Control Interfaces and Handling Qualities for Space Telerobotic Vehicles

by

Anna Gibbs Cinniger

B.S. Aeronautical and Astronautical Engineering, University of Washington
(1989)

Submitted to the Department of Aeronautics and Astronautics
In Partial Fulfillment of the Requirements for the Degree of

**Master of Science in
Aeronautics and Astronautics**

at the

**Massachusetts Institute of Technology
June, 1991**

Signature of Author _____
Department of Aeronautics and Astronautics
May 28, 1991

Certified by _____
Professor Harold L. Alexander
Thesis Supervisor
Department of Aeronautics and Astronautics

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

Control Interfaces and Handling Qualities for Space Telerobotic Vehicles

by

Anna Gibbs Cinniger

Submitted to the Department of Aeronautics and Astronautics in Partial Fulfillment of the Requirements for the Degree of Master of Science in Aeronautics and Astronautics

Abstract

The goal of this thesis was to improve control interface designs for remotely controlled vehicles. It is hypothesized that operator performance is substantially affected by the need to share attention between multiple control devices. It is further hypothesized that attention sharing between the hands and feet is easier than between the hands alone, and that interfaces which require hand/feet attention sharing will permit higher quality control than those which require hand/hand attention sharing. The impact of attention sharing and control interfaces on the handling qualities of a free flying, space telerobot is discussed. Experiments are described that use pilot ratings and performance indices to compare control interfaces which include hand and foot generated control devices.

The hypothesis was tested by comparing three different control device and command assignment configurations. The first configuration was the traditional two hand controller combination with translational degrees of freedom assigned to the left hand controller and rotational to the right. The second included a foot controller with one rotational degree of freedom assigned to it. The third included the foot controller with two rotational degrees of freedom assigned to it.

Eight pilots flew two different tasks for a total of 18 flights each. The first task required the pilots to reorient the vehicle from an unusual attitude. The second task required the pilots to hold position in the presence of disturbances. The flights were divided evenly between two tasks and the three configurations. The configurations were evaluated based on pilot rating, using a Cooper-Harper rating system, and on performance indicators like fuel expenditure and maximum velocity.

Conclusions are made that support the hypothesis to the extent that there appears to be no significant difference between the traditional configuration and the configuration with one degree of freedom assigned to the feet. Error source analysis reveals biases which suggest that improved experiments might support the hypothesis conclusively. It was also concluded that the foot controller design used in these experiments was unsuitable for pitch commands.

Acknowledgements

First of all, I really must thank all of the people who directly contributed to the contents of this research. Without their help it would have been a great struggle to even complete the experiments:

Matt (IRIS God) Machlis:	for developing the simulation and supporting the experiments and being one of the most all around helpful people I know.
Wonder UROP-ers:	Patrick Chu for pulling all of the pieces of the control station together, and Paul Stach for his most amazing foot controller.
Students of ASL:	for being so cooperative about scheduling IRIS time.
Test Subjects:	Patrick (Just let me tighten these bolts) Chu, Kurt Eberly, Beth (Is it moving?) Kader, Rob (Wildman) Sanner, Paul Stach, Mike Valdez, Harald Weigl and Franz Busse for making my rather hectic schedule possible.
Rob (Editor Extraordinaire) Sanner	for speed-editing my thesis and making incredibly clear and helpful suggestions.
Professor Graves	for loaning me his copy of SYSTAT and for letting me keep the documentation for so long.

To the powers that be:

Dave Akin:	for creating the lab and hiring me from afar.
Sandy Alexander:	for carrying on in the face of adversity and accomplishing good things with little or no money, for helpful advice, brainstorming, editing, and for working with me to get this thing done this term!!

And on a more personal level:

Sam Druker:	for all the food and affection that kept me going thru stressful times, for having the strength of personality to butt heads with me, and for being an incredibly great person despite his reputation.
Mom, Dad & especially Carmen!!!	for being the most awesome family possible and for inspiring curiosity about life and everything.

Table of Contents

Table of Figures

Chapter 1. Introduction

1.1 Space Robotics and Teleoperation

Even with the most advanced technology, robots of all types are limited in their ability to make decisions and to react to unexpected situations. Adding human controllers to systems will remove these limitations until such time as artificial intelligence is advanced sufficiently to allow true autonomy. Teleoperation, or remote control of robots, is especially well suited for working in dangerous environments; while secure in a less risky environment a human operator can teleoperate a robot to perform dangerous construction, maintenance or exploration tasks. Space is a good example of an inhospitable environment where telerobotics is already very effective; the remote manipulator arm on the Space Shuttle is regularly teleoperated to assist the astronauts.

Even with the remote manipulator arm, current satellite servicing activities and future plans for Space Station construction require humans to perform extravehicular activity (EVA). EVA's, however, are dangerous and time consuming; for example, today's space suit technology requires astronauts to pre-breath pure oxygen for two hours before they can begin an EVA. The problems of EVA might be solved with a more capable and mobile telerobot so that the the astronaut could remain in the shuttle or a space station

module, remotely pilot a robot to the worksite, and perform the necessary construction, maintenance, or repairs without requiring EVA.

Teleoperation of a remote vehicle, however, is a difficult activity which requires the coordination of several complex systems, the three most significant of which are the robot, the control station, and the human operator. The robot must combine coordinated mechanical and electronic systems which are capable of performing the desired task; the control station must provide the operator with the capability to control and communicate with the robot; and finally, perhaps most complex, the human operator must process the information provided by the robot and respond accordingly. Careful integration of all these elements is required to ensure that the total system accomplishes the task in a productive manner.

1.2 Problem Statement

The research described here aims at improving teleoperation by changing the interface between the vehicle and the human operator. Specifically, control device options and assignments of command degrees of freedom to the control devices were varied to investigate their impact on the handling qualities of a simulated space telerobot.

Remote vehicles are generally piloted by commands entered at a control station which are then transmitted to the vehicle. In previous control stations commands have been issued using a two-joystick combination; both joysticks were typically three degree of freedom, displacement-type controllers with one used to control translational motion and the other for rotational motion. Whenever possible, the translational controller's third degree of freedom was a push-pull action, and the rotational controller's was a twist action. Experience showed that this method of relating joysticks motion to the vehicle motion made it easier for the pilot to fly the vehicle. This rather subjective initial evaluation of joystick

types was one of the early motivations for the search for other, potentially superior, interface types and command assignments described herein.

The first step in this search was to identify *attention sharing* as an important factor in handling qualities. Attention sharing has been studied extensively in the context of cockpit scanning where the pilot must survey several different instruments during flight. This problem also occurs in multi-degree of freedom tasks where the pilot is trying to manage several simultaneous tasks. In the control of free-flying telerobots the pilot attempts to independently command six degrees of motion and must divide attention between the various degrees of freedom. With the traditional two-joystick configuration the alternation of attention is further divided between the hands.

The main hypothesis of this research is that, for certain types of command assignments and tasks, hand/foot attention sharing is easier than hand/hand. Human beings regularly attempt to simultaneously use their hands and feet. Driving a car with a manual transmission is a common example of a multi-task situation where one uses both hands and feet to perform different functions.

Virtual environment experiments were designed to evaluate the handling qualities of a teleoperated space vehicle as a function of the attentional demands of the control interface. These experiments compared three different control device configurations which divided the command assignments between the hands alone and the hands and feet. One configuration was the traditional two-joystick combination described above, and the other two substituted a foot controller for one and two degrees of freedom, respectively. Rotational commands were assigned to the foot controller because people are accustomed to using their feet and legs to balance and maintain their attitude while walking. The two degree of freedom foot controller was assigned pitch and roll commands. The one DOF foot controller configuration used the same foot controller, but only roll was assigned.

Eight pilots flew eighteen flights each. The flights were divided evenly between two tasks and the three configurations. The first task was a three DOF task which required the pilot to reorient the vehicle from an unusual attitude. The second task was a six DOF task which required the pilot to hold position and attitude in the presence of random disturbances.

1.3 Motivation and Background

The underlying assumption of this research is that success of vehicle teleoperation depends on the vehicle control interface independent of variations in the open-loop dynamics of the teleoperation system. Furthermore, it is assumed that different control interface configurations have different attention sharing characteristics, and that these contribute significantly to the handling qualities of the teleoperated vehicle. These two assumptions form the basis for the hypothesis of this research that control interfaces that require hand/feet attention sharing are superior to those that require hand/hand attention sharing. The purpose of this section is to summarize the results of past research which support these assumptions and which motivate the experiments presented in the following. The goal is to explain how handling qualities are evaluated and explain how attention demands are expected to affect the handling qualities.

Success of vehicle teleoperation is measured by the ease with which the human operator can fly the vehicle, which is commonly called the handling qualities of the vehicle by flight test engineers and human factors researchers. George Cooper and Robert Harper, pioneers in the application of pilot rating scales to the evaluation of aircraft handling qualities, define "Handling Qualities" as "those qualities or characteristics of an aircraft that govern the ease and precision with which a pilot is able to perform the tasks required in support of an aircraft role"[3]. The key here is that handling qualities are a combined

function of the closed loop stability and control characteristics of the man-vehicle system, of the display and control interfaces, and of the task being performed.

Pilot opinion ratings obtained using the Cooper-Harper Rating Scale are very generally used to evaluate handling qualities, especially in aerospace vehicles[3,6]. This rating scale, which can be seen in Figure 1.1 as it was used in these experiments, uses answers to specific questions about the controllability and performance of the vehicle to direct the pilot to a specific rating between "1" and "10", "1" being best. The directed nature of the pilot's evaluation reduces the subjectivity of the rating. The relative relationship between pilot rating and handling qualities has been found to be consistent, but the absolute ratings must be calibrated for each pilot[3,6].

Handling qualities research generally views the human being as a quasi-linear control element in the closed-loop system. The closed-loop, man-machine representation, which can be seen in Figure 1.2, illustrates the major contributions the human teleoperator makes as a control element: perception and compensation. The pilot perceives the information displayed, makes decisions and operates the control devices to command the vehicle. The more intuitive and less stressful the human processing is, the lower the workload and the better the handling qualities.

Attention sharing affects handling qualities whenever a pilot must make decisions, perform managerial tasks, or respond to a failure while still performing the primary task of flying the vehicle, or when the task of flying itself is too complex to manage all at once. When the pilot must divide his attention between the six degrees of motion of a free-flying vehicle, only a fraction of time is spent on any given degree of freedom. As attentional demands grow, the duration of attention given to any one of the degrees of freedom decreases and the performance suffers. Specifically, as attentional demands worsen the pilot-induced noise, pilot workload, and time delay increase[9].

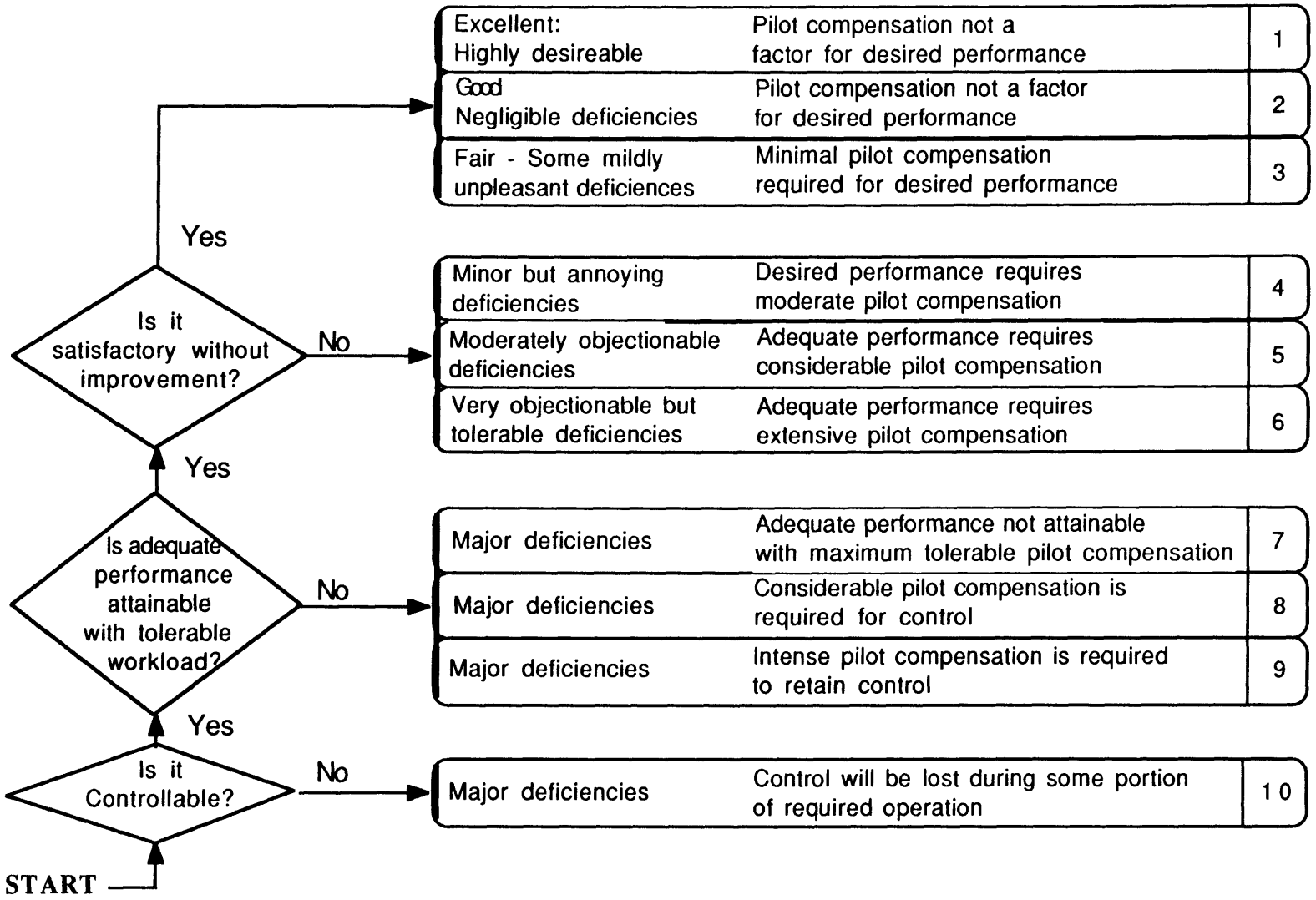

Figure 1.1 Pilot Evaluation Scale Based on the Cooper-Harper Pilot Opinion Rating Scale[3]. Notice the directed nature of the scale which reduces the subjectivity of the rating.

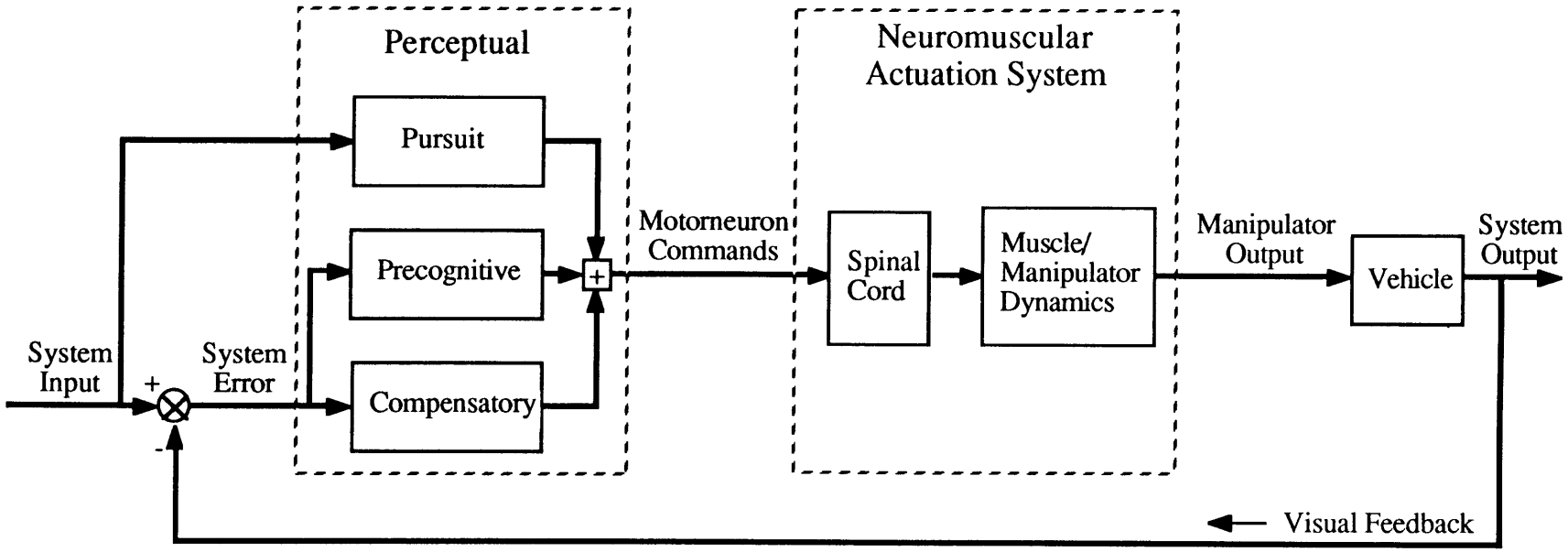

Figure 1.2 Closed-Loop Man-Machine Representation Adapted from McRuer, et al.[8].

In these experiments the task, the display, and the vehicle dynamics were held constant, and pilot ratings and performance indices were used to evaluate potentially superior configurations based on the hypothesis that foot controller configurations are superior because they have better attention sharing characteristics.

2.4 Thesis Overview

The remaining chapters describe the experiments and results.

Chapter 2, "Experiment Design", discusses in detail the decisions that were made in order to test the hypotheses described above. The discussion includes the human factors and technical issues which were relevant to the choice of control device configurations and command assignments, tasks, and the test matrix and parameters. The test procedure is also outlined.

Chapter 3, "Implementation", discusses the technical details of the equipment and software.

Chapter 4, "Results and Analysis", is divided into four sections. The first section, "General Observations", uses sample flights to illustrate typical results and uses these examples as a basis for discussion of observations made during the experiments and subsequent data analysis. The second section, "Data Analysis", describes the data reduction and analysis methods used in the third section, "Results", which is divided between the pilot evaluations and the performance evaluations. "Error Analysis", the fourth chapter, discusses error sources and their effects on the results.

Chapter 5, "Conclusion", summarizes the results of the evaluations and makes recommendations for future work.

Chapter 2. Experiment Design

The goal of these experiments was to evaluate three different control device configurations in an attempt to test the hypothesis that using feet to control one or two of the six degrees of freedom of a free flying robot is a good alternative to the traditional division of control of these motions between the two hands. The experiments required test subjects to fly a simulated telerobot from a control station using a reconfigurable control interface. In this chapter, each element of the experiment design is examined and the motivations behind the control device configurations and command assignments are discussed. Descriptions of the simulation, the tasks, and the test subjects and parameters are also given.

2.1 Control Devices and Command Assignment

Control devices contribute to the closed-loop, man-vehicle system dynamics in two ways: 1) The dynamics of control devices, such as lags, delays, and fly-by-wire control modes, contribute directly to the overall dynamics, and 2) The devices change the control response dynamics of the human operator. Attentionally demanding and non-intuitive configurations change the pilot transfer function because they change the processing the operator must do to respond to the input they receive. The *Human Engineering Guide for Equipment Designers* states, "Control movements that seem 'natural' for the operator are

more efficient and less fatiguing than those that seem awkward or difficult"[11]. Pitch and roll commands were assigned to the foot controller to make the command-motion relationship seem natural to a human pilot who would be accustomed to standing, walking and balancing with the feet and legs. The muscular actuation requirements of the devices may also affect the dynamics of the pilot by changing the neuro-muscular contribution to the pilot transfer function; and control devices that require excessive exertion or large forces will increase pilot workload and may degrade performance[4].

Control devices should be selected and oriented so that their motion consistently matches either the display or the vehicle motion. The relationship between the control device motion and the display motion is especially important to reaction and decision time when the number of control actions is high and when the control actions have a non-sequential nature; and all of the devices in a single configuration should have the same relationship[2]. Human factors research has shown that control-display compatibility is preferable for cockpit instrument design[12], however control-vehicle compatibility was selected for these experiments since the display was designed to simulate a video signal coming from cameras on board the vehicle. In most of the previous neutral buoyancy teleoperations research at MIT's Space Systems Laboratory (SSL) the only display has been the video signal.

Multi-degree of freedom joysticks or pressure sticks are recommended by human factors guidelines for multi-degree of freedom positioning of a vehicle such as space vehicle maneuvering. Foot pedals are recommended for continuous gross adjustments, like for a throttle or accelerator, or for large force applications, like braking or steering. In general, foot controls are useful when continuous control is required, as long as precision of control positioning is not important[2,4]. For the experiments described below a multi-degree of freedom foot controller was used in an attempt to combine the benefits which make a joystick-pedal device suitable for a space vehicle application.

Velocity or rate control is generally preferable over acceleration control, but in a space vehicle application the vehicle's position and translational velocity can not be sensed accurately enough to implement velocity control. Integrating the accelerometer readings to obtain velocity would be highly sensitive to initial conditions and noise. It is possible to have velocity control in the rotational degrees of freedom; however, since it is important in multi-degree of freedom manual control systems to have all degrees of freedom have the same dynamics, acceleration control was selected for all six degrees of freedom[12].

Since the control devices were the major point of interest in the experiment design careful thought was given both to their selection and to the commands assigned to each of their degrees of freedom. Both the type of controller and number of DOF's were important factors.

For the experiments described below, the pilots used three different configurations of control devices and command assignments. The first configuration was the traditional configuration of two, three degree of freedom, hand controllers, sometimes referred to as the no-foot-controller configuration. This configuration was arranged using the traditional translational/rotational division of control to provide a basis for comparison. Both hand controllers were displacement-type, proportional controllers. The translational controller's third DOF required a push/pull motion, while the rotational controller's required a twisting motion.

The commands were assigned in accordance with human engineering guidelines for control movement and vehicle response relationships[2,4,11]. The command assignments for all of the devices can be seen in Figure 2.1. The translational hand controller had vertical, lateral and horizontal commands assigned to it to make the motion of the joystick correspond to the intuitive motions of the vehicle. The rotational hand controller had roll, pitch and yaw command assigned to it in the traditional aircraft relations: pushing the

joystick forward commanded a pitch down; backward, pitch up; to the right, roll right; to the left, roll left; twisting to the right, yaw right; twisting to the left, yaw left.

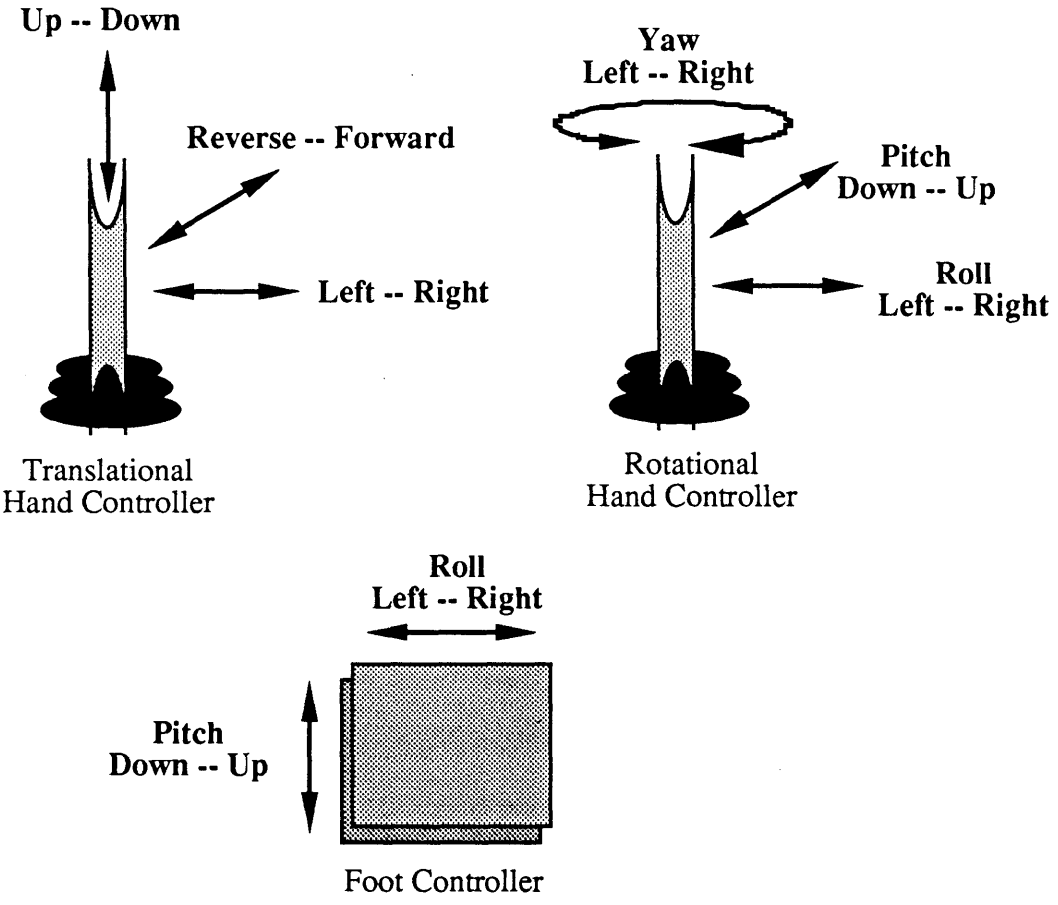

Figure 2.1 Illustration of Command Assignments to the Control Devices.

The second and third configuration used the same two hand controllers, but also included a two DOF, non-displacement-type, force-sensing foot controller. The second configuration, referred to as the two DOF foot controller configuration, had pitch and roll commands assigned to the foot controller. The foot controller was configured such that when the operator pushed flat with his right foot, he would command a roll to the right;

with his left foot, a roll to the left. When the operator pushed with the ball of his foot, he would command pitch down; with his heel, a pitch up. The translational and yaw commands were assigned to the hand controllers in the same fashion as the traditional configuration.

