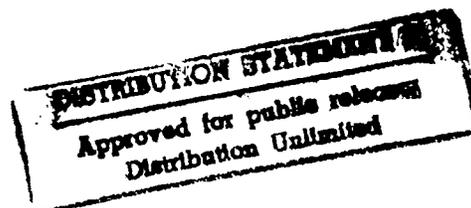19941228 037

TELEPRESENT SPACECRAFT DOCKING WITH
OBJECT-BASED BILATERAL CONTROL (OBBC)

THESIS

Paul Woznick, Captain, USAF

AFIT/GA/ENY/GA94D-9

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GA/ENY/GA94D-9

TELEPRESENT SPACECRAFT DOCKING WITH
OBJECT-BASED BILATERAL CONTROL (OBBC)

THESIS

Paul Woznick, Captain, USAF

AFIT/GA/ENY/GA94D-9

AFIT/GA/ENY/94D-9

TELEPRESENT SPACECRAFT DOCKING WITH
OBJECT-BASED BILATERAL CONTROL (OBBC)

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Astronautical Engineering

Paul Woznick, B.S.
Captain, USAF

December 1994

Acknowledgments

I wish to thank my advisor, Dr. Curtis Spenny for his support of my thesis, his time and his patience. I also want to thank the other committee members, Dr. Schneider and Dr. Hall as well as Mark Deriso, my lab technician and John Brohas for his machining skills. Special thanks goes to Mark Hunter and Tom Deeter, my fellow robotics students, for all their suggestions and support of my research and laboratory activities. It was the Robotics Group that made me feel like I belonged here. Finally, I beg forgiveness from my wife for the times I wasn't there for her and want her to know that our pregnancy means more to me than I could ever express in words.

Table of Contents

List of Figures

List of Tables

AFIT/GA/ENY94D-9

Abstract

The concept of object-based control is extended to the field of teleoperation, specifically to accomplish the task of spacecraft docking via a bilateral manual controller. An object-based controller with bilateral feedback controls the motion of the grasped object, not the trajectory of the manipulator. For this reason it can be designed with feedback that is intricately linked with the task kinematics. The benefits derived from anthropomorphicity and force feedback are possible without kinematically/geometrically similar master-slave systems, complex calibration and joint mapping schemes, or expensive, high degree-of-freedom force reflection. Object-based control is ideal for low-level telerobotic interfaces.

A hand controller and a spacecraft docking simulation are designed and constructed to demonstrate object-based bilateral control. The dominant task objective in spacecraft docking is the approach to a target vehicle along a single axis of motion. Several methods of bilateral feedback linked with this dominant objective are proposed in addition to simple force reflection. One method involves virtual forces and another utilizes velocity reflection. Each method, practical only with object-based control, enhances the man-machine interface by providing a heuristic method of manual control.

TELEPRESENT SPACECRAFT DOCKING WITH OBJECT-BASED

BILATERAL CONTROL (OBBC)

I. Introduction

1.1 Overview This thesis proposes a new control method for teleoperation. The task chosen to demonstrate this new control architecture is spacecraft docking. Figure 1-1 depicts teleoperated spacecraft docking. Major docking objectives are a translational approach to the target vehicle on one nominal axis with man-in-the-loop control and the quick and accurate reduction of the target and tracking vehicle separation distance.

TRACKING VEHICLE

TARGET VEHICLE

MONITOR

Visual Display

Master controller processing and communication equipment

MASTER HAND CONTROLLER
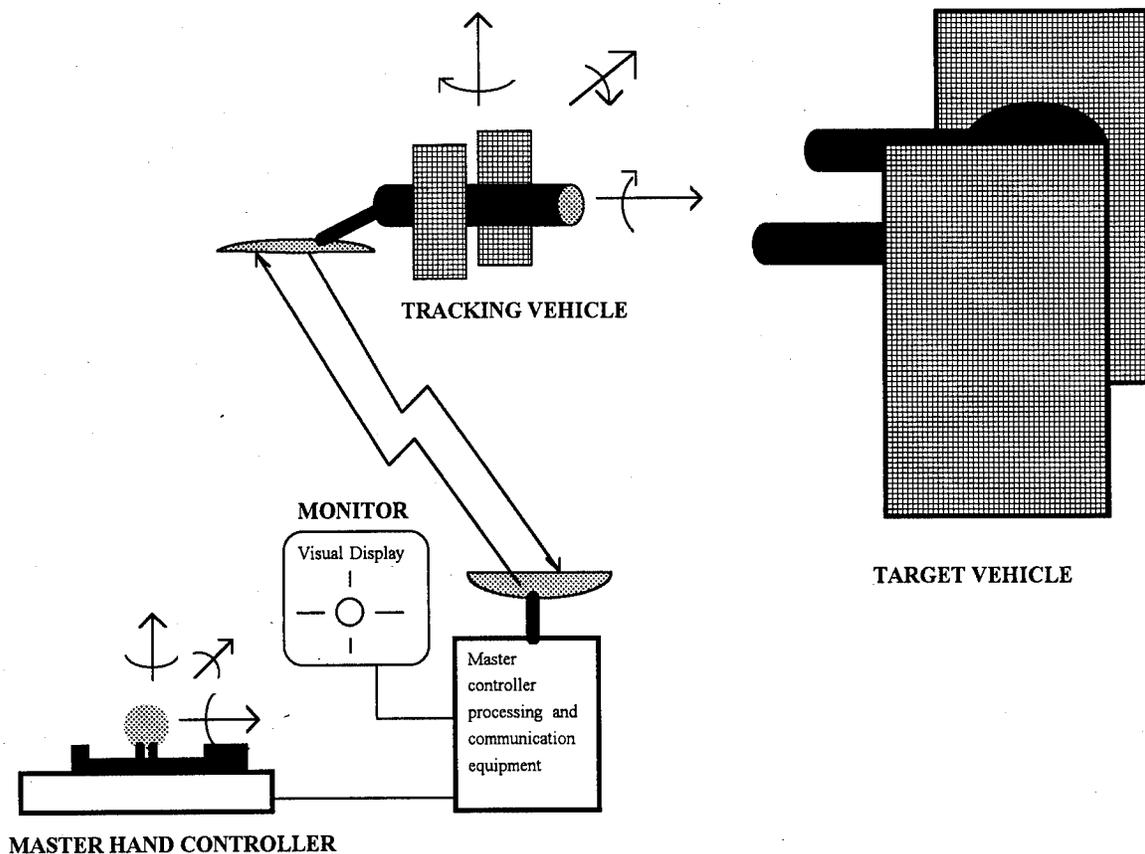
Figure 1-1. Teleoperated Spacecraft Docking

1.2 Background Space-based applications have historically required man's unique decision-making skills to implement contingency operations and manual control (1,2,3,4). Additionally, the consensus of the Department of Defense (DOD), major universities and industry is that man-in-the loop control will play a vital part in future applications that involve highly complex tasks in extremely dynamic environments (1). As a result, NASA's manned and unmanned space programs are based on the premise of man-in-the-loop control. The highly automated interplanetary missions such as Surveyor III and Viking Mars involved manual mission modifications and ground-based manipulator arm control (5). The Shuttle Transportation System's robotic arm, as well as the orbital docking accomplished throughout the Gemini, Apollo, and Apollo-Soyuz programs, required extensive manual control (2,5,6). The future requirements of space flight in the era of the space station will depend heavily on telemanipulated spacecraft docking (5). The history of spacecraft docking and the uncertainties of this task suggest the need for the complex decision capabilities of man, while the need for quick communication without time delay and the hazardous environment in which this task is performed suggest the need for autonomy. But at least until significant progress in automation is achieved, critical decision-making capability and emergency contingency operations in unpredictable environments will remain a human responsibility.

Past and current spacecraft rendezvous operations and proximity operations require either manual override or hand control (7,25). Astronauts review data displayed at consoles and either punch the necessary buttons and switches for automatic control or use combinations of simple interface devices similar to joysticks to perform manual spacecraft control (2,7,25). All proximity operations require crew visual queues and have manual override capability perhaps because virtually all of the U.S. and former Soviet Union's attempts at docking required some form of astronaut/cosmonaut manual control (2,7,25).

Review of the research establishes three categories of control in the field of telemanipulation. They are distinguished by the degree of man's input into the control

loop. The three categories are executive, supervisory, and manual control. Executive and manual control are easily defined because the represent the extremes in telemanipulation. Executive control is basically on/off control of an autonomous robot (4) and is typically associated with the term **telerobotics** (1). Manual control is total human control and is typically associated with the term **teleoperation** (1). Supervisory control, a term coined by Sheridan (8), is more difficult to precisely define because it combines both autonomous and human control (5,8). Because it involves some form of human input, it too is often considered teleoperation. Supervisory control similar to the telerobotic extreme implies human input, which augments pre-programmed computer control (6,8). Supervisory control similar to the teleoperation extreme implies computer input which augments human manual control (8). The degree to which the interface device provides a human operator with realistic information from the task environment is **telepresence** (1,5). Each broad category of control implies that the human operator is physically separated from the task environment, necessitating a man-machine interface device. This interface device is referred to as a master controller.

Since Ray Goertz created the world's first electrical master-slave manipulator in the 1950's at the Argonne National Laboratory (3,5), two master controller types have conclusively emerged: geometrically/kinematically identical master-slave controllers (hereafter referred to as master-slave controllers) and hand controllers (9). Emphasis has been on the master-slave controllers (10), probably because designers seeking ergonomically enhanced man-machine interface, have turned to anthropomorphicity (1,4,10) and force reflection (11). Anthropomorphicity and high degree of freedom (DOF) force reflection are predominantly identified with master-slave controllers. Force reflecting hand controllers have been limited to miniaturized replica robots (5). In other words, traditional force reflecting hand controllers are just miniaturized versions of the master-slave controllers. Hand controllers have made strides only because of their compactness. The need for this trait is particularly evident in space-based operations

3

where orbital work space is at a premium (4,9,12). The question of utility versus cost of a force reflecting hand controller has limited the use of force reflection in hand controller application (10). Still, force reflecting hand control is being pursued (12,13) and is a design specification for the free world's future space station. The space station has been the main force behind this technology for the past decade (5).

Research reveals many control theories that track actual slave motion to the desired motion. Common control concepts include position and force control. More recent methods are hybrid position-force control and impedance control (5). The foundation of the research in this thesis is object impedance control, a concept and theory introduced and developed by Schneider and Cannon (14). This concept focuses on the control of the object and not the kinematics of the manipulator. The authors proposed that object-based control is an extremely efficient technique due to instinctual control and transparent manipulator dynamics. If applied to teleoperation, the instinctual control and transparent manipulator dynamics would have three effects on a telemanipulation system. First, there is a need for shared control and increased autonomy. As a result, the basic operation requires little or no training because control is reduced to (at most) three simple translations and rotations in space, which are skills humans accomplish routinely many times per work day (1,4). Manipulator motion is automatically generated by the computer based on the commanded object motion. Secondly, the physical characteristics of the master controller need not be anthropomorphic or geometrically/kinematically similar to the slave. Instead, the controller can be similar to the grasped object and can provide DOF control equal to that required by the object. And thirdly, the feedback needed for bilateral control becomes easier to implement, no longer dependent on complex mathematical joint-mapping transformations, and can be designed kinematically similar to the task.

