization
#
.C.OBJ :
      cl $(stdcomp) $*.c

#
# Dependency block definitions
#
arm.obj : arm.c

control.obj : control.c

display.obj : display.c menu.h windows.h

spam2.obj : spam2.c spam2.h menu.h

spamio.obj : spamio.c spamio.h portdefs.h

video.obj : video.c video.h

#
# Create executable ...
#
$(program) : $(progdep)
      link $(stdlink) $(objects),$*,NUL,$(stdlibs)
```

```
/*--------------------------------------------------------------------------

ARM.H - Function declarations for ARM.C

    Author   : Terry Fong
    Created  : 03-12-90
    Modified       :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
--------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/* The following linked list implementation is used by backwards difference */
/* velocity calculation routines. A linked list (actually a ring) is used   */
/* to efficiently maintain previous positons. Instead of constantly updating*/
/* 'n' states (for a n-step backwards difference), a ring is defined and the*/
/* only thing we have to update is the pointer to the current node         */

#define BD_STEPS  3

typedef struct node {
      double position;
      struct node *next;
      struct node *prev;
} node, *node_ptr, **node_handle;

typedef node list[BD_STEPS];

void init_backdiff (void);
void init_list (node TheList[BD_STEPS]);
/*------------------------ End of linked list stuff ----------------------*/

joint_vector invkin (vector desired);
void update_arm_state (void);
void update_tip_state (void);
double calc_veloc (int FLAG);
void fwdkin (void);
joint_vector inv_jacobian (vector desired);
```

```
/*---------------------------------------------------------------------------

ARM.C - Manipulator calculation routines

    Author  : Terry Fong
    Created : 03-12-90
    Modified      :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
---------------------------------------------------------------------------*/

/* #define DEBUG */ /* if using DEBUG, use LINK /NOE ARM.OBJ */

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "spam2.h"

/*------------------------ Global Variables --------------------------------*/
#ifndef DEBUG
extern double T;              /* SPAMIO.C */
extern joint_state arm_current;         /* SPAM2.C */
extern cartesian_state tip_current; /* SPAM2.C */
extern cartesian_state tip_current; /* SPAM2.C */
extern double L1, L2;               /* SPAM2.C */
#else
double T;
cartesian_state tip_current = {0, 0, 0, 0, 0, 0};
cartesian_state tip_command = {0, 0, 0 ,0, 0, 0};
joint_state arm_current = {0, 0, 0, 0, 0, 0};
joint_state arm_command = {0, 0, 0, 0, 0, 0};
double L1 = 10, L2 = 10;
#endif

/*------------------------- Module Global Variables ------------------------*/

/* The following linked lists are used for backward difference */
static list tip_x;
static list tip_z;
static list shoulder_angle;
static list elbow_angle;

/*------------------------- End of Global Variables ------------------------*/


/*
** Initialize backwards difference linked lists.
*/
void init_backdiff (void)
{
      init_list (tip_x);
      init_list (tip_z);
      init_list (shoulder_angle);
      init_list (elbow_angle);
}

/*
** Create a circular list (aka ring) with forward and backward links, by
** initializing the passed array (TheList).
*/
void init_list (node TheList[BD_STEPS])
{
      register i;

      /* zero out all position values */
      for (i=0; i<BD_STEPS; i++) TheList[i].position=0;

      /* create forward links */
      for (i=0; i<BD_STEPS-1; i++) {
            TheList[i].next = &TheList[i+1];
      }
      TheList[BD_STEPS-1].next=&TheList[0];

      /* create backward links */
      for (i=BD_STEPS-1; i>0; i--) {
            TheList[i].prev = &TheList[i-1];
      }
      TheList[0].prev = &TheList[BD_STEPS-1];
}

/*
** Read joint positions, updating current and previous joint states
*/
void update_arm_state (void)
{
#ifndef     DEBUG
      arm_current.angle.elbow = read_angle (ELBOW);
      arm_current.angle.shoulder = read_angle (SHOULDER);
#endif
      arm_current.velocity.elbow = calc_veloc (ELBOW);
      arm_current.velocity.shoulder = calc_veloc (SHOULDER);
}

/*
** Update Tip position and velocity.
*/
void update_tip_state (void)
{
      fwdkin ();

      tip_current.velocity.x = calc_veloc (TIP_X);
      tip_current.velocity.z = calc_veloc (TIP_Z);
}

/*
** Calculate velocity using backwards differencing
*/
double calc_veloc (int FLAG)
{
      static node_ptr tip_x_ptr = tip_x;
      static node_ptr tip_z_ptr = tip_z;
      static node_ptr shoulder_angle_ptr = shoulder_angle;
      static node_ptr elbow_angle_ptr = elbow_angle;
      node_handle cur_handle;
      double velocity, cur_pos;

      switch (FLAG) {
            case TIP_X :
                  cur_handle = &tip_x_ptr;
                  cur_pos = tip_current.position.x;
                  break;
```

```
		case TIP_Z :
			cur_handle = &tip_z_ptr;
			cur_pos = tip_current.position.z;
			break;

		case ELBOW :
			cur_handle = &elbow_angle_ptr;
			cur_pos = arm_current.angle.elbow;
			break;

		case SHOULDER :
			cur_handle = &shoulder_angle_ptr;
			cur_pos = arm_current.angle.shoulder;
			break;
	}

	(*cur_handle)->position = cur_pos;

	velocity = ((*cur_handle)->position-(*cur_handle)->prev->position)
		/ (T*BD_STEPS);

	(*cur_handle) = (*cur_handle)->next;

	return (velocity);
}

/*
** Manipulator forward kinematics [theta] --> [cartesian tip]
**
** Note: To be consistent about what joint we are referencing, at all
**	times possible, joints should be referenced by NAME not by
**	number. The problem is that the Denavit-Hartenberg notation
**	given in CRAIG is problematic since it assigns joint 1 as
**	being the base joint and joint n as the tip joint. Since we are
**	trying to make this software as generic as possible (so that it
**	can be EASILY MODIFIED to change from a 2-link to 3-link version),
**	  joints will be numbered in reverse order (i.e. starting at tip).
**	From an analytical point of view, there is no difference
**	between the numbering systems since both will generate the
**	required base-to-tip transforms.
**
**	In practice, however, when one is concerned with using the
**	same code to control different manipulators (i.e. with different
**	numbers of joints), the latter scheme requires far less code
**	changes since the tip joint is ALWAYS referenced as joint 1.
**
**	Therefore, to be consistent, we will NOT USE D-H notation
**	throughout but rather assign #1 to the tip joint. This
**	means that for SPAM, the ELBOW is always joint #1, the SHOULDER
**	is joint #2, and the WAIST is always joint #3. This will allow
**	both 2 and 3 link arm configurations to be operated with MINIMAL
**	code changes.
**
**	Of course, numbers are not a problem if one does the kinematics
**	from a purely symbolic point of view (i.e. labelling the elbow
**	joint angle as THETA(ELBOW)) rather than a numerical(?) point of
**	view (i.e. elbow joint angle = THETA(1)). In any case, to keep
**	a high level of abstraction, joints should be referred to by name
**	(i.e. symbolically) rather than by number. I guess the real problem
**	is that it is much easier for code to handle things in a numerical
**	sense rather than a symbolic sense (i.e. the joint controller
```

```
**	 prefers to loop through the joints by number).
**
**	Joint		Defined as	D-H (2-link)	D-H (3-link)
**	----------------------------------------------------------------
**	ELBOW		1		2		3
**	SHOULDER	2		1		2
**	WAIST		3		--		1
*/
void fwdkin (void)
{
	double A, B, C1, S1;

	A = L1 + L2*cos(arm_current.angle.elbow*DEG2RAD);
	B = L2*sin(arm_current.angle.elbow*DEG2RAD);
	C1 = cos(arm_current.angle.shoulder*DEG2RAD);
	S1 = sin(arm_current.angle.shoulder*DEG2RAD);

	tip_current.position.z = A*C1 - B*S1;
	tip_current.position.x = A*S1 + B*C1;
}

/*
** Compute the inverse kinematics transform
*/
joint_vector invkin (vector desired)
{
	static byte blah= 0;
	static double blah1;
	static double blah2;
	joint_vector command;
	double C2, S2, K1, K2;

	if (!blah) {	/* this silly code fragment is because we cannot
			   do the initialization that C++ allows!! Blah! */
		blah = TRUE;
		blah1 = 2*L1*L2;
		blah2 = sq(L1) + sq(L2);
	}

	C2 = (sq(desired.x)+sq(desired.z)-blah1)/blah2;

	if (fabs(C2) > 1) {	/* No solution */
		command.elbow = 0;
		command.shoulder = 0;
#ifndef DEBUG
		message (" ERROR: INVKIN - No solution\n");
#endif
		return (command);
	}

	S2 = sqrt(1-sq(C2));

	command.elbow = atan2(S2, C2)*RAD2DEG;

	K1 = L1 + L2*C2;
	K2 = L2*S2;

	command.shoulder = (atan2(desired.x,desired.z)-atan2(K2,K1))*RAD2DEG;

	if (command.elbow < 0) command.elbow = 0;
	if (command.shoulder < 0) command.shoulder = 0;
	return (command);
}

/*
** Compute joint velocities using the inverse jacobian transform. NO, the
** jacobian is not inverted here, the solution has already been formulated
** and is implemented as such.
*/
joint_vector inv_jacobian (vector desired)
{
	FILE *fp;

	double C1, S1, C2, S2, C12, S12, DEN1, DEN2;
	joint_vector command;

	C1 = cos(arm_current.angle.shoulder*DEG2RAD);
	S1 = sin(arm_current.angle.shoulder*DEG2RAD);
	C2 = cos(arm_current.angle.elbow*DEG2RAD);
	S2 = sin(arm_current.angle.elbow*DEG2RAD);

	if (!S2) {  /* Elbow joint singularity */
		S2 = 0.09;  /* sin(5 degrees) = 0.09 */
#ifndef DEBUG
		message (" ERROR: JACOBIAN - Singularity trapped\n");
#else
		printf (" ERROR: JACOBIAN - Singularity trapped\n");
#endif
	}

	C12 = C1*C2-S1*S2;
	S12 = C1*S2+S1*C2;
	DEN1 = L1*S2;
	DEN2 = L1*L2*S2;

	command.shoulder = C12/DEN1*desired.z + S12/DEN1*desired.x;
	command.elbow = -(L1*C1+L2*C12)/DEN2*desired.z -
			(L1*S2+L2*S12)/DEN2*desired.x;

#ifdef DEBUG_JAC
	fp = fopen ("debug.out", "a");
	fprintf (fp, "------------------------------------\n");
	fprintf (fp, "elbow = %lf, shoulder = %lf\n",
		arm_current.angle.elbow, arm_current.angle.shoulder);
	fprintf (fp, "inv_jacobian: v_x = %lf, v_z = %lf\n",
		desired.x, desired.z);
	fprintf (fp, "inv_jacobian: elbow = %lf, shoulder = %lf\n",
		command.elbow, command.shoulder);
	fclose (fp);
#endif

	return (command);
}


/*
** This replaces the standard matherr() function to trap math functions
** errors. Module linking must be done with /NOE (No external dictionary)
*/
int matherr(struct exception *x)
{
	/* atan2(y,x) returns a DOMAIN error if both x,y are 0 */
	if (x->type == DOMAIN && strcmp(x->name, "atan2") == 0) {
		x->retval = 0.0;
#ifndef DEBUG
		message (" ERROR: ATAN2(0,0)\n");
#endif
		return (1);
	}
	return (0); /* use default matherr() action */
}

#ifdef DEBUG
main ()
{
	cartesian_state blah;
	joint_state command;

	while (1) {
	printf ("Enter elbow angle: ");
	scanf ("%lf", &arm_current.angle.elbow);
	printf ("Enter shoulder angle: ");
	scanf ("%lf", &arm_current.angle.shoulder);
	printf ("Enter Vx: ");
	scanf ("%lf", &blah.velocity.x);
	printf ("Enter Vz: ");
	scanf ("%lf", &blah.velocity.z);

	command.velocity = inv_jacobian (blah.velocity);

	printf ("Elbow rate = %lf\n", command.velocity.elbow);
	printf ("Shoulder rate = %lf\n", command.velocity.shoulder);
	}
}
#endif
```

```
/*---------------------------------------------------------------------------

CONTROL.H - Function declarations for CONTROL.C

    Author  : Terry Fong
    Created : 03-12-90
    Modified      :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
---------------------------------------------------------------------------*/
void OL_Joint (void);
void CL_Tip (void);
void OL_ResRate (void);
void PD_pos_servo (double deadband);
void PD_veloc_servo (void);
void init_PD_servo (void);
void zero_tip_command (void);

#define round(x)  ((int) floor(x+0.5))
```

```
/*---------------------------------------------------------------------------

CONTROL.C - The heart and soul of SPAM control stuff !!!