The third configuration, referred to as the one DOF foot controller configuration, had only roll assigned to the foot controller in the same sense as above; the other degree of freedom was not used. The translational, yaw and pitch commands were left unchanged from the traditional configuration. The third configuration was added because early testing suggested that the foot controller design was less suitable for pitch commands.

An ideal comparison of control division between just the hands and a combination of the feet and hands, would have employed a foot controller which was also a three DOF, displacement-type controller. However, it proved difficult to design such a controller, so the foot controller was designed with only two degrees of freedom. The differences between displacement control and force control are most noticeable at low frequency, where force control phase lag is higher than displacement control[9]. In general, force control is preferable for high order systems where excessive delay is a problem[12].

2.2 Telerobot Simulation

The simulation of the space telerobot was developed for these experiments by Matt Machlis while he was a graduate student in the MIT Laboratory for Space Teleoperations and Robotics (LSTAR).

The simulation used an extremely simple dynamic model for the space telerobot and it's environment. The robot was considered a rigid body moving with no damping or resistance in a drag free, zero gravity space environment. The mass of the vehicle was taken as 500 kg and the moments of inertia around each of the axes were 4000 kg-m^2.

Commands from the control station were exerted as forces on the vehicle with respect to the vehicle coordinate frame as shown in Figure 2.2. The maximum commandable force was 100 N, which would result in a maximum translational acceleration of 0.2 m/s^2; and the maximum commandable torque was 100 N-m, resulting in a maximum rotational acceleration of 0.025 rad/s^2.

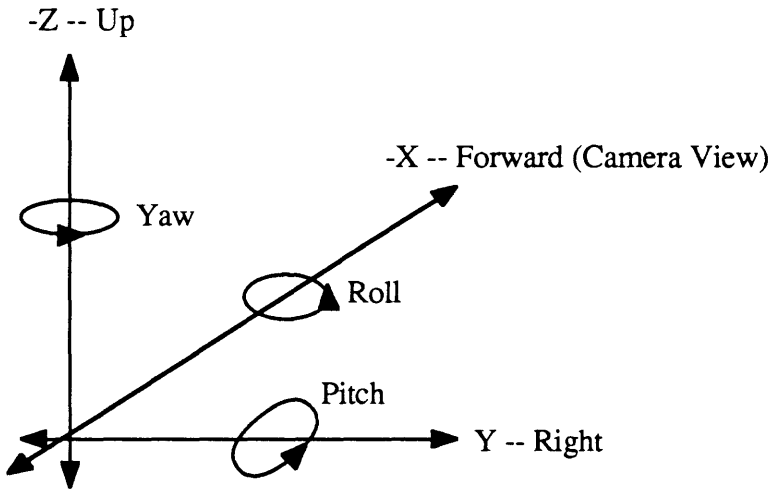

Figure 2.2 Command Assignment Directions with Respect to Vehicle Coordinate Frame. Notice that the vehicle coordinate frame is not a "right-hand" coordinate frame.

The simulation software polled the control station for commands every 0.1 seconds and the equations of motion were integrated using the backward Euler method. The control station and simulation software is listed in Appendix A. The dynamics calculation algorithm can be found on page 91.

2.3 Tasks and Display

Two different tasks were selected for the experiments. The first task required the operator to reorient the vehicle from an unusual attitude. For this task, motion was limited

to the three rotational DOF's; no translation was required. This task was selected to reduce the division of attention between degrees of freedom, hence it concentrates the attention sharing between the feet and hands. The results from this task were used to directly compare foot control and manual control modes.

The second task required the operator to maintain the vehicle's position and attitude in the presence of disturbances. These disturbances acted on the body in all six DOF's and were uniformly distributed random forces and torques with an absolute maximum magnitude of 25 N and 25 N-m, respectively. This task forces the operator to act like a regulator and reject disturbances and challenges attention sharing more than in the reorientation task.

For both tasks the display was the same. It was designed to attempt to provide the operator with enough information to determine the translation and rotation of the vehicle. The "space" in which the vehicle moved contained a grid wall with a wire frame cube in front of it. Both the grid wall and the wire frame cube were fixed in the "space" and the vehicle moved with respect to them. Refer to Figure 2.3 for an illustration of the display. Notice the two diagonal lines which form an arrow indicating up on the front face of the cube. This arrow was added to indicate the zero-roll orientation.

Since commands acted with respect to the moving vehicle coordinate frame, vehicle reference marks in the form of corner markers were added to give the operator additional information about how commands would affect the vehicle's motion. These reference marks can be seen in Figure 2.3. Without the vehicle reference marks the pilot could have encountered problems similar to those associated with high canopy cockpits where the pilot loses sense of how the vehicle is oriented and moving with respect to the fixed world. Past teleoperation research at SSL has shown vehicle reference marks to be useful in docking tasks[10].

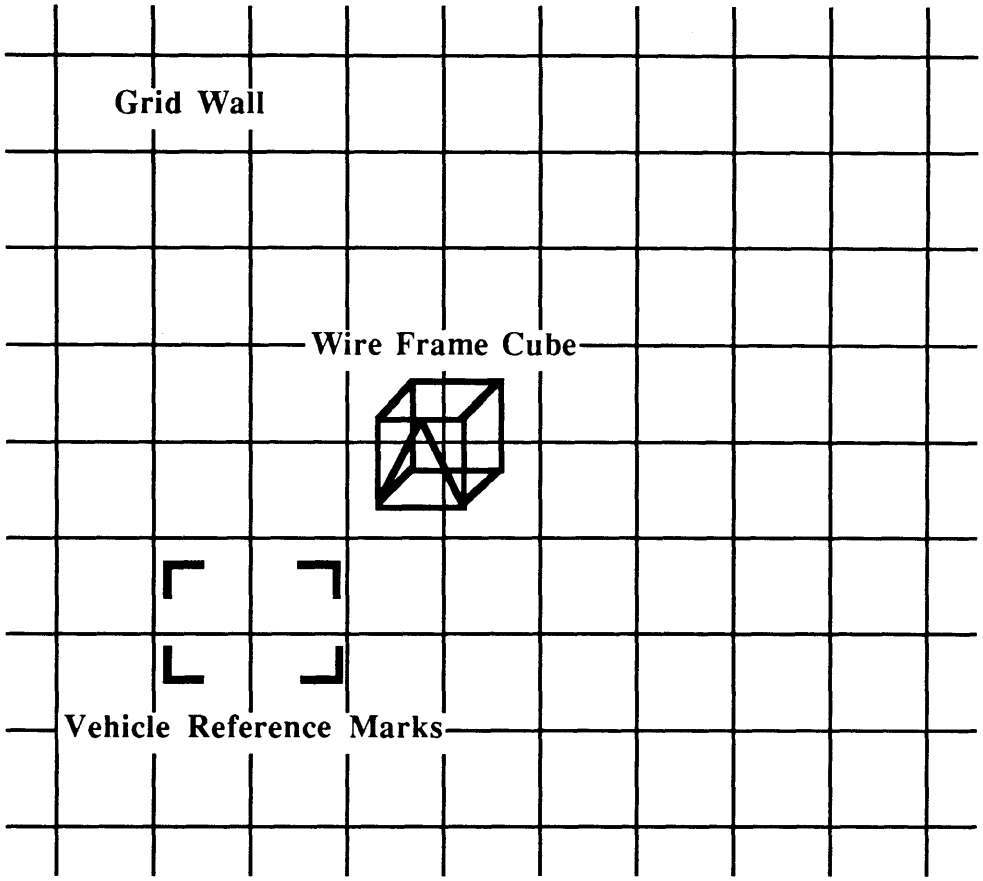

Figure 2.3 Three Dimensional Task Environment Display Including Vehicle Reference Marks.

A stereo display was selected because the 3-dimensional illusion provides better depth cues to the operator than a flat 2-dimensional image. Stereo displays have been observed to reduce docking time in a teleoperator docking task[10].

2.4 Subjects and Initial Evaluation

The eight test subjects were volunteers from LSTAR. All of the pilots were familiar with at least the concept of teleoperating a vehicle. A few had limited experience flying an actual or simulated telerobot. Before any experiments were performed the test subjects were told that they would be flying the robot using three different control configurations.

Each subject filled out a survey which recorded certain relevant skills such as driving, flying and video game playing. The survey, which can be found in Appendix B, also recorded the subjects' vision, any tendency for vertigo or confusion of left with right.

In order to assess the pilots' initial skill level and provide each with equivalent training, they flew a series of evaluation flights before the experiment began. The evaluation flights involved flying the simulated telerobot through the center of three rectangles. The rectangles were arranged such that they each defined a vertical plane, and the center of each rectangle was marked by a cross. The display for the evaluation flights is illustrated in Figure 2.4. As the pilots flew the telerobot through each of the planes the distance between the vehicle and the center of the rectangle associated with that plane was recorded. The pilots flew the evaluation flight three times, once with each of the configurations.The results of the evaluation flights were used to create an initial ranking of the pilots.

2.5 Test Matrix and Parameters

Experiments using human test subjects must be carefully designed to prevent excessive random error and systematic biases. Human vary from day to day, and they learn and adapt easily. Furthermore, factors such as fatigue and emotion can affect their performance. These experiments made use of two techniques, multiple trials and counter balanced trials, to reduce the impact of random errors and systematic biases, respectively[1].

The experiments were arranged so that each pilot flew a total of eighteen flights distributed over three days, six on each day. Each set of six flights was called a run. The three configurations were each used twice during a run, once for each of the two tasks described above. The order of flights in a run was such that three flights were flown for

one task and then three flights for the other task. Half of the pilots flew the experiment in an AB-BA-AB sequence (reorientation task = 'A', position hold task = 'B') and the other half in a BA-AB-BA sequence to prevent experience from one task from influencing performance in second task. For each task, the order of control device configurations changed from run to run; within a run, the order of configurations was different for each new task. For every task the pilots used the configurations in a 1-2-3, 2-3-1, or 3-1-2 order (no foot controller = '1', 2 DOF foot controller = '2', and 1 DOF foot controller = '3'). Figure 2.5 shows how the task and configurations were assigned for each flight for each pilot.

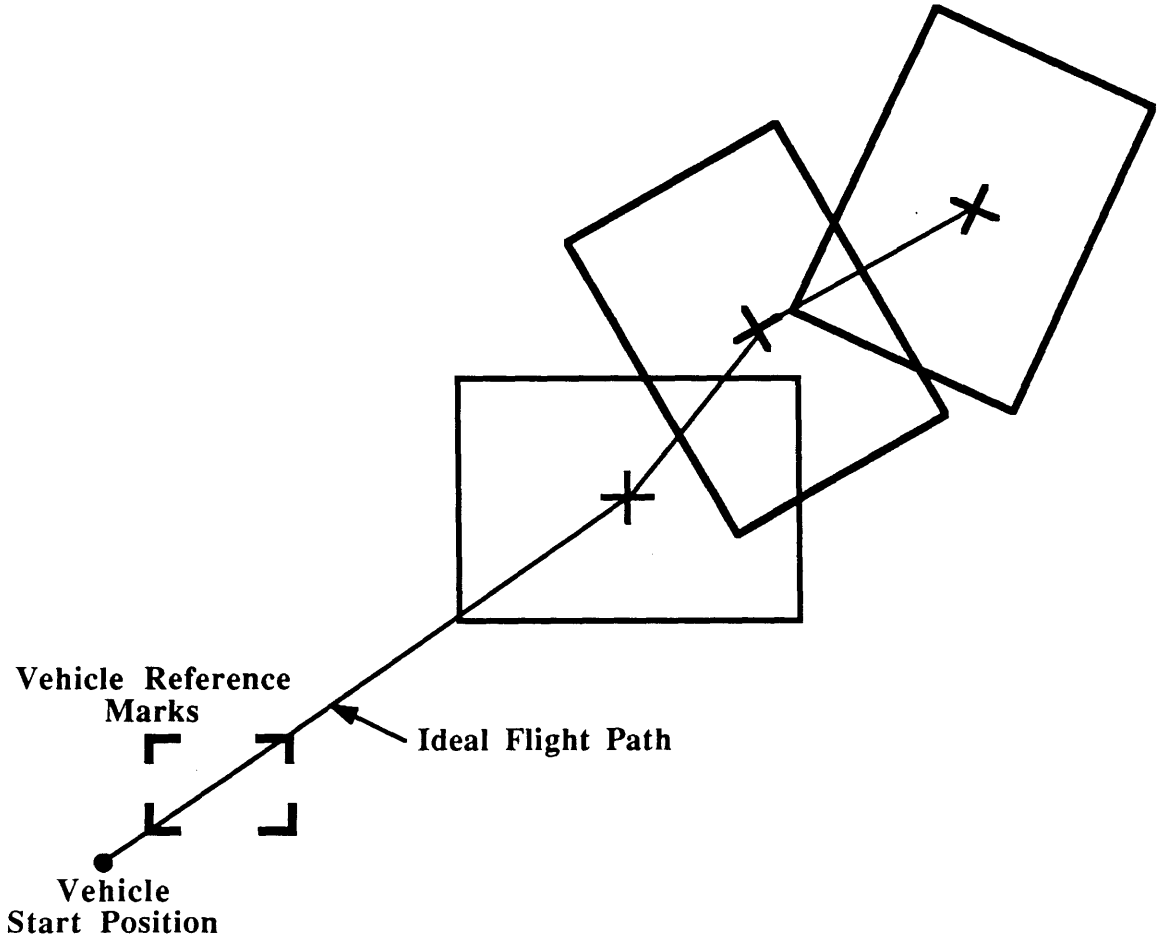

Figure 2.4 Three Dimensional Environment Display Used to Evaluate Pilot Skill. Pilots attempted to fly from the vehicle start position through the center of each successive rectangle.

Subject	Run #1 Flights 1 - 6						Run #2 Flights 7-12						Run #3 Flights 13-18					
a	A 2	A 3	A 1	B 1	B 2	B 3	B 2	B 3	B 1	A 3	A 1	A 2	A 1	A 2	A 3	B 3	B 1	B 2
b	B 3	B 1	B 2	A 1	A 2	A 3	A 2	A 3	A 1	B 1	B 2	B 3	B 2	B 3	B 1	A 3	A 1	A 2
c	A 3	A 1	A 2	B 2	B 3	B 1	B 3	B 1	B 2	A 1	A 2	A 3	A 2	A 3	A 1	B 1	B 2	B 3
d	B 1	B 2	B 3	A 2	A 3	A 1	A 1	A 2	A 3	B 3	B 1	B 2	B 2	B 3	B 1	A 3	A 1	A 2
e	A 2	A 3	A 1	B 1	B 2	B 3	B 2	B 3	B 1	A 3	A 1	A 2	A 1	A 2	A 3	B 3	B 1	B 2
f	B 3	B 1	B 2	A 1	A 2	A 3	A 2	A 3	A 1	B 1	B 2	B 3	B 2	B 3	B 1	A 3	A 1	A 2
g	A 3	A 1	A 2	B 2	B 3	B 1	B 3	B 1	B 2	A 1	A 2	A 3	A 2	A 3	A 1	B 1	B 2	B 3
h	B 1	B 2	B 3	A 2	A 3	A 1	A 1	A 2	A 3	B 3	B 1	B 2	B 2	B 3	B 1	A 3	A 1	A 2

KEY: "A" = Reorientation Task, "B" = Position Hold Task
Number corresponds to Inital Condition or Random Number Seed

Figure 2.5 Test Matrix

As mentioned above in section 2.4, the evaluation flight was used to rank the pilots by skill. These rankings were used to design the test matrix by matching each of the first four pilots with the last four based on relative skill. Figure 2.5 was created by first listing the test subjects in order of skill as determined in the evaluation flight. The three best pilots and the three configurations were used to form six 3x3 Latin squares, one for each task for each run. The fourth pilot repeated the pattern of the first pilot in the first run, the third pilot in the second run, and the second pilot in the last run. The pattern for the four best pilots was repeated for the four worst pilots. Creating matched sets in this fashion reduces the degrees of freedom in the statistical analysis and enables the use of a smaller data set. The matched sets were not used in the final data analysis because the validity of the rankings was questioned.

The parameters changed for each flight, except the practice flights, so that learning would not affect the pilots' performance. Within each run the same parameters were used for each of the practice flights so that the difficulty of practices remained consistent for each pilot. In the reorientation task the parameters were the initial roll, pitch and yaw angular offsets, which are listed in Table 2.1. These initial angles were generated randomly using dice and entered into the simulation input files. In order to make sure that the pilot could initially see at least part of the grid wall, no dice roll was allowed that resulted in an angle over 90 degrees, and any combination of angles that did not have the grid was in the field of view was discarded.

In the position hold task the random disturbance was generated by a random number generator based on a particular "seed" number. It is important to note that the same seed always produced the same set of random numbers. For each run, four three-digit seed numbers were selected with dice: one seed to be used repeatedly in the practice flights and one seed each for the three experiment flights. There were a total of three practice seeds and nine experiment seeds so that the pilots did not learn what to expect from the random

disturbances as time went on. Every pilot used the same seeds to make sure that each experienced similar levels of task difficulty.

Table 2.1: Initial Rotation Values for the Reorientation Task

Run	Parameter Number	Roll [deg]	Pitch [deg]	Yaw [deg]
1	P1	42.97	-24.64	42.97
	1	17.76	26.36	23.49
	2	-77.92	-29.79	71.05
	3	-77.35	-69.90	42.97
2	P2	-86.52	57.87	46.98
	4	-67.04	-55.58	-20.63
	5	-1.15	-31.51	4.58
	6	-22.92	-48.70	-10.89
3	P3	3.44	58.44	41.25
	7	69.90	56.72	17.19
	8	-10.31	-63.03	24.06
	9	-67.04	42.40	-81.93

2.6 Experiment Procedure

For each of the evaluation flights, the practice flights, and the actual experiment flights the test procedure went as follows:

1) The pilot was informed which task and configuration would be used.

2) The pilot put on the Eyephones helmet mounted display and was told to "relax and get neutral" which meant to put his hands and feet on the appropriate control devices and to relax so that the calibration reading in the software would set the zero point at the pilot's relaxed neutral point.

3) The control station software was started.

4) The pilot was told to close his eyes and the simulation video output was switched from the IRIS monitor to the Eyephones. This step was added to prevent eye fatigue from the static and flashes associated with switching the video

5) The simulation software was started and the pilot was told to open his eyes.

6) In the position hold task the disturbances were started by the experimenter five seconds after the simulation began.

7) The simulation ended automatically in the reorientation task and was terminated by the experimenter two minutes after the start of the disturbances in the the position hold task.

In the actual experiment flights the pilot finished by removing the Eyephones and filling out the pilot evaluation form shown in Appendix B.

Chapter 3. Implementation

A control station was built with the desired command interfaces and integrated with a computer simulation of a telerobot to perform the experiments described in the previous chapter. This chapter describes the computer simulation and display, as well as the equipment, including the control station and the control devices. The control station and simulation software can be found in Appendix A.

3.1 Virtual Environment Simulation and Display

The virtual environment simulation of the space telerobot, created by Matt Machlis while he was a Graduate Research Assistant in LSTAR, was programmed in C to run on a Silicon Graphics Personal IRIS Graphics Workstation. The simulation received the commands from the control station, calculated the motion of the vehicle, and produced updated stereo video images which were transmitted to the helmet mounted display. The helmet mounted display used for the experiments was the VPL Eyephones system, which consists of a video display inside a set of specially made goggles and a weight at the back of the head to counter the weight of the display.

3.2 Control Devices

The control devices used in the experiment were two joysticks and a foot controller. The joysticks were manufactured by P-Q Controls. The first joystick was a three DOF, displacement-type, proportional hand controller with a push pull action, P-Q Controls Model 220-19. The second joystick was a three DOF, displacement-type, proportional hand controller with a twist action, P-Q Controls Model 220-21. Both joysticks were purchased with the 30% voltage swing option, so that when supplied with ±8 volts DC each degree of freedom would output between ±4.8 volts DC.

The foot controller was designed and built by Paul Stach while he was an Undergraduate Research Assistant at LSTAR. The foot controller was a two degree of freedom, force sensing, non-displacement-type controller. Force sensitive resistors, manufactured by Interlink Electronics, sensed the force exerted by the operator's feet. Preliminary evaluation revealed that the pilots would commonly exert maximum forces of around 35 lbs. Circuitry was included to output a voltage proportional to the force exerted by the operator. The circuitry was tuned such that when the foot controller was supplied with ±15 Volts DC the output would range between ±5 volts for forces up to 35 lbs.

3.3 Control Station

The control station was comprised primarily of a micro-computer, the control devices, and a helmet mounted video display. The interface between the control devices and the micro-computer consisted of a custom I/O box and an analog to digital converter (A/D) board mounted in one of the computer's bus slots. Communications to and from the vehicle simulator were via the serial port on the computer. The system block diagram shown in Figure 3.1 illustrates the relationship between the components. Each of these

components is described below, except for the control devices and the video display which are treated separately above.

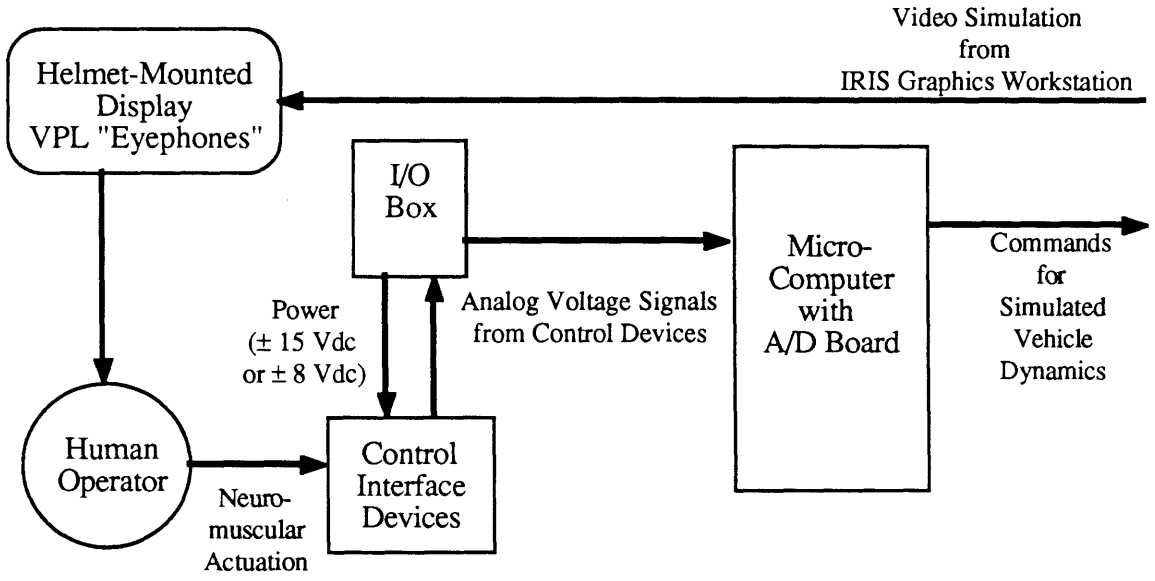

Figure 3.1 Teleoperation Control Station System Block Diagram. This figure illustrates the relationships between each of the control station components including the human operator. The input from and the output to the computer simulation are also noted.

The control station computer was a 20 MHz 80386 computer purchased from Gateway 2000. The operating system was QNX Version 2.15E, a real-time, multi-tasking, message-passing operating system produced by Quantum Software of Ottawa, Canada. The A/D board was model PC-74, a sixteen channel, variable gain board, purchased from Industrial Computer Source.

The I/O Box was custom designed and built for easy control station assembly and modularity. The purpose of the I/O Box was to supply power to the control devices and to connect the outputs of the control devices to the A/D board. Each of the control devices

was connected to the I/O box via DB-15 connectors. The I/O Box contained a ±15 volt DC power supply which was regulated to ±8 Volt DC for the two hand controllers. The analog voltage signals from the control devices were output by the box at a 50 pin DB to Ribbon Cable connector. A 50 wire ribbon cable connected the I/O box to the A/D board.

The hand controllers were mounted on a wooden support such that when the operator was holding them the operator's arms and shoulders were not raised in an uncomfortable fashion and were extended evenly. The translational controller was mounted horizontally on the left so that the push/pull action would be forward and back. The rotational controller was mounted on a fifteen degree incline. The foot controller was placed flat on the ground at the operators feet. Figure 3.2 shows the placement of the control devices on the control station.

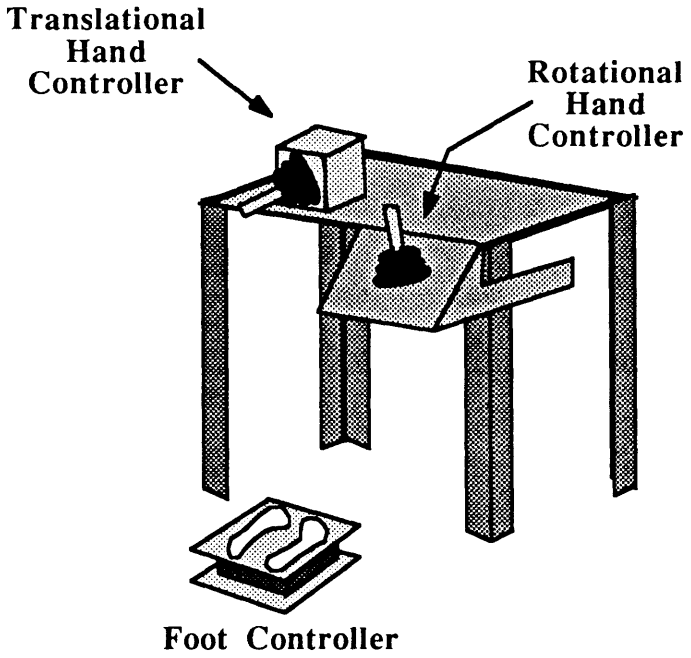

Figure 3.2 Control Device Layout. This figure illustrates the relative position of each of the control devices on the control station.