Up to now, little research has been done in this area. Dr. Nat Durlach of MIT introduced and researched Tool Handle Teleoperation specifically for the field of tele-

surgery. This method allows a surgeon to manipulate a surgical instrument mounted to a robot as if there was a handle attached to the instrument (15). Most recently, Dr. Paul Michelman and Dr. Peter Allen of Columbia University have proposed object-space teleoperation (16). They use a simple input device to control elementary rotation and translation motion of a Utah/MIT dexterous hand. The input device commands are based on the grasped object space, rather than joint position space, which is the traditional method to control motion of the dexterous hand.

The telepresence provided by a master controller is achieved mainly through the coupling of visual feedback with bilateral control, the two-way communication of position and force information between the user and the task environment (17,18). Research indicates that telemanipulated task completion is up to 50% quicker when some kind of force information is returned to the operator (8,12,13,16,19) and shows that force reflection used for bilateral control is extremely effective if the operator desires to control [contact] forces in the task environment (11,17,18). Traditional teleoperation architecture derives feedback in one of two ways (17). In the first method, the effect of object/environment interaction on each slave joint is directly measured, and impedance or hybrid position/force control is used to force the corresponding actuated joints of the master to track these slave joints. Master-slave symmetry makes this method practical but dictates master controller geometric design. The second method, which is required for hand controllers, is to form complex Jacobians that mathematically transform slave end-effector forces into master joint torques. For either method, slave/environment forces are corrupted by the compliance of the systems, while task and grasped object dynamics are completely neglected. Furthermore, the complex mathematical transformations make it impossible to orchestrate this information so that it can be used to provide a more instinctual feedback.

This thesis proposes a new teleoperation concept called object-based bilateral control (OBBC), and demonstrates its implementation with a spacecraft docking

simulation. OBBC extends the object control theory of Schneider and Canon (14) to teleoperation and enhances operator telepresence via the addition of bilateral control. OBBC provides object-space control to facilitate simple and realistic reflection of the interaction of the object with the environment while transparently controlling the motion of the slave. Complex schemes to transform end-effector dynamics to joint-space torques for effective bilateral control are not required; instead, the feedback is tailored to exploit the task's dominant kinematic objectives. OBBC makes 1-DOF tactile feedback a practical and useful design trait. Additionally, this type of control simplifies the correlation of multiple DOF data for feedback via a controller with a single or low-level DOF feedback capability.

Spacecraft docking is an ideal task for OBBC implementation because dominant kinematic objectives are clear and easily integrated into the master controller design. Additionally, the use of the OBBC master controller is compatible with the practice of using hand controllers in spacecraft proximity operations.

The following chapter compares object-based teleoperation architecture with the current architectures and makes general comparisons of each. The design characteristics and operational traits of the master controller's four operational modes are also discussed. Presently, AFIT does not possess a hand controller that offers bilateral control capability; therefore, a prototype master controller has been designed and fabricated to enable OBBC implementation. The description of this hardware is the subject of chapter 3. The master controller is designed bilaterally, that is, it is used to input commands for the six degrees of freedom (3 translational and 3 rotational) of a spacecraft and it is actuated for feedback in one degree of freedom, specifically for spacecraft approach axis feedback. The simple integration of two special types of heuristic feedback designed with task kinematic similarity is discussed. Both methods utilize virtual feedback that includes a shared control between the operator and the computer thereby increasing the overall system autonomy. Chapter 4 describes the overall OBBC system controller design and its link

with the graphical spacecraft docking simulation. Also discussed in this chapter is the OBBC communication network. Standard UNIX C programs have been written to perform master controller serial communication and ethernet communication between Silicon Graphics, Sun Sparc workstations, and a real-time microprocessor. Chapter 5 documents results recorded during spacecraft docking simulation with the four operational modes of the prototype OBBC system. Chapter 6 summarizes conclusions and recommendations about the thesis. Finally, the appendices supply design drawings, code listings and flowcharts, and simulation operation instructions.

1.3 Principal Accomplishments The principal accomplishments of this research are:

- development and demonstration of an object-based bilateral teleoperation concept, a concept unique to the field of teleoperation.

- fabrication of a 6-degree-of-freedom hand controller with 1-degree-of-freedom feedback to the operator, offering bilateral control

- demonstration of the utility of multi-axis to single-axis feedback for bilateral control, specifically in spacecraft docking

- development and implementation of two bilateral controllers unique to telemanipulation and

- the design of a low-level telemanipulation communication hub.

II. OBBC Concepts and Current Implementation Methods

2.1 OBBC Telemanipulation Architecture Research implies that a paradigm has evolved with the control architecture of telemanipulation. The physical design of the slave has dictated the foundation of the overall system control theory, the physical design of the master controller, and the design and implementation of the system's feedback. Additionally, due to the benefits derived from ergonomics, designers have been forced to incorporate anthropomorphic and high DOF force reflection characteristics in master controller design, putting further limitations on the slave design. This paradigm necessitates complex calibration and joint mapping schemes, geometrically/kinematically identical master-slave systems, and detailed knowledge of task and manipulator. It has also forced the emergence of basically two types of master controllers. The two controller types are force reflecting hand controllers (FRHC) and the geometrically/kinematically identical master-slave controllers with force reflection. The basic control architecture for these type of controllers is illustrated in Figure 2-1a.

The OBBC challenges this paradigm because the control theory is object based; therefore, the operator controls the motion of the object, not the slave. The OBBC implementation used in this thesis to perform teleoperated spacecraft docking demonstrates a master controller physical design not restricted by a need for geometric symmetry. OBBC also demonstrates bilateral control that utilizes feedback designed to exploit the docking task. Furthermore, ergonomic benefits are demonstrated without anthropomorphism. The proposed general architecture is seen in Figure 2-1b.

Figure 2-1a. TRADITIONAL TELEOPERATION ARCHITECTURE

Figure 2-1b. OBBC TELEOPERATION ARCHITECTURE
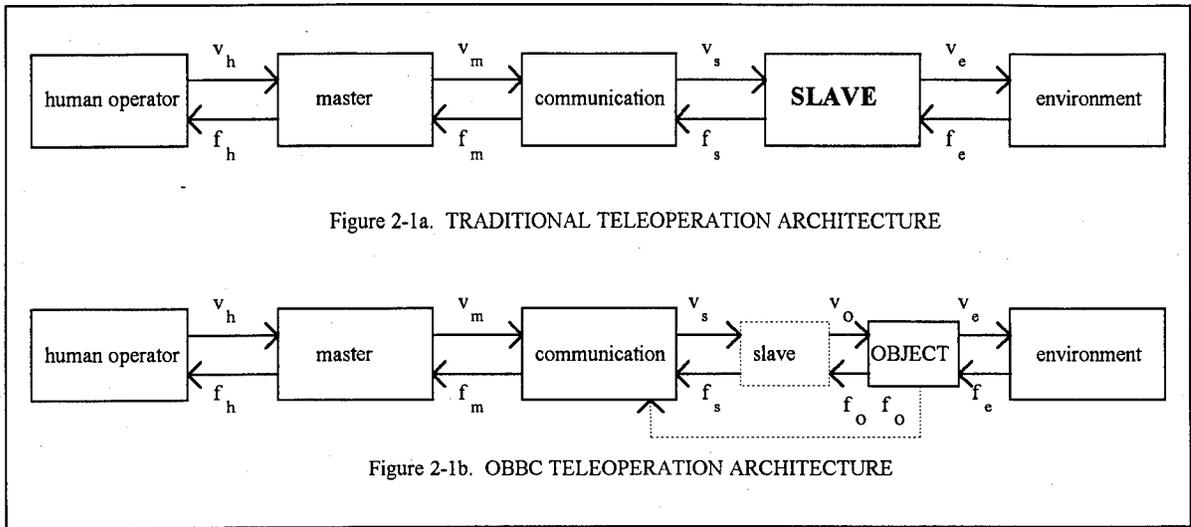
Figure 2-1. Block Diagrams of Teleoperation Architectures

Figure 2-1 illustrates the difference between object-based control and the traditional teleoperation architecture defined by Anderson and Spong (18). Ideally, for the traditional architecture, the desired effect of the slave (v_s) is controlled to equal the input's effect on the master (v_m), and the reflected force from the environment (f_e) is controlled to equal the force sensed by the operator (f_h). The object and task dynamics are neglected when implementing traditional teleoperation. For the OBBC architecture, the desired effect of the object (v_o) is controlled to equal the input on the master (v_m) via automatic computer generated slave input (v_s). In Figure 2-1b the slave box in the OBBC architecture is drawn dotted to illustrate that the slave dynamics are transparent to the operator. It is also important to note that object forces need not be manifested through the slave before communication back to the master controller.

The terms in the teleoperation architectures are related to the spacecraft docking task (pictured in Figure 1-1 of chapter 1) in the following discussion. The human operator or teleoperator is an astronaut or ground personnel manning the master controller. The master controller is the interface device that provides input for the control of a remote

9

slave and object. The slave, specifically for docking, is the attitude control and propulsion subsystems of the object or tracking vehicle at a remote environment. Finally, the environment is the approach and interaction between the tracking and target vehicles.

2.2 Qualitative Comparison of Teleoperation Techniques   Table 2-1 compares information obtained from the design and implementation of OBBC with existing master controllers' characteristics. Automatic control is included in this table because it represents the ultimate evolution in telemanipulation, otherwise known as telerobotics. The assessed characteristics for FRHC, master-slave and autonomous controllers are extracted from 4,5,10,11,12,13,16,17,20, and 21 and are divided into the two categories of engineering and human factors. In Table 2-1, the OBBC and Autonomous controllers are shaded to highlight their similarity. The letters "H", "M", and "L" stand for high, medium and low respectively and represent the degree to which each characteristic was manifested.

Controller	Characteristics	FRHC	Master-Slave	OBBC	Autono-mous
engineering factors	Design simplicity	L	H	H	L
	Ease of object/environment feedback implementation	L	M	H	
	Degree of autonomy			M	H
	Anthropomorphic dependency	L	H	L	L
	Tailored to task dynamics	L	L	H	H
	Universal	H	L	M	L
	Cost	H	M	L	
human factors	Ease of use	M	M	H	H
	Required training	H	M	L	L
	Requires coupling w/ visual f/b	H	H	L	L
	Ergonomic feedback	M	L	H	

Table 2-1. Qualitative Comparison of Teleoperation Techniques

General assessments could not be made about certain controllers for some of the characteristics, thus their respective table entries are left blank. The table illustrates two key points. It compares the human and engineering characteristics of each controller to show that the OBBC controller is a specialized FRHC which makes the controller more autonomous but less universal.