    Author    :  Terry Fong
    Created   :  03-14-90
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#define DEBUG

#include <gnleaf\ibmkeys.h>
#include <math.h>
#include <stdio.h>
#include "menu.h"
#include "spam2.h"

/*---------------------------- Global Variables ------------------------------*/
extern double T;                /* SPAMIO.C */
extern joint_sols elbow_sols;       /* SPAMIO.C */
extern joint_sols shoulder_sols;    /* SPAMIO.C */
extern cartesian_state tip_command; /* SPAM2.C */
extern cartesian_state tip_current; /* SPAM2.C */
extern joint_state arm_current;          /* SPAM2.C */
extern joint_state arm_command;          /* SPAM2.C */

/* Low-level joint controller constants. These must be global since Kp, Kd are
   read by READ_CFG. */
double Kp, Kd;

/*------------------------- Module global variables ------------------------*/
/* Low-level joint controller flag and constants */
static byte CONTROL_ON;
static double K1, K2;

/* The following array points to joint solenoid groups. The array is indexed
   according to the ordering given in SPAM2.H in accordance with D-H notation.
   For more info, see FWDKIN note in ARM.C */
static joint_sols *joint[4] = {NULL, &elbow_sols, &shoulder_sols, NULL};

/*
** Open Loop Joint Control.
*/
void OL_Joint (void)
{
      /* The following two arrays contain current joint solenoid pulse-
         widths. Indexing is as defined in SPAM2.H. Note that index
         zero is never used for anything since joint numbers start at 1 */
      byte C_pw[NUM_JOINTS+1], D_pw[NUM_JOINTS+1];

      byte DATASAVE = FALSE;
      FILE *fp;
      string filename, buffer;
      unsigned int key = 0;
      register i;

      /* Initialize to PWM 7=50% PWM duty for sol C,D pulse */
      for (i=1; i<=NUM_JOINTS; i++) { C_pw[i] = 7; D_pw[i] = 7; }
```

```
     draw_menu (OL_JOINT);
     show_jctrl ("Open Loop");
     show_mode ("Open Loop Joint Control");

     while (key != ESC) {
          synch ();
          update_arm_state ();
          show_joints (arm_current);

          update_tip_state ();
          show_tip (tip_current);

          if (DATASAVE) {
            fprintf (fp, "%.21f,%.21f,%.21f,%.21f\n",
            arm_current.angle.elbow, arm_current.velocity.elbow,
            arm_current.angle.shoulder, arm_current.velocity.shoulder);
            }

          if (gfkbhit ()) {
               switch (key = getkey ()) {
                     case F1 :
                          toggle_sol (joint[ELBOW]->A);
                          break;

                     case F2 :
                          toggle_sol (joint[ELBOW]->B);
                          break;

                     case F3 :
                          toggle_sol (joint[ELBOW]->C);
                          break;

                     case F4 :
                          toggle_sol (joint[ELBOW]->D);
                          break;

                     case F5 :
                          toggle_sol (joint[SHOULDER]->A);
                          break;

                     case F6 :
                          toggle_sol (joint[SHOULDER]->B);
                          break;

                     case F7 :
                          toggle_sol (joint[SHOULDER]->C);
                          break;

                     case F8 :
                          toggle_sol (joint[SHOULDER]->D);
                          break;
/*
 * Reserved for waist joint.
 *                   case F9 :
 *                   case F10 :
 *                   case F11 :
 *                   case F12 :
 */
                     case '1' :
                          zero_angle (ELBOW);
                          break;

                     case '2' :
                          zero_angle (SHOULDER);
                          break;
/*
 * Reserved for waist joint.
 *
 *                   case '3' :
 */
                     case CF3 :
                     case CF4 :
                          get_pw (&C_pw[ELBOW], &D_pw[ELBOW]);
                          draw_menu (OL_JOINT);
                          sprintf (buffer, " ELBOW: C_pw = %2hu / D_pw =
%2hu\n", C_pw[ELBOW], D_pw[ELBOW]);
                          message (buffer);
                          break;

                     case CF7 :
                     case CF8 :
                          get_pw(&C_pw[SHOULDER],&D_pw[SHOULDER]);
                          draw_menu (OL_JOINT);
                          sprintf (buffer, " SHOULDER: C_pw = %2hu / D_pw =
%2hu\n",C_pw[SHOULDER], D_pw[SHOULDER]);
                          message (buffer);
                          break;

/*
 * Reserved for waist joint.
 *                   case CF11 :
 *                   case CF12 :
 */
                     case SF3 :
                          control_sol (joint[ELBOW]->C, C_pw[ELBOW]);
                          break;

                     case SF4 :
                          control_sol (joint[ELBOW]->D, D_pw[ELBOW]);
                          break;

                     case SF7 :
                          control_sol (joint[SHOULDER]->C, C_pw[SHOULDER]);
                          break;

                     case SF8 :
                          control_sol (joint[SHOULDER]->D, D_pw[SHOULDER]);
                          break;

/*
 * Reserved for waist joint.
 *                   case SF11 :
 *                   case SF12 :
 */
                     case '4' :
                          control_sol (joint[ELBOW]->A,SOL_ON);
                          control_sol (joint[ELBOW]->B,SOL_OFF);
                          control_sol (joint[ELBOW]->C,SOL_ON);
                          control_sol (joint[ELBOW]->D,D_pw[ELBOW]);
                          break;
```

```
                case '5' :
                      control_sol (joint[ELBOW]->A,SOL_OFF);
                      control_sol (joint[ELBOW]->B,SOL_ON);
                      control_sol (joint[ELBOW]->C,C_pw[ELBOW]);

                      control_sol (joint[ELBOW]->D,SOL_ON);
                      break;
                case '6' :
                      control_sol (joint[SHOULDER]->A,SOL_ON);
                      control_sol (joint[SHOULDER]->B,SOL_OFF);

                      control_sol (joint[SHOULDER]->C,SOL_ON);
                      control_sol (joint[SHOULDER]->D,D_pw[SHOULDER]);
                      break;
                case '7' :
                      control_sol (joint[SHOULDER]->A, SOL_OFF);

                      control_sol (joint[SHOULDER]->B, SOL_ON);
                      control_sol      (joint[SHOULDER]->C,
C_pw[SHOULDER]);
                      control_sol (joint[SHOULDER]->D, SOL_ON);
                      break;

/*
 * Reserved for waist joint.
 *              case '8' :
 *              case '9' :
 */

                case ' ' :
                      close_all_air ();
                      close_all_water ();
                      break;

                case 's' :
                      if (DATASAVE) {
                          if (close_save (fp, filename))
                                DATASAVE = FALSE;
                      }
                      else {
                          if (start_save (&fp, filename))
                                DATASAVE = TRUE;
                                draw_menu (OL_JOINT);
                        }
                      break;

                case ESC :
                      if (DATASAVE)
                            close_save (fp, filename);
                      close_all_air ();
                      close_all_water ();
                      break;
             }
         }
    }
    show_jctrl ("");
    show_mode ("");
}
```

```
/*
** Closed Loop Tip Position Control.
*/
void CL_Tip (void)
{
      byte DATASAVE = FALSE;
      FILE *fp;
      string filename, buffer;
      unsigned int key = 0;
      double step = 1.0;
      double angle_deadband = 1.0;  /* 1 degree deadband */

      draw_menu (CL_POSITION);
      show_jctrl ("PosServo");
      show_mode ("CL Tip Position Control - OFF");
      CONTROL_ON = FALSE;

      update_arm_state ();
      update_tip_state ();
      tip_command = tip_current;

      while (key != ESC) {
            synch ();
            update_arm_state ();
            show_joints (arm_current);

            update_tip_state ();
            show_tip (tip_current);

            show_command (tip_command);

            arm_command.angle = invkin (tip_command.position);
            PD_pos_servo (angle_deadband);

            if (DATASAVE) {
/*            fprintf (fp, "%.2lf,%.2lf,%.2lf,%.2lf\n",
              arm_current.angle.elbow, arm_current.velocity.elbow,
              arm_current.angle.shoulder, arm_current.velocity.shoulder);
*/
                  fprintf (fp, "%.2lf,%.2lf,%.2lf,%.2lf\n",
                  tip_command.position.x, tip_command.position.z,
                  arm_current.angle.elbow, arm_current.angle.shoulder);
              }

            if (gfkbhit ()) {
                  switch (key = getkey ()) {

                        case CURLF :
                              tip_command.position.x -= step;
                              break;

                        case CURRT :
                              tip_command.position.x += step;
                              break;

/*                      case CURUP :
 *                            tip_command.position.y += step;
 *                            break;
 *
 *                      case CURDN :
```

```
 *
 *
 */
                        tip_command.position.y -= step;
                        break;

                case '+' :
                        tip_command.position.z += step;
                        break;

                case '-' :
                        tip_command.position.z -= step;
                        break;

                case 'i' :
                        if (CONTROL_ON) {
                                CONTROL_ON = FALSE;
                                show_mode  ("CL  Tip  Position  Control  -
OFF");
                        }
                        PD_pos_servo (angle_deadband);
                        step = get_step ();
                        draw_menu (CL_POSITION);
                        sprintf  (buffer,  "  Position  Step  Increment  =
%.2lf\n", step);
                        message (buffer);
                        break;

                case ' ' :
                        arm_command = arm_current;
                        CONTROL_ON = FALSE;
                        show_mode ("CL Tip Position Control - OFF");
                        break;

                case CR :
                        if (CONTROL_ON) {
                                CONTROL_ON = FALSE;
                                show_mode  ("CL  Tip  Position  Control  -
OFF");
                        }
                        else {
                                CONTROL_ON = TRUE;
                                show_mode ("CL Tip Position Control - ON");
                        }
                        break;

                case 's' :
                        if (DATASAVE) {
                            if (close_save (fp, filename))
                                  DATASAVE = FALSE;
                        }
                        else {
                              if (CONTROL_ON) {
                                    CONTROL_ON = FALSE;
                                    show_mode ("CL Tip Position Control -
OFF");
                              }
                              PD_pos_servo (angle_deadband);
                              if(start_save (&fp, filename))
                                  DATASAVE = TRUE;
                                  draw_menu (CL_POSITION);
                          }
                        break;
```