Chapter 4. Results and Analysis

The purpose of these experiments was to test the hypothesis that division of attention between the hands and feet is better than between the hands alone. As described in the preceding chapters this was accomplished by contrasting three different configurations of control devices and command assignments. This chapter reports the results of these experiments while evaluating and comparing the different configurations. In the first section, sample flights and general observations are briefly described for each of the tasks. The second section includes a description of the data reduction and statistical analyses and presents the results of the evaluations. The pilot evaluation results are given first, followed by the performance evaluation results. The evaluation results are summarized at the end of the second section. The third section describes errors and problems which may have affected the data.

4.1 General Observations

This section discusses the data recorded and describes the flights, including example time histories of variables representing various motion and command degrees of freedom. In both tasks these time histories reveal interesting information about how the pilots commanded the vehicle and how unexpected problems affected the data.

During the experiments the following data was recorded for both tasks: time histories of state values and commands and pilot evaluations. For the position hold task the disturbance time histories were also recorded. Time of completion was also recorded for the reorientation task.

The reorientation task was designed to provide the operator with a reasonably simple, multi-degree-of-freedom task. Figure 4.1 shows time histories for each of the three rotational degrees of freedom. The pitch overshoot in the flight shown is characteristic of many of the flights where overshoot occurred frequently, independent of pilot or configuration.

During the experiments it was observed that the pilots typically commanded one degree of freedom at a time. Occasionally they would command pitch and yaw together, but rarely combined roll with the other two degrees of freedom. Figure 4.1 shows command time histories for the reorientation task for each of the degrees of freedom. In the reorientation task the intervals of time in which one degree of freedom was being commanded rarely overlapped; in the example shown they did not overlap at all. This style of discretely controlling each degree of freedom meant that the pilots did not multi-task between the degrees of freedom. As a result, the comparison of the configurations for the reorientation task highlights the differences between the devices rather than the differences in the attentional demands of the devices.

The position hold task was designed to force a great deal of attention sharing. Figures 4.2.a and 4.2.b show the X, Y and Z position and roll, pitch and yaw angle time histories. Figures 4.3.a and 4.3.b show the command time histories for each of the degrees of freedom.

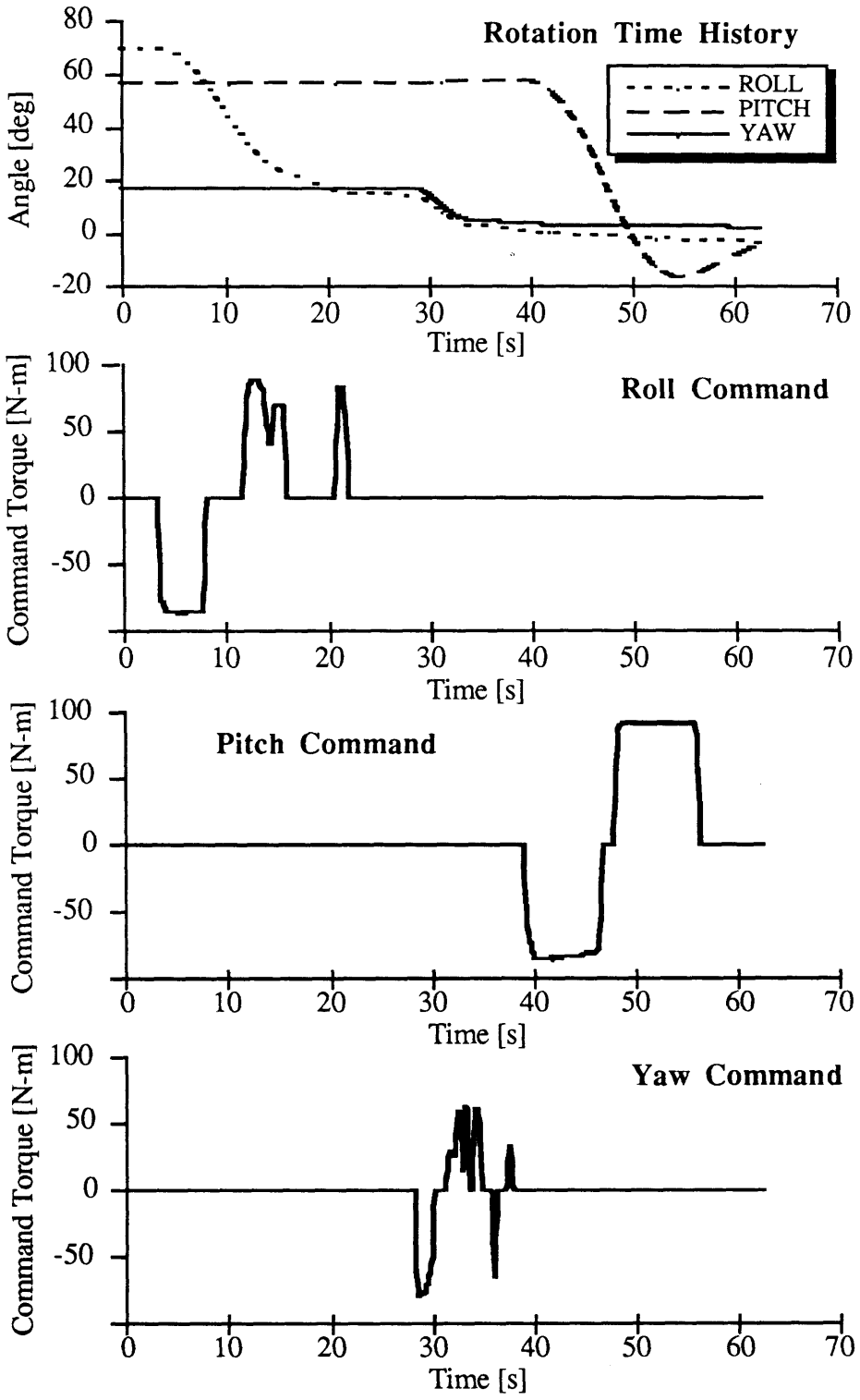

Figure 4.1 Example Rotation Angle and Command Time Histories for the Reorientation Task. The pilot was using the Traditional two-joystick configuration. Notice the pitch overshoot at 50 seconds and how the pilot broke the flight up into three segments: roll first, then yaw, and pitch last. Both of these phenomena were typical for the reorientation task independent of pilot or configuration. (run = 3, subject = "a", initial condition set = 7).

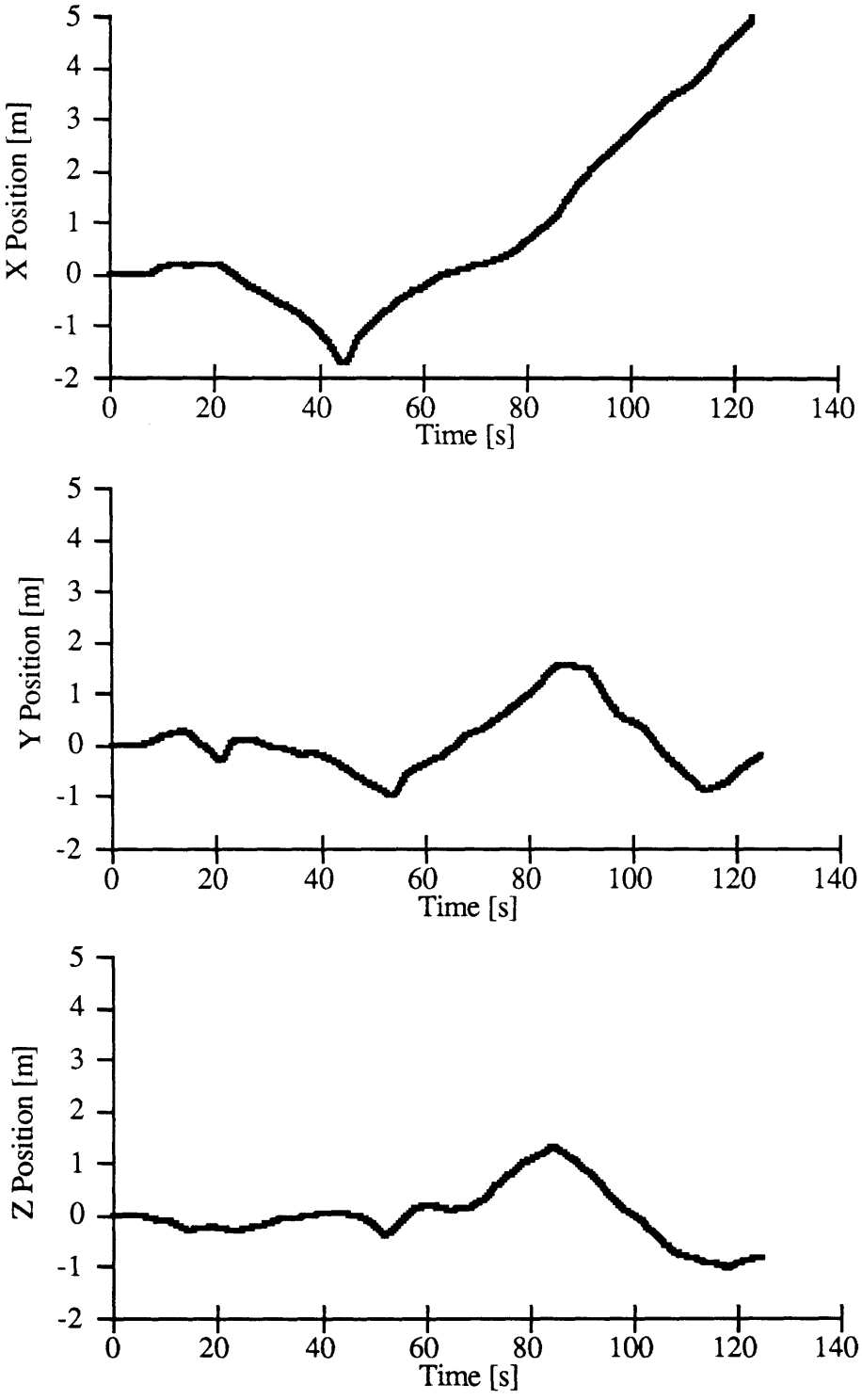

Figure 4.2.a Example X, Y, and Z Position Time Histories for the Position Hold Task. The pilot was using the Two DOF Foot Controller configuration. The corresponding command time histories are shown in Figure 4.3.a. (run = 3, subject = "b", random seed parameter #8)

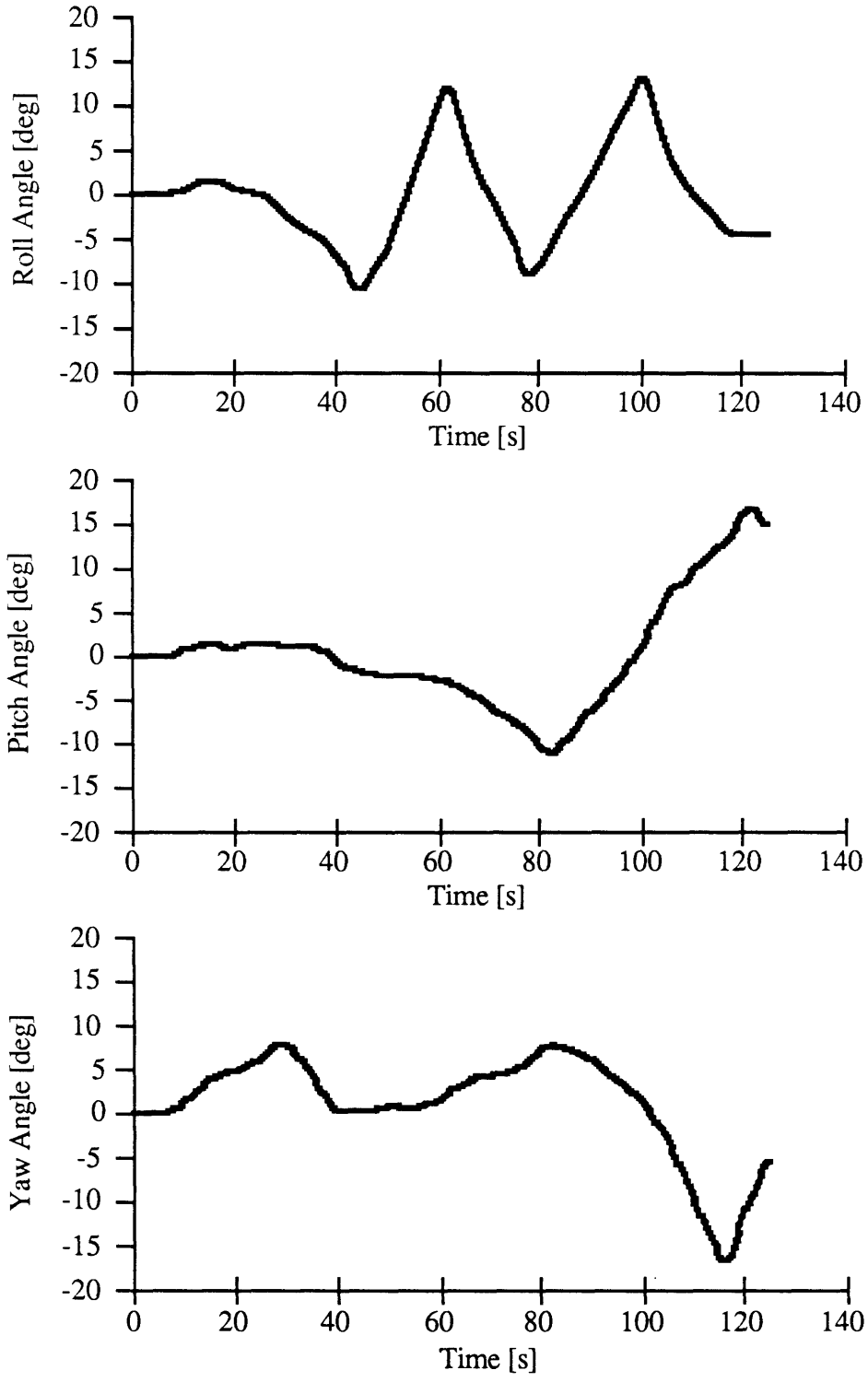

Figure 4.2.b Example Roll, Pitch and Yaw Rotation Angle Time Histories for the Position Hold Task. The pilot was using the Two DOF Foot Controller configuration. The corresponding command time histories can be found in Figure 4.3.b. (run = 3, subject = "b", random seed parameter #8)

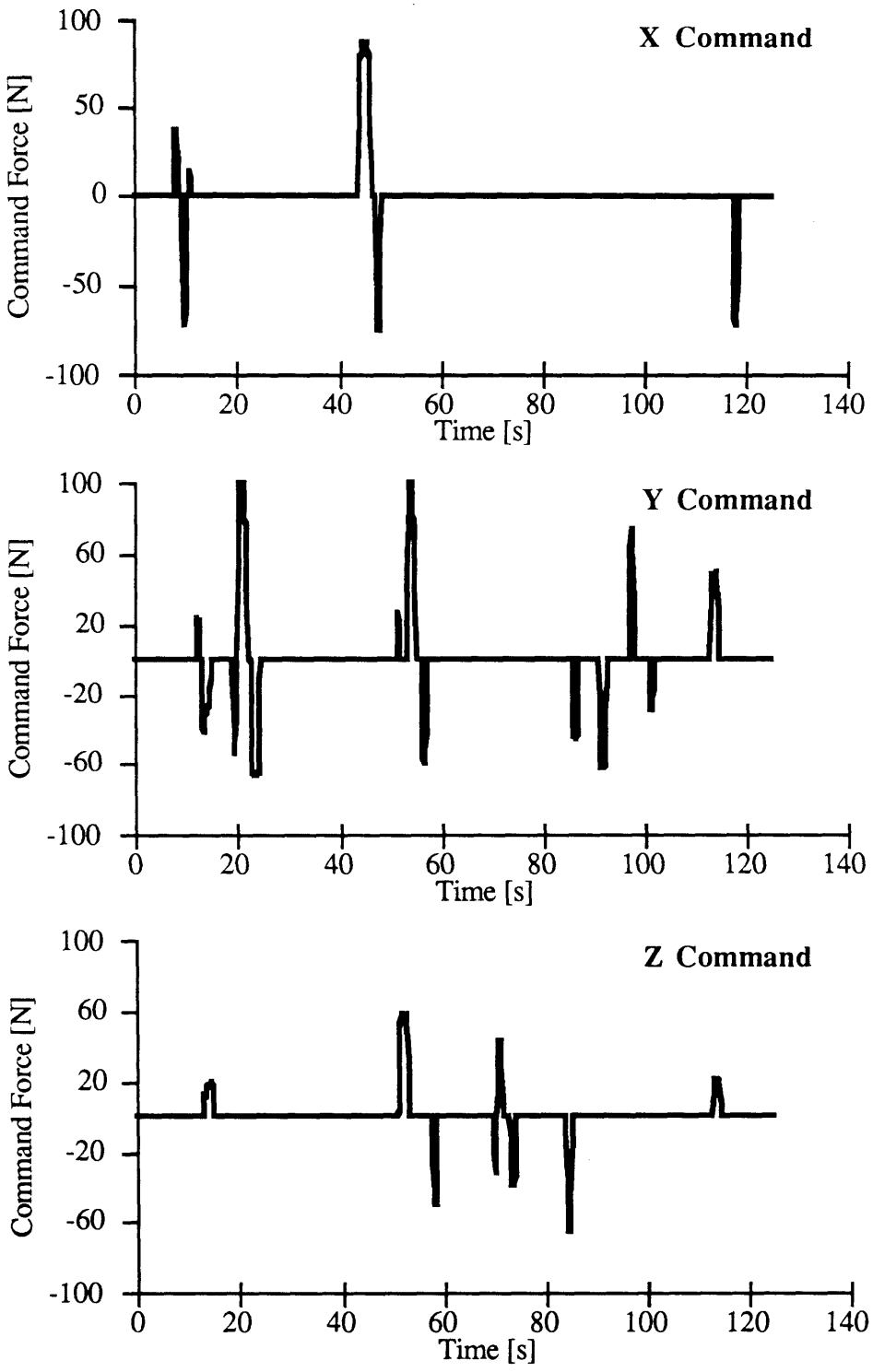

Figure 4.3.a Example X, Y, and Z Command Time Histories for the Position Hold Task. The Pilot was using the Two DOF Foot Controller configuration. The corresponding position time histories can be found in Figure 4.2.a.

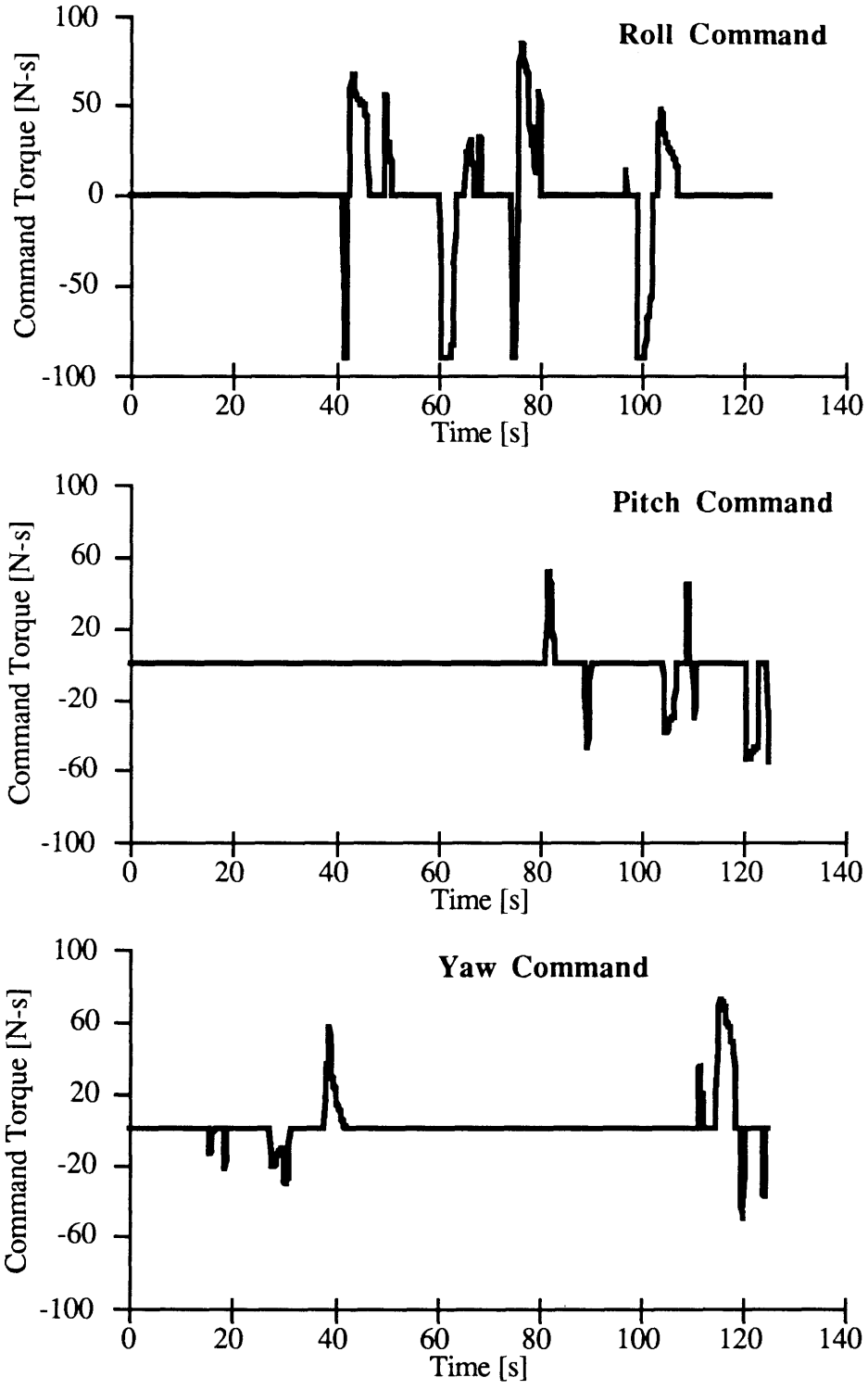

Figure 4.3.b Roll, Pitch and Yaw Command Time Histories for the Position Hold Task. The pilot was using the Two DOF Foot Controller configuration. See related Figures 4.2.a, 4.2.b, and 4.3.a.

4.2 Data Analysis

Data reduction included integrating the magnitude of the commands from time zero to the end of the flight to create a total command effort for each DOF. The integration was performed using the trapezoidal approximation method. The individual absolute command efforts were added to create a total command effort for each flight. For the position hold task, displacement errors were calculated at each time increment, where displacement error was defined to be the sum of the squares of the displacements from the zero point in each of the translational degrees of freedom. The displacement error was equivalent to the magnitude of the vector pointing from the zero point to the vehicle. For both tasks maximum, minimum, and average state values and commands were determined for each flight. In addition maximum, minimum, and average displacement errors and disturbances were determined for each of the position hold task flights.

This data reduction produced a set of numbers for each flight which were prepared along with pilot pre-flight survey information for statistical analysis, which included analysis of variance to investigate the relationships of the important variables to the configurations and other parameters if necessary.

Analysis of variance was used to test the significance of the differences in the mean pilot ranking and performance index values between each configuration. The result of analysis of variance indicates the probability that the differences in the means is not due to chance and is likely to be a function of the specified factor. In most engineering applications a probability of 95% or greater is considered significant and a probability of 99% or greater is highly significant. For the purposes of the pilot and performance evaluations discussed below, a probability of 90% or greater was classified as "almost" significant, and was treated as significant for the purposes of contrasting the effects of one

configuration against another. The analysis of variance and contrasting of effects was performed using SYSTAT 5.0 for the Macintosh.

4.3 Results

4.3.1 Pilot Evaluation

Evaluation of the different configurations was based partly on pilot evaluations of the handling qualities. Pilot evaluations were obtained using a Cooper-Harper pilot rating scale. The Cooper-Harper rating scale is commonly used in flight test evaluations of aircraft handling qualities. It is a directed evaluation which rates the system from 1 to 10, 1 being the best rating and 10 being the worst. The pilot determines the rating by answering a sequence of questions relating to the controllability and performance of the system. The scale used in these experiments is shown in Figure 1.1. The pilot ratings were used to rank the configurations for each task within each run. The best rating was assigned a ranking of "1"; the next best, a "2"; and the worst, a "3"; in that order. When more than one configuration had the same rating they were given the same ranking.

In both tasks the pilots preferred the traditional configuration over the two configurations with the foot controller. The distributions of pilot ranking for each of the configurations are shown in Figures 4.4 and 4.5 for the reorientation and position hold tasks, respectively. In the reorientation task the no-foot-controller configuration had significantly higher rankings than the other two, and there was no significant difference reported between the one and two DOF foot controller configurations. In the position hold task the two DOF foot controller was rated significantly worse than the other two, and there was no significant difference between the traditional configuration and the one DOF foot controller configuration.