2.3 Prototype OBBC Mode Characteristics The prototype OBBC system is designed with four operational modes. All modes provide six degree of freedom (6-DOF) control of a tracking vehicle in a spacecraft docking simulation. Detailed information about the docking simulation is documented in chapters 3 and 4. The baseline mode offers only unilateral control with no feedback other than the visual feedback inherent to the graphical simulation. The remaining three modes additionally offer 1-DOF bilateral control reflecting information about the axis of approach. The four modes are as follows:

- Mode 1 : unilateral control
- Mode 2 : force reflection
- Mode 3 : virtual force reflection
- Mode 4 : electronic funneling

Table 2-2 is a comparison of the prototype's operational modes and a recommended alternative mode. The table summarizes the characteristics derived from each mode's feedback design and implementation discussed in sections 2.2.1 through 2.2.5. As in Table 2-1, the characteristics are assigned a grade of "H", "M "or "L". Assessments are made from the actual implementation of the four operational modes. The alternative mode's assessment is hypothesized.

2.3.1 Mode 1 - Unilateral Control. The unilateral control mode enables the teleoperator to provide the 6-DOF input required for control of the tracking vehicle during docking simulation. No feedback is returned to the operator in this mode. The operator

uses only visual feedback to accomplish the spacecraft docking. The target vehicle was assumed to have been designed with the necessary compliance to produce acceptable impact forces for a range of nominal terminal closure rates. This mode is most representative of current spacecraft proximity operations in that it is performed via a hand controller without bilateral control. One could postulate that a highly skilled and practiced operator using unilateral control with visual feedback could eventually perform the task optimally, without the need for special, non-visual feedback.

The performance, operation, and design of this mode is used as the baseline for the comparison of the design and operation of the other master controller modes. The human factor traits in this discussion are based on initial operation of the master controller in its respective modes.

2.3.2 Mode 2 - Force Reflection. The major kinematic objective in spacecraft docking is a translational approach along only one axis of motion (2). In mode two, force reflection, the approach axis force resulting from object/environment (i.e. tracking and target vehicle) interaction is reflected back to the operator. During the docking task, force on the approach axis due to the environment is non-existent unless contact between the target and tracking vehicles occur. Upon contact, either from task completion or an unplanned collision, contact force on the approach axis is modeled as the force generated by a simple spring that has sufficient force-length to allow the operator to "feel" contact for sufficient time to sense the rate of closure and to make required adjustments. This type of feedback would be extremely effective for docking if the target vehicle has a recoiling probe that is extended during docking to absorb impact forces.

During operation in this mode, the operator feels no interaction until actual contact is made with the target vehicle at task completion. Thus, force reflection indicates only that contact has occurred, regardless of whether this contact is from a successful docking or from a catastrophic collision. Though the operator easily recognizes task completion, force reflection information does nothing to enhance the operator's pre-contact

performance because the operator's ability to react is limited by the limitations on the spring force-length. Details about this mode and its integration into the OBBC prototype controller are found in section 4.2.3.1.

2.3.3 Mode 3 - Virtual Force Reflection. A unique feedback method is mode three, virtual force reflection. Virtual force reflection employs a virtual spring to provide the teleoperator with feedback throughout the task duration. The computer generates this virtual spring environment that continuously interacts with the tracking vehicle along the approach axis. The object/environment interaction is monitored by the computer and reflected back to the operator via the actuated master controller. The operator must apply forces to the master controller to overcome a growing virtual force. If the applied forces are greater than the reflected force, the operator can feel the master controller move in the nominal docking direction. The virtual force, calculated in object-space, grows linearly as the tracking vehicle approaches the target on the approach axis. Mode three is designed with a 100m linear virtual spring, but any spring length and force/displacement relationship could have been substituted. The virtual spring of mode 3 extends from the docking port of the target vehicle. The operator can apply forces at the master controller only sufficient enough to equal the maximum force of the virtual spring, making impact with the target vehicle theoretically impossible.

The virtual force is designed to moderate the negative effect of the feedback on the task performance, so the feedback is reflected only when motion is commanded with the manual controller, acting like a dead man trigger. This mode demonstrates how the OBBC concept permits the simple manipulability of the data derived from object/environment interaction, transforming it into ergonomic feedback by selecting only useful information for return to the teleoperator. The feedback thus allows the operator to concentrate on other task objectives.

During simulation, the motion along the axis of approach is automatically regulated as the tracking vehicle approaches because the virtual force limits the input

commanded at the master controller, leaving the operator free to align the remaining translational and rotational degrees of freedom. The commanded translation over the separation distance is extremely fast and returns continuous feedback to the teleoperator. While the operator remains dependent on visual feedback despite virtual force reflection, this mode enhances the manual control of the docking simulation and demonstrates the utility of single-axis force reflection. Details about this mode's integration into the prototype OBBC controller are found in section 4.2.3.2.

2.3.4 Velocity Reflection with Electronic Funneling. OBBC makes it possible to propose two methods of feedback, the previously mentioned virtual force reflection and velocity reflection with electronic funneling. These methods are unique to telemanipulation because OBBC allows the feedback design to integrate the major kinematic objective of the docking task, and OBBC facilitates the orchestration of all information derived from object and environment interaction. The second unique feedback, implemented in mode 4, is velocity reflection derived from funneled multi-axis object/environment interaction. Because the prototype OBBC hand controller is object-based, a simulated constant spacecraft velocity can be reflected back to the master controller. Approach motion of both the object and the master is prohibited until the operator uses the master controller to satisfy position and orientation constraints on the tracking vehicle. The constraints are dictated by a virtual hypercone originating at the docking port of the target vehicle and centered on the approach axis. If all constraints are satisfied, the docking approach can proceed. The operator "feels" the approach proceeding when the master begins to move at a constant velocity scaled from the actual spacecraft velocity. The master controller is actuated with forces derived from a constant force virtual spring that exerts a force on the master on the nominal docking axis. This force is actuated at the master only when a hypercone of constraints around the ultimate target position and orientation are satisfied and only when motion along the approach axis is commanded.

14

The constraints are described by a virtual hypercone, since they are dependent on information from all six degrees of freedom. The translational constraints of the hypercone are depicted in Figure 2-2. The virtual hypercone extends out from the nominal docking position. It is 25m in length, and its radius at any given point is dependent on the separation distance between the target and tracking vehicle along the axis of nominal docking direction. Thus the closer the tracking vehicle is to the target vehicle, the stricter the tolerances become on the error between the commanded position and orientation of the tracking vehicle and the nominally docked position and orientation. Inherent in this
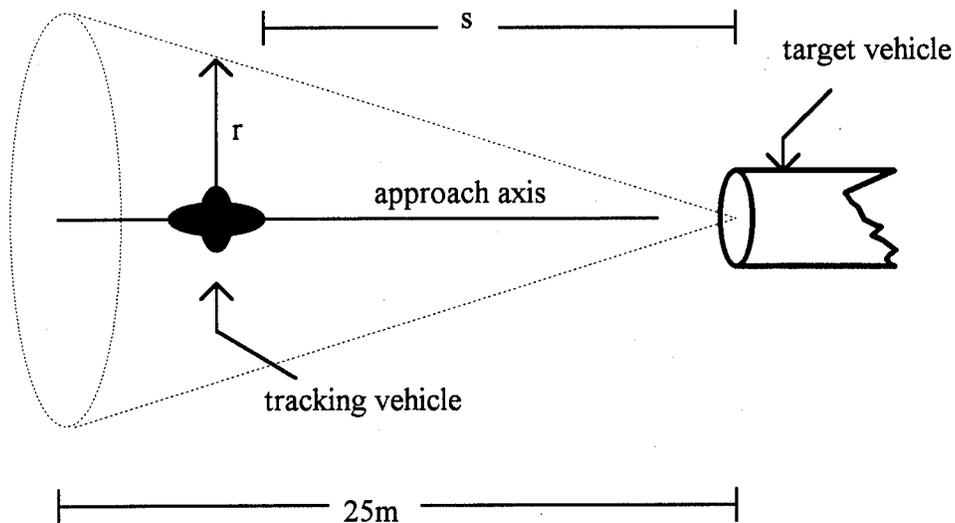
Figure 2-2. Depiction of Translational Hypercone Constraints

mode is a shared control between the teleoperator and the computer. Motion along the approach axis is handled by the computer, with executive control dictated by the operator's alignment of the remaining position and orientation axes. The virtual hypercone places constraints on the off-axes data to "funnel" the object's approach to a pre-planned target position. The feedback is unique to teleoperation because it is derived from multi-axis data but is fed back to the teleoperator via single-axis actuation of the master

controller; therefore, both the approach to the target vehicle and the multi-axis feedback are, in a sense, funneled.

During the simulation, velocity feedback with electronic funneling allows quick and accurate task completion when coupled with visual feedback. Small simulation modifications could make task completion with this mode possible even without visual feedback. Teleoperator control is enhanced with the use of various input filtering functions programmed for the spaceball which modify the programmed velocity or sensitize off-axis input, thereby increasing the effectiveness of the mode. Final docking position and attitude are completely accurate but spacecraft terminal closure rate is high. Safety concerns related to this high terminal velocity can be addressed by modifying the ultimate target position, which can be accomplished by modifying the hypercone constraints.

The limited range of motion on the prototype hand controller's feedback actuation system forced the interruption of the docking simulation and hampered the evaluation of this feedback mode. Details about this mode's integration into the prototype OBBC controller are found in section 4.2.3.3.

2.3.5 Mode 5 - Virtual Environment. A virtual environment is suggested as an alternative master controller mode, designed to reflect position, velocity, and (virtual) interaction forces to the teleoperator. The virtual environment is practical and easily implemented because OBBC sensors at the object provide object-space data. This mode would require the implementation of the optical encoder and augmentation of its associated processing hardware. As a result, closed loop motor control can be utilized. The hand controller could then be designed with position references which are calibrated to match actual positions. Position commands are input via the hand controller, and the actual position/velocity is bilaterally returned to the operator. Virtual forces could be reflected to complete the package. The controller becomes a mini-virtual environment, and visual feedback is no longer necessary for manual control. This alternative mode

16

which represents nominal telepresence, is discussed for comparative purposes and was not implemented.