```
                                case ESC :
                                        if (DATASAVE)
                                                close_save (fp, filename);
                                        close_all_air ();
                                        close_all_water ();
                                        break;
                        }
                }
        }
        show_jctrl ("");
        show_mode ("");
}

/*
** Open Loop Resolved Rate Tip Control.
*/
void OL_ResRate (void)
{
        byte DATASAVE = FALSE;
        FILE *fp;
        string filename, buffer;
        unsigned int key = 0;
        double step = 1.0;
        double angle_deadband = 1.0;  /* 1 degree deadband */

        draw_menu (OL_RESRATE);
        show_jctrl ("VelocServo");
        show_mode ("Open Loop Resolved Rate - OFF");
        CONTROL_ON = FALSE;

        update_arm_state ();
        update_tip_state ();
        zero_tip_command ();

        while (key != ESC) {
                synch ();
                update_arm_state ();
                show_joints (arm_current);

                update_tip_state ();
                show_tip (tip_current);

                show_command (tip_command);

                arm_command.velocity = inv_jacobian (tip_command.velocity);

                PD_veloc_servo ();

                if (DATASAVE) {
        fprintf(fp,"%.2lf,%.2lf,%.2lf,%.2lf,%.2lf,%.2lf,%.2lf,%.2lf\n",
                   tip_command.velocity.x, tip_command.velocity.z,
                   tip_current.velocity.x, tip_current.velocity.z,
                   arm_current.angle.elbow, arm_current.angle.shoulder,
                   arm_current.velocity.elbow, arm_current.velocity.shoulder);
                }

                if (gfkbhit ()) {
                    switch (key = getkey ()) {

                        case CURLF :
                                tip_command.velocity.x -= step;
                break;

          case CURRT :
                tip_command.velocity.x += step;
                break;

/*        case CURUP :
 *              tip_command.velocity.y += step;
 *              break;
 *
 *        case CURDN :
 *              tip_command.velocity.y -= step;
 *              break;
 */
          case '+' :
                tip_command.velocity.z += step;
                break;

          case '-' :
                tip_command.velocity.z -= step;
                break;

          case 'i' :
                if (CONTROL_ON) {
                      CONTROL_ON = FALSE;
                      show_mode  ("Open  Loop  Resolved  Rate  -
OFF");
                }
                PD_veloc_servo ();
                step = get_step ();
                draw_menu (OL_RESRATE);
                sprintf  (buffer,  "  Velocity  Step  Increment  =
%.2lf\n", step);
                message (buffer);
                break;

          case ' ' :
                zero_tip_command ();
                CONTROL_ON = FALSE;
                PD_veloc_servo ();
                show_mode ("Open Loop Resolved Rate - OFF");

                break;

          case CR :
                if (CONTROL_ON) {
                      CONTROL_ON = FALSE;
                      show_mode  ("Open  Loop  Resolved  Rate  -
OFF");
                }
                else {
                      CONTROL_ON = TRUE;
                      show_mode ("Open Loop Resolved Rate - ON");
                }
                break;

          case 's' :
                if (DATASAVE) {
                    if (close_save (fp, filename))
                          DATASAVE = FALSE;
                }
```

```
                                          else {
                                                if (CONTROL_ON) {
                                                      CONTROL_ON = FALSE;
                                                      show_mode  ("Open Loop Resolved Rate -
OFF");
                                                }
                                                PD_veloc_servo ();
                                                if(start_save (&fp, filename))
                                                    DATASAVE = TRUE;
                                                    draw_menu (OL_RESRATE);
                                          }
                                          break;

                                    case ESC :
                                          if (DATASAVE)
                                                close_save (fp, filename);
                                          close_all_air ();
                                          close_all_water ();
                                          break;
                              }
                        }
            }
            show_jctrl ("");
            show_mode ("");
}

/*
** Proportional-Derivative Joint Position Servo.
**
** This routine is a single-pass control algorithm. The joint (PWM) command
** is generated by using PD control to minimize position error. Since
** this routine is single-pass, it is meant to be called at regular time
** intervals when CONTROL_ON is TRUE. In fact, it would be useful to make
** the servo be interrupt-driven. Short of that (since we would need a hard-
** ware based clocked interrupt to keep sampling constant), anytime the
** CONTROL_ON flag is set FALSE, THIS ROUTINE MUST IMMEDIATELY BE CALLED.
** This will ensure that the joints are stopped (since we don't want them
** wandering by themselves.
**
** The servo uses DEADBAND as a criteria to determine if control action
** should be used on an individual joint.
*/
void PD_pos_servo (double deadband)
{
      FILE *fp;
      static double error[NUM_JOINTS+1], last_error[NUM_JOINTS+1];
      double command;
      register i;

      if (!CONTROL_ON) {      /* Halt everything if control is off */
            close_all_air ();
            close_all_water ();
            return;
      }

#ifdef DEBUG
      fp = fopen ("posservo.out", "a");
      if (!fp) {
            prompt ("Unable to open POSSERVO.OUT\n");
            exit (0);
      }
```

```
#endif

        error[ELBOW]    =arm_command.angle.elbow-arm_current.angle.elbow;
        error[SHOULDER]=arm_command.angle.shoulder-arm_current.angle.shoulder;

#ifdef DEBUG
        fprintf (fp, "---------------------------\n");
        fprintf (fp, "%7.2lf        %7.2lf\n", arm_command.angle.elbow,
                arm_command.angle.shoulder);
#endif

        for (i=1; i<=NUM_JOINTS; i++) {
                if (fabs(error[i]) > deadband)
                        command = (K1*error[i]-K2*last_error[i]);
                else
                        command = 0;

                last_error[i] = error[i];

#ifdef DEBUG
                fprintf (fp, "%7.2lf %7.2lf", error[i], command);
#endif

                if (!command) {    /* zero command, stop joint */
                        control_sol (joint[i]->A, SOL_OFF);
                        control_sol (joint[i]->B, SOL_OFF);
                        control_sol (joint[i]->C, SOL_OFF);
                        control_sol (joint[i]->D, SOL_OFF);
                }
                else {
                        if (command > 0) { /* close joint */
                                if (command > 15) command = 15;
                                control_sol (joint[i]->A, SOL_OFF);
                                control_sol (joint[i]->B, SOL_ON);
                                control_sol (joint[i]->C, round(command));
                                control_sol (joint[i]->D, SOL_ON);
                        }
                        else { /* open joint */
                                if (command < -15) command = -15;
                                control_sol (joint[i]->A, SOL_ON);
                                control_sol (joint[i]->B, SOL_OFF);
                                control_sol (joint[i]->C, SOL_ON);
                                control_sol (joint[i]->D, round(-command));
                        }
                }
        }
#ifdef DEBUG
        fprintf (fp, "\n");
        fclose (fp);
#endif
}

/*
** Proportional-Derivative Joint Velocity Servo.
**
** This routine is a single-pass control algorithm. The joint (PWM) command
** is generated by using PD control to minimize position error. Since
** this routine is single-pass, it is meant to be called at regular time
** intervals when CONTROL_ON is TRUE. In fact, it would be useful to make
** the servo be interrupt-driven. Short of that (since we would need a hard-
** ware based clocked interrupt to keep sampling constant), anytime the
** CONTROL_ON flag is set FALSE, THIS ROUTINE MUST IMMEDIATELY BE CALLED.
** This will ensure that the joints are stopped (since we don't want them
** wandering by themselves.
**
*/
void PD_veloc_servo (void)
{
        FILE *fp;
        static double error[NUM_JOINTS+1], last_error[NUM_JOINTS+1];
        static double pw[NUM_JOINTS+1];
        static byte dir[NUM_JOINTS+1];
        double command;
        register i;

        if (!CONTROL_ON) {       /* Halt everything if control is off */
                close_all_air ();
                close_all_water ();
                return;
        }

#ifdef DEBUG
        fp = fopen ("velservo.out", "a");
        if (!fp) {
                prompt ("Unable to open VELSERVO.OUT\n");
                exit (0);
        }
#endif

        error[ELBOW]    =arm_command.velocity.elbow-arm_current.velocity.elbow;
        error[SHOULDER]=arm_command.velocity.shoulder-
arm_current.velocity.shoulder;
        dir[ELBOW] = (arm_command.velocity.elbow > 0) ? CLOSE : OPEN;
        dir[SHOULDER] = (arm_command.velocity.shoulder > 0) ? CLOSE : OPEN;

#ifdef DEBUG
        fprintf (fp, "---------------------------\n");
        fprintf (fp, "%7.2lf        %7.2lf\n", arm_command.velocity.elbow,
                arm_command.velocity.shoulder);
#endif

        for (i=1; i<=NUM_JOINTS; i++) {
                command = (K1*error[i]-K2*last_error[i]);

                last_error[i] = error[i];

#ifdef DEBUG
                fprintf (fp, "%7.2lf %7.2lf", error[i], command);
#endif

                pw[i] += command;

                if (pw[i] < 0) pw[i] = 0;
                else if (pw[i] > 15) pw[i] = 15;

                if (dir[i] == CLOSE) { /* close joint */
                        control_sol (joint[i]->A, SOL_OFF);
                        control_sol (joint[i]->B, SOL_ON);
                        control_sol (joint[i]->C, round(pw[i]));
                        control_sol (joint[i]->D, SOL_ON);
                }
                else { /* open joint */
                control_sol (joint[i]->A, SOL_ON);
                control_sol (joint[i]->B, SOL_OFF);
                control_sol (joint[i]->C, SOL_ON);
                control_sol (joint[i]->D, round(pw[i]));
            }
      }

#ifdef DEBUG
      fprintf (fp, "\n");
      fclose (fp);
#endif
}

void zero_tip_command (void)
{
      tip_command.position.x = 0;
      tip_command.position.y = 0;
      tip_command.position.z = 0;
      tip_command.velocity.x = 0;
      tip_command.velocity.y = 0;
      tip_command.velocity.z = 0;
}

/*
** Yes, this is indeed a stupid little routine which could be better
** handled in C++ (i.e. as variable initializations since K1 and K2 do
** not change once they are calculated). It would be possible to read
** K1 and K2 from SPAM.CFG if T was a constant, but since we may want
** to experiment with changing the sampling rate . . .
**
** This routine is called from INIT_SYSTEM in SPAMIO.C as part of the
** general SPAM initialization.
*/
void init_PD_servo (void)
{
#ifdef DEBUG
      FILE *fp;

      fp = fopen ("velservo.out", "a");
      fprintf (fp, "Vd_elbow         Vd_shoulder\n");
      fprintf (fp, "  error command  error command\n");
      fclose (fp);
      fp = fopen ("posservo.out", "a");
      fprintf (fp, "THETAd_elbow     THETAd_shouulder\n");
      fprintf (fp, "  error command  error command\n");
      fclose (fp);
#endif

      K1 = Kp + Kd/T;
      K2 = Kd/T;
}
```

```
/*---------------------------------------------------------------------------

DISPLAY.H

    Author  : Terry Fong
    Created : 08-11-89
    Modified      :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
---------------------------------------------------------------------------*/
void draw_display (void);
void erase_display (void);
void show_command (cartesian_state command);
void show_filename (char *filename, int display);
void show_time (int start);
void show_jcontrol (char *string);
void prompt (char *string);
void message (char *string);
void clear_messages (void);
unsigned int do_menu (int menu);
void draw_menu (int menu);
int get_command (unsigned int selection);
void show_jctrl (char *string);
void show_mode (char *string);
void show_joints (joint_state current);
int start_save (FILE **fp, char *filename);
int close_save (FILE *fp, char *filename);
void show_tip (cartesian_state current);
double get_step (void);
```

```
/*---------------------------------------------------------------------------

DISPLAY.C - Routines to set up the display screen and to display various
            things in various windows.