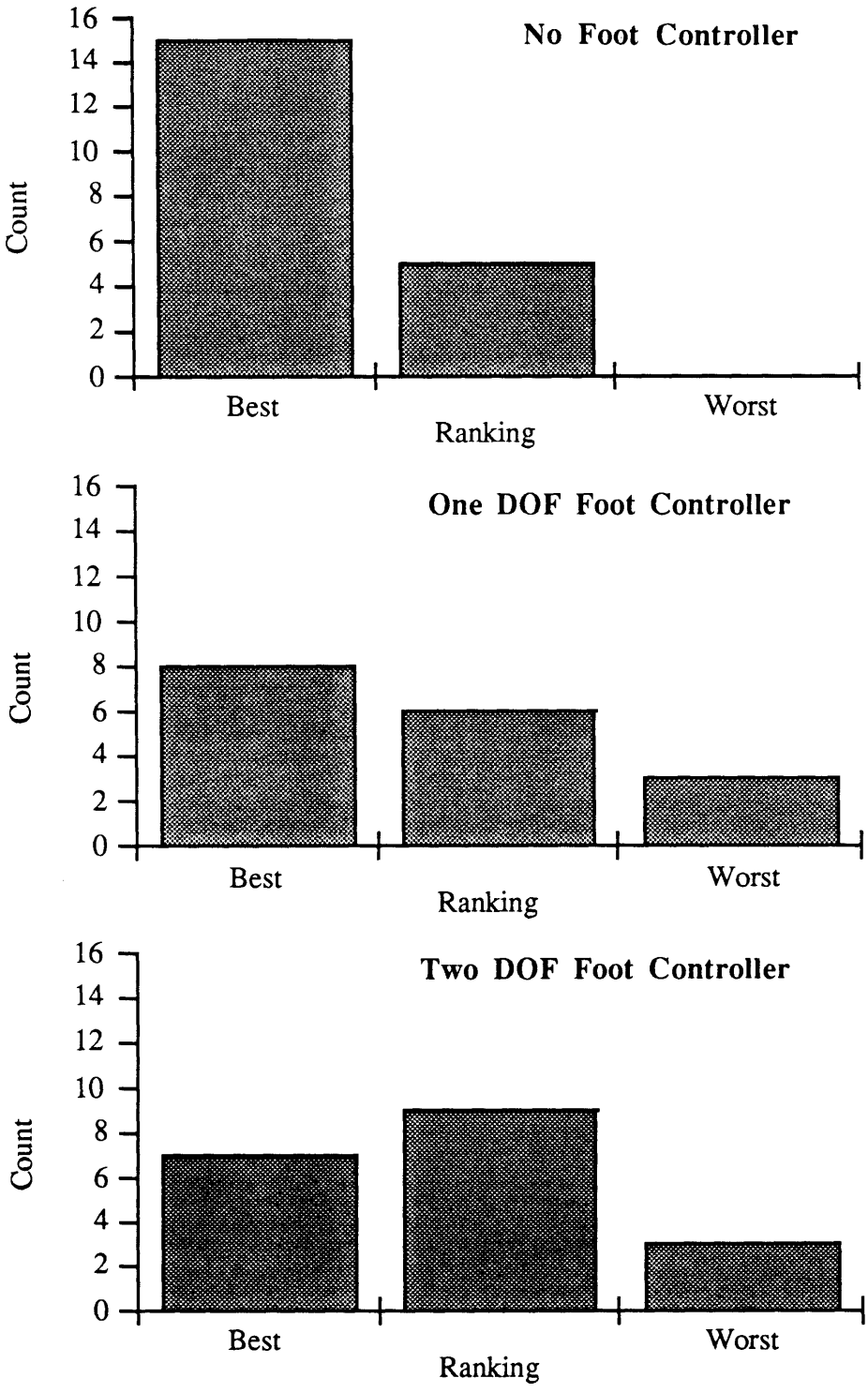

Figure 4.4 Distribution of Pilot Rankings of Each Control Device Configuration for the Reorientation Task. These rankings were determined from pilot opinion ratings (ties were given the same ranking).

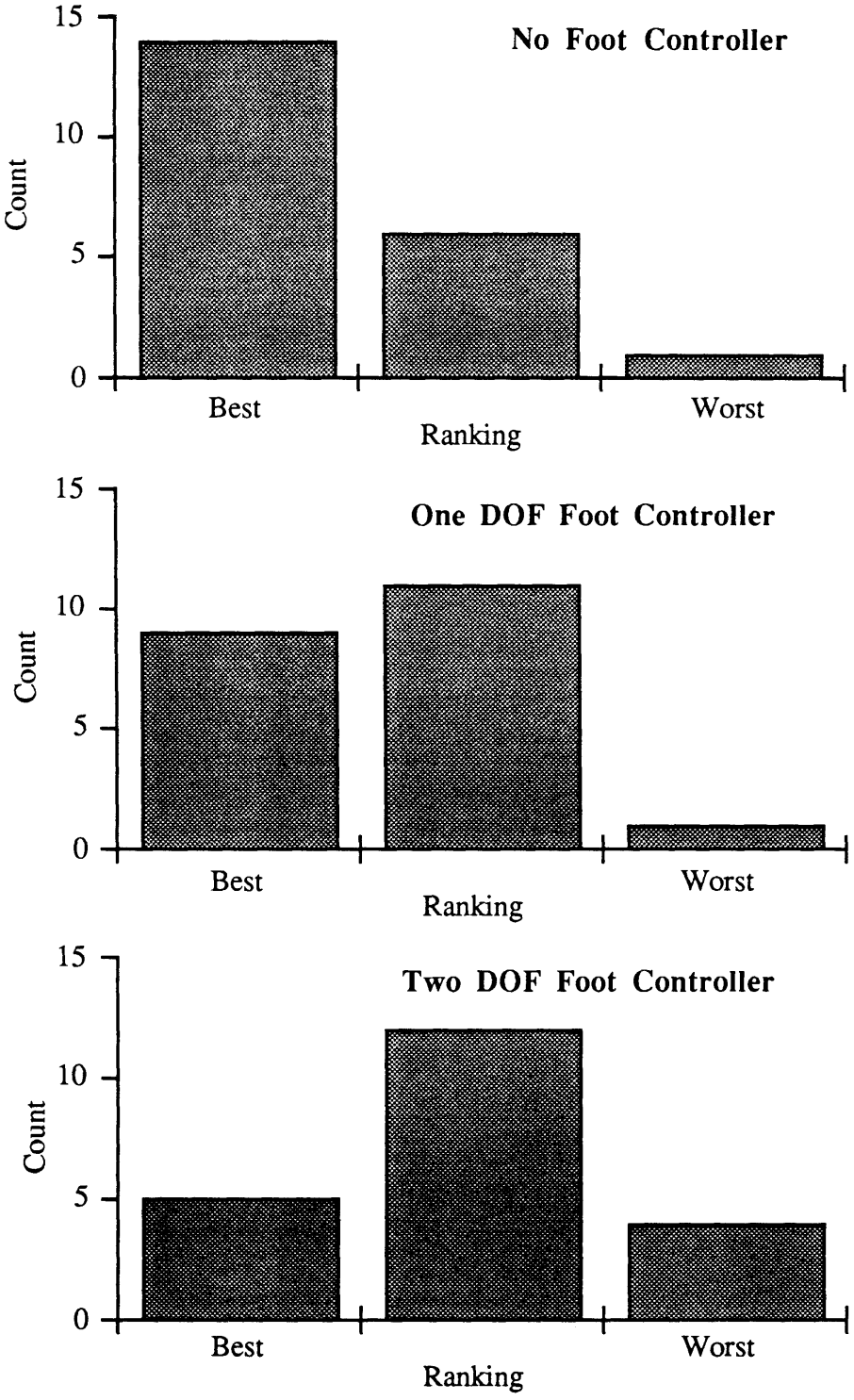

Figure 4.5 Distribution of Pilot Rankings of Each Control Device Configuration for the Position Hold Task. These rankings were determined from pilot opinion ratings (ties were given the same ranking).

4.3.2 Performance Evaluation

Evaluation of the configurations was also based on performance parameters. For both tasks performance was based on command effort and velocity. Additionally performance was based on on displacement error in the position hold task and time of completion in the reorientation task.

COMMAND EFFORT: For both tasks command effort was considered important because it would represent fuel expenditure in a real world situation. In space activities fuel mass is a critical cost driver. A configuration that resulted in a smaller command effort would require less fuel and would therefore be a superior choice. It is assumed, also, that if attention sharing were easier with the foot controller then the command efforts would be smaller for a given task. This assumption is based on the fact that poor attention sharing adds lag and therefore increases the tendency for overshoot and for unnecessary extra compensation.

A good evaluation of handling qualities is obtained from the total command effort because this measure reflects the overall performance. Total command effort is the sum of the individual command efforts for each of the degrees of freedom and represents the total fuel expenditure for the task. The command efforts for the reorientation task were smaller than the command efforts for the position hold task.

The total command efforts for the reorientation task were not significantly different for the three configurations. For the position hold task, the distribution of the total command efforts for each of the configurations are shown in Figure 4.6. The mean values for each are indicated to allow easy comparison. Statistical comparison of the total command effort for each of the configurations reveals that there is a greater than 90% chance that the difference is due to the configurations. By contrasting the individual configurations, the two DOF foot controller configuration was found to be significantly

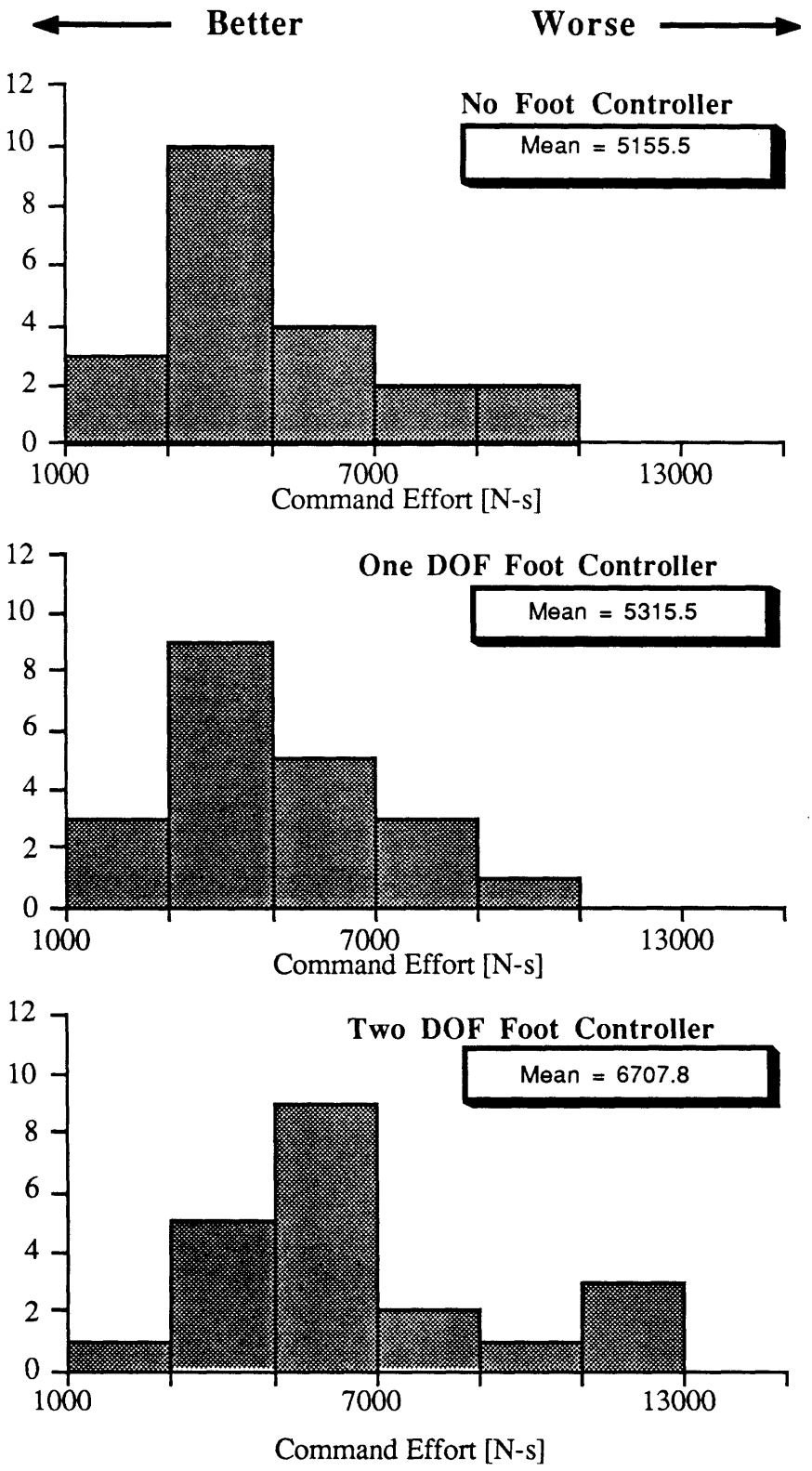

Figure 4.6 Histogram of Integrated Command Magnitudes for Each of the Control Device Configurations for the Position Hold Task. Command Effort is an indication of fuel expenditure and workload.

worse than the no foot controller configuration, there was no significant difference between the one DOF foot controller configuration and the traditional configuration.

By examining the results for pitch and roll command efforts, the differences between the control devices and command assignments can be analyzed more carefully. For both tasks, the roll command effort was not significantly different for any of the configurations. This result indicates that fuel expenditure due to roll commands would be approximately the same for each of the configurations.

In the reorientation task the differences between the pitch command efforts for each of the configurations were not significant. In the position hold task both the no foot controller configuration and the one DOF foot controller configuration had significantly lower pitch command efforts than the two DOF foot controller configuration. Furthermore there was no significant difference between the no foot controller configuration and the one DOF foot controller configuration. This result indicates that fuel expenditure due to pitch commands would be significantly worse for the two DOF foot controller configuration.

VELOCITY: Velocity was considered important for different reasons in each task. Maximum absolute value of the velocity is important for the reorientation task because it is an indication of the ease with which the pilot can command the vehicle. It is assumed that confident and comfortable pilots would command more authoritatively and fly the vehicle faster; a higher velocity would hence signify a better configuration. This assumption was supported by the fact that the better pilots generally had shorter completion times in both the preliminary evaluation task and the reorientation task. Alternatively, velocity can be considered an indicator of the bandwidth of the closed loop, man-vehicle system: again, the higher the velocity, the higher the bandwidth, and possibly, the better the configuration. Conversely, in the position hold task, a large velocity is an indication of the lack of control the pilot exhibited during the flight; since the object or goal of the task was to remain

stationary; the faster the vehicle was moving, the less success the pilot was having. In this task a lower velocity would denote a better configuration.

Maximum overall velocity was defined as the fastest the vehicle was moving in the direction of any of the principal axes of the fixed reference frame, or the fastest it was rotating around any of the Euler axes, whichever was greater.

Statistical analysis of the maximum overall velocity for the reorientation task did not indicate any significant difference between the three configurations. Figure 4.7 shows the distribution of the maximum overall velocities for each of the configurations in the position hold task. Analysis of variance of the data revealed a probability of greater than 90% that the differences between the mean overall velocities for each of the configurations were likely to have been due to the configurations. Contrasting each of the configurations suggested that there was no significant difference between the traditional configuration and the configuration with a one DOF foot controller. When these two controllers were contrasted with the two DOF foot controller configuration the analysis indicated that the two DOF foot controller configuration was significantly worse than the other two.

By looking at the results for pitch and roll velocities the effects of the differences of the configurations can be seen more directly.

Statistical analysis of the maximum roll velocities for both tasks did not indicate any significant difference between the three configurations. Statistical analysis of the maximum pitch velocity data for the reorientation task produced results very similar to the overall velocity results for the position hold task. Figure 4.8 shows the distributions of the maximum pitch velocities for the reorientation task. The probability that the differences in the means were due to the configurations is not quite 95%. However, contrasting the configurations revealed that the configuration with the two DOF foot controller was significantly worse than the configuration with no foot controller. Also there was no

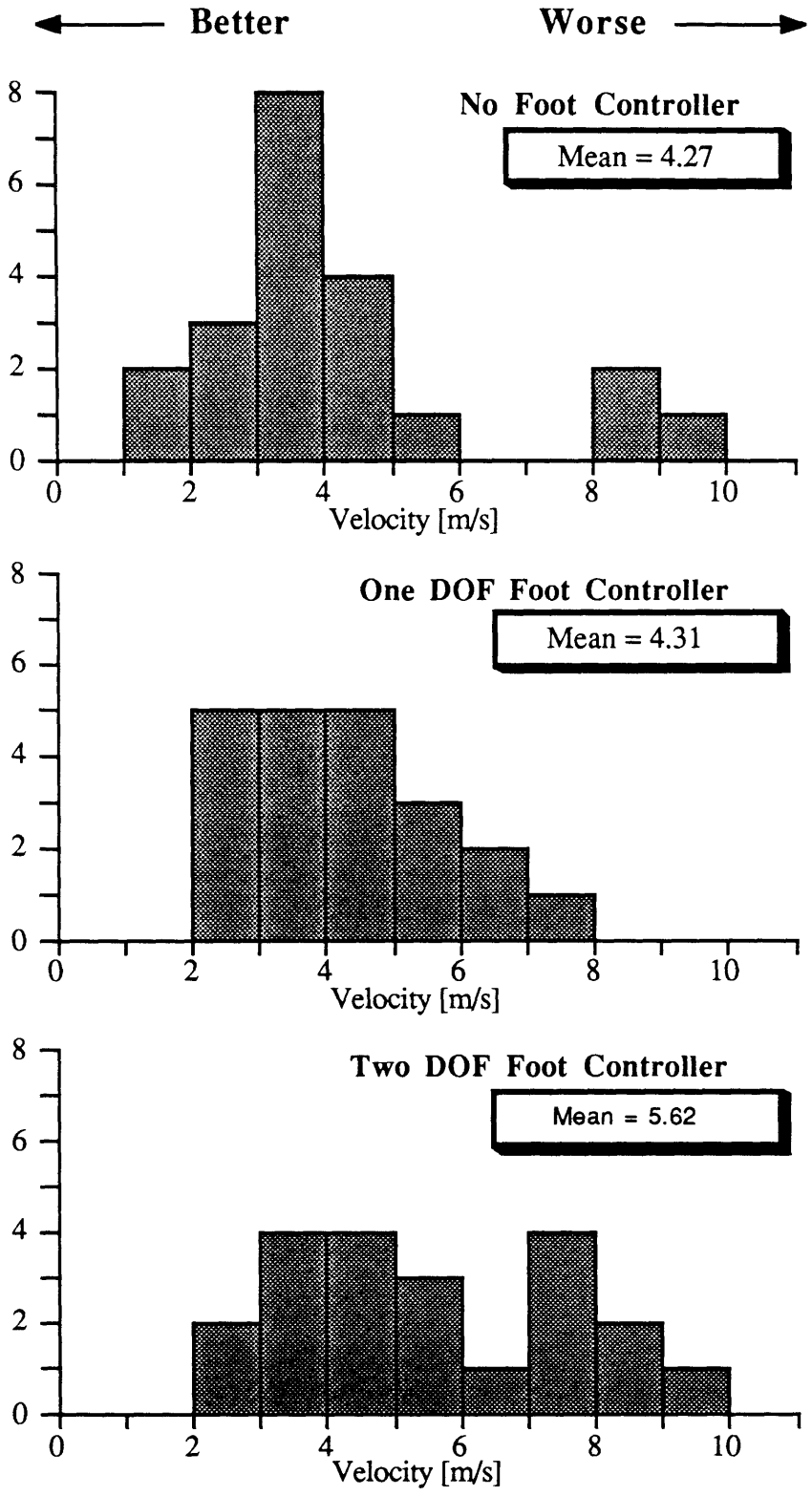

Figure 4.7 Histogram of Maximum Single Degree of Freedom Velocity Magnitudes for Each of the Control Device Configurations for the Position Hold Task. High Maximum Velocity is an indication of poor task performance.

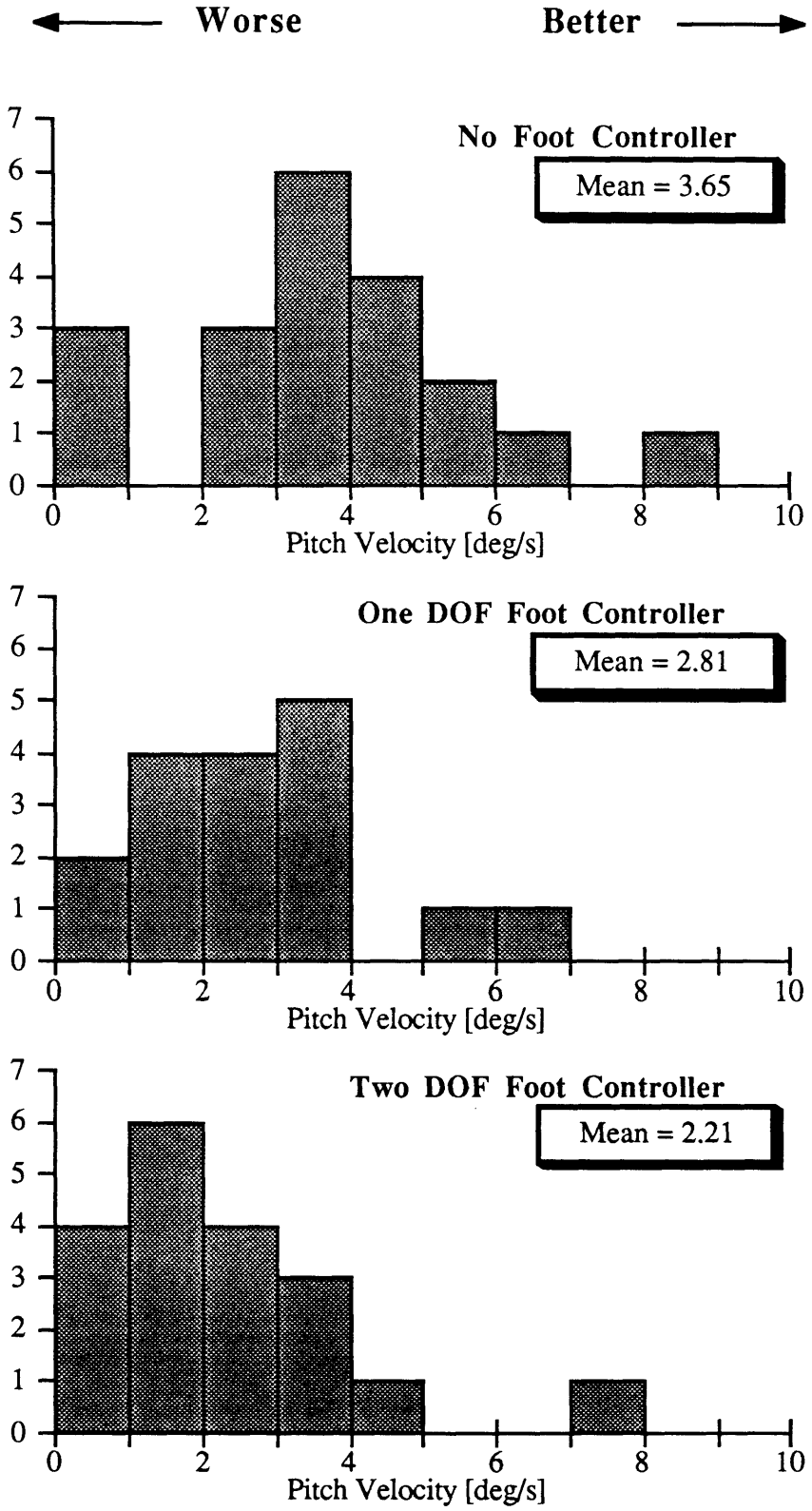

Figure 4.8 Histogram of Maximum Pitch Velocity Magnitudes for Each Control Device Configuration for the Reorientation Task. High velocity is assumed to correspond to good pilot comfort and confidence.

significant difference between the one DOF foot controller configuration and the others. This result indicates that the pilots tended to pitch more slowly with the two DOF foot controller configuration.

The maximum pitch velocity data for the position hold task is shown in Figure 4.9. There is a probability of over 99% that the differences in the means was due to the configurations. This is a highly significant result. Further comparison of the configurations showed that the one DOF foot controller configuration was significantly better than the two DOF foot controller configuration, and that there was no significant difference between the one DOF foot controller configuration and the traditional configuration. This result indicates that the pilots were less successful with the two DOF foot controller configuration.

DISPLACEMENT ERROR: In the position hold task displacement error was another indicator of the pilots' success at maintaining the vehicle's position in the presence of disturbances. A smaller displacement error would suggest a better configuration. Figure 4.10 shows the distributions of the maximum displacement errors for each of the control device configurations. Statistical comparison of displacement errors revealed that there was only approximately a 90% probability that the difference wasn't due to chance. Contrasting the different configurations indicated that there was no significant difference between the no foot controller configuration and the one DOF foot controller configuration.