FEEDBACK DESIGN	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5
ergonomically designed	L	L	M	H	H
tailored to task kinematics	L	L	M	H	H
ease of implementation	H	H	M	M	M
derived from multiple axis data				H	H
TASK PERFORMANCE					
completion speed	L	L	H	M	H
completion accuracy	L	L	M	H	H
contingency capability (manual)	H	H	H	H	H
dependency on visual feedback	H	H	H	M	L

Table 2-2. Comparison of Implemented and Alternative Feedback Modes

III. Description of Prototype Hardware for OBBC

3.1 Overview To investigate and demonstrate OBBC, a six degree of freedom (6-DOF) master hand controller actuated to provide feedback had to be designed and constructed. The fabricated master controller is a position control interface device for a spacecraft docking simulation. The complexity and cost of multi-axis feedback, as well as the nature of the docking task, suggested that a master controller with feedback in only the nominal docking direction would supply the teleoperator with sufficient feedback for enhanced telepresence. The prototype master controller is pictured in Figure 3-1. The four major components of the controller are the actuator, the servo-amplifier and power supply, the CIS Dimension 6 spaceball, and the spaceball platform and actuator casing.
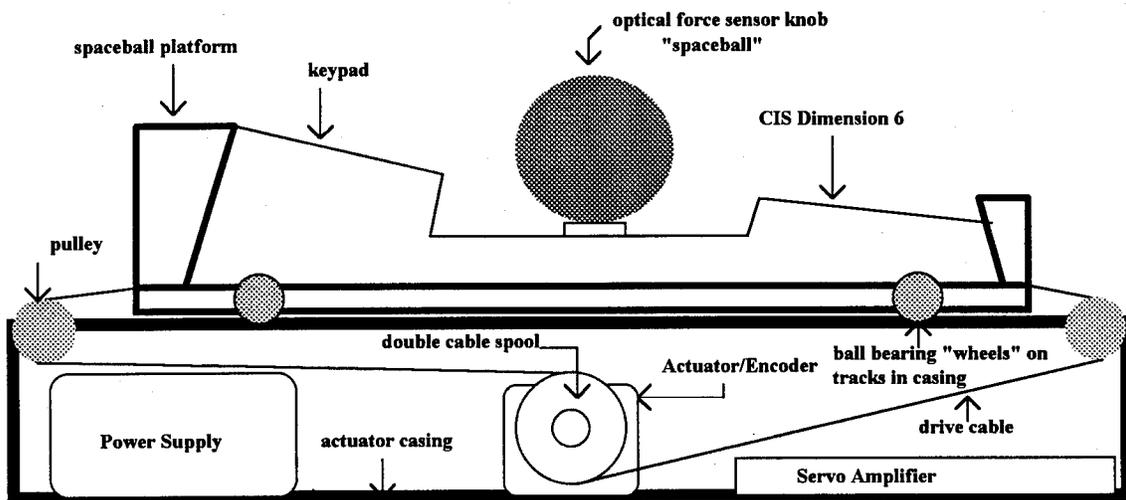
Figure 3-1. Prototype OBBC Master Controller

During the simulation, the operator envisions the master controller spaceball platform as the spacecraft in the simulation display and inputs changes to the spacecraft position and attitude by imparting a force or torque on the spaceball knob, acting as if

using his hand to position the spacecraft itself. The actuator casing always remains immobile, but in modes 2 through 4, described in the previous chapter, the approach axis force sensed at the spacecraft from its interaction with the environment is converted to a proportional torque that is applied to the platform actuator. The operator then "feels" one axis of spacecraft/environment force (or motion in mode 4) with his hand.

3.2 Actuator The actuator is a Globe 24VDC electric motor, selected because it was back-driveable and already in AFIT's possession. The actuator generates the necessary reflected feedback. Attached to its motor shaft is a double cable spool, which is designed to simultaneously reel in and out six gauge metal cable. The drive cable is secured to each end of the spaceball platform and pulls on the platform to move it in either direction along the tracks, the direction being dependent on the direction of the reflected force. The spaceball platform is immobile due to the mechanical impedance of the drive system unless voltage is supplied to the actuator. Also attached to the motor is a Hewlett Packard 500 count resolution optical encoder.

3.3 Servo-amplifier and power supply The servo-amplifier is a Copley Controls Corp. 300 Series Amplifier powered with a Series 600 unregulated DC power supply, requiring a standard 120V AC outlet. The amplifier is used as a voltage to current converter and is operated in the flat-gain mode, requiring a reference or control voltage of ± 75mV. Specific amplifier information can be obtained from its user manual (22). Unforced movement of the spaceball platform requires at least a ± 35mV reference voltage signal to the servo-amplifier which overcomes the impedance of the drive system.

3.4 CIS Dimension 6 Spaceball The CIS Dimension 6 Spaceball (Dim 6) contains a 6-DOF optical force sensor integrated into a spherical knob. It is this force sensor that is used to modify the tracking vehicle's position vector. A standard RS-232 connection and a null modem device allows serial interface with a Silicon Graphics IRIS Indigo workstation (SGI). The Dim 6 has dip switch selectable communication parameters, the settings for which are found in Appendix F.

19

In addition to the optical force sensor, the Dim 6 has a keypad complete with 8 function buttons and 3 transmission modes. Function buttons are user definable, and these functions are defined in section 4.2.1. Figure 3-2 depicts the buttons used for the three transmission modes, labeled as "TRA", "DOM", and "ROT". When initially powered up, the Dim 6 is in default mode and will transmit forces sensed in all six degrees of freedom. The teleoperator can use the TRA button to transmit only translational inputs and the ROT button to transmit only rotational inputs. The DOM button transmits only the dominant force sensed by the spaceball and can be used in conjunction with any transmission mode.
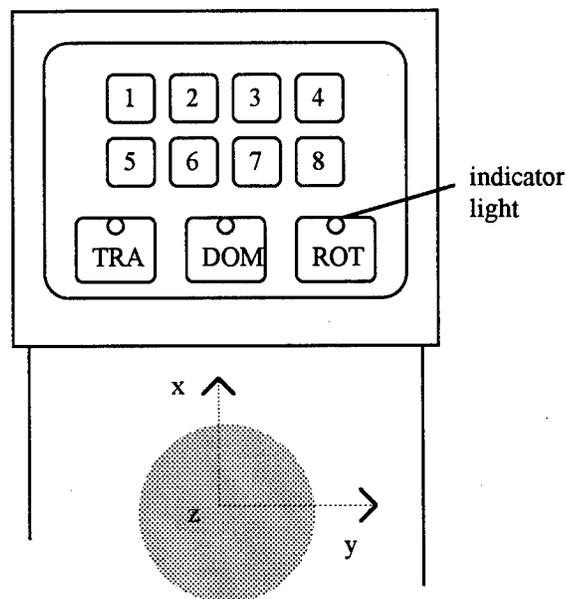
Figure 3-2. Dim 6 Keypad

Additional information about the Dim 6 can be obtained from its user manual (26).

3.5 Spaceball Platform and Actuator Casing The Dim 6 provides the required 6-DOF command input to the spacecraft, but by itself is incapable of reflecting feedback to the

teleoperator. Thus a platform capable of "frictionless" motion in the $\pm$ x (approach axis) direction was constructed with aluminum and four ball-bearing wheels. The Dim 6 is mounted to the platform, and the drive cable of the actuator is attached at both ends. The platform glides on tracks machined into the top of the actuator casing, prohibiting out of plane motion. The range of translational travel is up to three inches. The actuator casing provides a housing for the actuator motor and spool, as well as the servo-amplifier and the power supply. Additionally, the casing provides a pulley interface for the connection between the actuator cable and the spaceball platform.

IV. Prototype Object-based Bilateral Controller Design

4.1 Overview   Several programs were required to perform overall system control. The server_frobmc.c program of appendix B, in conjunction with the frobmc.gsl program of appendix C, performs two main functions.   First the programs serve as a fully resolved position control system that performs tracking of a desired position command input. Secondly, these programs function together as a master control system that processes operator input from a master controller operating in one of four operational modes and actuate the master controller to return feedback to the operator.   A secondary function of server_frobmc.c allows user- selectable scaling of hand controller input. The simulation driven by frobmc.gsl with input from server_frobmc.c displays the approach of a 6-DOF rigid body spacecraft to an immobile target vehicle. The main visual display provided to monitor docking is a cockpit view and is complete with dual crosshairs to enhance visual feedback. A set of crosshairs is attached to each docking point on the tracking and target vehicles. A second display provides a top view of the docking. The target vehicle is a space station constructed from graphics provided by Deneb Robotics for Interactive Graphics for Robotic Interface Program (IGRIP). The tracking vehicle is modeled after the Precision Orbital Tracking Vehicle (POTV), the spacecraft of Lawrence's thesis (23). Both vehicles' graphics and the basic simulation design were originally programmed by Tom Bridgman (24). User keyboard interface is required to input the tracking method and operational mode preferences, and the initial separation distance necessary to initiate the simulation. The server_frobmc.c coding is accomplished in standard UNIX C, and a complete listing as well as a flowchart can be found in Appendix B. This software is intricately linked with the Graphics simulation code, which drives the visual display of the docking on an SGI Indigo workstation. The frobmc.gsl coding is written in Graphics

22

Simulation Language (GSL) for IGRIP, and a complete listing as well as its flowchart can be found in Appendix C.