    Author  : Terry Fong
    Created : 08-11-89
    Modified      :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
---------------------------------------------------------------------------*/

#include <graph.h>
#include <stdio.h>
#include <gnfuns\music.h>
#include <gnfuns\timedate.h>
#include <gnleaf\ibmkeys.h>

#include "spam2.h"
#include "menu.h"

#include "windows.h"    /* window definitions and pointer declarations */

/* local global for controlling sound */
static char play = TRUE;

void show_jctrl (char *string)
{
      clear_window (jctrl_win);
      wprintrc (jctrl_win, 1, 2, string);
}

void show_mode (char *string)
{
      clear_window (mode_win);
      wprintrc (mode_win, 1, 2, string);
}

/*
**  Initialize the video system, draw the user interface display
*/
void draw_display (void)
{
      register i;
      extern joint_state arm_current;
      extern cartesian_state tip_current;

      init_video ();
      show_time (TRUE);
      create_window (title_win, "");
      create_window (menu_win, "");
      create_window (error_win, " MESSAGES ");
      create_window (joint_win, " JOINTS ");
      create_window (mode_win, "CONTROL MODE");
      create_window (jctrl_win, "JOINT CTRL");
      create_window (tip_win, " TIP STATE ");
      create_window (command_win, " COMMAND ");

      wprintf_ctr (title_win, 1, "SPAM 3-DOF Arm Controller v2.0");
      wprintf_ctr (title_win, 2, "MIT Space Systems / LOOP Group");
      wprintrc (joint_win, 1, 2, "°e ");
      wprintrc (joint_win, 2, 2, "°s ");
      wprintrc (joint_win, 3, 2, "°w ");
      wprintrc (joint_win, 4, 2, "We ");
      wprintrc (joint_win, 5, 2, "Ws ");
      wprintrc (joint_win, 6, 2, "Ww ");
      show_joints (arm_current);

      wprintrc (tip_win, 1, 2, "X  ");
      wprintrc (tip_win, 2, 2, "Y  ");
      wprintrc (tip_win, 3, 2, "Z  ");
      wprintrc (tip_win, 4, 2, "Vx ");
      wprintrc (tip_win, 5, 2, "Vy ");
      wprintrc (tip_win, 6, 2, "Vz ");
      show_tip (tip_current);

      wprintrc (command_win, 1, 2, "X    0.00");
      wprintrc (command_win, 2, 2, "Y    0.00");
      wprintrc (command_win, 3, 2, "Z    0.00");
      wprintrc (command_win, 4, 2, "Vx   0.00");
      wprintrc (command_win, 5, 2, "Vy   0.00");
      wprintrc (command_win, 6, 2, "Vz   0.00");

      play = FALSE;
      if (play) {
            sound (NA2, 5);
            sound (560, 5);    /* Middle C# */
            sound (NE2, 5);
            sound (NA3, 10);
      }
}

void show_joints (joint_state current)
{
      string buffer;

      sprintf (buffer, "%7.21f", current.angle.elbow);
      wprintrc (joint_win, 1, 7, buffer);
      sprintf (buffer, "%7.21f", current.angle.shoulder);
      wprintrc (joint_win, 2, 7, buffer);
      sprintf (buffer, "%7.21f", current.angle.waist);
      wprintrc (joint_win, 3, 7, buffer);
      sprintf (buffer, "%7.21f", current.velocity.elbow);
      wprintrc (joint_win, 4, 7, buffer);
      sprintf (buffer, "%7.21f", current.velocity.shoulder);
      wprintrc (joint_win, 5, 7, buffer);
      sprintf (buffer, "%7.21f", current.velocity.waist);
      wprintrc (joint_win, 6, 7, buffer);
}

void show_tip (cartesian_state current)
{
      string buffer;

      sprintf (buffer, "%7.21f", current.position.x);
      wprintrc (tip_win, 1, 6, buffer);
      sprintf (buffer, "%7.21f", current.position.y);
      wprintrc (tip_win, 2, 6, buffer);
      sprintf (buffer, "%7.21f", current.position.z);
      wprintrc (tip_win, 3, 6, buffer);
      sprintf (buffer, "%7.21f", current.velocity.x);
```

```
     wprintrc (tip_win, 4, 6, buffer);
     sprintf (buffer, "%7.2lf", current.velocity.y);
     wprintrc (tip_win, 5, 6, buffer);
     sprintf (buffer, "%7.2lf", current.velocity.z);
     wprintrc (tip_win, 6, 6, buffer);
}

void show_command (cartesian_state command)
{
     string buffer;

     sprintf (buffer, "%7.2lf", command.position.x);
     wprintrc (command_win, 1, 6, buffer);
     sprintf (buffer, "%7.2lf", command.position.y);
     wprintrc (command_win, 2, 6, buffer);
     sprintf (buffer, "%7.2lf", command.position.z);
     wprintrc (command_win, 3, 6, buffer);
     sprintf (buffer, "%7.2lf", command.velocity.x);
     wprintrc (command_win, 4, 6, buffer);
     sprintf (buffer, "%7.2lf", command.velocity.y);
     wprintrc (command_win, 5, 6, buffer);
     sprintf (buffer, "%7.2lf", command.velocity.z);
     wprintrc (command_win, 6, 6, buffer);
}

/*---------------------------------------------------------------------------
  Print a message in the error window and beep. Wait for a user keypress.
---------------------------------------------------------------------------*/
void prompt (char *string)
{
     extern window_ptr error_win;

     clear_window (error_win);
     wprintf (error_win, string);
     error_win->color.text += 16; /* set blinking */
     wprintf (error_win, " Press any key to continue ...\n");

     sound (NA2, 10);
     while (!kbhit ());
     getch ();

     error_win->color.text -= 16;
     error_win->curpos.row--;
     wprintf (error_win, "                                  \n");
}

/*---------------------------------------------------------------------------
  Display a message in the error window
---------------------------------------------------------------------------*/
void message (char *string)
{
     extern window_ptr error_win;

     wprintf (error_win, string);
}

/*
** Clear all messages.
*/
void clear_messages (void)
{
     extern window_ptr error_win;
     clear_window (error_win);
}

/*
**  Clear the display and display exit time. Call immediately before exiting.
*/
void erase_display (void)
{
     restore_video ();
     show_time (FALSE);
}

/*
**  Display name of an open datasave file on the bottom display line.
*/
void show_filename (char *filename, int display)
{
     string buffer;

     _settextwindow (1, 1, 25, 79);
     _setbkcolor ((long) BLUE);
     _settextcolor (LTGREY);
     _settextposition (25, 50);

     if (display) {
          sprintf (buffer, "Data saved to: %s", strupr(filename));
          _outtext (buffer);
     }
     else
          _outtext("                              ");
}

/*
**  Time stamp the display. If start is TRUE, then print "started on", else
**  print "exited at" on bottom display line.
*/
void show_time (int start)
{
     struct TIMEDATE *ptd, *sgettime();
     string buffer;

     ptd = sgettime (8);
     if (start)
          sprintf (buffer, "SPAM v2.0 started on: %s", ptd->dateline);
     else
          sprintf (buffer, "SPAM v2.0 exited at: %s", ptd->dateline);

     _settextwindow (1, 1, 25, 79);
     _setbkcolor ((long) BLUE);
     _settextcolor (LTGREY);
     _settextposition (25, 2);
     _outtext (buffer);
}

/*
** Display menus to the user and scan for keypress. The Greenleaf Comm func-
** tion used here (getkey()) returns the key as an unsigned int (so that
** extended code keys, such as cursors and function keys, can be trapped too).
** Refer to IBMKEYS.H for list of recognized key codes.
**
```

```
** Note: The key is returned as an ASCII value (i.e. pressing the '1' key
** returns a '1' char = 49 decimal).
*/
unsigned int do_menu (int menu)
{
        draw_menu (menu);

        while (!gfkbhit ());

        return (getkey());
}

/*
** Clear TheWindow and display a menu. Menu index is contained in MENU.H
*/
void draw_menu (int menu)
{
        string buffer;
        window_ptr TheWindow = menu_win;

        clear_window (TheWindow);

        switch (menu) {

                case TOP_LEVEL :
                        wprintf (TheWindow, "  F1 - About SPAM v2.0..\n\n");
                        wprintf (TheWindow, "   1 - OL Joint Control\n");
                        wprintf (TheWindow, "   2 - CL Position Control\n");
                        wprintf (TheWindow, "   3 - OL Resolved Rate\n");
                        wprintf (TheWindow, "   4 - CL Resolved Rate\n\n");
                        wprintf (TheWindow, " ESC - Exit program.\n");
                        break;

                case CL_RESRATE_1 :
                        wprintf (TheWindow, "   1 - Enter targets\n");
                        wprintf (TheWindow, "   2 - Run CL Control\n\n");
                        wprintf (TheWindow, " ESC - Quit\n");
                        break;

                case CL_RESRATE_2 :
                        wprintf (TheWindow, "   1 - Enter X_d\n");
                        wprintf (TheWindow, "   2 - Enter Y_d\n");
                        wprintf (TheWindow, "   3 - Enter Z_d\n");
                        wprintf (TheWindow, "   4 - Enter Vx_d\n");
                        wprintf (TheWindow, "   5 - Enter Vy_d\n");
                        wprintf (TheWindow, "   6 - Enter Vz_d\n\n");
                        wprintf (TheWindow, " ESC - Quit\n");
                        break;

                case ABOUT_SPAM :
                        wprintf_ctr (TheWindow, 1, "SPAM v2.0");
                        wprintf_ctr (TheWindow, 2, "3-DOF Arm Controller");
                        wprintf_ctr (TheWindow, 4, "written by");
                        wprintf_ctr (TheWindow, 5, "Terry Fong");
                        wprintf_ctr (TheWindow, 8, " (C) 1990 by Blahware, Inc.");
                        wprintf_ctr (TheWindow, 9, "Blahburgh, USA");
                        break;

                case OL_JOINT :
                        wprintf (TheWindow, " F1..F12  Toggle A1..D3\n");
                        wprintf (TheWindow, " Shift F  Pulse C,D 1-3\n");
```