COMPLETION TIME: It was originally thought that completion time would be an indication of the ease and confidence with which the pilot had commanded the vehicle in the reorientation task. While many factors may have contributed to the pilot's ease and confidence, completion time turned out to be more of an indication of the pilot's skill and style than an evaluation of the configurations. The talented pilots generally had shorter

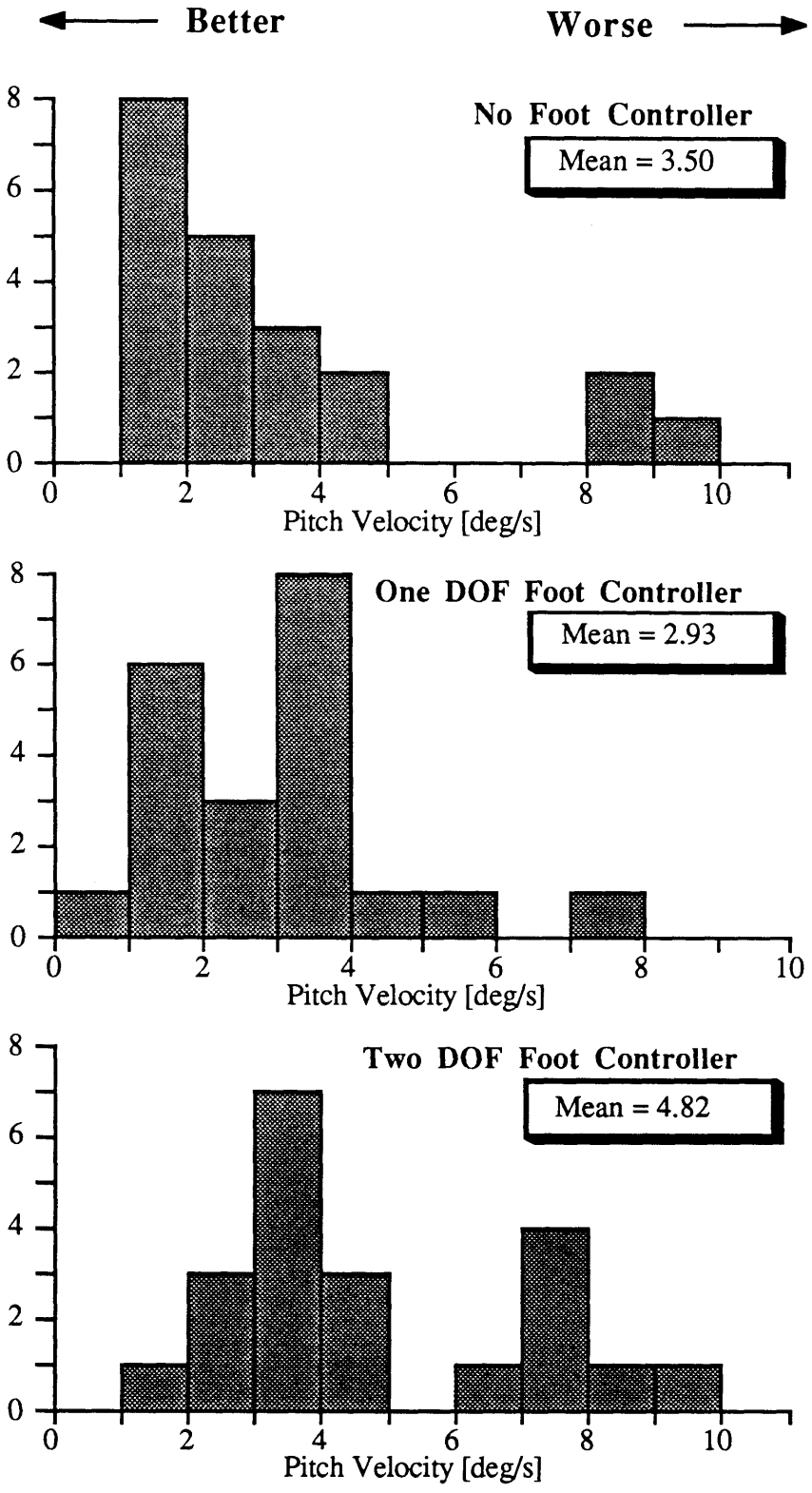

Figure 4.9 Histogram of Maximum Pitch Velocity Magnitudes for Each Control Device Configuration for the Reorientation Task. High velocity corresponds to poor task performance.

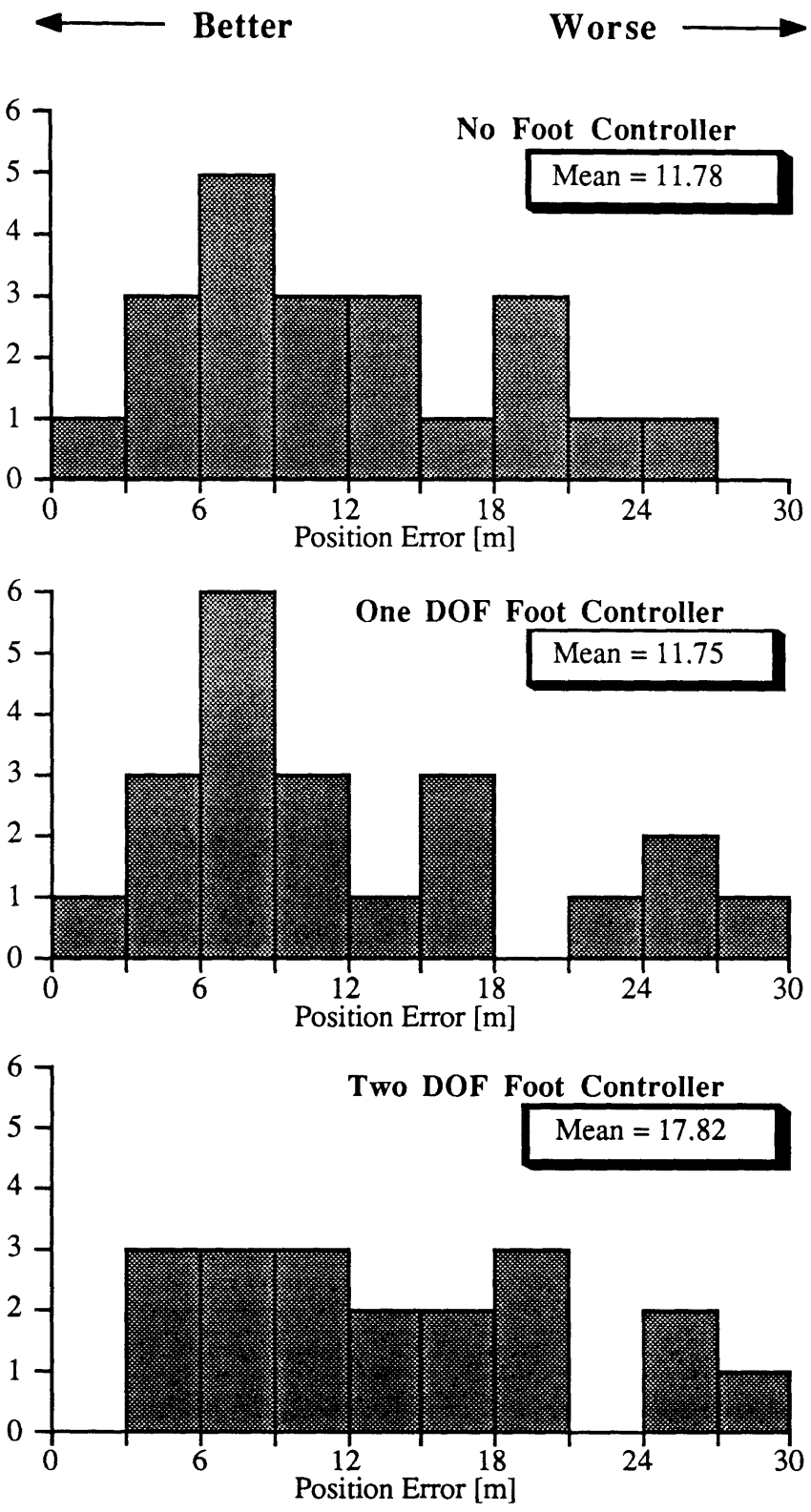

Figure 4.10 Histogram of Maximum Displacement Errors for Each Control Device Configuration for the Position Hold Task. High Displacement Error is an indication of poor task performance.

completion times, and the less adept pilots had longer completion times, regardless of configuration.

4.3.3 Summary

A survey of the data analysis reveals patterns associated with each of the tasks. For the reorientation task pilots preferred the traditional configuration, but the performance evaluation did not show any significant difference between the configurations, except in the case of pitch velocity.

The pilot evaluation result for the reorientation task suggests that the workload for the configurations with the foot controller was higher than that for the traditional configuration. Since the pilots typically broke the flight up into three segments, one for each of the degrees of freedom, it is not likely that high attentional demands was the source of the high workload.

Since the reorientation task showcases the differences in the rotational command assignments the performance evaluation result suggests two things: 1) that the differences in the individual devices did not contribute significantly to the command effort, and 2) that the pilots ability to command with his feet was not significantly different than his ability to command with his hands, except in the case of pitch commands. It is important to note that neither the differences in the control devices nor the pilots' ability to command are the same as the attention sharing characteristics of the configurations.

The position hold task results can be used to compare the attention sharing characteristics of each of the configurations because the task is very demanding and requires the pilot to command all six degrees of freedom. None of the evaluation parameters indicate any significant difference between the traditional configuration and the

one DOF foot controller configuration; and all of the evaluation parameters argue that the two DOF foot controller configuration is undesirable.

In general the results indicate that the foot controller configurations were not superior to the traditional configuration, and that the pilots especially had difficulty with pitch commands when assigned to the foot controller. For the position hold task the traditional configuration and the one DOF foot controller configuration produced similar results and it seems likely that with an improved foot controller the hypothesis would be supported. The next section discusses sources of error which probably contributed to the poor performance of the foot controller configurations.

4.4 Error Analysis

There were four primary sources of error: spurious commands, problems with the foot controller, problems with the display, and other equipment related problems. All of these error sources would have tended to bias the data against the foot controller configurations or have equal effect on all three of the configurations.

SPURIOUS COMMANDS: Examination of the data showed that in over 50% of the flights there were spurious commands lasting for one time increment that were too large to have been generated by the operator. The magnitude of the commands was always approximately 300% of the normal maximum commandable force or torque and they did not occur at any regular interval. Figure 4.11 illustrates example of spurious commands affecting the time histories for the reorientation task. These spurious commands would have resulted in a higher total command effort and would have increased the velocities slightly, which would be especially noticeable in the reorientation task. However, the spurious commands occurred in flights using all three configurations so the net result in the comparison would not have been apparent.

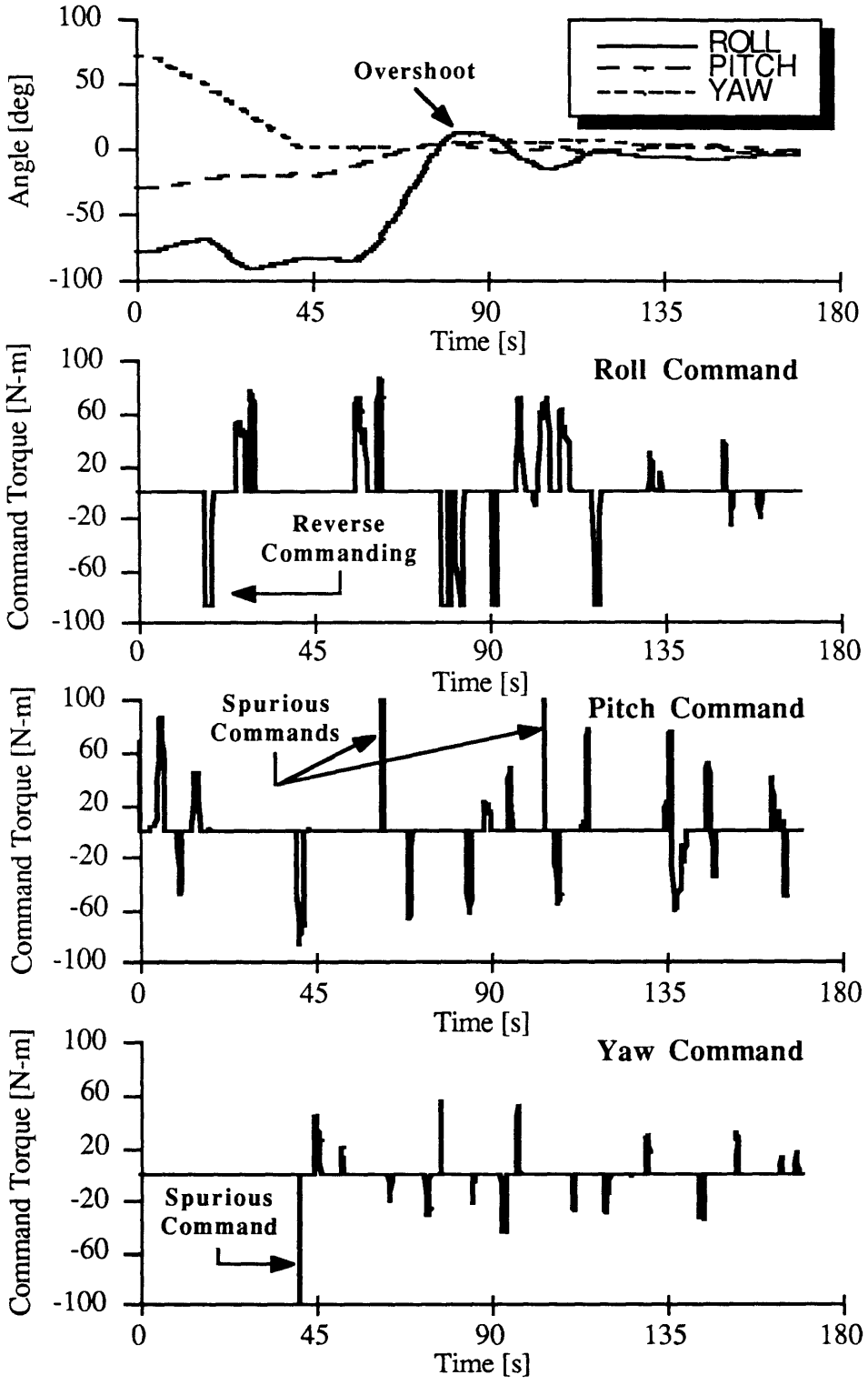

Figure 4.11 Illustration of Overshoot, Spurious Commands, and Reverse Commanding in the Reorientation Task. The pilot was using the One DOF Foot Controller configuration. Notice the significant effect the spurious yaw command has on the yaw angle. (run = 2, subject = "b", initial condition set = 5)

The source of this problem was found to be a serial communications protocol setting that caused the serial port drivers on the control station computer to expand DEL (ASCII 127) to backspace-space-backspace (ASCII 8-32-8). As a result the commands read by the simulation software were incorrect. Occasionally this problem affected the commands in more than one degree of freedom at a time.

Because the spurious commands typically lasted for only one time increment, their effect on the vehicle's motion remained small as long as they did not occur frequently. In the reorientation task, the flights which had spurious commands occurring more than three times were not used in the data analysis. In the position hold task spurious commands were not considered as much of a problem because they were, for the most part, indistinguishable from the random disturbance forces. Although, they required additional reaction from the pilots, the spurious commands were averaged out of the position hold flights before the data was reduced because their magnitude was high compared to the disturbance forces.

FOOT CONTROLLER PROBLEMS: During the experiments the test subjects expressed two concerns about the foot controller, one of which affected the commands. The most common complaint was that it was difficult to sense how much command was being applied. The primary effect that this problem had on the flights was that the pilots occasionally over compensated in response to undesired motion; they would think the foot controller wasn't working until the vehicle's acceleration or deceleration was large. This over compensation in turn contributed to the occurrence of overshoot. The sample flights shown in Figure 4.1 and 4.11 include instances of overshoot. The over compensation problem was further exacerbated by the fact that in some attitudes acceleration and deceleration were difficult to detect. The implications of overcompensation in the data analysis are that pitch and roll command efforts and displacement errors could be expected to be larger for the configurations which use the foot controller. It's possible that the two

DOF foot controller configuration is not significantly worse than the traditional configuration and that the one DOF foot controller configuration is actually better.

The second problem with the foot controller related to pilot discomfort and only occurred with the test subjects who had small feet. The placement of the force sensors in the foot controller made it easier for people with medium-size or large feet to use the foot controller effectively. Observations of the pilots with small feet showed that they compensated for the problem by moving their feet around on the foot controller. One pilot used the balls of her feet for both positive and negative pitch by sliding her feet back and forth as needed. Pilot discomfort associated with the foot controller would have contributed most significantly to the workload. It is also very likely that pilot prejudices affected the pilot ratings. Most pilots expressed dissatisfaction with a control device configuration at one time or another.

DISPLAY PROBLEMS: Another problem that affected the commands was associated with the display. The pilots frequently seemed to experience the common visual illusion that the display was moving and the vehicle was fixed. As a result they would affect commands as if to move the grid wall and cube, rather than move the vehicle with respect to the grid wall and cube. For example, the pilots would react to a left roll motion (the grid wall and cube rolling to the right) by commanding the vehicle further to the left, thinking that they were moving the grid wall and cube to the left. The pilots usually realized their mistake quickly, but that did not prevent it from occurring again. The impact of this problem was that the pilots would have to then correct for their mistake and therefore would increase the command effort. In the position hold task, reverse commanding may have resulted in increased displacement error as well as command effort. Reverse commanding effects can be seen in Figure 4.11.

Over commanding due to foot controller problems may have also increased the velocities in pitch and roll making the one and two DOF foot controller configuration seem worse than they really are. So it is quite possible that the one DOF foot controller configuration is actually superior to the traditional configuration and that the two DOF foot controller configuration is only slightly worse.

Although the display was designed to provide motion cues in all six degrees of freedom, the pilots had some difficulty detecting motion in certain attitudes. Pilots would frequently mistake a sideways translation for a yaw motion. Also, as in the reorientation task, when the pilots were not facing the grid wall dead on, the relationship between the rotational commands and the vehicle's rotation with respect to the grid wall presented the operator with some problem.

It was also observed that the pilots got disoriented easily, especially when commanding more than one degree of freedom at a time. This disorientation was caused most likely by the complexity of the transformation between the commands in the vehicle coordinate frame and the vehicle's motion in the fixed reference frame. In the case of one pilot the resolution of the display interfered with flying slowly. Disorientation and motion detection difficulties were not a function of the configurations, however may have increased pilot noise and increased the ambiguity of the results.

GENERAL EQUIPMENT PROBLEMS: It was observed that the control station configuration worked quite well for the pilots whose height was in the range of about 5'6" to 6'. The pilots who were shorter than 5'6" could only reach the foot controller if they sat toward the edge of the chair. The pilots who were taller than 6' would sometimes bang their knees on the underside of the hand controller mounting. The pilots seemed to adjust as needed and this problem is not thought to have any significant effect on the results. In general, pilot discomfort would increase pilot stress and workload and could in turn affect

the pilot's rating of the flight. At worst the discomfort or awkwardness may have slightly lowered the pilot evaluations of the configurations which used the foot controller.

Another problem relating to discomfort was associated with the helmet mounted display. The helmet's counter-weight at the back of the operator's head pulled the top of the goggles into the operator's forehead. This problem was lessened by carefully adjusting the helmet for each pilot and by giving the pilots ample time to position the helmet comfortably before every flight. Since this problem did not relate specifically to any of the control device configurations, it was not expected to be significant.

Another problem with the control station was that the hand controller mounting and the controller cart were not completely secure. Occasionally, when the operator would exert substantial force on the hand controllers, the mounting would tip or the controller cart would move. The motion was small, however, and did not seem to interfere with the pilots' ability to fly the simulation.

The only effect of any software problems was procedural. The software had been set up so that the program did not have to be restarted for each flight; it could simply be reset. However, in the position hold task, not all of the variables were reset properly so the experiment procedure was adjusted slightly to allow stopping and restarting of the program before each flight. This problem did not have any significant effect on the results.

Chapter 5. Conclusion

5.1 Summary of Results

The results of these experiments did not conclusively support the hypothesis that attention sharing between the hands and feet is superior to attention sharing between the hands alone. It is important to note, however, that the results did not overwhelmingly dispute the hypothesis either and there were considerable factors that may have increased the ambiguity of the data. Furthermore, interpretation of the attention sharing characteristics of the configurations revealed some interesting trends and patterns which did illuminate the configurations' impact on the handling qualities of the vehicle.

1) The attention sharing characteristics of the traditional configuration and the configuration with roll assigned to the foot controller were not found to be significantly different, and it is possible that an improved experiment would reveal that the one DOF foot controller configuration was actually superior to the traditional configuration. Suggestions for improving the experiment are discussed in the next section.

2) The ability of the pilots to manipulate the various command devices did not appear to make any significant difference in the handling qualities of the vehicles, except in the case of the foot operated pitch command.

3) The foot controller design was not well suited for pitch commands. When pitch command was assigned to the foot controller the handling qualities of the vehicle were compromised.

5.2 Recommendations for Future Work

The recommendations for future work, described in this section, are divided into two categories. The first category addresses the problems encountered during these experiments and makes recommendations for improvements. The second category suggests additional experiments or investigations which were inspired by observations made during these experiments.

Most of the problems with these experiments were associated with the foot controller design and may have biased the data in favor of the traditional configuration. Because the pilots expressed difficulty in determining how much force they were applying with the foot controller and because they did not have this problem with the hand controllers, the pilots may have developed preferences for the hand controllers. These preferences would not have been related to the attention sharing characteristics of the configurations and would have influenced both the pilot evaluation and the performance evaluation of the configurations.

This problem could be solved by including in the display an indication of the amount of force being applied to the foot controller. A simple analog display, similar to a light meter needle in a camera, could be placed alongside one edge of the image. The information could also be presented to the pilot via an audio signal, where, for example, the pitch of a steady tone changed as the force changed. An alternative to adding a display would be to change the foot controller to a displacement-type controller. Making both the hand and foot controllers displacement type would allow a more direct comparison of

manual control and foot control. Matching the controller types would reduce any interaction between the type of controller and the performance variables which was not related to the attention sharing characteristics.

Because the foot controller had only two degrees of freedom, the configurations which used the foot controller also required the pilot to use both hands. A better comparison of hand-feet and hand-hand attention sharing could be made with a three DOF foot controller. The foot controller configurations demanded three-way attention sharing whereas the traditional configuration only required two-way, and this problem may have biased the results against the foot controller configurations. Any attempt to solve this problem should be carefully considered to address the ramifications of some of the difficulties associated with designing a three DOF foot controller. These difficulties were discussed in Section 2.1.

Other problems were a result of the experiment design and may have contributed to the ambiguity of the results. These experimental design problems involved both the task design and the display design. The reorientation task did not provide for a balanced comparison of the command assignments. The addition of a fourth configuration which had pitch only assigned to the foot controller would allow a better understanding of how the foot controller and the command assignments contribute to the evaluations.

The hold position task was good for forcing the pilot to divide his attention between the degrees of freedom, but it was very demanding, and the associated high workload may have reduced the variance of the pilot evaluation into a tight range. It would be interesting to simplify the task by reducing the total number of degrees of freedom. A total of two or four degrees of freedom evenly divided between translation and rotation would reduce the pilot workload and might better highlight the attention sharing characteristics of the control device configurations.

The problem of reverse commanding was frequently encountered during both tasks and with all configurations. It would be interesting to experiment with reversing the direction of the commands assigned to the foot controller. This would make the control-display relationship compatible rather than the control-vehicle relationship. Also, since the balancing forces a person typically applies with the feet and legs are reaction forces a reverse command configuration would probably be more intuitive.

The remaining recommendations for future work fall under the second category of suggested experiments. Since the pilots typically commanded the vehicle with large discrete commands it would be interesting to investigate whether or not bang-bang (on-off) control would work well for teleoperation. Bang-bang control could be implemented easily with a foot controller by making each degree of freedom behave like an on-off-on switch. The foot controller could be modified to snap into each of the positions to make it clear to the operator that the command had been affected.

Another interesting investigation of the handling qualities of a telerobot could be made by varying display designs. Certain displays would make it easier to determine the motion of the vehicle. Also, an instrument panel could be added to the display to indicate velocity which would reduce the amount of lead equalization required by the operator. Improving the display would also reduce pilot induced noise due to disorientation.

References

[1] Chapanis, Alphonse, *Research Techniques in Human Engineering*, The Johns Hopkins Press, Baltimore, 1959

[2] Chapanis, Alphonse and R.G. Kinkade, "Design of Controls", *Human Engineering Guide to Equipment Design*, Van Cott and Kinkade, editors, 1972

[3] Cooper, George E. and R.P. Harper, "The Use of Pilot Rating in the Evaluation of Aircraft Handling Qualities", NASA TN D-5153, 1969

[4] Fogel, Lawrence J., *Biotechnology: Concepts and Applications*, Prentice-Hall, Englewood Cliffs, 1963

[5] Frost, George, "Man-Machine Dynamics", *Human Engineering Guide to Equipment Design*, Van Cott and Kinkade, editors, 1972

[6] McDonnell, John D., "Pilot Rating Techniques for the Estimation and Evaluation of Handling Qualities", AFFDL-TR-68-76, 1968

[7] McRuer, Duane and D.K. Schmidt, "Pilot-Vehicle Analysis of Multi-Axis Tasks", AIAA Paper No. 87-2538-CP, 1987

[8] McRuer, Duane, et al., "Minimum Flying Qualities Volume II: Pilot Modelling for Flying Qualities Applications", Technical Report No. 1235-1 Preliminary Version, Systems Technology, Inc., Hawthorne, CA, 1989

[9] McRuer, Duane and R.E. Magdaleno, "Human Pilot Dynamics with Various Manipulators", AFFDL-TR-66-138, 1966

[10] Rowley, Vicky M., Effects of Stereovision and Graphics Overlay on a Teleoperator Docking Task, Masters Thesis, MIT, 1989

[11] Woodson, Wesley E., and D.W. Conover, *Human Engineering Guide for Equipment Designers*, 2nd Ed.,University of California Press, Berkeley, 1964

[12] Young, Laurence R., "Human Control Capabilities", *Bioastronautics Data Book*, NASA SP-3006, Parker and West, editors, 1973

Appendix A.