4.2 Prototype OBBC Architecture   The function of the OBBC control system is to force the position of the object to track the position input at the master controller while simultaneously allowing the master controller actuation system to track the influence of the environment on the object. Thus the OBBC teleoperation system can be considered a two-port network. One port represents the interaction between the operator and the master controller, while the other port represents the interaction between the spacecraft (object) and its environment. The object/environment port must be created to allow the slave (spacecraft attitude control/propulsion subsystems) dynamics to remain transparent to the operator. One way to represent the OBBC two-port architecture is illustrated in Figure 4-1 and is discussed in the following sections.
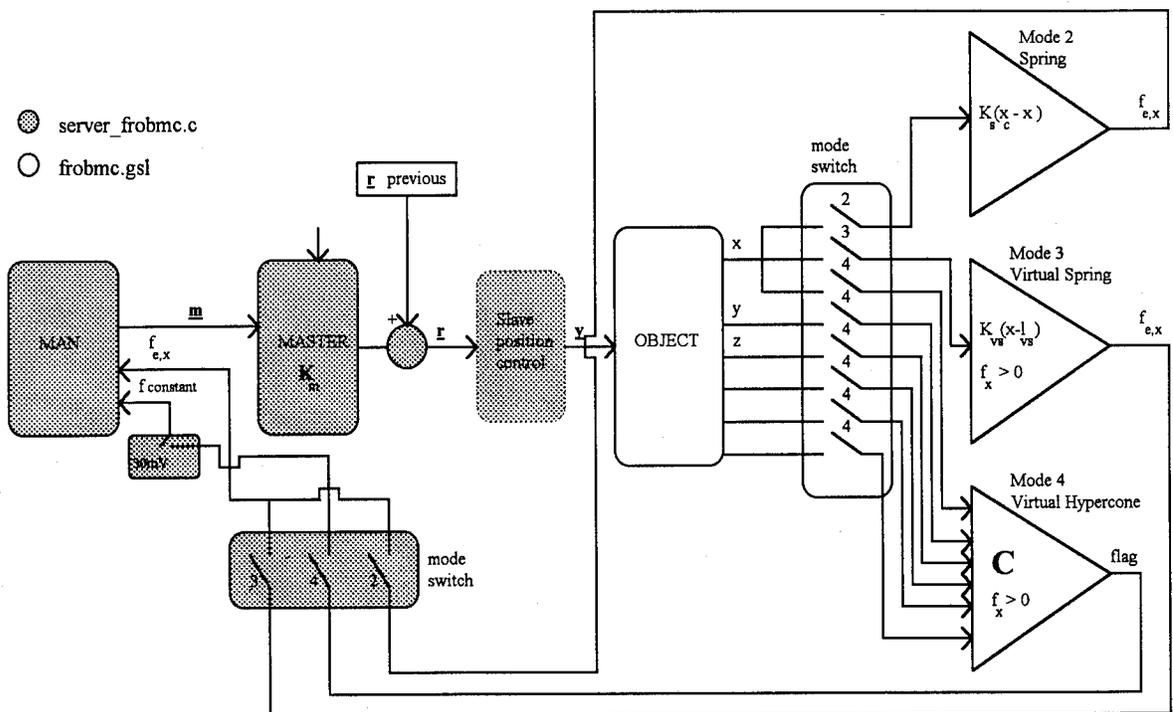
Figure 4-1. Prototype OBBC Architecture

4.2.1 Master Control. To move the simulated object (tracking vehicle), the teleoperator imparts a force or a torque in the desired translational or rotational direction on the hand controller knob. This input forms the master input vector and is made up of three forces and three torques:

$$\mathbf{m} = [f_x \ f_y \ f_z \ t_x \ t_y \ t_z]^{\mathrm{T}}$$

The server_frobmc.c program then translates this into meters/degrees with the mapping function, K_m. K_m maps forces and torques exerted on the control knob to values between 0 and 1 meters and between 0 and 1 degrees respectively.

If operating the master in mode 1, the mapped input is simply summed with the current desired position command vector. A keyboard input of the initial desired position is required before the simulation will commence. Operation in modes 2 and 3 sums the mapped input with the current desired position command vector, as well as with the force information derived from the object/environment interaction. In mode 4 operation, server_frobmc.c evaluates a flag returned from frobmc.gsl. This flag indicates whether or not the mode 4 constraints are satisfied. When the constraints are satisfied, the desired command vector is updated with a constant 1m input. Unsatisfied constraints allow the desired command vector to remain unchanged. The updated desired position command vector is made up of three translational positions and three relative attitude angles:

$$\mathbf{r} = [x_{des} \ y_{des} \ z_{des} \ \theta_{des} \ \phi_{des} \ \psi_{des} \ ]^{T}$$

At simulation initiation, the initial desired position command is input via the keyboard.

Server_frobmc.c enables the teleoperator to use the function keys on the keypad of the master controller to modify the scale and sensitivity of the commanded input.

24

Function button #4 reduces input to 1/10 the default input capability. Depressing button #5 cuts commanded input of the off-axes by half while button #8 doubles the input of the off-axes. The operator can double the input of the approach axis with button #6. Finally, the input can be restored to default levels with button #2. Table 4-1 summarizes the keypad functions.

Button #	Function
2	reset all
4	$0.1[f_y\, f_z\, t_x\, t_y\, t_z]$
5	$0.5[f_y\, f_z\, t_x\, t_y\, t_z]$
6	$2f_x$
8	$2[f_y\, f_z\, t_x\, t_y\, t_z]$

Table 4-1. Master Controller Keypad Functions

4.2.2 Slave Position Controller Design. The updated desired position command vector becomes the input to a slave position controller. For a spacecraft, this slave controller would be the attitude control and propulsion subsystems. The design of the controller (see Figure 4-2) is borrowed and modified from a thesis by Richard E. Lawrence (23). This design includes the controller system, input, pre-filter, and gain matrices optimized for both the V-Bar and R-Bar tracking methods. The R-Bar tracking method is an approach to the target along the radius vector of the target vehicle, while V-bar tracking is an approach to the target vehicle along a horizontal component of the target vehicle's tangential velocity vector (25).
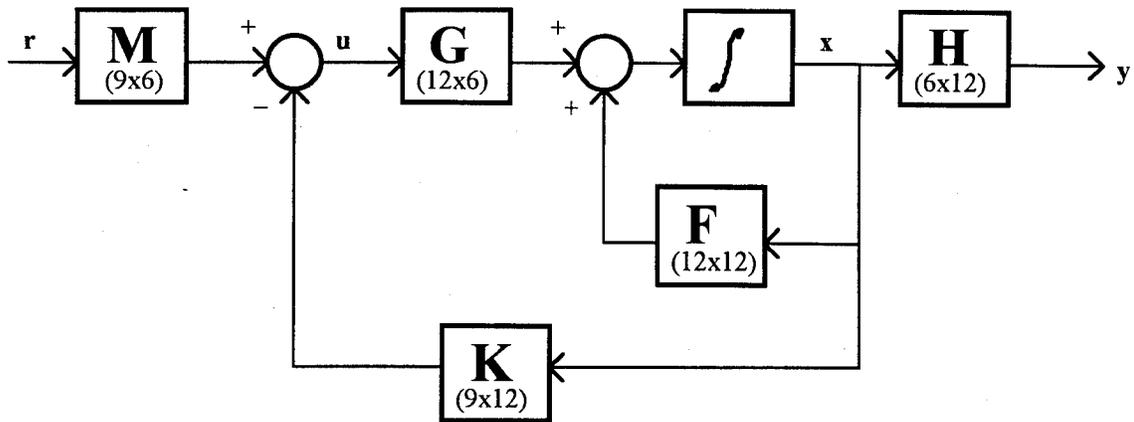
Figure 4-2. Position Controller Block Diagram

The vectors and matrices in Figure 4-2 are defined as follows:

$r^{6x1} \equiv$ desired position command vector $F \equiv$ system matrix

$u^{9x1} \equiv$ control vector $G \equiv$ input matrix

$x^{12x1} \equiv$ system state vector $H \equiv$ output matrix

$y^{6x1} \equiv$ output position vector $M \equiv$ pre-filter matrix

$K \equiv$ controller gain matrix

The controller code utilizes a 4th-order Runge-Kutta method for the necessary integration of the relative equations of motion and is extracted from a program by Tom Bridgman (24). The linearized equations of motion of the tracking vehicle relative to the target vehicle, as well as the non-gravitational forces and moments, are completely derived by Lawrence (23). The complete set of both translational and rotational relative equations of motion are as follows:

$$x - 2ny - 3n^2 x = \frac{F_x}{m} \qquad (4.1)$$

$$y + 2nx = \frac{F_y}{m} \qquad (4.2)$$

$$z + n^2 z = \frac{F_z}{m} \qquad (4.3)$$

$$A(\theta - n\phi) + (C - B)(n\phi + n^2\theta) = M_x \qquad (4.4)$$

$$B(\phi + n\theta) + (A - C)(n\theta - 4n^2\phi - \frac{3n^2 z}{r_o}) = M_y \qquad (4.5)$$

$$C(\psi - \frac{3nx}{2r_o}) + (B - A)(3n^2\psi - \frac{3n^2 y}{r_o}) = M_z \qquad (4.6)$$

where F_x, F_y, and F_z are non-gravitational forces; M_x, M_y, and M_z are non-gravitational moments; $n = \sqrt{GM_\oplus / r^3}$; A, B, and C are principal moments of inertia of the tracking vehicle; m is the mass of the tracking vehicle; r_o is the magnitude of the target vehicle's position vector; and r is the magnitude of the tracking vehicles position vector.

4.2.3 Object/Environment Interaction. The slave controller output position vector **y** is used by frobmc.gsl to drive the position and attitude of the object (tracking vehicle) in the docking simulation. The operator then uses this simulation as visual feedback. At this point, unilateral master control (mode 1) is completely described. For the remaining operational modes, frobmc.gsl creates a simulated environment to provide a source for the non-visual feedback.

4.2.3.1 Mode 2 - Force Reflection. In mode 2, the force on the axis of approach from any contact between the tracking and target vehicles is modeled as the force of a linear spring. This force, limited to the approach axis due to master controller design, is calculated in object space as described in Figure 4-1, where x_c is the position of contact, and x is the commanded position of the object that caused the contact. K_s is a spring constant and makes the force proportional with f_x. The resulting object-based force $f_{e,x}$ is summed with the f_x input sensed at the master. The summed force is then

converted to the necessary voltage units to actuate the spaceball platform on the master controller.

4.2.3.2 Mode 3 - Virtual Force Reflection. In mode 3, the source of the feedback created by frobmc.gsl is a virtual spring. The operator must overcome the virtual force generated by a 100m long virtual spring that originates from the target vehicle. For the prototype, this virtual spring is infinitely wide and the force $f_{e,x}$ is calculated as described in Figure 4-1, where l_{vs} is the unforced length of the virtual spring. As before, this force is limited to the axis of approach because it is this axis for which feedback is available at the master controller. K_{vs} is a spring constant and makes the force proportional to f_x. No force is created until the separation distance is less than the 100m length of the spring. Furthermore, the force is fed back only when the teleoperator makes a command input in the nominal docking direction ($f_x > 0$). The feedback imparted to the operator by the master is the sum of the commanded force f_x and the virtual force $f_{e,x}$. This summed force is then converted to voltage and relayed to the manual controller. If a collision with another part of the target vehicle should occur in this mode, this force is reflected back by the same method used in mode two.

4.2.3.3 Mode 4 - Velocity Reflection with Electronic Funneling. In mode 4, the code again generates an environment for object interaction that serves as the source for the non-visual feedback. Unlike the previous two modes, all of the object's position and attitude components are utilized in the derivation of the feedback. A hypercone of off-axes alignment constraints is created from the separation distance between the tracking and target vehicles. As can be seen in Figure 4-1, the feedback is only possible when the constraints **C** of the virtual hypercone are satisfied by the ongoing simulation and only when the operator is commanding motion in the nominal docking direction ($f_x > 0$).