```
                        wprintf (TheWindow, " Ctrl  F  Set C,D 1-3 pw\n");
                        wprintf (TheWindow, "   1..3   Reset ENC 1-3\n");
                        wprintf (TheWindow, "   4..9   Open/Close 1-3\n");
                        wprintf (TheWindow, " <SPACE>  CLOSE ALL SOLS\n");
                        wprintf (TheWindow, "    S     Toggle Save\n\n");
                        wprintf (TheWindow, "  <ESC>   Exit Open Loop");
                        break;

                case CL_POSITION :
                        /* yessssss--A STUPID DOS LIMITATION!!!! You
                                simply CANNOT include an ESC char in a
                                string because DOS thinks that it is the
                                EOF marker!!!! Arrrrgh !!!*/
                        sprintf (buffer, "      \x1a    +/- Tip X_d\n");
                        wprintf (TheWindow, buffer);
                        wprintf (TheWindow, "              +/- Tip Y_d\n");
                        wprintf (TheWindow, "    + -       +/- Tip Z_d\n");
                        wprintf (TheWindow, "     i        Set step inc.\n");
                        wprintf (TheWindow, " <ENTER>  Toggle control\n");
                        wprintf (TheWindow, " <SPACE>  Command=Current\n");
                        wprintf (TheWindow, "    S     Toggle Save\n\n");
                        wprintf (TheWindow, "  <ESC>   Exit Open Loop");
                        break;

                case OL_RESRATE :
                        sprintf (buffer, "      \x1a    +/- Tip Vx_d\n");
                        wprintf (TheWindow, buffer);
                        wprintf (TheWindow, "              +/- Tip Vy_d\n");
                        wprintf (TheWindow, "    + -       +/- Tip Vz_d\n");
                        wprintf (TheWindow, "     i        Set step inc.\n");
                        wprintf (TheWindow, " <ENTER>  Toggle control\n");
                        wprintf (TheWindow, " <SPACE>  Zero Velocities\n");
                        wprintf (TheWindow, "    S     Toggle Save\n\n");
                        wprintf (TheWindow, "  <ESC>   Exit Open Loop");
                        break;
        }
}

/*
** Input Solenoid pulse width
*/
void get_pw (byte *C_pw, byte *D_pw)
{
        window_ptr TheWindow = menu_win;

        clear_window (TheWindow);
        cursor_on ();
        wprintf (TheWindow, " 'C' pulse width: ");
        TheWindow->curpos.row++;
        TheWindow->curpos.col = 1;
        scanf ("%uh", C_pw);
        wprintf (TheWindow, " 'D' pulse width: ");
        TheWindow->curpos.row++;
        TheWindow->curpos.col = 1;
        scanf ("%uh",D_pw);
        cursor_off ();
}

/*
** Input step size.
*/
```

```
double get_step (void)
{
      window_ptr TheWindow = menu_win;

      clear_window (TheWindow);
      return (input (TheWindow, " Enter Step Size: "));
}

/*
** Input state command. TRUE is returned if a command has been entered or if
** ESC has been pressed. Anything else returns a FALSE
*/
int get_command (unsigned int selection)
{
      window_ptr TheWindow = menu_win;
      extern cartesian_state tip_command;

      clear_window (TheWindow);

      switch (selection) {

            case '1' :
                  tip_command.position.x =
                        input (TheWindow, " Enter x [feet]: ");
                  break;

            case '2' :
                  tip_command.position.y =
                        input (TheWindow, " Enter y [feet]: ");
                  break;

            case '3' :
                  tip_command.position.z =
                        input (TheWindow, " Enter z [feet]: ");
                  break;

            case '4' :
                  tip_command.velocity.x =
                        input (TheWindow, " Enter x velocity: ");
                  break;

            case '5' :
                  tip_command.velocity.y =
                        input (TheWindow, " Enter y velocity: ");
                  break;

            case '6' :
                  tip_command.velocity.z =
                        input (TheWindow, " Enter z velocity: ");
                  break;

            case ESC :
                  break;

            default :
                  return (FALSE);
      }
      return (TRUE);
}

/*---------------------------------------------------------------------------
  Open a file for data output. Returns a TRUE if file is opened successfully.
---------------------------------------------------------------------------*/
int start_save (FILE **fp, char *filename)
{
      string buffer;
      struct TIMEDATE *ptd, *sgettime();
      window_ptr TheWindow = menu_win;

      clear_window (TheWindow);
      wprintf (TheWindow, " Filename [.PRN]: ");
      TheWindow->curpos.row++;
      TheWindow->curpos.col = 1;
      cursor_on ();
      scanf ("%s", filename);
      cursor_off ();

      strcat (filename, ".prn");

      if ((*fp=fopen(filename, "a+"))==NULL) {
            prompt (" ERROR: Cannot open file !!!!!\n");
            return (FALSE);
      }
      else {
            sprintf (buffer, " %s opened for data savc.\n",
                  strupr(filename));
            message (buffer);
            show_filename (filename, TRUE);

            ptd = sgettime (8);
            fprintf (*fp, "%s\n\n", ptd->dateline);
            fprintf (*fp, "SPAM v2.0 Data Save File\n");
            fprintf (*fp, "------------------------\n");

            return (TRUE);
      }
}

/*---------------------------------------------------------------------------
  Close a data save file
---------------------------------------------------------------------------*/
int close_save (FILE *fp, char *filename)
{
      string buffer;

      if (fclose (fp)==NULL) {
            sprintf(buffer, " %s closed.\n", strupr(filename));
            prompt (buffer);
            show_filename ("", FALSE);
            return (TRUE);
      }
      else {
            sprintf (buffer, " ERROR: Cannot close %s !!!!!\n",
                  strupr(filename));
            prompt (buffer);
            return (FALSE);
      }
}

/*------------------------ End of DISPLAY.C ------------------------------*/
```

```
/*-----------------------------------------------------------------------------

MENU.H - Menu level definitions.

    Author   :  Terry Fong
    Created  :  08-11-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-----------------------------------------------------------------------------*/

#define TOP_LEVEL       0
#define CL_RESRATE_1    1
#define     CL_RESRATE_2      2
#define ABOUT_SPAM      4
#define OL_JOINT  5
#define CL_POSITION     6
#define OL_RESRATE      7
```

```
/*-----------------------------------------------------------------------------

PORTDEFS.H - Prototype Expansion Board I/O Port definitions

    Author   :  Terry Fong
    Created  :  05-04-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-----------------------------------------------------------------------------*/

#define CTRL_PORT1      0x3E3 /* 8255-1 port definitions */
#define PORT_C1         0x3E2
#define PORT_B1         0x3E1
#define PORT_A1         0x3E0

#define CTRL_PORT2      0X3E7 /* 8255-2 port definitions */
#define PORT_C2         0x3E6
#define PORT_B2         0x3E5
#define PORT_A2         0x3E4
```

```
7.0   7.0   1.0   0.1
L1    L2    Kp    Kd

Spam v2.0 Configuration file.
Only numbers in the first line are read.
/*------------------------------End of SPAM.CFG-------------------------*/
```

```
/*
**
** SPAM2.H - 3 DOF Manipulator OL/CL Controller
**
**    Author  : Terry Fong
**    Created    : 03-13-90
**    Modified:
**
** MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
*/

#include <gnleaf\gf.h>
#include <stdio.h>

#define sq(x)     (x*x)

/*------------------------- Some cool constants ---------------------------*/
#define PI        3.1415926
#define DEG2RAD   PI/180     /* By convention: ALL VARS are in DEGREES ! */
#define RAD2DEG   180/PI

#define ELBOW         1      /* see FWDKIN note in ARM.C */
#define SHOULDER  2
#define WAIST         3
#define TIP_X         4
#define TIP_Y         5
#define TIP_Z         6

#define NUM_JOINTS    2      /* 2 link arm for now !!! */

/*--------------------- Define and typedef some vars ----------------------*/
typedef struct {        /* Here is a vector in position space */
      double x;
      double y;
      double z;
} pos_vector;

typedef struct {        /* Here is a vector in orientation space */
      double roll;
      double pitch;
      double yaw;
} ang_vector;

typedef struct {        /* A vector in joint space. */
      double elbow;
      double shoulder;
      double waist;
} joint_vector;

typedef pos_vector vector;    /* A "normal" vector in 3 space. This is just
                                a matter of semantics since POS_VECTOR was
                                defined to be symmetric to ANG_VECTOR. */

typedef struct {        /* A 6-DOF location is called a "frame" */
      vector position;
      ang_vector angle;
} frame;

typedef struct {        /* cartesian state vector has 6 states */
      vector position;  /* X, Y, Z, Vx, Vy, Vz */
      vector velocity;
} cartesian_state;

typedef struct {
      joint_vector angle;
      joint_vector velocity;
} joint_state;

typedef char string[80];
typedef unsigned char byte;

/*--------------------------- Function declarations -------------------------*/

/*** DISPLAY.C ***/
#include "display.h"

/*** CONTROL.C ***/
#include "control.h"

/*** ARM.C ***/
#include "arm.h"

/*** VIDEO.C ***/
#include "video.h"

/*** SPAMIO.H ***/
#include "spamio.h"
/*---------------------------- End of SPAM2.H --------------------------------*/
```

```
/*---------------------------------------------------------------------------------

SPAM2.C - 2-link planar arm control software

    Author   :  Terry Fong
    Created  :  05-04-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------------*/
#include <stdio.h>
#include <gnleaf\ibmkeys.h>

#include "spam2.h"
#include "menu.h"

/*-------------------------- Global variables ------------------------------*/
cartesian_state tip_current = {0, 0, 0, 0, 0, 0};
cartesian_state tip_command = {0, 0, 0 ,0, 0, 0};
joint_state arm_current = {0, 0, 0, 0, 0, 0};
joint_state arm_command = {0, 0, 0, 0, 0, 0};
double L1 = 0, L2 = 0;

/*-------------------------- End of Global Variables -----------------------*/

int (*jcontrol) ();     /* function pointer to low level joint controller */

main ()
{
      init_system ();
      draw_display ();
      while (dispatch ());
      erase_display ();
}

/*
** Dispatch commands from the TOPLEVEL menu. Return FALSE if users wishes to
** exit the program.
*/
int dispatch ()
{
      int menu;

      clear_messages ();
      message ("\n Top Level: Select control method or quit.\n");
      switch (do_menu (TOP_LEVEL)) {
            case '1' : /* OL Joint Control */
                  clear_messages ();
                  OL_Joint ();
                  break;

            case '2' : /* CL Tip Control */
                  clear_messages ();
                  CL_Tip ();
                  break;

            case '3' : /* OL Resolved Rate */
                  clear_messages ();
                  OL_ResRate ();
                  break;
```

```
        case '4' :
            menu = CL_RESRATE_1;
            while (menu == CL_RESRATE_1) {
                clear_messages ();
                message ("\n Specify end target or run CL control");
                switch (do_menu (menu)) {
                    case '1' :
                        menu = CL_RESRATE_2;
                        while (menu == CL_RESRATE_2) {
                            clear_messages ();
                            message ("\n Select target state to
change.\n");

                            if (get_command (do_menu(menu))){
                                menu = CL_RESRATE_1;
                                show_command (tip_command);
                            }
                        }
                        break;

                    case '2' :
                        clear_messages ();
                        message ("\n CL control\n");
                        /* CL_ResRate (); */
                        break;

                    case ESC :
                        menu = TOP_LEVEL;
                        break;
                }
            }
            break;


        case F1 :
            clear_messages ();
            message ("\n Press a key to continue...\n");
            do_menu (ABOUT_SPAM);
            break;

        case ESC :
            return (FALSE);
    }
    return (TRUE);
}
```

```
/*---------------------------------------------------------------------------

SPAMIO.H - Define the Super Spiffy Solenoid Object and various things about
           those wondefully wacky solenoids.