Control Station and Simulation Software

A1. Control Station Software

```
/*      iriscom.c                                                    */
/*      Anna G. Cinniger                                             */
/*      8/13/90                                                      */
/*      A preliminary program to communicate with the iris           */

#include <stdio.h>
#include <dev.h>
#include "iris.h"

#define         RED             0x8440
#define         CYAN            0x8340
#define         GREEN           0x8240

main()
{
        char monitor;

        screen_init();                  /*      Prepare screen for display
        */

        comm_init();                    /*      Get serial port ready
        */

                term_printf(9,3,CYAN,"Run IRIS graphics program");

        hc_read();                      /*      Read Rotational handcontroller
        */
}
```

```
/*************************************************************/
/*      iris_funcs.c                                         */
/*      Anna G. Cinniger                                     */
/*      8/15/90                                              */
/*      Functions for use with iriscom.c                     */
/*************************************************************/

#include <stdio.h>
#include <dev.h>
#include <math.h>
#include "newreg74.h"
#define data_74
#include "newpc74.h"
#include "iris.h"

#define         RED             0x8440
#define         CYAN            0x8340
#define         GREEN           0x8240
#define         BRIGHT          0x2
#define         x_chan          5
#define         y_chan          3
#define         z_chan          4
#define         r_chan          0
#define         p_chan          1
#define         yaw_chan        2

#define         TRUE            1

static          FILE *serial_r;
static          FILE *serial_w;
extern int      Error_status;

/*************************************************************************/
/*      screen_init() is a function to set up the computer screen        */
/*                              displaying information                   */

void screen_init()
{
        int     e_attr, c_attr, s_attr, d_attr;
        int     shadow, blank;

        set_option(stdout,get_option(stdout) & ~(EDIT|ECHO));

        if (term_load(stdout))
        {
                term_clear(0);
                term_box_fill(0,0,80,25,(0x80 | 7) << 8,0,0xdb);
                e_attr= (4 | 0x80) << 8 | 0;
                c_attr = (0x80 | 2) << 8;
                s_attr = (0x80 | 5) << 8;
                d_attr = (0x80 | 1) << 8;
                blank = 0x80 << 8;
                shadow = 0x80 << 8 | 0x2;
```

```
                term_box_fill(2,2,37,4,shadow,0,0xdb);      /* Status Box */
                term_box_fill(1,1,36,4,blank,1,0xdb);
                term_box(1,1,36,4,s_attr,0,0xdb);

                term_box_fill(8,2,37,16,shadow,0,0xdb);     /* Data Box */
                term_box_fill(7,1,36,16,blank,1,0xdb);
                term_box(7,1,36,16,d_attr,0,0xdb);

                term_box_fill(2,41,38,6,shadow,0,0xdb);     /* Error Box */
                term_box_fill(1,40,37,6,blank,1,0xdb);
                term_box(1,40,37,6,e_attr,0,0xdb);

                term_box_fill(10,41,38,14,shadow,0,0xdb);           /* Command Box */
                term_box_fill(9,40,37,14,blank,1,0xdb);
                term_box(9,40,37,14,c_attr,0,0xdb);
        }
}


/*****************************************************************/
/*      comm_init() is a function to open the device (serial port)       */
/*                          and set the stty parameters                  */

void comm_init()
{
   struct stty_entry{
                char    mtype;
                char    stty_devno;
                char    stty_type;
                unsigned        stty_baud;
                char    stty_parity;
                char    stty_stop_bits;
                char    stty_data_bits;
        }buf;

        serial_r = fopen("$mdm","r");
        serial_w = fopen("$mdm","w");

        if (!serial_r || !serial_w)
        {
                term_printf(3,42,RED,"serial file(s) could not be opened");
                exit(-1);
        }

        set_option(serial_r,get_option(serial_r) & ~(EDIT|ECHO|ETAB|ERS));
        set_option(serial_w,get_option(serial_w) & ~(EDIT|ECHO|ETAB|ERS));

        if (tty_get_stty(serial_r, &buf))
        {
                term_printf(3,42,RED,"serial port parameters not got");
                exit(-1);
        }
        buf.stty_baud = 19200;
```

```
        buf.stty_parity = 0;
        buf.stty_stop_bits = 1;
        buf.stty_data_bits = 8;
        if ((tty_set_stty(serial_r,&buf)) || (tty_set_stty(serial_w,&buf)))
        {
                term_printf(3,42,RED,"serial port parameters not set");
                exit(-1);
        }
}


/*****************************************************************/
/*      console_init is a function to prepare the monitor screen */
/*                              for display                      */

void console_init(monitor)
char monitor;
{
}



/*****************************************************************/
/*      s_write is a function to write data to the serial port   */
/*                                                               */

void s_write(x,y,z,roll,pitch,yaw)
unsigned char  x[2], y[2], z[2];
unsigned char  roll[2], pitch[2], yaw[2];
{
        fput(&x[0], 1, serial_w);
        fput(&x[1], 1, serial_w);
        term_printf(12,60, GREEN,"%3d   %3d  ", (int) x[0], (int) x[1]);
        fput(&y[0], 1, serial_w);
        fput(&y[1], 1, serial_w);
        term_printf(13,60, GREEN,"%3d   %3d  ", (int) y[0], (int) y[1]);
        fput(&z[0], 1, serial_w);
        fput(&z[1], 1, serial_w);
        term_printf(14,60, GREEN,"%3d   %3d  ", (int) z[0], (int) z[1]);
        fput(&roll[0], 1, serial_w);
        fput(&roll[1], 1, serial_w);
        term_printf(15,60, GREEN,"%3d   %3d  ", (int) roll[0], (int) roll[1]);
        fput(&pitch[0], 1, serial_w);
        fput(&pitch[1], 1, serial_w);
        term_printf(16,60,GREEN,"%3d    %3d  ",(int) pitch[0], (int) pitch[1]);
        fput(&yaw[0], 1, serial_w);
        fput(&yaw[1], 1, serial_w);
        term_printf(17,60,GREEN,"%3d    %3d  ",(int) yaw[0], (int) yaw[1]);
        fflush(serial_w);
}


/*****************************************************************/
```

```
/*      hc_read()       is a function to read the hand controller input          */
/*                              from the a2d board                               */

void hc_read()
{
        unsigned char           roll[2], pitch[2], yaw[2];
        unsigned char           x[2], y[2], z[2];
        unsigned char           r_offset[2], p_offset[2], yaw_offset[2];
        unsigned char           x_offset[2], y_offset[2], z_offset[2];
        unsigned char           c = ' ';
        int                     i,k,n,l;
        int                     xc, yc, zc, rollc, pitchc, yawc;
        int                     r_offsetc, p_offsetc, yaw_offsetc;
        int                     x_offsetc, y_offsetc, z_offsetc;
        float                   xcom, ycom, zcom, rollcom, pitcom, yawcom;

        base_74 = 0x300;

        k=0;

        init();
        clk_md(0);                      /* Single conversion upon ctrlstat load  */
        ad_clock(0x08);                 /* Clock circuit operating freqency = 60KHz */

        i_ad_in(x_chan, x_offset);
        i_ad_in(y_chan, y_offset);
        i_ad_in(z_chan, z_offset);
        i_ad_in(r_chan, r_offset);
        i_ad_in(p_chan, p_offset);
        i_ad_in(yaw_chan, yaw_offset);

        x_offsetc= (((int) x_offset[0]) << 8) | ((int) x_offset[1]);
        y_offsetc= (((int) y_offset[0]) << 8) | ((int) y_offset[1]);
        z_offsetc= (((int) z_offset[0]) << 8) | ((int) z_offset[1]);
        r_offsetc= (((int) r_offset[0]) << 8) | ((int) r_offset[1]);
        p_offsetc= (((int) p_offset[0]) << 8) | ((int) p_offset[1]);
        yaw_offsetc= (((int) yaw_offset[0]) << 8) | ((int) yaw_offset[1]);

        term_printf(10,41,GREEN | BRIGHT,"          COMMANDS");
        term_printf(12,43,GREEN,"X:  ");
        term_printf(13,43,GREEN,"Y:  ");
        term_printf(14,43,GREEN,"Z:  ");
        term_printf(15,43,GREEN,"ROLL:  ");
        term_printf(16,43,GREEN,"PITCH:  ");
        term_printf(17,43,GREEN,"YAW:  ");

        k=1;

        while (i > -1)
        {
                l=1;

                while (c != '~' && c != '!')
                {
```

```
                term_printf(3,42,RED,"Waiting for IRIS");
                term_printf(4,42,RED,"%d",l++);
                c=getc(serial_r);
                term_printf(4,45,RED,"%c %d",c, (int) c);
        }

        if (c == '!')
        s_write(x_offset,y_offset,z_offset,r_offset,p_offset,yaw_offset);
        else if (c == '~')
        {
                i_ad_in(x_chan, x);
                i_ad_in(y_chan, y);
                i_ad_in(z_chan, z);
                i_ad_in(r_chan, roll);
                i_ad_in(p_chan, pitch);
                i_ad_in(yaw_chan, yaw);

                s_write(x,y,z,roll,pitch,yaw);
        }

        c=' ';

        xc= (((int) x[0]) << 8) | ((int) x[1]);
        yc= (((int) y[0]) << 8) | ((int) y[1]);
        zc= (((int) z[0]) << 8) | ((int) z[1]);
        rollc= (((int) roll[0]) << 8) | ((int) roll[1]);
        pitchc= (((int) pitch[0]) << 8) | ((int) pitch[1]);
        yawc= (((int) yaw[0]) << 8) | ((int) yaw[1]);

        if ( fabs( xcom=(xc-x_offsetc)*5.0/2048) < 0.2)
                xcom = 0;
        if (fabs(ycom = (yc-y_offsetc)*5.0/2048) < 0.3)
                ycom = 0;
        if (fabs(zcom = (zc-z_offsetc)*5.0/2048) < 0.2)
                zcom = 0;
        if ( fabs( rollcom=(rollc-r_offsetc)*5.0/2048) < 0.2)
                rollcom = 0;
        if (fabs(pitcom = (pitchc-p_offsetc)*5.0/2048) < 0.3)
                pitcom = 0;
        if (fabs(yawcom = (yawc-yaw_offsetc)*5.0/2048) < 0.15)
                yawcom = 0;

        term_printf(12, 50, GREEN,"%5.2f", xcom);
        term_printf(13, 50, GREEN,"%5.2f", ycom);
        term_printf(14, 50, GREEN,"%5.2f", zcom);
        term_printf(15, 50, GREEN,"%5.2f", rollcom);
        term_printf(16, 50, GREEN,"%5.2f", pitcom);
        term_printf(17, 50, GREEN,"%5.2f", yawcom);

        term_printf(2,42,RED,"Did loop %d times",k++);
}
term_printf(18,3,CYAN, "Closing files");
fclose("serial_r");
fclose("serial_w"); }
```

```
/*      File:  pc74s.c
Copyright (C) 1988      A.D. McGuffog
*************************************************************************

                        Module description :

    This module provides simple I/O functions, which are not language
            dependant. These are the following :

                    Initialization functions.
                    Clock functions.
                    Digital I/O functions.
                    Simple polled I/O A/D input functions.

      This module can be compiled by almost all C compilers, and is
            suitable for use on UNIX/XENIX systems.

*************************************************************************
*************************************************************************

    Modifications :     by Anna G. Cinniger
                        7/26/90  Changed variable names
                                 Re-wrote Clean() and inserted in Init()

                        8/22/90  Added i_ad_in for iris programs

*************************************************************************
*************************************************************************/
#include <stdio.h>
#include "newreg74.h"
#include "newpc74.h"


void    rtc_off()
{
        imr_bits        imr_d;

        imr_d.by = io_in(int_mr);
        imr_d.bi.irq0 = masked;
        io_out(int_mr, imr_d.by);
}


void    rtc_on()
{
        imr_bits        imr_d;

        imr_d.by = io_in(int_mr);
        imr_d.bi.irq0 = enabled;
        io_out(int_mr, imr_d.by);
}


void clean()
```

```
{
        ctrlstat_bits    ctrlstat_d;
        gainchan_bits    gainchan_d;
        register int     i;

/* First we disable everything. */

/*      ctrlstat_d.bi.mode = 0;
        ctrlstat_d.bi.i_en = 0;
        ctrlstat_d.bi.d_en = 0;
        ctrlstat_d.bi.clr_err = 1;
*/

        ctrlstat_d.by = 0x10;                   /* Does the same thing as above */

        io_out(ctrlstat, ctrlstat_d.by);

        io_in(A2Dlow);                          /* Then we clear the A/D done bit. */
        io_in(A2Dhigh);

        w_busy;                                 /* Wait for any current conversion to end. */

/*      Now wait either for 100 uS or for done.
        for (i = 0; ((i < 20) && (!(io_in(ctrlstat) & 0x80))); i++);
*/

        io_in(A2Dlow);                          /* Then we clear the A/D done bit. */
        io_in(A2Dhigh);

        io_out(ctrlstat, ctrlstat_d.by);        /* and clear any error bits. */
}


void init()
{
        ctrlstat_bits           ctrlstat_d;
        gainchan_bits           gainchan_d;
        register int    i;

        ctrlstat_d.by = 0x10;                   /* Clear error and disable */
        io_out(ctrlstat, ctrlstat_d.by);
        io_in(A2Dlow);                          /* Then we clear the A/D done bit. */
        io_in(A2Dhigh);
        w_busy;                                 /* Wait for any current conversion to end. */
        io_in(A2Dlow);                          /* Then we clear the A/D done bit. */
        io_in(A2Dhigh);
        io_out(ctrlstat, ctrlstat_d.by);        /* and clear any error bits. */
        io_out(tmrctr, 0x35);                   /* Sets a frequency of 1 KHz */
}


void clk_md(c_mode)
int c_mode;
{
```

```
        ctrlstat_bits           ctrlstat_d;

        ctrlstat_d.by = io_in(ctrlstat);        /* Preserve the other stuff. */
        ctrlstat_d.bi.mode = c_mode;
        io_out(ctrlstat, ctrlstat_d.by);
}


void    ad_clock(clk_val)
int clk_val;
{
        io_out(tmrctr, clk_val);
}


int     read_clock()
{
        return(io_in(tmrctr));
}


int ad_in(chan, in_val)
int chan;
int *in_val;
{
        ctrlstat_bits           ctrlstat_d;
        gainchan_bits           gainchan_d;

        if ((chan < 16) && (chan > -1)) {
                gainchan_d.bi.chan = chan;
                gainchan_d.bi.gain = gain_74[chan];
                io_out(gainchan, gainchan_d.by);
                w_done;                                 /* Wait for A/D completion. */
                *in_val = (io_in(A2Dhigh) << 8) | io_in(A2Dlow);
                ctrlstat_d.by = io_in(ctrlstat);
                return((ctrlstat_d.bi.error) ? err_74 : ok_74);
                }
        else return(par_74);
}

int i_ad_in(chan, in_val)                       /*  ad_in() for the IRIS programs      */
int chan;                                       /*      doesn't combine high & low     */
char in_val[2];                                 /*    bytes                            */
{
        ctrlstat_bits           ctrlstat_d;
        gainchan_bits           gainchan_d;

        if ((chan < 16) && (chan > -1)) {
                gainchan_d.bi.chan = chan;
                gainchan_d.bi.gain = gain_74[chan];
                io_out(gainchan, gainchan_d.by);
                w_done;                                 /* Wait for A/D completion. */
                in_val[0] = io_in(A2Dhigh);
                in_val[1] = io_in(A2Dlow);
```

```
                ctrlstat_d.by = io_in(ctrlstat);
                return((ctrlstat_d.bi.error) ? err_74 : ok_74);
                }
        else return(par_74);
}

int ccalb(chan)
int chan;
{
        int i, offset, offset_sum;

        for (i=1;i<=10;i++)
        {
                ad_in(chan, &offset);
                offset_sum = offset_sum + offset;
        }
        return(offset_sum/10);
}
```

```
/*************************************************************/
/*      iris.h                                               */
/*      Anna G. Cinniger                                     */
/*      8/15/90                                              */
/*      Function declarations for iriscom.c.  Code for functions */
/*      is in iris_funcs.c                                   */
/*************************************************************/

void screen_init();

void comm_init();

void console_init();

void hc_read();
```

```
/*      File:  pc74.h
Copyright (C) 1988      A.D. McGuffog
*************************************************************************

                        Module description :

                This module provides function and data definitions for the PC-74
                driver system.

                This module can be compiled by almost all C compilers, and is
                suitable for use on UNIX/XENIX systems.

*************************************************************************
*************************************************************************

    Modifications :  by Anna G. Cinniger
                        7/27/90  Eliminated unnecessary items for
                                        preliminary a2dread.c

                        8/22/90  Added i_ad_in for iris programs.

*************************************************************************
*************************************************************************/
#ifndef data_74
extern int              gain_74[16];
extern unsigned         base_74;
#else
int                     base_74;
int                     gain_74[16];
#endif

#define ok_74           0
#define par_74          -1
#define err_74          -2
#define n_comp_74       1


void    init();

void    clean();

void    rtc_off();

void    rtc_on();

int     ad_in();

int     i_ad_in();

void    clk_md();

int     read_clock();

void    ad_clock();
```

```
int     diag();
int     ccalb();
```

```
/*      File:  reg_74.h
Copyright (C) 1988    A.D. McGuffog
*************************************************************************
*************************************************************************

                        Module description :

    This module provides register definitions for both the PC-74 and
            certain PC hardware.

            This module can be compiled by almost all C compilers, and is
            suitable for use on UNIX/XENIX systems.

*************************************************************************
*************************************************************************

    Modifications :   by Anna G. Cinniger
                      7/26/90       Changed some variable names
                                    Removed interrupt & DMA stuff

*************************************************************************
*************************************************************************/


/* Now the def's for the PC-74 itself. */

#define     int_mr      0x21        /* Interrupt mask register. */
#define     masked      1           /* Interrupt masked (also DMA). */
#define     enabled     0           /* Interrupt enabled. */

/* Now we define a bit field for the IMR. */

typedef union {
        int by;
        struct {
                unsigned irq0 : 1;      /* Timer mask. */
                unsigned irq1 : 1;      /* Keyboard/RTC/Pointing device. */
                unsigned irq2 : 1;                  /* Video. */
                unsigned irq3 : 1;                  /* Serial. */
                unsigned irq4 : 1;                  /* Serial. */
                unsigned irq5 : 1;                  /* Fixed disk. */
                unsigned irq6 : 1;                  /* Diskette. */
                unsigned irq7 : 1;                  /* Parallel port. */
                } bi;
} imr_bits;

#define     ctrlstat    base_74                 /* Control/status register */
#define     gainchan    (base_74    + 1)        /* Gain/channel register */

#define     A2Dlow      (base_74    + 2)        /* A/D low byte. R. */
#define     A2Dhigh     (base_74    + 3)        /* A/D high byte. R. */

#define     D2Alow0     (base_74    + 2)        /* D/A 0 low byte. W. */
#define     D2Ahigh0    (base_74    + 3)        /* D/A 0 high byte. W. */
```

```
#define     D2Alow1     (base_74     + 4)     /* D/A 1 low byte. W. */
#define     D2Ahigh1    (base_74     + 5)     /* D/A 1 high byte. W. */

#define     diginp      (base_74     + 6)     /* Digital input port. R. */
#define     digoutp     (base_74     + 6)     /* Digital output port. W. */

#define     tmrctr      (base_74     + 7)     /* Timer/counter. R/W. */

#define     w_done  while (!(io_in(ctrlstat)&0x80))   /* Macro to check for A/D
done. */
#define     w_busy  while ((io_in(ctrlstat)&0x20))    /* Macro to wait until current
conversion done. */

/* Now we define a bit field for the control/status register. */
/* Some of these bits are read, some write and some read/write. */

typedef union {
        int by;
        struct {
                unsigned mode : 2;          /* Clock mode bits. R/W. */
                unsigned i_en : 1;          /* Interrupt enable. R/W. */
                unsigned d_en : 1;          /* DMA enable. R/W. W only in PC-74
mode. */
                unsigned clr_err : 1;       /* Clear error condition. W. */
                unsigned busy : 1;          /* A/D busy. R. */
                unsigned error : 1;         /* Error. R. */
                unsigned done : 1;          /* A/D done. R. */
        } bi;
} ctrlstat_bits;

/* Now we define a bit field for the gain/channel register. */

typedef union {
        int by;
        struct {
                unsigned chan : 4;          /* Channel address. R/W. */
                unsigned  : 2;              /* Not used. */
                unsigned gain : 2;          /* Gain bits. R/W. */
        } bi;
} gainchan_bits;
```