The constraints **C** are explicitly defined below:

$$|y - y_t| < 0.25s$$
$$|z - z_t| < 0.25s$$
$$|\theta - \theta_t| < 0.25(\tfrac{\circ}{m})s$$
$$|\phi - \phi_t| < 0.25(\tfrac{\circ}{m})s$$
$$|\psi - \psi_t| < 0.25(\tfrac{\circ}{m})s$$

where the subscript t indicates a target vehicle variable and the separation distance is s = $(x-x_t)$. The 0.25 constant was arbitrarily chosen. The constraints were specifically made dependent on the separation distance, tailoring the feedback design to task dynamics. The constant can be changed to make the constraints more stringent. In order to provide a concrete example of the tolerances created by the constraints, consider two vehicles with a separation distance of 12m. The translational position error on the off- axes can be no larger than 3m and the attitude angle error can be no larger than 3°.

The constraints are monitored by frobmc.gsl with additional visual feedback provided to the operator if the constraints are not satisfied. This additional visual feedback is in the form of the background color of the simulation display and the set of crosshairs representing the target and tracking vehicles. The operator adjusts off-axes translational positions by "superimposing" the tracking vehicle crosshairs on the target vehicles crosshairs. The background color of the cockpit view display will turn green if the tracking vehicle with respect to the nominal docking attitude is off in yaw, yellow if off in pitch, and purple if off in roll. Because motion of the tracking vehicle, and thus the master controller, is only permitted when the software verifies that all constraints are satisfied, increased system autonomy and shared control is implied.

4.3 Communication Software Communication is a very important design trait for a teleoperation system because time delay is a major problem source for teleoperation (18). Total round-trip communication for the prototype controller is less than 1/5 of a second, which causes a slight discrepancy between the visual feedback and the feedback used to

provide bilateral control. The communication network for the prototype OBBC simulation consists of the basic components needed to perform teleoperation but lacks realism in its representation of all components necessary for true remote teleoperation. For example, the displays provided for the visual feedback during simulation would actually require radio communication to remote cameras. This type of communication was not part of the OBBC simulation.

The communication software that provides the necessary teleoperation communication to support the data flow illustrated in Figure 4-3 is written in standard UNIX C language. A complete source listing can be found in appendix D.
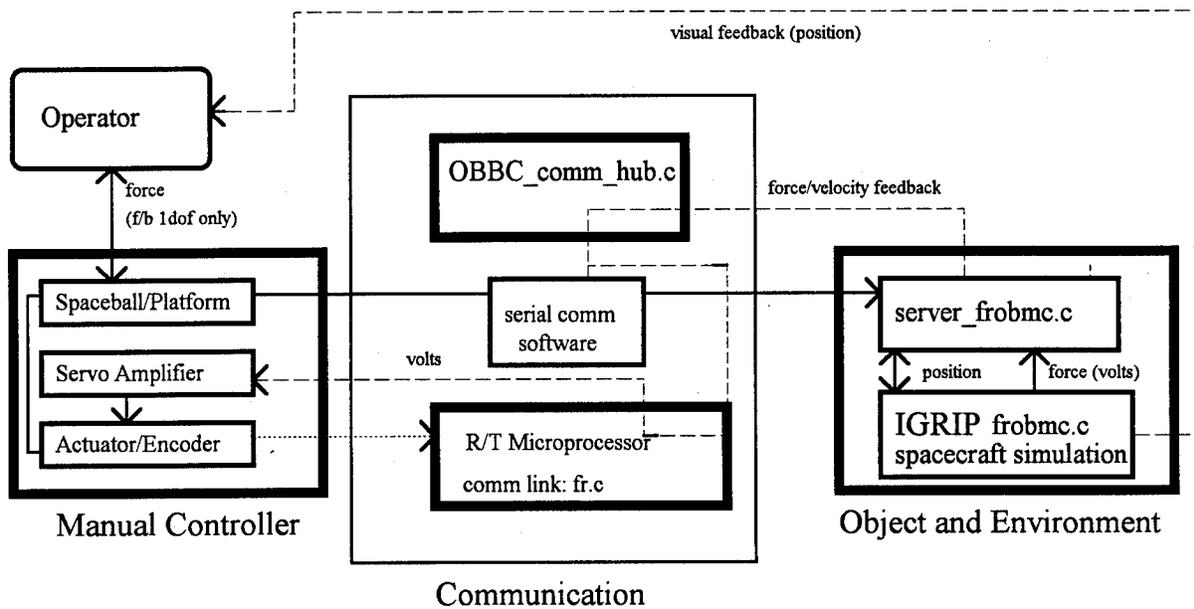
Figure 4-3. System Data Flow

4.3.1 Ethernet Communication. The central hub of all ethernet communication is OBBC_comm_hub.c . It is executed on a Sun Microsystems SPARC 2 Workstation. This hub provides communication between the SGI workstation and the CHIMERA-operated real-time VME microprocessor. The code providing the communication link at

the microprocessor necessary for D/A and A/D interface with the manual controller is the program fr.c.

4.3.2 Master Controller Input Communication. The communication software also provides serial communication between the SGI workstation and the CIS Dimension 6 spaceball portion of the master controller. Because the master controller control is performed in the same location as the creation of the virtual environment and the control of the simulated object, this communication is not truly indicative of remote teleoperation. This communication satisfies simulation requirements, but again, reality would require a radio communication component necessary to relay information between the master and a remote object. The source code is written to support all selectable communication parameters of the Dimension 6 spaceball.

4.4 IGRIP Environment The Deneb Robotics IGRIP software provides many interactive tools that the teleoperator may find useful during the simulation. Three that have aided in a successful completion of the docking task are the **joint values, magnify, and rotate world** functions. The operation of these tools is described in the operation instructions found in Appendix E. The joint value function allows the user to view the tracking vehicle's translational and rotational position values. The magnify tool, when used in conjunction with the rotate world tool, allows for a close inspection of the mating between the target and tracking vehicles. These tools can be used to enhance the views generated automatically by frobmc.gsl.

V. Demonstration of Prototype Operation

5.1 Task Description The operation of the prototype OBBC master was limited. A simple spacecraft docking simulation was used to demonstrate the functional capability of the controller in each of its four modes. In each case the tracking vehicle made an R-bar approach to the target vehicle and master control was initiated at a separation distance of 15 meters. No attitude angles were commanded. The approach was free of any possible contingency operation and accuracy of the docking position at task completion was not recorded. The task was considered complete when the separation distance was reduced to 15mm or less. The operator was expected to perform the task as accurately as possible and achieve a small terminal velocity. Table 5-1 summarizes each mode's simulation performance.

Characteristic	Mode 1	Mode 2	Mode 3	Mode 4
Initial separation distance	15m	15m	15m	15m
Tracking method	R-bar	R-bar	R-bar	R-bar
Maximum velocity (m/s)	0.0047	0.0071	0.0114	0.0165
Velocity at task completion (m/s)	0.0015	0.0003	0.0	0.0
Duration (s)	3800	4920	2320	4360

Table 5-1. Master Mode Simulation Summary

5.2 Closure Rate Profiles For each of the master controller modes, tracking vehicle velocity was recorded to illustrate functional differences. Figure 5-1 shows their separation closure rates. These plots should not be interpreted as an evaluation of the controller since insufficient controls were applied during data collection. Possible controls would have included such factors as accuracy at task completion, skilled or unskilled operators, task duration limits, and contingency response. The results are specific to the

operator and could change even if rerun due to their sensitivity to operator skill and style. For all modes, manual operation was initiated at 3900 seconds because all mode simulations were initiated with identical initial command vectors. The profiles for operation in modes 1 and 2 illustrate each mode's sensitivity to operator style. The mode 1 profile shows a non aggressive style indicated by small initial commanded velocities. The operator was forced to command input near task completion, resulting in a terminal velocity larger than any mode. In contrast, the profile recorded for mode 2 shows an aggressive operator style indicated by the large initial commanded velocities, which resulted in a slow terminal velocity.

When making further comparisons, it should be noted from Figure 5-1 that the mode 4 zero closure rates indicate translational corrections in the off-axes. The mode was designed so that the task could not proceed until the errors in the alignment of the lateral axes were corrected. This design characteristic allows unequaled accuracy in task performance. Errors in the lateral translational axis at task completion were significantly less than those of any other mode. In fact, a perfectly piloted tracking vehicle with mode 4 operation, could achieve significant reductions in task duration due to its high constant velocity input (note the maximum velocities in Table 5-1). Neglecting the time for off-axes correction in mode 4's profiled data, task duration is 2520s.

The profiles for modes 3 and 4 show no velocity at task completion, indicating that contact between the tracking and target vehicle is never achieved. This is possible only because the sensors are assumed to be perfectly accurate and reliable.
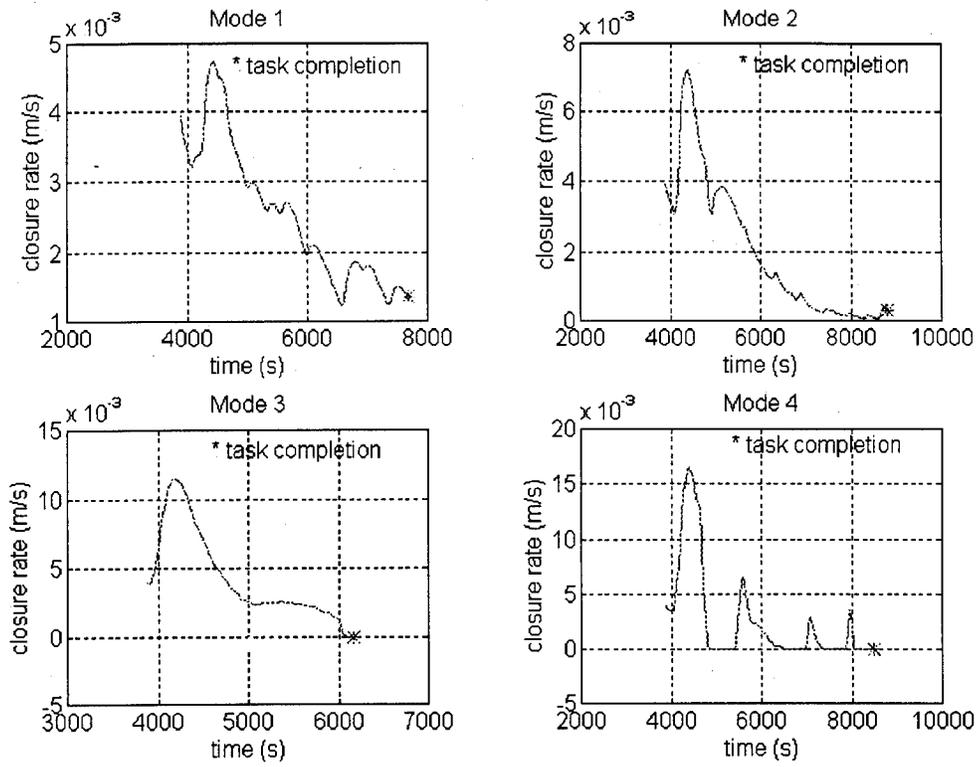
Figure 5-1. Spacecraft Closure Rates

VI. Conclusion and Recommendations

6.1 Conclusion Current teleoperation systems are based fundamentally on the slave that manipulates the controlled object, and master-slave symmetry is either required or assumed in order to provide enhanced telepresence to the operator via the master controller. Object-based bilateral control (OBBC) offers a new control concept for teleoperation and for master controller design. The benefits of anthropomorphicity and bilateral control are obtained without the need for kinematically/geometrically identical master-slave systems, complex calibration and joint mapping schemes, detailed task and manipulator knowledge, and expensive high DOF force reflection. Manipulator motion is transparent to the user and automatically generated. Subsequently, the controller autonomy is increased, thus enhancing task performance and significantly reducing the operator's required training for the task. Both the master controller and its feedback are tailored to the task and designed ergonomically with no geometrical limitations imposed by the design of the manipulator. The obviated need for complex joint mapping transformations implies that multiple axis feedback can be selectively reflected to a low DOF feedback capable controller, making single-axis feedback a desirable design trait. Mode 3 of the prototype controller demonstrates how OBBC can be used to increase the utility of single-axis feedback. Furthermore, OBBC facilitates the orchestration of multi-axis object/environment information, allowing for the return of useful information to a low DOF feedback controller. As a result, practical tactile feedback, like the velocity feedback of mode 4, can be implemented to provide enhanced telepresence without dependency on force reflection.