    Author   :  Terry Fong
    Created  :  07-23-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

typedef struct solenoid {
      byte on_code;             /* Air sols have a single ON code */
      byte off_code;            /*  "   "    "    "    "  OFF  "  */
      byte pw;          /* Water sols have 16 pwm levels */
      byte nybble;              /* Water sols wired either UPPER or LOWER */
      unsigned int port;        /* Port to send output codes to */
      struct solenoid *buddy; /* Pointer to link up solenoids. Only water */
                          /*   sols have a buddy, air sols point NULL */
} solenoid;

typedef struct {
      solenoid *A;
      solenoid *B;
      solenoid *C;
      solenoid *D;
} joint_sols;

#define NULL       0
#define SOL_ON  15       /* for a water solenoid, this is 100% duty */
#define SOL_OFF    0

#define UPPER      1
#define LOWER      0

#define UPPER_MASK 0x0f
#define LOWER_MASK 0xf0

#define OPEN       1
#define CLOSE      0

#define INPUT      1
#define OUTPUT     0

#define NUM_SOLS   4

/*** SPAMIO.C ***/
void control_sol (solenoid *sol, byte code);
void close_all (solenoid sols[NUM_SOLS]);
void link_water ();
void toggle_sol (solenoid *sol);
double read_angle (int sensor);
void init_sols ();
void init_joints ();
void close_all_water (void);
void close_all_air (void);
void init_system ();
void zero_angle (int sensor);
void synch (void);
void calib_T (void);
void read_cfg (void);
void set8255 (unsigned int port, int a, int b, int cl, int ch);
```

```
/*-----------------------------------------------------------------------------

SPAMIO2.C - Hardware I/O routines. Control solenoids, read encoder.
          Version 2 since v.1 only handled 4 solenoids!!!

    Author   :  Terry Fong
    Created  :  12-7-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-----------------------------------------------------------------------------*/
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <sys\timeb.h>

#include "portdefs.h"
#include "spam2.h"

#define CAL_ITERS 30    /* number of loops for calib_T */

/*---------------------- Module global variables ---------------------------*/

/* Solenoid groups are defined as arrays of structures so that all solenoids
   in a group (i.e. air, water) can "efficiently" be operated at once. This
   is important since we may want to quickly stop all arm motion by closing
   all solenoids. Caution! see LINK_WATER note!!! */
static solenoid air[NUM_SOLS] =
      {5,4,0,0,CTRL_PORT2, NULL,    /* A1 */
       7,6,0,0,CTRL_PORT2, NULL,    /* B1 */
       1,0,0,0,CTRL_PORT2, NULL,    /* A2 */
       3,2,0,0,CTRL_PORT2, NULL};   /* B2 */

static solenoid water[NUM_SOLS] =
      {0,0,0,LOWER,PORT_B2, NULL,   /* C1 */
       0,0,0,UPPER,PORT_B2, NULL,   /* D1 */
       0,0,0,LOWER,PORT_A2, NULL,   /* C2 */
       0,0,0,UPPER,PORT_A2, NULL};  /* D2 */

/*------------------------- Global variables -------------------------------*/
extern double L1, L2;   /* SPAM2.C */
extern double Kp, Kd;   /* CONTROL.C */

/* sampling period (actually this is the PWM period) */
double T = 0;

/* joint objects are groupings of solenoids */
joint_sols elbow_sols = {&air[0], &air[1], &water[0], &water[1]};
joint_sols shoulder_sols = {&air[2], &air[3], &water[2], &water[3]};

/*---------------------- End of variable definitions ----------------------*/

/*
** Solenoid control. The air solenoids are wired individually and only have
** 2 states (on/off). The water solenoids are wired in "buddy" pairs and have
** 16 states (on/off and 14 PWM levels). To determine what kind of solenoid
** has been passed, a check is made of sol->buddy. If this is NULL, then we
** are dealing with an air solenoid, else we have a water solenoid.
**
** Control "code" definition and notes
**      SOL_OFF = 0          Shuts off any solenoid
**      SOL_ON  = 15          '15' is 100% duty for a water solenoid. The
**                       following code uses '15' to turn on a water
**                       solenoid. The actual value is ignored when
**                       turning on an air solenoid.
**      code = [1..14]  Specifies PWM duty cycle. 1 is approx. 1/15
**                       duty (6.7%) and 14 is approx. 14/15 duty (93%)
*/
void control_sol (solenoid *sol, byte code)
{
      if (sol->buddy == NULL)
            switch (code) { /* switch is used to ensure that only ON and
                          OFF are valid Koganei codes, all other
                          codes "fall off the end of the switch" */

                  case SOL_ON:
                        outp (sol->port, sol->on_code);
                        sol->pw = SOL_ON;
                        break;

                  case SOL_OFF:
                        outp (sol->port, sol->off_code);
                        sol->pw = SOL_OFF;
                        break;
            }
      else {      /* Here we deal with a water solenoid. If it is wired to the
               upper nybble of a 8255 port, we must shift the 4-bit
               CODE to the upper nybble of SOL->PW. The actual value
               output to the 8255 port is created by combining the PW
               values of both solenoids hooked to the port. The mask of
               of 0x0f ensures that we only have a 4-bit CODE value */

            if (sol->nybble == UPPER)
                  sol->pw = ((code & 0x0f) << 4);
            else
                  sol->pw = (code & 0x0f);

            outp (sol->port, (sol->pw | sol->buddy->pw));
      }
}

/*
** Initialize the solenoids by "linking" kips and closing all valves.
*/
void init_sols ()
{
      link_water ();
      close_all_water ();
      close_all_air ();
}

/*
** Close all water solenoids.
*/
void close_all_water (void)
{
      close_all (water);
}

/*
** Close all air solenoids.
*/
void close_all_air (void)
{
      close_all (air);
}

/*
** Close an entire set of solenoids.
*/
void close_all (solenoid sols[NUM_SOLS])
{
      register i;

      for (i=0; i<NUM_SOLS; i++) control_sol (&sols[i], SOL_OFF);
}

/*
** Connect the buddy pairs of water solenoids.
**
** CAUTION: This routine assumes that the array of water solenoids has
**          been defined with "buddy pairs" of solenoids located on even
**          numbered array boundaries (i.e. 0 and 1 are a pair, 2 and 3
**          are a pair, etc.).
*/
void link_water ()
{
      register i;

      for (i=0; i<NUM_SOLS-1; i=i+2) {
            water[i].buddy   = &water[i+1];
            water[i+1].buddy = &water[i];
      }
}

/*
** Toggle a solenoid. if (sol->buddy == NULL) then air solenoid, else water.
*/
void toggle_sol (solenoid *sol)
{
      if (sol->buddy == NULL) {
            if (sol->pw)
                  control_sol (sol, SOL_OFF);
            else
                  control_sol (sol, SOL_ON);
      }
      else {
            if (sol->nybble == UPPER) {
                  if (sol->pw & LOWER_MASK) /* zero lower nybble */
                        control_sol (sol, SOL_OFF);
                  else
                        control_sol (sol, SOL_ON);
            }
            else {
                  if (sol->pw & UPPER_MASK) /* zero lower nybble */
                        control_sol (sol, SOL_OFF);
                  else
                        control_sol (sol, SOL_ON);
            }
      }
}
```

```
/*
** Initialize the 8255's, zero the HCTL-2000 counters, close all solenoids.
*/
void init_system ()
{
      calib_T ();
      read_cfg ();
      set8255 (CTRL_PORT1, INPUT, OUTPUT, OUTPUT, OUTPUT);
      set8255 (CTRL_PORT2, OUTPUT, OUTPUT, OUTPUT, INPUT);
      read_angle (SHOULDER);   /* make sure all encoder OE* are high by */
      read_angle (ELBOW);      /*    calling READ_ANGLE for all joints  */
      zero_angle (ELBOW);      /* reset encoders by calling ZERO_ENCODER */
      zero_angle (SHOULDER);   /*    for all joints */
      init_sols ();            /* make "buddy pairs" & close all sols */
      init_PD_servo ();  /* calculate PD parameters */
      init_backdiff ();  /* initialize backward difference routines */
}

/*
** Set the zero position for the optical encoder by sending a low/high pulse
** to the HCTL RST* (pin 5). The reset triggers on the pulse rising edge.
*/
void zero_angle (int sensor)
{
      switch (sensor) {
            case ELBOW :
                  outp (CTRL_PORT1, 0xA); /* reset 8255-1 PC5 */
                  outp (CTRL_PORT1, 0xB); /* set   8255-1 PC5 */
                  break;
            case SHOULDER :
                  outp (CTRL_PORT1, 0x4); /* reset 8255-1 PC2 */
                  outp (CTRL_PORT1, 0x5); /* set   8255-1 PC2 */
                  break;

            case WAIST :
                  break;
      }
}

/*
** Wait for a PWM synch pulse at start of duty cycle. The PWM driver clock
** (a 555 circuit with two 47K resistors and a 10 mFD capacitor) synchs every
** 110 millisecs (approx)., so that we have a 10 Hz synch rate.
*/
void synch (void)
{
      /* wait for synch line to be set */
      while (!(inp(PORT_C2) & 0x10)); /* mask to check only bit 0 */

      /* The following waits for the falling edge */
      while ((inp(PORT_C2) & 0x10));
}

/*
** Read the 12 bit optical encoder position output through the HCTL-2000.
** The HCTL must be controlled as:
**
**       (1) OE* line --> low
**       (2) SEL line --> low
**       (3) grab MSB from data lines
**       (4) SEL line --> high
**       (5) grab LSB from data lines
**       (6) OE* line --> high
**
** If step (6) is NOT done, the HCTL inhibit logic will not reset and the data
** on the data lines will remain unchanged !!!
*/
double read_angle (int sensor)
{
      register int lb, hb;
      double phi_e, phi, theta;

      switch (sensor) {
            case ELBOW :
                  outp (CTRL_PORT1, 0xE); /* pc7 --> OE* = reset */
                  outp (CTRL_PORT1, 0xC); /* pc6 --> SEL = reset */
                  hb = inp (PORT_A1);     /* PORT_A1 --> data lines */
                  outp (CTRL_PORT1, 0xD); /* pc6 --> SEL = set */
                  lb = inp (PORT_A1);     /* PORT_A1 --> data lines */
                  outp (CTRL_PORT1, 0xF); /* pc7 --> OE* = set */
                  break;

            case SHOULDER :
                  outp (CTRL_PORT1, 0x8); /* pc4 --> OE* = reset */
                  outp (CTRL_PORT1, 0x6); /* pc3 --> SEL = reset */
                  hb = inp (PORT_A1);     /* PORT_A1 --> data lines */
                  outp (CTRL_PORT1, 0x7); /* pc3 --> SEL = high */
                  lb = inp (PORT_A1);     /* PORT_A1 --> data lines */
                  outp (CTRL_PORT1, 0x9); /* pc7 --> OE* = set */
                  break;

            case WAIST :
                  break;
      }

      /* calculate position 12-bit position (stored in a 16-bit word) */
      phi_e = (double) (hb*256 + lb) / 22.7555556;
      phi = (phi_e + 15.15) * 0.017453;   /* in radians */