A2. Simulation Software

```
/* sim.c */
/* by Matthew Machlis */
/* updated 3/6/91 */

#include "sim.h"

main(argc, argv)
int argc;
char **argv;
{
int i;

        if (sys_setup(argc, argv) == 1) exit(0);

        ginit();

        mmode(MVIEWING);
        resetls(TRUE);
        linewidth(3);
        backface(FALSE);
        zbuffer(FALSE);
        doublebuffer();
        cursoff();
        gconfig();

        init_queue();
        init_colors();

        if (pcmode == '1') init_pc();
        if (htmode == '1') init_vpl();

        while (inp_next() && !getbutton(ESCKEY)) {
                outpfile = strcmp(".",outfname);
                init_env();
                init_monitor();
                dynamics(1);

                while (!getbutton(NKEY) && !endflag) {
                        get_control();
                        if (htmode == '1') get_track();
                        dynamics(0);
                        endflag = set_flag();
                        writemask(4095);
                        color(BLACK);
                        clear();
                        if (vmode == '1' || vmode == '2') {

                        /* Helmet mounted display view */
```

```
        /* Left eye (red) */
        pushmatrix();
        writemask(63);
        window (ocular - (SCREENSEP/2 + SCREENWIDTH),
                ocular -SCREENSEP/2, -SCREENDN,
                SCREENUP, SCREENDIST, FARCLIP);
        if (vmode == '2') translate(ocular,0.,0.);
        draw_env();
        popmatrix();

        /* Right eye (blue) */
        pushmatrix();
        writemask(4032);
        window (SCREENSEP/2 - ocular, (SCREENSEP/2 +
                SCREENWIDTH) - ocular, -SCREENDN,
                SCREENUP, SCREENDIST, FARCLIP);
        if (vmode == '2') translate(-ocular,0.,0.);
        draw_env();
        popmatrix();

} else if (vmode=='3' || vmode=='5') {

/* Mono monitor view */

        pushmatrix();
                perspective(500,1.25,0.1,1000.);
                draw_env();
        popmatrix();

} else {

/* Stereo monitor view */

        /* Left eye (red) */
        pushmatrix();
                writemask(63);
                perspective(500,1.25,0.1,1000.);
                translate(0.025,0.,0.);
                draw_env();
        popmatrix();

        /* Right eye (blue) */
        pushmatrix();
                writemask(4032);
                perspective(500,1.25,0.1,1000.);
                translate(-0.025,0.,0.);
                draw_env();
        popmatrix();

}

/* Write text on screen */
```

```
        if ( vmode=='5' || vmode=='6') {

    pushmatrix();
        ortho2(-100.5,100.5,-100.5,100.5);
        writemask(4095);
        color(4095);


        if (mouseflag == 0) {
            cmov2i(-90,-70);
            sprintf(mess,"xcmd= %6.4f",cx);
            charstr(mess);
            cmov2i(-90,-80);
            sprintf(mess,"ycmd= %6.4f",cy);
            charstr(mess);
            cmov2i(-90,-90);
            sprintf(mess,"zcmd= %6.4f",cz);
            charstr(mess);
        } else if (mouseflag == 1) {
            cmov2i(-90,-70);
            sprintf(mess,"roll cmd=       %6.4f",crx);
            charstr(mess);
            cmov2i(-90,-80);
            sprintf(mess,"pitch cmd= %6.4f",cry);
            charstr(mess);
            cmov2i(-90,-90);
            sprintf(mess,"yaw cmd=        %6.4f",crz);
            charstr(mess);
        }
        cmov2i(0,-70);
        sprintf(mess,"xw= %6.4f",xw);
        charstr(mess);
        cmov2i(0,-80);
        sprintf(mess,"yw= %6.4f",yw);
        charstr(mess);
        cmov2i(0,-90);
        sprintf(mess,"zw= %6.4f",zw);
        charstr(mess);
        cmov2i(60,-70);
        sprintf(mess,"roll=     %10.7f",phi*180./PI);
        charstr(mess);
        cmov2i(60,-80);
        sprintf(mess,"pitch= %10.7f",theta*180./PI);
        charstr(mess);
        cmov2i(60,-90);
        sprintf(mess,"yaw=     %10.7f",psi*180./PI);
        charstr(mess);
    popmatrix();

    }

    swapbuffers();
}
```

```
                printf("# skipped screen updates = %d\n",numskip);
                numskip=0;
                writemask(4095);
                color(BLACK);
                clear();
                swapbuffers();
                clear();
                if (outpfile) out_data();
                ringbell();
                while (getbutton(NKEY)) {};
                while (!getbutton(NKEY) && !getbutton(ESCKEY)) {};
                while (getbutton(NKEY)) {};
        }

        if (pcmode == '1') close(pc);
        if (htmode == '1') close(vpl);
        tpon();
        curson();
        qreset();
        setmonitor(HZ60);
        system("blanktime 67000");
        greset();
        gexit();
        exit(0);
}

int set_flag()
/* Set end-of-run flag if appropriate conditions are met */
{
        register int tmpflag=0;
        float v1,v2,v3,w1,w2,w3,val;

        if (loop >= 9990) tmpflag=1;
        else if (pmmode == '2') {
                v1=fabs(psi);  v2=fabs(theta);          v3=fabs(phi);
                w1=fabs(p);             w2=fabs(q);                     w3=fabs(r);
                if ((v1<0.0698) || (v1>(TWOPI-0.0698)))
                        if ((v2<0.0698) || (v2>(TWOPI-0.0698)))
                                if ((v3<0.0698) || (v3>(TWOPI-0.0698)))
                                        if ((w1<0.0349) || (w1>(TWOPI-0.0698)))
                                                if ((w2<0.0349) || (w2>(TWOPI-0.0698)))
                                                        if ((w3< 0.0349) || (w3>(TWOPI-
                                                                0.0698))) tmpflag=1;
        } else if (pmmode == '3') {
                v1 = vec3[rectnum][0];
                v2 = vec3[rectnum][1];
                v3 = vec3[rectnum][2];
                w1 = x - targloc[rectnum+1][0];
                w2 = y - targloc[rectnum+1][1];
                w3 = z - targloc[rectnum+1][2];
                val = v1*w1 + v2*w2 + v3*w3;
                if (val > 0.) {
                        rectt[rectnum] = (float)(inttime - start_time)/(float)HZ;
                        rectd[rectnum++] = fsqrt(w1*w1 + w2*w2 + w3*w3);
```

```
                        if (rectnum > 2) tmpflag=1;
                }
        }
        return(tmpflag);
}

init_env()
/* Initialize environment if necessary */
{
        mass = masses[emode-'1'];
        inertia = inertias[emode-'1'];
        if (emode == '2') init_env2();
        if (dsave != '1') srand48(rndseed);
}

init_env2()
/* Initialize fly-through rectangles */
{
        int i,j;
        float vec1[3], vec2[3], rvec[3], uvec[3];
        float targhvec, targwvec;

        for (i=1;i<NUMTARG-1;i++) {
                for (j=0;j<3;j++) {
                        vec1[j] = targloc[i][j] - targloc[i-1][j];
                        vec2[j] = targloc[i+1][j] - targloc[i][j];
                        vec3[i-1][j] = targloc[i+1][j] - targloc[i-1][j];
                }
                cross_product (vec1, vec2, rvec);
                normalize(rvec);
                cross_product (rvec, vec3[i-1], uvec);
                normalize(uvec);
                for (j=0;j<3;j++) {
                        targhvec = uvec[j] * fcos(targroll[i])
                                        + rvec[j] * fsin(targroll[i]);
                        targwvec = rvec[j] * fcos(targroll[i])
                                        - uvec[j] * fsin(targroll[i]);
                        targv1[i][j] = targloc[i][j] - TARGWD * targwvec
                                        + TARGHT * targhvec;
                        targv2[i][j] = targloc[i][j] + TARGWD * targwvec
                                        + TARGHT * targhvec;
                        targv3[i][j] = targloc[i][j] + TARGWD * targwvec
                                        - TARGHT * targhvec;
                        targv4[i][j] = targloc[i][j] - TARGWD * targwvec
                                        - TARGHT * targhvec;
                        targv5[i][j] = targloc[i][j] - XSIZE * targhvec;
                        targv6[i][j] = targloc[i][j] - XSIZE * targwvec;
                        targv7[i][j] = targloc[i][j] + XSIZE * targhvec;
                        targv8[i][j] = targloc[i][j] + XSIZE * targwvec;
                }
        }
}

draw_env()
```

```
/* Draw environment */
{
        if (htmode == '1') {
                rotate((Angle)(1800.*t2/PI),'x');
                rotate((Angle)(1800.*t1/PI),'y');
        }
        if (bfmode != '1') getmatrix(bodmat);
        if (camht != 0.) translate(0.,-camht,0.);
        rotate(rz,'z');
        rotate(rx,'x');
        rotate(ry,'y');
        translate(0.,0.,-z);
        translate(-x,-y,0.);
        if (emode == '1') draw_env1();
        else if (emode == '2') draw_env2();
        else if (emode == '3') draw_env3();
        else if (emode == '4') draw_env4();
        if (bfmode != '1') {
                loadmatrix(bodmat);
                draw_bodref();
        }
}

draw_bodref()
/* Draw body-fixed references */
{
        register int i;

        color(4095);
        for (i=0;i<=9;i+=3) {
                bgnline();
                        v3f(bfv[i]);
                        v3f(bfv[i+1]);
                        v3f(bfv[i+2]);
                endline();
        }
        if (bfmode == '3') {
                for (i=12;i<=21;i+=3) {
                        bgnline();
                                v3f(bfv[i]);
                                v3f(bfv[i+1]);
                                v3f(bfv[i+2]);
                        endline();
                }
                for (i=24;i<=54;i+=2) {
                        bgnline();
                                v3f(bfv[i]);
                                v3f(bfv[i+1]);
                        endline();
                }
        }
}

draw_env1()
```

```
/* Draw box environment */
{
        color(4095);
        translate(0.,0.,-50.);
        draw_box();
}

draw_box()
/* Draw a box */
{
        bgnclosedline();
                v3s(boxv[0]);
                v3s(boxv[1]);
                v3s(boxv[2]);
                v3s(boxv[3]);
        endclosedline();
        bgnclosedline();
                v3s(boxv[4]);
                v3s(boxv[5]);
                v3s(boxv[6]);
                v3s(boxv[7]);
        endclosedline();

        bgnline();
                v3s(boxv[0]);
                v3s(boxv[4]);
        endline();
        bgnline();
                v3s(boxv[1]);
                v3s(boxv[5]);
        endline();
        bgnline();
                v3s(boxv[2]);
                v3s(boxv[6]);
        endline();
        bgnline();
                v3s(boxv[3]);
                v3s(boxv[7]);
        endline();

        bgnline();
                v3s(boxv[5]);
                v3s(boxv[8]);
                v3s(boxv[6]);
        endline();
}

draw_env2()
/* Draw fly-through rectangles environment */
{
        register int i;

        color(4095);
        for (i=0;i<NUMTARG;i++) {
```

```
                bgnclosedline();
                        v3f(targv1[i]);
                        v3f(targv2[i]);
                        v3f(targv3[i]);
                        v3f(targv4[i]);
                endclosedline();
                bgnline();
                        v3f(targv5[i]);
                        v3f(targv7[i]);
                endline();
                bgnline();
                        v3f(targv6[i]);
                        v3f(targv8[i]);
                endline();
        }
}

draw_env3()
/* Draw dual-grid environment */
{
        register float i;
        float tmpv[3];

        color(4030);
        tmpv[2]= -5.;
        for (i= -5.; i<=5.; i+=0.5) {
                tmpv[1]=i;
                tmpv[0]= -5.;
                bgnline();
                        v3f(tmpv);
                        tmpv[0]=5.;
                        v3f(tmpv);
                endline();
                tmpv[0]=i;
                tmpv[1]= -5.;
                bgnline();
                        v3f(tmpv);
                        tmpv[1]=5.;
                        v3f(tmpv);
                endline();
        }
        color(4095);
        tmpv[2]= -2.5;
        for (i= -5.; i<=5.; i+=0.5) {
                tmpv[1]=i;
                tmpv[0]= -5.;
                bgnline();
                        v3f(tmpv);
                        tmpv[0]=5.;
                        v3f(tmpv);
                endline();
                tmpv[0]=i;
                tmpv[1]= -5.;
                bgnline();
```

```
                        v3f(tmpv);
                        tmpv[1]=5.;
                        v3f(tmpv);
                endline();
        }
}

draw_env4()
/* Draw box plus grid environment */
{
        register short i;
        short tmpv[3];

        color(4030);
        tmpv[2]= -10;
        for (i= -10; i<=10; i++) {
                tmpv[1]=i;
                tmpv[0]= -10;
                bgnline();
                        v3s(tmpv);
                        tmpv[0]=10;
                        v3s(tmpv);
                endline();
                tmpv[0]=i;
                tmpv[1]= -10;
                bgnline();
                        v3s(tmpv);
                        tmpv[1]=10;
                        v3s(tmpv);
                endline();
        }
        color(4095);
        scale(0.05,0.05,0.05);
        translate(0.,0.,-140.);
        draw_box();
}

get_control()
/* Get commands from (Gateway or IRIS mouse) and IRIS keyboard */
{
        static float mxcmd=0., mycmd=0.;
        long xmouse, ymouse, btnid;
        short btndata, *bdata;

        bdata = &btndata;
        if (pcmode == '1') {
                /* Read command input from Gateway */
                get_pc();
                while (qtest() != 0) {
                        btnid = qread(bdata);
                        if (btndata == 1) {
                                switch (btnid) {
                                        case RKEY : dynamics(1);      break;
                                        case PKEY : while (!getbutton(OKEY)) {};
```

```
                                inttime=times(&buffer);         break;
                        case BKEY : if (bfmode=='3') bfmode='1';
                                else if (bfmode=='2') bfmode='3';
                                else bfmode='2';
                                break;
                        case CKEY : if (rcmode=='1') rcmode='2';
                                else rcmode='1';        break;
                        case DKEY : if (dmode=='1') dmode=dsave;
                                else {dmode='1';
                                xdist=ydist=zdist=pdist=qdist=rdist=0.;
                                 }break;
                        case MKEY : markline[marknum++]=loop;   break;
                    }
                }
            }
        }
else {
        /* Get command input from IRIS mouse */
        while (qtest() != 0) {
                btnid = qread(bdata);
                if (btndata == 1) {
                        switch (btnid) {
                                case MIDDLEMOUSE :
                                        mxoff = getvaluator(MOUSEX);
                                        myoff = getvaluator(MOUSEY);
                                        btncmd = 0.0;
                                        if (mouseflag == 0) {
                                                cx = cy = cz = 0.;
                                        } else {
                                                crx = cry = crz = 0.;
                                        }
                                        mouseflag = !mouseflag;
                                        break;
                                case LEFTMOUSE : btncmd = btncmd - 0.1;
                                        break;
                                case RIGHTMOUSE : btncmd = btncmd + 0.1;
                                        break;
                                case RKEY : dynamics(1);        break;
                                case PKEY : while (!getbutton(OKEY)) {};
                                        inttime=times(&buffer);
                                        break;
                                case BKEY : if (bfmode=='3') bfmode='1';
                                        else if (bfmode=='2') bfmode='3';
                                        else bfmode='2';
                                        break;
                                case CKEY : if (rcmode=='1') rcmode='2';
                                        else rcmode='1';
                                        break;
                                case DKEY : if (dmode=='1') dmode=dsave;
                                        else {dmode='1';
                                        xdist=ydist=zdist=pdist=qdist=rdist=0.;
                                         }break;
                                case MKEY : markline[marknum++]=loop;   break;
                        }
```

```
                }
            }
            mxcmd = (float)(getvaluator(MOUSEX) - mxoff) / 200.;
            if ((mxcmd>-0.1) && (mxcmd<0.1)) {
                mxcmd = 0.;
            } else if (mxcmd <= -0.1) {
                mxcmd = mxcmd + 0.1;
            } else {
                mxcmd = mxcmd - 0.1;
            }
            mycmd = (float)(getvaluator(MOUSEY) - myoff) / 200.;
            if ((mycmd>-0.1) && (mycmd<0.1)) {
                mycmd = 0.;
            } else if (mycmd <= -0.1) {
                mycmd = mycmd + 0.1;
            } else {
                mycmd = mycmd - 0.1;
            }
            if (mouseflag == 0) {
                cx = -mycmd;
                cy = mxcmd;
                cz = btncmd;
            } else if (mouseflag == 1) {
                crx = mxcmd;
                cry = mycmd;
                crz = -btncmd;
            }
        }
        if (vcmode == '1') {
            fxcmd = FMAX*cx;
            fycmd = FMAX*cy;
            fzcmd = FMAX*cz;
        } else
            fxcmd = fycmd = fzcmd = 0.;
        txcmd = TMAX*crx;
        tycmd = TMAX*cry;
        tzcmd = TMAX*crz;
}

get_track()
/* Get head dircos matrix from 3SPACE, calculate Euler angles */
{
        get_vpl();

        /* calculate yaw-pitch-roll Euler angles from 3SPACE dir cos matrix */
        /* t1=yaw t2=pitch t3=roll */
        t2 = fasin(dircos[2][0]);
        t1 = fasin(dircos[1][0]/fcos(t2));
        if (sgnf(dircos[0][0]/fcos(t1)) == -1) t1 = PI - t1;
        t3 = facos(dircos[2][2]/fcos(t2));
        if ( t3 > (PI/2.) ) {
            t1 = t1 + PI;
            t2 = (float)sgnf(t2) * PI - t2;
        }
```

```
}

dynamics(reset)
/* Dynamics code */
int reset;
{
        if (reset == 1) {
                /* Reset all dynamics & program control variables */
                xw = yw = zw = 0.;
                vxw = vyw = vzw = vxb = vyb = vzb = 0.;
                phi = init_phi;
                theta = init_theta;
                psi = init_psi;
                p = q = r = 0.;
                loop = 0;               endflag = 0;            marknum = 0;
        rectnum = 0;
                if (pcmode == '2') {
                        btncmd = 0.;
                        cx = cy = cz = crx = cry = crz = 0.;
                        mxoff = getvaluator(MOUSEX);
                        myoff = getvaluator(MOUSEY);
                }
                start_time = times(&buffer);
                inttime = times(&buffer);
                if (outpfile) store_data();
                return;
        }

        /* integrate forward one loop period */

        calc_dynamics(loopt);

        /* keep integrating forward until catch up to or pass current time */

        while (times(&buffer) > inttime) {
                calc_dynamics(loopt);
                numskip++;
        }

        /* wait for current time to catch up to integration time */

        while (inttime > times(&buffer)) {}

}

calc_dynamics(lstep)
/* Do dynamic calculation, integrations */
long lstep;
{
        register float phidot,psidot,thetdot;
        register float cthet,sthet,cphi,sphi,cpsi,spsi;
        register float tstep;

        tstep = (float)lstep / 100.;
```

```
/* do rotations first */
/* convert body rates to euler rates */

sphi = fsin(phi);
cphi = fcos(phi);

sthet = fsin(theta);
cthet = fcos(theta);
if (cthet == 0.) cthet=.0001; /* protect against singularity at theta = 0 */

spsi    = fsin(psi);
cpsi    = fcos(psi);

/* calculate new EULER rates */

phidot  = p + q*sphi*sthet/cthet + r*cphi*sthet/cthet;
thetdot = q*cphi - r*sphi;
psidot  = q*sphi/cthet + r*cphi/cthet;

/* get random disturbances if necessary */

if (dmode == '2' || dmode == '4') {
        xdist = TDIST * getrnd();
        ydist = TDIST * getrnd();
        zdist = TDIST * getrnd();
}
if (dmode == '3' || dmode == '4') {
        pdist = RDIST * getrnd();
        qdist = RDIST * getrnd();
        rdist = RDIST * getrnd();
}

/* do dynamics calculations */

vxbdot = (fxcmd + xdist) / mass;
vybdot = (fycmd + ydist) / mass;
vzbdot = (fzcmd + zdist) / mass;

if (rcmode == '1') {
        pdot = (txcmd + pdist) / inertia;
        qdot = (tycmd + qdist) / inertia;
        rdot = (tzcmd + rdist) / inertia;
} else {
        pdot = (ROTLAG * (ROTGAIN * txcmd - p) + pdist) / inertia;
        qdot = (ROTLAG * (ROTGAIN * tycmd - q) + qdist) / inertia;
        rdot = (ROTLAG * (ROTGAIN * tzcmd - r) + rdist) / inertia;
}

/* calculate translational motion */

vxwdot = cthet*cpsi*vxbdot + (sphi*sthet*cpsi-cphi*spsi)*vybdot
                + (cphi*sthet*cpsi+sphi*spsi)*vzbdot;
vywdot = cthet*spsi*vxbdot + (sphi*sthet*spsi+cphi*cpsi)*vybdot
```

```
                    + (cphi*sthet*spsi-sphi*cpsi)*vzbdot;
        vzwdot = -sthet*vxbdot + sphi*cthet*vybdot + cphi*cthet*vzbdot;

        /* integrate world accelerations to obtain world velocities */

        vxw += vxwdot * tstep;
        vyw += vywdot * tstep;
        vzw += vzwdot * tstep;

        /* integrate to obtain positions */

        xw += vxw * tstep;
        yw += vyw * tstep;
        zw += vzw * tstep;

        /* integrate rotations */

        p += pdot * tstep;
        q += qdot * tstep;
        r += rdot * tstep;

        phi += phidot * tstep;
        while (phi > TWOPI) phi -= TWOPI;
        while (phi < 0.) phi += TWOPI;

        psi += psidot * tstep;
        while (psi > TWOPI) psi -= TWOPI;
        while (psi < 0.) psi += TWOPI;

        theta += thetdot * tstep;
        while (theta > TWOPI) theta -= TWOPI;
        while (theta < 0.) theta += TWOPI;

        /* assign world rotations to graphics rotations */

        rx      =       (Angle)(theta*1800./PI);
        ry      =       (Angle)(-psi*1800./PI);
        rz      =       (Angle)(phi*1800./PI);

        /* assign world positions to graphics positions */

        x = yw;         y = -zw; z = xw;

        /* store state data if necessary */

        inttime += lstep;
        if (outpfile) store_data();
}

store_data()
/* Temporarily save output data in arrays */
{
        atime[loop]=(float)(inttime-start_time)/HZ;
        axw[loop]=xw;                   ayw[loop]=yw;                   azw[loop]=zw;
```

```
        aphi[loop]=phi*180./PI;     atheta[loop]=theta*180./PI;
        apsi[loop]=psi*180./PI;
        avxw[loop]=vxw;             avyw[loop]=vyw;             avzw[loop]=vzw;
        ap[loop]=p*180./PI;         aq[loop]=q*180./PI;         ar[loop]=r*180./PI;
        afxc[loop]=fxcmd;           afyc[loop]=fycmd;           afzc[loop]=fzcmd;
        atxc[loop]=txcmd;           atyc[loop]=tycmd;           atzc[loop]=tzcmd;
        if (dmode == '2' || dmode == '4') {
                axd[loop]=xdist;    ayd[loop]=ydist;    azd[loop]=zdist;
        }
        if (dmode == '3' || dmode == '4') {
                apd[loop]=pdist;    aqd[loop]=qdist;    ard[loop]=rdist;
        }
        loop++;
}

out_data()
/* Write output data to disk file(s) */
{
        register int oloop, tmpmknum=0;

        if (pmmode == '3') {
                outf_ptr = fopen(rectfname, "w");
                for (oloop=0; oloop<=2; oloop++)
                        fprintf(outf_ptr,"rectangle #%1d   time=%7.2f   dist=%7.4f\n",
                                oloop, rectt[oloop], rectd[oloop]);
                fclose(outf_ptr);
        }

        outf_ptr = fopen(outfname, "w");
        for (oloop=0; oloop<loop; oloop++) {
                if (tmpmknum < marknum)
                        if (oloop == markline[tmpmknum])
                                fprintf(outf_ptr,"MARK %d\n",++tmpmknum);

                fprintf(outf_ptr,"%ld",oloop);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",atime[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",axw[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",ayw[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",azw[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",aphi[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",atheta[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",apsi[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",avxw[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",avyw[oloop]);
                        putc(TAB,outf_ptr);
```

```
                fprintf(outf_ptr,"%.2f",avzw[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",ap[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",aq[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",ar[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",afxc[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",afyc[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",afzc[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",atxc[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",atyc[oloop]);
                        putc(TAB,outf_ptr);
                fprintf(outf_ptr,"%.2f",atzc[oloop]);
                if (dsave == '2' || dsave == '4') {
                                putc(TAB,outf_ptr);
                        fprintf(outf_ptr,"%.2f",axd[oloop]);
                                putc(TAB,outf_ptr);
                        fprintf(outf_ptr,"%.2f",ayd[oloop]);
                                putc(TAB,outf_ptr);
                        fprintf(outf_ptr,"%.2f",azd[oloop]);
                }
                if (dsave == '3' || dsave == '4') {
                                putc(TAB,outf_ptr);
                        fprintf(outf_ptr,"%.2f",apd[oloop]);
                                putc(TAB,outf_ptr);
                        fprintf(outf_ptr,"%.2f",aqd[oloop]);
                                putc(TAB,outf_ptr);
                        fprintf(outf_ptr,"%.2f",ard[oloop]);
                }
                fprintf(outf_ptr,"\n");
        }
        fclose(outf_ptr);
}

init_queue()
{
        qdevice(RIGHTMOUSE);
        qdevice(MIDDLEMOUSE);
        qdevice(LEFTMOUSE);
        qdevice(RKEY);
        qdevice(OKEY);
        qdevice(PKEY);
        qdevice(BKEY);
        qdevice(CKEY);
        qdevice(DKEY);
        qdevice(MKEY);
}
```

```
init_colors()
/* Initialize color map */
{
        unsigned short i;
        int bind,rind;
        float bintens,rintens;
        unsigned char gred,ggre,gblu;

        mapcolor(62,40,40,0);
        mapcolor(63,255,255,0);

        mapcolor(3968,0,0,40);
        mapcolor(4032,0,0,255);

        mapcolor(4030,40,40,40);
        mapcolor(4031,255,255,40);
        mapcolor(4094,40,40,255);
        mapcolor(4095,255,255,255);
}

init_monitor()
{
        system("blanktime 0");                  /* disable screen saver*/
        color(BLACK);
        clear();
        swapbuffers();
        clear();
        if ( vmode!='5' && vmode!='6' ) {
                /* VPL Eyephones or NTSC monitor */
                setvideo(DE_R1, DER1_170 | DER1_UNBLANK | DER1_SYNCG);
                viewport(0,NTSC_XMAX,0,NTSC_YMAX);
        }
}

int sys_setup(argc, argv)
/* Read all parameter values from keyboard or disk file */
int argc;
char **argv;
{
        int tmp;
        char iomode;

        vmode = emode = iomode = pmmode = '0';
        rcmode = bfmode = vcmode = pcmode = htmode = '0';
        dmode = '1';

        camht = 0.;

        if (argc == 1) {

                inmode = '1';

                printf("\n\nOutput filename (.=none)? ");
                scanf("%s",outfname);
```

```
printf("\n\nVideo mode:");
printf("\n1 - VPL Eyephones (monoptic)");
printf("\n2 - VPL Eyephones (stereoptic)");
printf("\n3 - NTSC RGB Monitor (mono)");
printf("\n4 - NTSC RGB Monitor (stereo)");
printf("\n5 - IRIS Screen (mono)");
printf("\n6 - IRIS Screen (stereo)");
printf("\n (1-6)? ");
while ( vmode<'1' || vmode>'6') {
        scanf("%c",&vmode);
}

printf("\n\nEnvironment mode:");
printf("\n1 - Box target");
printf("\n2 - Fly-through rectangles");
printf("\n3 - Grids");
printf("\n4 - Box in foreground, grid behind");
printf("\n (1-4)? ");

while ( emode<'1' || emode>'4') {
        scanf("%c",&emode);
}

printf("\n\nInitial orientation:");
printf("\n1 - Aligned with world frame");
printf("\n2 - User-entered initial Euler angles");
printf("\n (1-2)? ");
while ( iomode<'1' || iomode>'2') {
        scanf("%c",&iomode);
}

if ( iomode == '2') {
        printf("\n\nInitial orientation:");
        printf("\nRoll (int,degrees) ?");
        scanf("%f",&tmp);
        init_phi = (float)tmp * PI / 180.;
        printf("\nPitch (int,degrees) ?");
        scanf("%f",&tmp);
        init_theta = (float)tmp * PI / 180.;
        printf("\nYaw (int,degrees) ?");
        scanf("%f",&tmp);
        init_psi = (float)tmp * PI / 180.;
}

printf("\n\nEnd-of-run flag mode:");
printf("\n1 - None");
printf("\n2 - Orientation-derived");
printf("\n (1-2)? ");
while ( pmmode<'1' || pmmode>'2') {
        scanf("%c",&pmmode);
}

printf("\n\nRandom disturbance mode:");
```

```
printf("\n1 - None");
printf("\n2 - Translational only");
printf("\n3 - Rotational only");
printf("\n4 - Translational and rotational");
printf("\n (1-4)? ");
while ( dsave<'1' || dsave>'4') {
        scanf("%c",&dsave);
}

if ( dmode != '1' ) {
        printf("\n\nDisturbance random number generator seed:");
        printf("\n (long)? ");
        scanf("%ld",&rndseed);
}

printf("\n\nVehicle control mode:");
printf("\n1 - Rotation and translation");
printf("\n2 - Rotation only");
printf("\n (1-2)? ");
while ( vcmode<'1' || vcmode>'2') {
        scanf("%c",&vcmode);
}

printf("\n\nRotational control mode:");
printf("\n1 - Acceleration control");
printf("\n2 - Velocity control (with first-order lag)");
printf("\n (1-2)? ");
while ( rcmode<'1' || rcmode>'2') {
        scanf("%c",&rcmode);
}

printf("\n\nBody-fixed references:");
printf("\n1 - None");
printf("\n2 - Inner set only");
printf("\n3 - Full set");
printf("\n (1-3)? ");
while ( bfmode<'1' || bfmode>'3' ) {
        scanf("%c",&bfmode);
}
bfsave = bfmode;

printf("\n\nCamera offset from vehicle center? ");
scanf("%f",&camht);

printf("\n\nControl hardware mode:");
printf("\n1 - Control station");
printf("\n2 - IRIS");
printf("\n (1-2)? ");
while ( pcmode<'1' || pcmode>'2') {
        scanf("%c",&pcmode);
}

printf("\n\nHead-tracking mode:");
printf("\n1 - 3SPACE");
```

```
                printf("\n2 - None");
                printf("\n (1-2)? ");
                while ( htmode!='1' && htmode!='2' ) {
                        scanf("%c",&htmode);
                }

                printf("\n\nLoop time in hundredths of a second:");
                printf("\n (long)? ");
                scanf("%ld",&loopt);

                if ( vmode=='1' || vmode=='2' ) {if (getocular(&ocular) == 1) return(1);}

        } else if (argc == 2) {
                inmode = '2';
                if ((inf_ptr = fopen(argv[1], "r")) == NULL) return(1);
                fscanf(inf_ptr,"%c %c %c ", &vmode, &pcmode, &htmode);
                if ( vmode=='1' || vmode=='2' ) {if (getocular(&ocular) == 1) return(1);}

        } else return(1);

        return(0);
}

int inp_next()
{
        char mmode, iomode;

        if (inmode == '1') {inmode='0';       return(1);}
        if (inmode == '0') return(0);

        fscanf(inf_ptr, "%s ", outfname);
        if (!strcmp("EOF",outfname)) return(0);

        fscanf(inf_ptr, "%c ", &mmode);
        if (mmode == '1') {

                        /* Anna's random initial orientation setup */
                emode='4';    pmmode='2'; dsave='1';    rcmode='1';   bfmode='2';
                camht=0.;     loopt=10;       vcmode='2';
                fscanf(inf_ptr, "%f %f %f ",&init_phi,&init_theta,&init_psi);

        } else if (mmode == '2') {

                        /* Anna's random disturbances setup */
                emode='4';    pmmode='1'; dsave='4';    rcmode='1';   bfmode='2';
                camht=0.;     loopt=10;       vcmode='1';
                init_phi = init_theta = init_psi = 0.;
                fscanf(inf_ptr, "%ld ",&rndseed);

        } else if (mmode == '3') {

                        /* Anna's subject matching test setup */
                emode='2';    pmmode='3'; dsave='1';    rcmode='1';   bfmode='2';
                camht=0.;     loopt=10;       vcmode='1';
```

```
                init_phi = init_theta = init_psi = 0.;
                fscanf(inf_ptr, "%s ",rectfname);

        } else {

                                /* Explicit parameter values */
                fscanf(inf_ptr, "%c %c ", &emode, &iomode);
                if (iomode == '2') fscanf(inf_ptr, "%f %f %f ",&init_phi,&init_theta,
                                        &init_psi);
                else init_phi = init_theta = init_psi = 0.;
                fscanf(inf_ptr, "%c %c ", &pmmode, &dsave);
                if (dsave != '1') fscanf(inf_ptr, "%ld ", &rndseed);
                fscanf(inf_ptr, "%c %c %f %ld ", &rcmode, &bfmode, &camht, &loopt);
        }

        dmode = '1';
        bfsave = bfmode;
        return(1);
}

int getocular(ocu)
float *ocu;
{
        float iod;

        iodf_ptr = fopen("iod.data", "r");
        if (iodf_ptr == NULL) return(1);
        else {
                fscanf(iodf_ptr, "%f", &iod);
                fclose(iodf_ptr);
                *ocu = (iod / 2.) * 2.54 / 100.;
                return(0);
        }
}
```

```
/* com.c */
/* Communications routines for sim.c simulation */
/* by Matthew Machlis */
/* updated 3/6/91 */

#include <gl.h>
#include <device.h>
#include <stdio.h>
#include <fcntl.h>
#include <termio.h>
#include <get.h>
#include <string.h>
#include <math.h>

PC_abort()
{
        extern int pc;

        tpon();
        curson();
        qreset();
        setmonitor(HZ60);
        system("blanktime 670000");
        close(pc);
        greset();
        gexit();
        printf("PC comm error\n");
        exit(0);
}

VPL_abort()
{
        extern int vpl;

        tpon();
        curson();
        qreset();
        setmonitor(HZ60);
        system("blanktime 670000");
        close(vpl);
        greset();
        gexit();
        printf("VPL comm error\n");
        exit(0);
}

init_pc()
{
/* Open & set up port #1 to communicate with Gateway */
        int i,j,insize;
        unsigned char s;
        unsigned int datastr[12];
        extern unsigned int x_0,y_0,z_0,rx_0,ry_0,rz_0;
        extern int pc;
```

```
        static struct termio comport1 = {
        0,0,B19200|CS8|CREAD|HUPCL|CLOCAL,0,0,{0,0,0,0,0,0,1,0}};

        pc = open("/dev/ttyd1",O_RDWR|O_NDELAY);
        ioctl(pc,TCSETAF,&comport1);                    /* Set port parameters */

        s = '!';
        if (write(pc,&s,1) != 1) PC_abort();
        i = j = 0;
        while (i<12 & j<5000) {
                if (insize=read(pc,&s,1) == 1) {datastr[i++] = (int) s;
}
                else if (insize == -1) {PC_abort();}
                j++;
        }
        if (j == 5000) PC_abort();

        x_0 = ((datastr[0]<<8) | datastr[1]);
        y_0 = ((datastr[2]<<8) | datastr[3]);
        z_0 = ((datastr[4]<<8) | datastr[5]);
        rx_0 = ((datastr[6]<<8) | datastr[7]);
        ry_0 = ((datastr[8]<<8) | datastr[9]);
        rz_0 = ((datastr[10]<<8) | datastr[11]);
}

init_vpl()
{
/* Open & set up port #2 to communicate with 3SPACE */
        extern int vpl;
        extern char cmpst1[22], cmpst2[22], outstr[8];
        char s, instr[60];
        int i,j,insize;
        static struct termio comport2 = {
        0,0,B19200|CS8|CREAD|HUPCL|CLOCAL,0,0,{0,0,0,0,0,0,1,0}};

        vpl = open("/dev/ttyd2",O_RDWR|O_NDELAY);
        ioctl(vpl,TCSETAF,&comport2);                   /* Set port parameters */

        s = 'S';                        /* Request status */
        if (write(vpl,&s,1) != 1) VPL_abort();
        i = j = 0;
        while (i<55 & j<5000) {
                if (insize=read(vpl,&s,1) == 1) {instr[i++]=s;}
                else if (insize == -1) VPL_abort();
                j++;
        }
        if (j==5000) VPL_abort();
        instr[21] = '\000';
        if (strcmp(cmpst1,instr) & strcmp(cmpst2,instr)) VPL_abort();
                /* Next set output to direction cosines matrix */
        if (write(vpl,outstr,7) != 7) VPL_abort();
        s = 'f';                        /* Change VPL to binary output format */
        if (write(vpl,&s,1) != 1) VPL_abort();
}
```

```
get_pc()
{
        extern float cx,cy,cz,crx,cry,crz;
        extern unsigned int x_0,y_0,z_0,rx_0,ry_0,rz_0;
        int insize;
        unsigned int tx,ty,tz,trx,try,trz;
        unsigned char s,garbin[12];
        unsigned int datastr[12];
        int i,j;

        read(pc,garbin,12);
        s = '~';
        if (write(pc,&s,1) != 1) PC_abort();
        i = j = 0;
        while (i<12 & j<5000) {
                if (insize=read(pc,&s,1) == 1) {datastr[i++] = (unsigned int) s;
        }
                else if (insize == -1) {PC_abort();}
                j++;
        }
        if (j == 5000) PC_abort();

        tx = ((datastr[0]<<8) | datastr[1]);
        ty = ((datastr[2]<<8) | datastr[3]);
        tz = ((datastr[4]<<8) | datastr[5]);
        trx = ((datastr[6]<<8) | datastr[7]);
        try = ((datastr[8]<<8) | datastr[9]);
        trz = ((datastr[10]<<8) | datastr[11]);

        cx = ((int)tx-(int)x_0) / 2048.;
        cy = ((int)ty-(int)y_0) / 2048.;
        cz = -((int)tz-(int)z_0) / 2048.;
        crx = ((int)trx-(int)rx_0) / 2048.;
        cry = ((int)try-(int)ry_0) / 2048.;
        crz = -((int)trz-(int)rz_0) / 2048.;

        if (fabs(cx)<0.10) cx=0.;
        else if (cx>0.) cx=cx-0.1;
        else cx=cx+0.1;

        if (fabs(cy)<0.10) cy=0.;
        else if (cx>0.) cy=cy-0.1;
        else cy=cy+0.1;

        if (fabs(cz)<0.10) cz=0.;
        else if (cz>0.) cz=cz-0.1;
        else cz=cz+0.1;

        if (fabs(crx)<0.10) crx=0.;
        else if (crx>0.) crx=crx-0.1;
        else crx=crx+0.1;

        if (fabs(cry)<0.10) cry=0.;
```

```
        else if (cry>0.) cry=cry-0.1;
        else cry=cry+0.1;

        if (fabs(crz)<0.10) crz=0.;
        else if (crz>0.) crz=crz-0.1;
        else crz=crz+0.1;

}

get_vpl()
{
        unsigned char datastr[21];
        extern float dircos[3][3];
        int i,j,tmpint,insize;
        char s;

        s = 'P';
        if (write(vpl,&s,1) != 1) VPL_abort;
        i = j = 0;
        while (i<21 & j<1500) {
                if (insize=read(vpl,&s,1) == 1) {datastr[i++] = s;}
                else if (insize == -1) {VPL_abort;}
                j++;
        }
        if (j == 1500) VPL_abort();
        for (i=0; i<3; i++)
                for (j=0; j<3; j++) {
                        tmpint = datastr[3 + i*2 + j*6] + 256 * datastr[4 + i*2 + j*6];
                        if (tmpint > 32767) tmpint = tmpint - 65536;
                        dircos[i][j] = tmpint / 32768.;
                }
}
```

```
/* mymath.c */
/* updated 1/17/91 */

#include <math.h>
#include "gl.h"
#include "mymath.h"

sgnf(x)
/*---------------------------------------------------------------------------
 * Returns the sign of the floating point number x.
 *---------------------------------------------------------------------------
 */
float x;
{
        if (x < 0.0)
                return(-1);
        else if (x > 0.0)
                return(1);
        else
                return(0);
}

float dot_product(v1, v2)
/*---------------------------------------------------------------------------
 * Returns the dot product of the two vectors v1 and v2.
 *---------------------------------------------------------------------------
 */
Vector v1, v2;
{
        register float res;
        register int i;

        res = 0.0;

        for (i = X ; i < W ; i++)
                res += v1[i]*v2[i];

        return(res);
}

vect_cpy(v1, v2)
/*---------------------------------------------------------------------------
 * Copy vector v2 into v1.
 *---------------------------------------------------------------------------
 */
Vector v1, v2;
{
        register int i;

        for (i = X ; i <= Z ; i++)
                v1[i] = v2[i];
}

vect_add(v1, v2, res)
```

```
/*-----------------------------------------------------------------------
 * Add vector v1 to vector v2 and put the result in res.
 *-----------------------------------------------------------------------
 */
Vector v1, v2, res;
{
        register int i;

        for (i = X ; i <= Z ; i++)
                res[i] = v1[i] + v2[i];
}

vect_mult(v1, n)
/*-----------------------------------------------------------------------
 * Multiply all of the elements of the vector v1 by n and put the result back
 * in v1.
 *-----------------------------------------------------------------------
 */
Vector v1;
float n;
{
        register int i;

        for (i = X ; i <= Z ; i++)
                v1[i] *= n;
}

cross_product(v1, v2, res)
/*-----------------------------------------------------------------------
 * Loads the cross product of v1 with v2 into the vector res.
 *-----------------------------------------------------------------------
 */
Vector v1, v2, res;
{
        res[X] = v1[Y]*v2[Z] - v1[Z]*v2[Y];
        res[Y] = v1[Z]*v2[X] - v1[X]*v2[Z];
        res[Z] = v1[X]*v2[Y] - v1[Y]*v2[X];
}

float norm(v1)
/*-----------------------------------------------------------------------
 * Returns the norm (ie length) of the vector v1.
 *-----------------------------------------------------------------------
 */
Vector v1;
{
        return(sqrt(dot_product(v1, v1)));
}

normalize(v1)
/*-----------------------------------------------------------------------
 * Normalize the vector v1.
 *-----------------------------------------------------------------------
 */
```

```
Vector v1;
{
        register float n;

        n = norm(v1);
        vect_mult(v1, 1.0/n);
}

vector_print(vec)
/*-------------------------------------------------------------------------
 * Print the 3 diminsional vector vec.
 *-------------------------------------------------------------------------
 */
Vector vec;
{
        register int i;

        printf("[");
        for (i = 0 ; i <= Z ; i++)
                printf("%4.6f ", vec[i]);
        printf("]\n");
}

hvector_print(vec)
/*-------------------------------------------------------------------------
 * Print the 4 diminsional vector vec.
 *-------------------------------------------------------------------------
 */
Vector vec;
{
        register int i;

        printf("[");
        for (i = 0 ; i <= W ; i++)
                printf("%4.6f ", vec[i]);
        printf("]\n");
}

matrix_print(mat)
/*-------------------------------------------------------------------------
 * Print the 4x4 matrix mat.
 *-------------------------------------------------------------------------
 */
Matrix mat;
{
        register int i, j;

        for (i = 0 ; i < 4 ; i++){
                for (j = 0 ; j < 4 ; j++)
                        printf("%4.6f ", mat[i][j]);
                printf("\n");
                }
}
```

```
matrix_cpy(mat1, mat2)
/*-------------------------------------------------------------------------
 * Copy matrix mat2 to matrix mat1.
 *-------------------------------------------------------------------------
 */
Matrix mat1, mat2;
{
        register int i, j;

        for (i = 0 ; i < 4 ; i++)
                for (j = 0 ; j < 4 ; j++)
                        mat1[i][j] = mat2[i][j];
}

float getrnd()
/* Generate Gaussian-distributed random number */
{
        static int iset=0;
        static float gset;
        float fac,r,v1,v2;

        if (iset == 0) {
                do {
                        v1 = 2.0 * drand48() - 1.0;
                        v2 = 2.0 * drand48() - 1.0;
                        r = v1*v1 + v2*v2;
                } while (r >= 1.0 || r == 0.0);
                fac = fsqrt(-2.0 * flog(r) / r);
                gset = v1 * fac;
                iset = 1;
                return (v2 * fac);
        } else {
                iset = 0;
                return (gset);
        }
}
```

```
/* iod.c */
/* Program to determine iod for Eyephone projection calculations */
/* by Matthew Machlis */
/* based on Eyephones manual example */
/* updated 3/6/91 */


#include "iod.h"
#include "stdio.h"
#include </usr/include/gl/cg2vme.h>
#include </usr/include/gl/addrs.h>
main()
{
        FILE *fopen(), *file_ptr;
        int monitor;
        int sy,i;
        int angle;
        float iod;
        long dev;
        short value;

        ginit();

        resetls(TRUE);
        linewidth(3);
        backface(FALSE);
        zbuffer(FALSE);
        doublebuffer();
        gconfig();

        color(BLACK);
        clear();
        swapbuffers();
        clear();

        sys_setup();

        iod = 2.5;

        dev = 0;
        while (dev != ESCKEY) {

                i++;
                writemask(4095);
                color(BLACK);
                clear();

        drawcal(iod);

        swapbuffers();

        dev = 0;
        while (dev!=ESCKEY & dev!=LEFTMOUSE & dev!=RIGHTMOUSE)
        dev=qread(&value);
```

```
        if (dev == LEFTMOUSE & value == 1) iod = iod - 0.005;
        if (dev == RIGHTMOUSE & value == 1) iod = iod + 0.005;

        }

        tpon();
        curson();
        qreset();
        setmonitor(HZ60);
        system("blanktime 67000");
        greset();
        gexit();

        printf("\n\nIOD = %5.3f\n\n",iod);
        file_ptr = fopen("iod.data", "w");
        fprintf(file_ptr, "%5.3f\n", iod);
        fclose(file_ptr);

        exit(0);
}

drawcal(iod)
float iod;
{
        float ocular;
        ocular = iod / 2;

        /* Draw circle for right eye */
        writemask(4032);
        color(4032);
        window (SCREENSEP/2 - ocular, (SCREENSEP/2 + SCREENWIDTH) - ocular,
                        -SCREENDN, SCREENUP, SCREENDIST, FARCLIP);
        pushmatrix();
        translate (ocular, 0, -FARAWAY);
        circf(0,0,50);
        popmatrix();

        /* Draw rectangle for left eye */
        writemask(63);
        color(63);
        window (ocular - (SCREENSEP/2 + SCREENWIDTH), ocular - SCREENSEP/2,
                        -SCREENDN, SCREENUP, SCREENDIST, FARCLIP);
        pushmatrix();
        translate (-ocular, 0, -FARAWAY);
        rect(-5, 50, 5, 250);
        popmatrix();
}

sys_setup()
{
/* set Monitor type */
        setvideo(DE_R1, DER1_170 | DER1_UNBLANK | DER1_SYNCG);
        viewport(0,NTSC_XMAX,0,NTSC_YMAX);
        system("blanktime 0");/* disable screen saver*/
```

```
        mapcolor(63,255,0,0);
        mapcolor(4032,0,0,255);

/* enqueue iod-adjustment keys */
        qdevice(RIGHTMOUSE);
        qdevice(LEFTMOUSE);
        qdevice(ESCKEY);
}
```

```
/* sim.h */
/* by Matthew Machlis */
/* updated 3/6/91 */

#include "stdio.h"
#include "math.h"
#include "mymath.h"
#include "gl.h"
#include "device.h"
#include "get.h"
#include "string.h"
#include </usr/include/gl/cg2vme.h>
#include </usr/include/gl/addrs.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/param.h>
#include <sys/time.h>

/* NTSC monitor screen boundaries */
#define NTSC_XMAX 645
#define NTSC_YMAX 484

/* Vehicle definition constants */
#define FMAX 100.
#define TMAX 100.

#define MASS 500.
#define I 4000.

#define CDX 1.
#define CDY 1.
#define CDZ 1.

#define CDROTX 1.
#define CDROTY 1.
#define CDROTZ 1.

#define DTOR 0.0174533

/* Communications stuff */
int pc,vpl;
FILE *fopen(), *inf_ptr, *outf_ptr, *iodf_ptr;
char outfname[20], rectfname[20];
float cx,cy,cz,crx,cry,crz;
unsigned int x_0,y_0,z_0,rx_0,ry_0,rz_0;
#define TAB 9
float atime[10000],axw[10000],ayw[10000],azw[10000],aphi[10000],atheta[10000],
        apsi[10000],avxw[10000],avyw[10000],avzw[10000],ap[10000],aq[10000],
        ar[10000],afxc[10000],afyc[10000],afzc[10000],atxc[10000],atyc[10000],
        atzc[10000],axd[10000],ayd[10000],azd[10000],apd[10000],aqd[10000],
        ard[10000];
float rectt[3], rectd[3];
char mess[40];
```

```
/* Timing variables */
struct tms buffer;
long loopt, inttime, start_time;          /* time steps */
int numskip=0;                            /* # of skipped screen updates */

/* Random number generator variables */
struct timeval tp;
struct timezone tzp;
long rndseed;
float getrnd();

/* Program control variables */
int loop=0, markline[50];
char vmode, emode, pmmode, dmode, rcmode;
char vcmode, htmode, bfmode, inmode, pcmode;
char dsave, bfsave;
short marknum=0;
Boolean endflag=FALSE, outpfile;

/* Control input variables */
float fxcmd=0., fycmd=0., fzcmd=0.;
float txcmd=0., tycmd=0., tzcmd=0.;
short mouseflag=2;
long mxoff=0, myoff=0;
float btncmd=0.;

/* Graphics transformation variables */
Angle rx,ry,rz;
Coord x,y,z;
float camht;
float t1,t2,t3;
Matrix bodmat;

/* Vehicle dynamics variables */
#define ROTGAIN 0.003
#define ROTLAG 3000.
float mass,inertia;
float masses[5]={500.,500.,500.,500.,500.};
float inertias[5]={4000.,4000.,4000.,4000.,4000.};
float xw, yw, zw;                         /* world coord position */
float p, q, r;                            /* body rotation rates */
float   phi,                              /* roll (Euler angle) */
        theta,                            /* pitch */
        psi;                              /* yaw */
float init_theta=0., init_phi=0., init_psi=0.;
float vxw, vyw, vzw;                      /* world coord velocities */
float vxb, vyb, vzb;                      /* body coord velocities */

float vxbdot,vybdot,vzbdot;               /* body coord accels */
float vxwdot,vywdot,vzwdot;               /* world coord accels */
float pdot,qdot,rdot;                     /* body coord angular accels */

#define TDIST 25.                         /* max translational dist magnitude */
#define RDIST 25.                         /* max rotational disturbance magnitude */
```

```
float xdist=0.,ydist=0.,zdist=0.;/      * translational disturbance forces */
float pdist=0.,qdist=0.,rdist=0.;       /* rotational disturbance forces */

/* Body-fixed reference data */
float bfv[56][3] = {{15.,9.,-100.},{15.,11.,-100.},{13.,11.,-100.},
        {-13.,11.,-100.},{-15.,11.,-100.},{-15.,9.,-100.},
        {-15.,-9.,-100.},{-15.,-11.,-100.},{-13.,-11.,-100.},
        {13.,-11.,-100.},{15.,-11.,-100.},{15.,-9.,-100.},
        {33.,24.,-100.},{33.,26.,-100.},{31.,26.,-100.},
        {-31.,26.,-100.},{-33.,26.,-100.},{-33.,24.,-100.},
        {-33.,-24.,-100.},{-33.,-26.,-100.},{-31.,-26.,-100.},
        {31.,-26.,-100.},{33.,-26.,-100.},{33.,-24.,-100.},

        {-1.,100.,-100.},{1.,100.,-100.},{0.,100.,-100.},{0.,97.,-100.},
        {-72.,70.,-100.},{-70.,72.,-100.},{-71.,71.,-100.},{-69.,69.,-100.},
        {70.,72.,-100.},{72.,70.,-100.},{71.,71.,-100.},{69.,69.,-100.},
        {100.,1.,-100.},{100.,-1.,-100.},{100.,0.,-100.},{97.,0.,-100.},
        {72.,-70.,-100.},{70.,-72.,-100.},{71.,-71.,-100.},{69.,-69.,-100.},
        {-1.,-100.,-100.},{1.,-100.,-100.},{0.,-100.,-100.},{0.,-97.,-100.},
        {-70.,-72.,-100.},{-72.,-70.,-100.},{-71.,-71.,-100.},{-69.,-69.,-100.},
        {-100.,-1.,-100.},{-100.,1.,-100.},{-100.,0.,-100.},{-97.,0.,-100.}};

/* Fly-through rectangle data */
#define NUMTARG 5
#define TARGHT 6.
#define TARGWD 7.5
#define XSIZE 1.
/*
float targloc[NUMTARG][3] = {{0.,0.,-30.},{0.,5.,-60},{0.,15.,-85},
        {10.,20.,-100.},{20.,25.,-125.},{35.,25.,-150.},{40.,30.,-170.},
        {40.,30.,-200.},{50.,30.,-225.},{60.,32.,-250.}};
float targroll[NUMTARG] = {0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0.};
*/
float targloc[NUMTARG][3] = {{0.,0.,-30.},{0.,5.,-60},{0.,15.,-85},
        {10.,20.,-100.},{20.,25.,-125.}};
float targroll[NUMTARG] = {0., 0., 0., 0., 0.};
float vec3[3][3];
short rectnum=0;
float targv1[NUMTARG][3], targv2[NUMTARG][3], targv3[NUMTARG][3];
float targv4[NUMTARG][3], targv5[NUMTARG][3], targv6[NUMTARG][3];
float targv7[NUMTARG][3], targv8[NUMTARG][3];

/* Cube data */
short boxv[9][3] = {{10,10,-10},{10,-10,-10},{-10,-10,-10},
        {-10,10,-10},{10,10,10},{10,-10,10},{-10,-10,10},
        {-10,10,10},{0,10,10}};

/* 3SPACE stuff */
float dircos[3][3];
char cmpst1[22] = "21S208 0    0 2 3";
char cmpst2[22] = "21S209 0    0 2 3";
char outstr[8] = "O5,6,7\r";
float borevec[3];
```

```
/* Eyephone stuff */
#define SCREENSEP 0.0222
#define SCREENWIDTH 0.0542
#define SCREENUP 0.0218
#define SCREENDN 0.0185
#define SCREENDIST 0.0351
#define FARCLIP 100
float ocular;
```

```
/* mymath.h */
/* Type definitions etc for the math library. */
/* updated 1/17/91 */

#define     X     0
#define     Y     1
#define     Z     2
#define     W     3

#define ABS(A)      (A < 0.0 ? -A:A)
#define SGN(A)  ( (A < -A)  ? -1:1)
#define PI 3.141592654
#define TWOPI 6.283185308
typedef float   Vector[3],
                HVector[4];

float dot_product();
float norm();
int ascii_to_int();
float getrnd();
```

```
/* iod.h */
/* by Matthew Machlis */
/* updated 1/17/91 */

#include "math.h"
#include "gl.h"
#include "device.h"
#include "get.h"

/* NTSC monitor screen boundaries */
#define NTSC_XMAX 645
#define NTSC_YMAX 484

/* Eyephone constants */
#define SCREENSEP 0.875
#define SCREENWIDTH 2.133
#define SCREENUP 0.860
#define SCREENDN 0.730
#define SCREENDIST 1.383
#define FARCLIP 100000
#define FARAWAY 500
```