The feasibility of the telepresent OBBC architecture and the utility of its unique feedback has been demonstrated through implementation in a spacecraft docking

simulation. OBBC application is not limited to spacecraft docking, but includes aircraft munitions loading, micro-chemistry, and tele-surgery.

6.2 Recommendations OBBC is unique because its control is performed in object-space, and its design utilizes bilateral control. Though object-based teleoperation has very recently been demonstrated, never before has it been accomplished with bilateral control. Object-based teleoperation is in its infancy, and its general concept deserves further exploration.

6.2.1 Feedback Design. With OBBC, feedback is not just a means to provide heuristic, natural manual control. The feedback itself is designed ergonomically. Useful feedback for bilateral control is not limited to force reflection. Many types of tactile feedback can now be practical because the feedback design can be tailored to specific task kinematics. The many possibilities of feedback design, including optimal feedback "mixing" and selective feedback, are the most significant benefit of OBBC and should be explored. Specifically to the thesis work, Mode 4 could be enhanced by substituting the constant force spring with linearly increasing force derived similarly from the off-axis deviation.

6.2.2 Multi-axis to Single-axis Feedback. OBBC has demonstrated the utility of a single-axis feedback capable controller by funneling multi-axis data. This idea of "funneled feedback" should be researched. Master controllers designed with limited DOF feedback capability could significantly reduce their cost. The potential cost savings alone is reason enough to pursue multi-axis to single-axis feedback.

6.2.3 Closed Loop Motor Control. The feedback actuation on the prototype hand controller has been designed with an open loop motor control. The actuator that provides the 1-DOF bilateral control is equipped with an optical encoder. The communication software has been designed to handle this required communication flow. However, the hardware and software necessary to process encoder data must be augmented. Closed loop motor control would enhance the current controller's feedback modes by providing

accurate position feedback and would allow additional feedback designs, such as the

virtual environment suggested in chapter 2, to be more intricately coupled to the task.

Bibliography

1. Whitaker, D. L. (Ed.) Advanced Robotics for Air Force Operations. Washington DC: National Academy Press, 1989.

2. Syromiatnikov, V. S. Spacecraft Docking Devices. Princeton: Soviet Technical Paper NO. SSIVSS-1, 1990.

3. Corliss, W. R. and Edwin G. Johnson. Teleoperators and Human Augmentation. Washington DC: NASA SP-5070, 1967.

4. Corliss, W. R. and Edwin G. Johnson. Teleoperator Controls. Washington DC: NASA SP-5070, 1968.

5. Jacobus, H.N., A.J. Riggs, C.J. Jacobus, and Y. Weinstein. "Implementation Issues for Telerobotic Handcontrollers: Human-Robot Ergonomics." Human-Robot Interaction. London: Taylor and Francis, 1992.

6. Hirzinger, G., K. Landzettel, and J. Heindl. "ROTEX - Space Telerobotic Flight Experiment." SPIE Proceedings on Telemanipulator Technology and Space Telerobotics., edited by W. S. Kim. 2057: 51-72. Boston, 1993.

7. NASA Memorandum DM6-86-81, "Rendevous Flight Rules and Rationale." Trajectory Operations Branch, Johnson Space Center.

8. Sheridan, T. B. Telerobotics, Automation, and Human Supervisory Control. Cambridge: The MIT Press, 1992.

9. Milgram, Paul. "Human Performance Evaluation of Isometric and Elastic Rate Controllers in a 6 DOF Tracking Task." SPIE Proceedings on Telemanipulator Technology and Space Telerobotics., edited by W. S. Kim. 2057: 130-141. Boston, 1993.

10. Eberman, B. and Bin An. "EXOS Research on Force-Reflecting Controllers." SPIE Proceedings on Telemanipulator Technology., edited by Hari Das. 1833: 9-19. Boston, 1992.

11. Draper J. V., J. N. Herndon, B. S. Weil and W. E. Moore. "Effects of Force Reflection on Servomanipulator Performance." Proceedings of International Topical Meeting on Remote Handling and Robotics in Hostile Environments. pp. 654-660, American Nuclear Society, IL. 1987.

12. Handlykken, M. and T. Turner. "Control System Analysis and Synthesis for a Six Degree-of-Freedom Universal Force-Reflecting Hand Controller." IEEE, 1980.

13. Bejczy, A. K. and J.K. Salisbury. "Controlling Remote Manipulators Through Kinesthetic Coupling." Computers in Mechanical Engineering. July, 1983: 48-60.

14. Schneider, S. A. and Robert H. Cannon, Jr. "Object Impedance Control for Cooperative Manipulation: Theory and Experimental Results." IEEE Proceedings of the NASA Conference on Space Telerobotics, (Pasadena CA), NASA, February 1989.

15. Durlach, N. "Human-Machine Interfaces for Virtual Environment and Teleoperation Systems." Briefing, Armstrong Labratory, 21 June,1994.

16 Michelman, P. and P. Allen. "Shared Autonomy in a Robot Hand Teleoperation System." IEEE International Conference on Intelligent Robots and Systems. 1994.

17. Hannaford, B. "A Design Framework for Teleoperators with Kinesthetic Feedback." IEEE Transactions on Robotics and Automation. 5: 426-434. August 1989.

18. Anderson, R. J. and M. W. Spong. "Bilateral Control of Teleoperators with Time Delay." IEEE Transactions on Automatic Control. 34: 494-501. May 1989.

19. Draper, John V. "Human Factors in Telemanipulation: Perspectives from the Oakridge National Laboratory Experience." SPIE Proceedings on Telemanipulator Technology and Space Telerobotics., edited by W. S. Kim. 2057: 162-174. Boston, 1993.

20. Bejczy, A. K. "Teleoperation: The Language of the Human Hand." IEEE International Workshop on Robot and Human Communication. 32-43: April, 1992.

21. Vertut, J. and P. Coiffet. Teleoperation and Robotics Evolution and Development. 3A & 3B, London: Kogan Page, 1986.

22. Copley Controls Corporation. 300 Series Amplifier User's Guide. Rev. 6: March, 93.

23. Lawrence, R. E. Jr. "An Electromagnetically Controlled Precision Orbital Tracking Vehicle (POTV)." Thesis. AFIT, 1992.

24. Bridgman, T. and C. H. Spenny. "Simulation of POTV." Proceedings of 1993 Deneb User Group, 35-37: Auburn Hill, MI, December 1993.

25. Hall, William M. An Introduction to Shuttle/LDEF Retrieval Operations: The R-bar Approach Option. NASA TM-78668, February 1978.

26. CIS Graphics Incorporated. Dimension 6 User's Manual. Version A: July, 1988.

A.1 Design Drawings: Spaceball Platform (cm, not to scale)
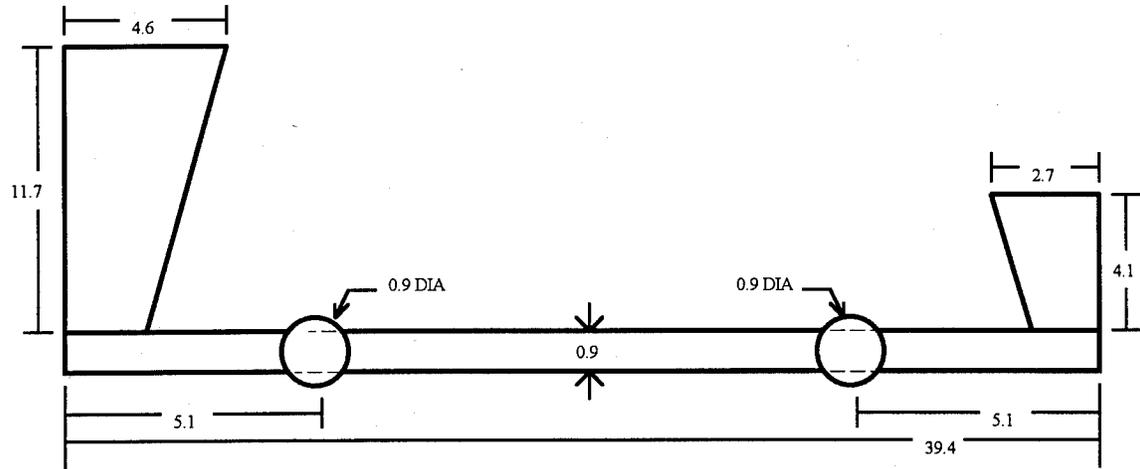
Figure A-1. Side View - Spaceball Platform
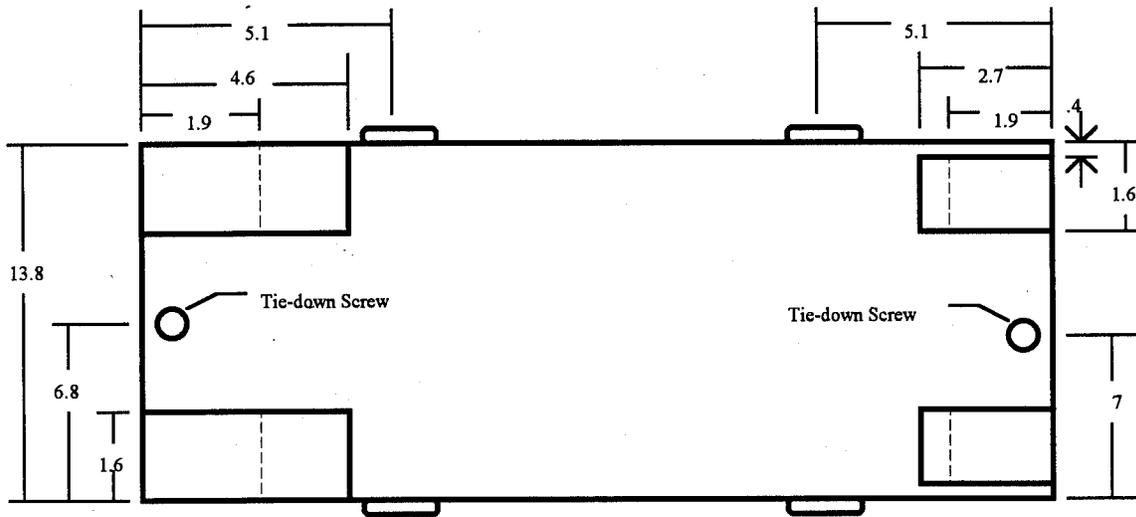
Figure A-2. Top View - Spaceball Platform
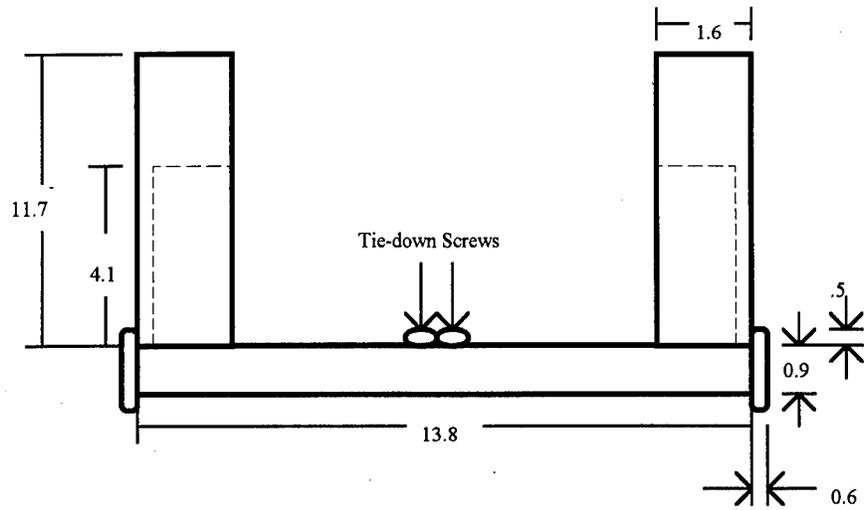
Figure A-3. Front View - Spaceball Platform