      /* 180/pi*2 = 114.592 */
      theta = 180 - 114.592*atan(6.216*sin(phi)/(6-6.216*cos(phi)));

      return (theta);

      /* 12-bit range is -2048 to +2048 */
      /* if (hb > 7) pos -= 4096; */
      /* return the position in degrees:
         (2048 counts) * (4 quadrature) / (360 degrees) = 22.755556 */
}

/*
** Calculate Sampling Period (T) by comparing to the system clock.
** The result is stored in the module global variable "T".
*/
void calib_T (void)
{
      struct timeb start, end;
      double start_time, end_time;
      register int i;

      printf ("\n\n Please wait...testing sampling interval!!!\n");
      printf (" Press any key to stop calibration...\n\t");
```

```
	for (i=0; i<CAL_ITERS; i++) printf(".");
	for (i=0; i<CAL_ITERS; i++) printf("\b");

	ftime (&start);   /* grab start time from system clock */
	for (i=0; i<CAL_ITERS; i++) { /* average over CAL_ITERS synchs */
		synch ();
		printf (" ");
		if (kbhit()) {
			getch ();
			break;
		}
	}
	ftime (&end); /* grab end time from system clock */

	/* convert structure time to decimal seconds */
	start_time= (double) start.time + (double) start.millitm/1000;
	end_time= (double) end.time + (double) end.millitm/1000;

	T = (end_time-start_time) / i;

	printf ("\n\n\t Sampling Interval (T) = %5.3lf seconds\n", T);
	printf ("\t Pulse Width Mod. Freq = %5.3lf Hertz\n", 1/T);
	printf ("\n Press any key to continue.\n\n\n");
	while (!kbhit());
	getch ();
}

void read_cfg (void)
{
	FILE *fp;

	if (!(fp = fopen ("spam.cfg", "r"))) {
		prompt (" ERROR: Could not read SPAM.CFG !!!\n");
		exit (0);
	}

	fscanf (fp, "%lf %lf %lf %lf", &L1, &L2, &Kp, &Kd);
}

/*
** This routine calculates the 8-bit control word to setup the ports of
** an 8255 for Mode 0 operation (Basic Input/Output). The control word
** is output to the specified I/O port.
**
** The flags (a, b, cl, ch) should be 0 for output, 1 for input.
*/
void set8255 (unsigned int port, int a, int b, int cl, int ch)
{
	int word = 0x80;  /* Mode set flag active */

	if (a)  word |= 0x10;
	if (b)  word |= 0x02;
	if (cl) word |= 0x01;
	if (ch) word |= 0x08;

	outp (port, word);
}

/*-------------------------------End of SPAMIO.C------------------------------*/
```

```
/*----------------------------------------------------------------------------

VIDEO.H - Yo! It's FONGMASTER's graphics stuff !!!

   Author   :  Terry Fong
   Created  :  08-03-89
   Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
----------------------------------------------------------------------------*/

#include <gnfuns\color.h>
#include <graph.h>

/*--------------------------- define a window structure ----------------------*/
typedef struct {
	struct rccoord top;
	struct rccoord bottom;
	struct rccoord curpos;
	struct {
		short background;
		short border;
		short text;
	} color;
} window, *window_ptr;

/*------------------------- Function Prototypes -----------------------------*/

void init_video (void);
void restore_video (void);
void create_window (window_ptr x, char *string);
void clear_window (window_ptr x);
void wprintrc (window_ptr x, short row, short col, char *string);
void wprintf (window_ptr x, char *string);
void wprintf_ctr (window_ptr x, short row, char *string);
void cursor_off (void);
void cursor_on (void);
double input (window_ptr TheWindow, char *string);

/*---------------------------- End of VIDEO.H -------------------------------*/
```

```
/*---------------------------------------------------------------------------

VIDEO.C - IBM PC Video and color text screen graphic primitives

    Author   :  Terry Fong
    Created  :  07-31-89
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
---------------------------------------------------------------------------*/

#include <dos.h>
#include <stdio.h>
#include <string.h>
#include "video.h"

/*---------------------------------------------------------------------------
  Initialize the video to 80 column color, turn wrapping off.
---------------------------------------------------------------------------*/
void init_video ()
{
      _setvideomode (_TEXTC80);
      _setbkcolor ((long) BLUE);
      _clearscreen (_GCLEARSCREEN);
      _wrapon (_GWRAPOFF);
      cursor_off ();
}

/*
** Turn the cursor back on, clear the display.
*/
void restore_video ()
{
      _setvideomode (_TEXTC80);
      _setbkcolor ((long) BLUE);
      _clearscreen (_GCLEARSCREEN);
      cursor_on ();
}

/*---------------------------------------------------------------------------
  Input a floating point number
---------------------------------------------------------------------------*/
double input (window_ptr TheWindow, char *string)
{
      float result;

      wprintf (TheWindow, string);
      TheWindow->curpos.row++;
      TheWindow->curpos.col = 1;
      cursor_on ();
      scanf ("%f", &result);
      cursor_off ();

      return (result);
}

/*---------------------------------------------------------------------------
  Clear a specific window and fill with window background color. The window
  cursor is "homed" to (1, 1).
---------------------------------------------------------------------------*/
void clear_window (window_ptr x)
{
      _settextwindow (x->top.row+1, x->top.col+1,
                      x->bottom.row-1, x->bottom.col-1);
      _setbkcolor ((long) x->color.background);
      _clearscreen (_GWINDOW);
      x->curpos.row = 1;
      x->curpos.col = 1;
}

/*---------------------------------------------------------------------------
  Create a window on the screen by drawing a border and filling with color
  if NAME is not NULL, print a centered title on the upper border line.
---------------------------------------------------------------------------*/
void create_window (window_ptr x, char *name)
{
      char  top[81], middle[81], bottom[81];    /* Max box is 80 char wide */
      register i;
      long  old_back;
      short old_text;

      /* Save the current colors before messing with them */
      old_back = _getbkcolor ();
      old_text = _gettextcolor ();

      /* Compile the strings for the box */
      top [0] = '●';          /* char 201 */
      middle [0] = '●';       /* char 186 */
      bottom [0] = '●';       /* char 200 */
      for (i = 1; i < (x->bottom.col - x->top.col); i++) {
            top [i] = '●';          /* char 205 */
            middle [i] = ' '; /* char 32  */
            bottom [i] = '●';       /* char 205 */
      }
      top [i] = '●';          /* char 187 */
      middle [i] = '●';       /* char 186 */
      bottom [i] = '●';       /* char 188 */

      /* terminate all the strings with a null char. */
      i++;
      top [i] = '\0';
      middle [i] = '\0';
      bottom [i] = '\0';

      _setbkcolor ((long) x->color.background);
      _settextcolor (x->color.border);

      /* Draw the top line */
        _settextposition (x->top.row, x->top.col);
      _outtext (top);

      if (name != NULL) {
            _settextposition(x->top.row,
                  (x->bottom.col + x->top.col - strlen(name))/2 + 1);
            _outtext (name);
      }

      /* Draw the middle lines */
      for (i=1; i < (x->bottom.row - x->top.row); i++) {
            _settextposition (x->top.row + i, x->top.col);
            _outtext (middle);
      }
      /* Draw the bottom of the box */
      _settextposition (x->bottom.row, x->top.col);
      _outtext (bottom);

      /* Restore the original colors */
      _setbkcolor (old_back);
      _settextcolor (old_text);
}

/*---------------------------------------------------------------------------
  Print a text string at a specified row and column (in window coords)
  using the window text color. If string exceeds right window edge, it is
  truncated. Does not update the window CURPOS. Note: the origin is (1,1)
  and is located inside the upper left corner window border:

      
      *%               * = row 1, col 1     % = row 1, col 2
      +                + = row 2, col 1     etc...

  Minimal escape char line control is provided (_OUTTEXT really only prints
  text strings well). '\n' works correctly though ...
---------------------------------------------------------------------------*/
void wprintrc (window_ptr x, short row, short col, char *string)
{
      _settextwindow (x->top.row+1, x->top.col+1,
                     x->bottom.row-1, x->bottom.col-1);
      _settextcolor (x->color.text);
      _setbkcolor ((long) x->color.background);
      _settextposition (row, col);
      _outtext (string);
      x->curpos = _gettextposition();
}

/*---------------------------------------------------------------------------
  Print a text string starting at the current text position. If printing at
  last line, all text in window is scrolled up. Note: no word wrapping!!!
---------------------------------------------------------------------------*/
void wprintf (window_ptr x, char *string)
{
      _settextwindow (x->top.row+1, x->top.col+1,
                     x->bottom.row-1, x->bottom.col-1);
      _settextcolor (x->color.text);
      _setbkcolor ((long) x->color.background);

      /* if past last row, scroll up */
      if (x->curpos.row == x->bottom.row-x->top.row) {
           x->curpos.row--;
           upscroll (1, x->top.row, x->bottom.row-2, x->top.col,
                     x->bottom.col-2, x->color.background<<4);
      }

      _settextposition (x->curpos.row, x->curpos.col);
      _outtext (string);
      x->curpos = _gettextposition();
}

/*---------------------------------------------------------------------------
  Print centered text on a specified window row
---------------------------------------------------------------------------*/
void wprintf_ctr (window_ptr x, short row, char *string)
{
      wprintrc (x, row,(x->bottom.col-x->top.col-strlen(string))/2, string);
}

/*---------------------------------------------------------------------------
  Turn off the cursor by playing with INT 10H.
---------------------------------------------------------------------------*/
void cursor_off ()
{
      union REGS regs;

      regs.h.ah = 1;
      regs.h.ch = 0x20; /* set bit 5 to disable cursor */

      int86 (0x10, &regs, &regs);
}

/*---------------------------------------------------------------------------
  Turn on the cursor with INT 10H
---------------------------------------------------------------------------*/
void cursor_on ()
{
      union REGS regs;

      regs.h.ah = 1;
      regs.h.ch = 6;
      regs.h.cl = 7;

      int86 (0x10, &regs, &regs);
}

/*------------------------- End of VIDEO.C -------------------------------*/
```

```
/*
**
**  WINDOWS.H - Window definitions
**
**    Author      : Terry Fong
**    Created     : 03-13-90
**    Modified:
**
**  MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY
**
*/

/* top row,col, bot row,col, start row,col, back,border,text color */
static window title_w   ={ 2,21,  5,57, 1,1, WHITE,BLACK,BLACK};
static window error_w   ={20, 2, 24,79, 1,1, RED,BLACK,YELLOW};
static window tip_w     ={11,32, 18,46, 1,1, CYAN,BLACK,BLACK};
static window command_w ={11,47, 18,61, 1,1, CYAN,BLACK,BLACK};
static window joint_w   ={11,63, 18,78, 1,1, CYAN,BLACK,BLACK};
static window mode_w    ={ 7,32,  9,64, 1,1, BLACK,LTGREY,RED};
static window jctrl_w   ={ 7,65,  9,78, 1,1, BLACK,LTGREY,RED};
static window menu_w    ={ 7, 2, 18,30, 1,1, LTGREEN,BLACK,BLUE};

static window_ptr       title_win   = &title_w,
                  error_win   = &error_w,
                  joint_win   = &joint_w,
                  tip_win         = &tip_w,
                  command_win = &command_w,
                  mode_win    = &mode_w,
                  jctrl_win   = &jctrl_w,
                  menu_win    = &menu_w;
```

QUADOBS Listing

```
#----------------------------------------------------------------------------
#
# QUADOBS - Makefile for QUADOBS.EXE
#
#   Author   :  Terry Fong
#   Created  :  4-10-90
#   Modified :
#
#   NOTES: Exercise caution when compiling with /Oa (relax alias checking) !!!
#          (remember:  /Ox = /Oailt /Gs )
#
#----------------------------------------------------------------------------
#
# Macro definitions
#
model=/AS /FPi87         # small memory model, inline 8087 support
stdcomp=/c /Od $(model)  # optimize loops & speed, no stack probes
stdlink=/NOI /NOD        # case-sensitive, no default library search
stdlibs=slibc7+graphics

#
# QUADOBS dependencies
#
quadobs.obj : quadobs.c quadobs.h
      cl $(stdcomp) $*.c

#
# Create QUADOBS.EXE
#
quadobs.exe : quadobs.obj
      link $(stdlink) quadobs,$*,NUL,$(stdlibs)
```

```
/*----------------------------------------------------------------------------

QUADOBS.H - Function declarations for QUADOBS.C
            Observer board I/O port definitions.