Appendix B.

Test Subject Survey Questionaire and Pilot Evaluation Comment Sheet

B1. Test Subject Survey Questionaire

Date: _____, 1991

All of the questions below are optional. Please do not feel pressured to answer any questions you do not feel comfortable answering. This questionaire will be kept on file with the experiment data for future bias analysis and is not part of the actual experiment. If you have any questions please feel free to ask them. Thank you for your cooperation.

Name: _____

Male _____, Female _____

Birthdate: _____

Skills Questions:

1. Do you know how to drive? Yes _____, No _____

   If yes:

   1a. How often do you drive? (Daily, Regularly, Rarely)

   _____

2. Are you a pilot? Yes _____, No _____

   If yes:

   2a. How often do you fly? (Weekly, Monthly, Rarely)

   _____

   2b. How many hours do you have?

   _____

3. Do you play video or computer games? Yes _____, No _____

   If yes:

   3a. Do you play flight-type video games? Yes _____, No _____

   3b. How often do you play? (Daily, Weekly, Rarely)

Vision Questions:

4. What is your vision? (20/400, for example) _____

5. Do you have any vision irregularities? (Astigmatism, ...) Yes _____, No _____

   If yes:

   Please Describe:

   _____

6. Do you wear lenses of any type? Yes _____, No _____

   If yes:

   6a. Why? _____

   6b. What is your vision with your lenses? _____

   6c. Do you wear contacts? Yes _____, No _____

7. Are you colorblind? Yes _____, No _____

   If yes:

   Please Describe:

   _____

8. Do you have any chronic or frequent problems with depth perception? Yes _____, No _____

   If yes:

   Please Describe:

   _____

Miscellaneous Questions:

9. Are you dyslexic? Yes _____, No _____

10. Do you experience chronic or frequent vertigo? Yes _____, No _____

11. Do you have serious difficulty telling your right from your left? Yes _____, No _____

B2. Pilot Evaluation Comment Sheet

Name: _____

Flight _____

Date: _____

Please answer the following questions:

Rate the flight using the rating scale: _____

Comment on your ability to accomplish the task:

_____

_____

_____

Comment on your ability to control the vehicle:

_____

_____

_____

Comment on your ability to perceive the translation and rotational motion of the vehicle:

_____

_____

_____

During the flight did you get confused about the task objective?

Yes _____, No _____

During the flight did you get confused about the controls?

Yes _____, No _____

During the flight did you get confused about your translational motion?

Yes _____, No _____

During the flight did you get confused about your rotational motion?

Yes _____, No _____

During the flight did the display ever confuse you?

Yes _____, No _____

Please comment on any confusion you might have had:

_____

_____

_____

Please add any comments of your own:

_____

_____

_____