    Author    :  Terry Fong
    Created   :  04-10-90
    Modified  :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
----------------------------------------------------------------------------*/

#define CTRL_8255 0x3E3
#define PORT_A          0x3E0
#define PORT_C          0x3E2

#define CTRL_8253 0x3E7
#define CNTR_0          0x3E4

#define OSC_FREQ  1e6   /* 1 MHz oscillator */

void interrupt far sample (void);
void init_system (int sample_freq);
void disable_irq (void);
void enable_irq (void);
void cursor_on (void);
void cursor_off (void);
```

```
/*-----------------------------------------------------------------------------

QUADOBS.C - Quadrature Observer.

        This software is designed to be used in conjunction with the
        SPAM/OBSERVER BOARD to sample quadrature position data.

        Quadrature signals (e.g. optical encoder output) are fed to a
        HP HCTL-2000 chip which decodes the signals into a 12-bit
        position (stored in a counter on-board the HCTL-2000). The
        observer board is equippad with an Intel 8253 programmable
        interval timer which is used to generate regular interrupt
        signals. The interrupt signals are hooked to the IBM IRQ2 line,
        which is normally unused on the PC and PC/XT. It should be noted
        that the 8253 output mode used (#2: Rate Generator) produces
        low pulses only one clock period wide, meaning the IRQ2 line
        is high for most of the time. This is not a problem since the
        8259 interrupt controller is only triggered by rising edges.

        Since the 8253 is driven by a 1 MHz oscillator and has a 16-bit
        internal countdown register, the slowest sampling rate that
        may be requested is 15.23 Hz. The upper limit on sampling
        rate for a PC/XT running at 4.77 MHz is approximately 2 KHz.
        If quadrature position is displayed on the screen, sampling
        rates should be limited to 1 KHz, otherwise the system will
        operate VERY SLOWLY.

        The following code was written for Microsoft C v5.1

    Author   :  Terry Fong
    Created  :  04-10-90
    Modified :

MASSACHUSETTS INSTITUTE OF TECHNOLOGY - SPACE SYSTEMS LABORATORY (LOOP GROUP)
-----------------------------------------------------------------------------*/

/* #define DEBUG */

#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include <graph.h>
#include <malloc.h>
#include <gnfuns\color.h>

#include "quadobs.h"

FILE *fp;
char DATASAVE;
int old_mask;
void far *old_vector;
int lb, hb, *pos;
unsigned int samples=0, max_samples=0;

main ()
{
      char filename[80];
      int sample_freq=0;
      unsigned int i;
```

```
printf ("\nData save (y/n): ");
scanf ("%c", &DATASAVE);

if (DATASAVE == 'y') {
      while ((sample_freq < 16) | (sample_freq > 2000)) {
            printf ("\nEnter sampling freq (16-2,000 Hz): ");
            scanf ("%d", &sample_freq);
      }

      printf ("\nEnter data save filename: ");
      scanf ("%s", filename);

      if ((fp = fopen (filename, "w")) == NULL) {
            printf ("\nERROR: Could not create %s\n",
                  strupr (filename));
            exit (1);
      }

      printf ("\nEnter max. # of samples: ");
      scanf ("%u", &max_samples);

      if ((pos=(int *) calloc (max_samples, sizeof(int))) == NULL) {
            printf ("\nERROR: Could not allocate memory!\n");
            exit (1);
      }
}
else {
      while ((sample_freq < 16) | (sample_freq > 1000)) {
            printf ("\nEnter sampling freq (16-1,000 Hz): ");
            scanf ("%d", &sample_freq);
      }
      DATASAVE = 0;
}

init_system (sample_freq);

_setvideomode (_TEXTC80);
_setbkcolor ((long) BLUE);
_clearscreen (_GCLEARSCREEN);
cursor_off ();
printf ("\n Quadrature Position Sampler v1.0\n");
printf (" written by Terry Fong (04/10/90)\n");
printf (" --------------------------------\n");
printf (" SAMPLING FREQUENCY = %d Hz.\n", sample_freq);
printf (" --------------------------------\n\n");
printf (" The position counter has been zeroed.\n");

_settextposition (10, 1);
printf ("\a Press any key to START sampling !!!\n");

while (!kbhit ());
getch ();

_settextposition (10, 1);
printf ("\a Press any key to STOP sampling !!!\n");

enable_irq ();
while (!kbhit()) {
      if (DATASAVE & (samples == max_samples))
            break;
```

```
        _settextposition (13, 5);

        if (DATASAVE)
              printf ("Sample: %4u", samples);
        else
              printf ("Position: %4d", hb*256 + lb);
     }

     if (kbhit()) getch ();

     /* Disable interrupt */
     disable_irq ();
     cursor_on ();

     if (DATASAVE) {
        fprintf (fp,"Sampling freq. = %d Hz.\n", sample_freq);
        for (i=0; i<samples; i++) fprintf (fp, "%4d\n", pos[i]);
        fprintf (fp, "%u samples saved\n", samples);
        fclose (fp);
     }

     _settextposition (22, 1);
     printf ("\a QUADOBS finished.\n");
}

/*
** Interrupt Driven sampling routine
*/
void interrupt far sample ()
{
     outp (CTRL_8255, 0x2);  /* HCTL-2000 OE* low */
     outp (CTRL_8255, 0x6);  /* HCTL-2000 SEL low */
     hb = inp (PORT_A);      /* grab position MSB */
     outp (CTRL_8255, 0x7);  /* HCTL-2000 SEL high */
     lb = inp (PORT_A);      /* grab position LSB */
     outp (CTRL_8255, 0x3);  /* clear HCTL-2000 inhibit logic */

     if (DATASAVE && (samples<max_samples))
        pos[samples++] = hb*256 +lb;

     outp (0x20, 0x20);      /* Reset 8259 Interrupt Controller */
}

/*
** Set up the 8255 I/O ports, configure the 8253, reset the HCTL counter.
*/
void init_system (int sample_freq)
{
     unsigned int N, lsb, msb;

     N = OSC_FREQ / sample_freq;
     lsb = N % 256;
     msb = (N - lsb) / 256;

     /* Setup interrupt for IRQ2*/
     outp (CTRL_8255, 0x90);       /* init 8255: A=in, B=out, C=out */
     outp (CTRL_8255, 0x0);        /* make PC0 low, disable 8253 out */
     outp (CTRL_8253, 0x34);       /* set up 8253: CNTR0 set to mode 2
                               (rate generator), load lsb/msb */
     outp (CNTR_0, (int) lsb);     /* load 8253 CNTR0 with lsb */
     outp (PORT_C, 0x0);
     outp (CNTR_0, (int) msb);     /* load 8253 CNTR0 with msb */

     /* zero the HCTL-2000 internal counter */
     outp (CTRL_8255, 0x4);        /* send a pulse to HCTL-2000 RST* */
     outp (CTRL_8255, 0x5);

#ifdef DEBUG
     printf ("Sampling at %d Hz. --> N = %u (0x%X%X)\n",
          sample_freq, N, msb, lsb);
#endif
}

/*
** Activate handler for IRQ2 requests.
**
** The IBM PC handles interrupts through the Intel 8259 located on the
** PC motherboard. An 8-bit register, accessed through I/O port 0x21,
** contains a mask of all recognized interrupts. In order for an interrupt
** request to be recongnized, the corresponding bit of this mask must be
** set to zero. Since the SPAM/OBSERVER BOARD generates requests on
** IRQ2, bit #2 must be set to zero.
**
** Interrupt requests processed by the 8259 are dispatched through vectors
** stored in RAM by DOS. IRQ2, for example, is handled by dos vector 0x0A.
** Changing this vector (using _dos_setvect in Microsoft C) allows tasks
** to be performed upon reception of an interrupt request.
**
** IT IS IMPORTANT TO RESTORE ANY DOS VECTORS CHANGED DURING PROGRAM
** EXECUTION BEFORE EXITING. If vectors are not restored, then the system
** will crash if DOS tries to execute a no longer existent routine!!!
*/
void enable_irq (void)
{
     old_mask = inp (0x21);            /* read current 8259 mask register */
     outp (0x21, old_mask & 0xFB); /* set IRQ2 active bit in register */

     old_vector = _dos_getvect (0x0A);
     _dos_setvect (0x0A, sample);  /* point interrupt AH (Hardware IRQ 2)
                                  vector to interrupt routine */
     outp (0x20, 0x20);                /* clear the 8259 */

     outp (CTRL_8255, 0x1);            /* set 8253 GATE0 high, enabling
                                  output from CNTR0 */

#ifdef DEBUG
     printf ("8259 Register: 0x%X --> 0x%X\n", old_mask, old_mask&0xFB);
#endif
}

/*
** Disable IRQ2 handler. Only call this routine AFTER enable_irq() has been
** called!!!
*/
void disable_irq (void)
{
     outp (PORT_C, 0x0);               /* set GATE0 low, disabling output
                                  from CNTR0 */
     outp (0x21, old_mask);            /* restore 8259 mask register */
     outp (0x20, 0x20);                /* clear the 8259 */
     _dos_setvect (0x0A, old_vector);/* restore old 0x0A vector */
}
```

```
/*
** Turn off the cursor by playing with INT 10H.
*/
void cursor_off (void)
{
     union REGS regs;

     regs.h.ah = 1;
     regs.h.ch = 0x20; /* set bit 5 to disable cursor */

     int86 (0x10, &regs, &regs);
}

/*
** Turn on the cursor with INT 10H
*/
void cursor_on (void)
{
     union REGS regs;

     regs.h.ah = 1;
     regs.h.ch = 6;
     regs.h.cl = 7;

     int86 (0x10, &regs, &regs);
}
```