A.2 Design Drawings: Actuator Casing (cm, not to scale)
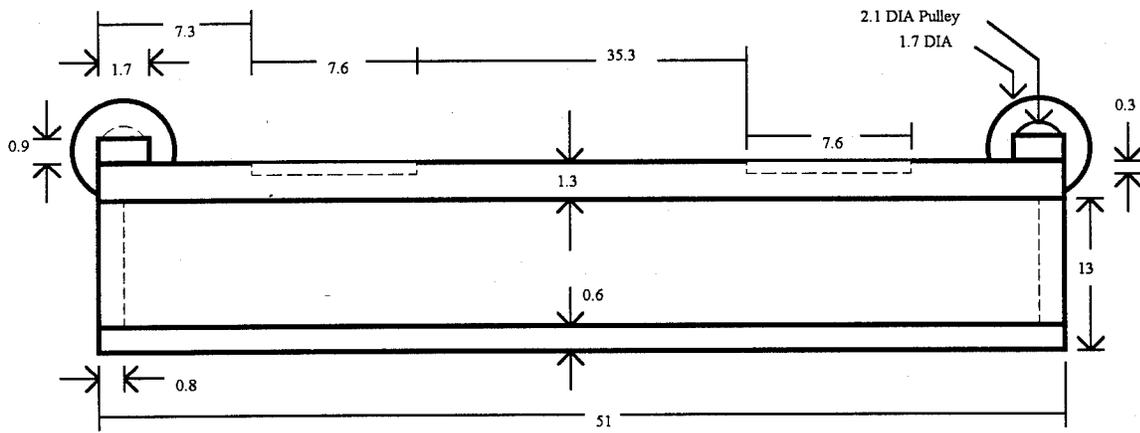
Figure A-4. Side View - Actuator Casing
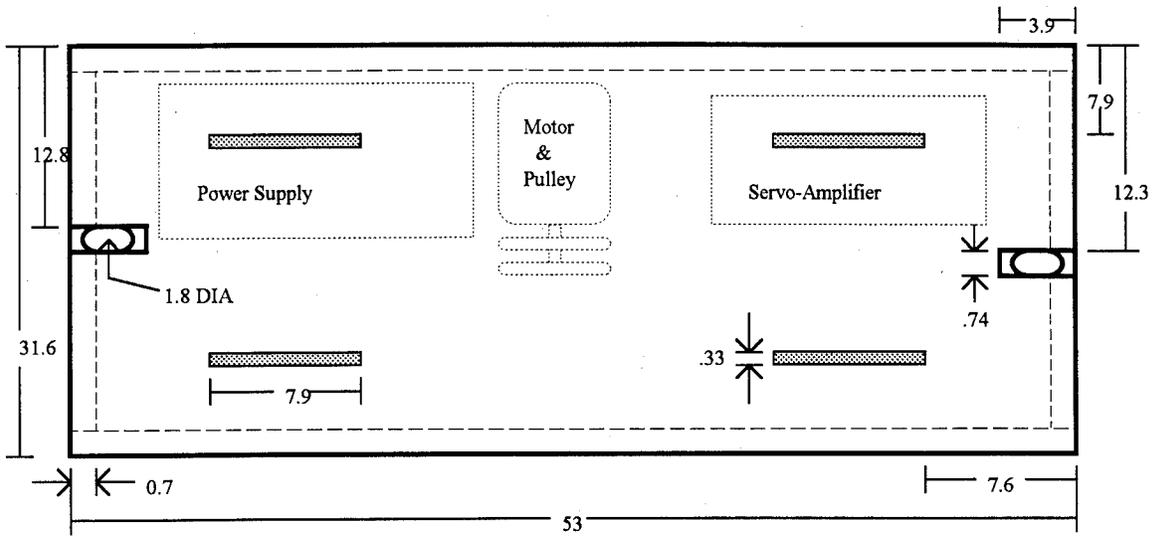
Figure A-5. Top View - Actuator Casing

Appendix B

B.1 Prototype's Server Software: Flowchart
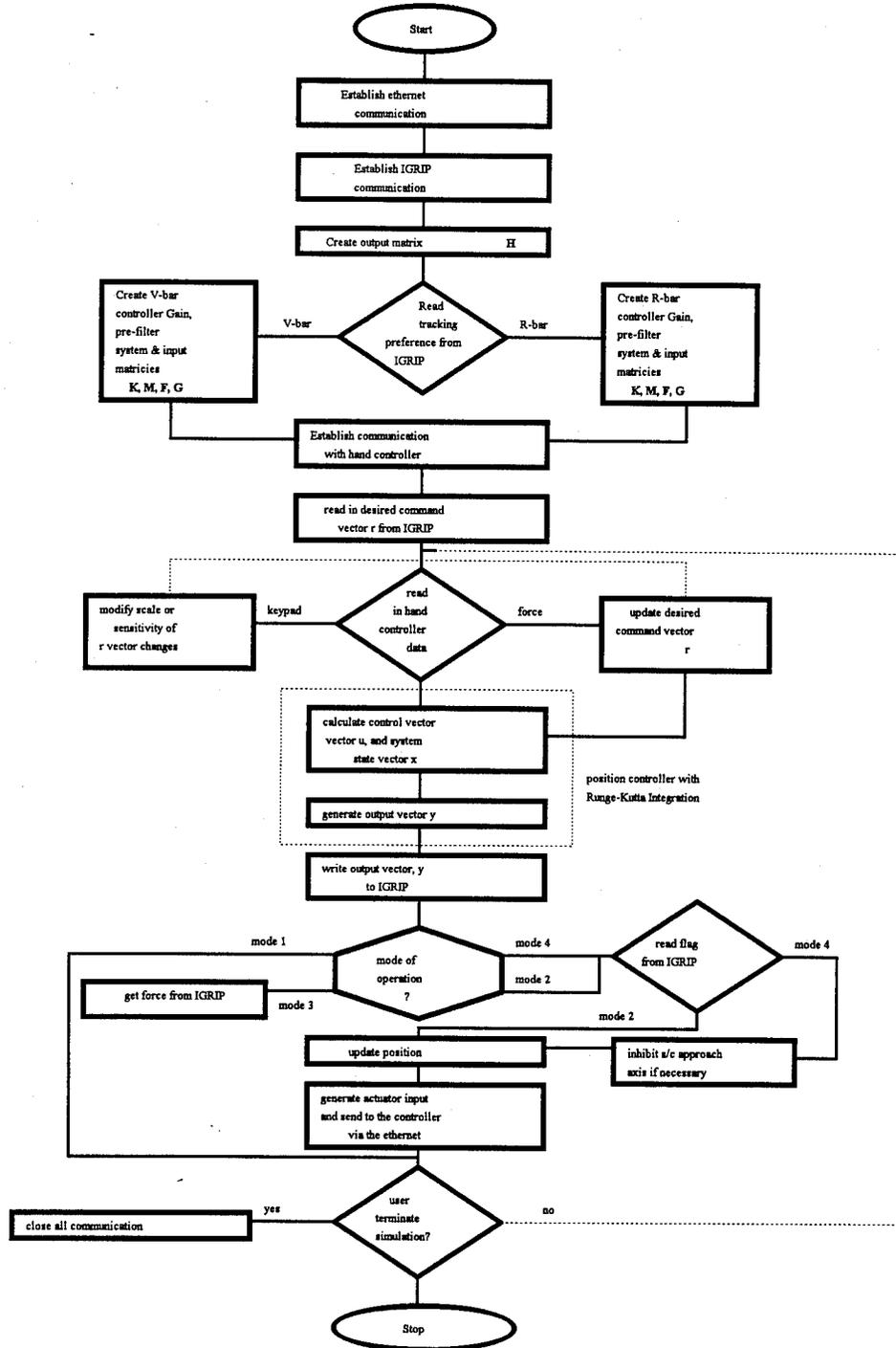
Figure B-1. Server_frobmc.c Flow Chart

```
/*********************************************************************
 * Name: Paul Woznick, Jul 94                                        *
 *   Server_frobmc.c Adapted from code provided by T. Bridgman       *
 *                                                         *         *
 * Purpose: The purpose if this program is to implement a            *
 *              4th order Runge-Kutta routine needed to solve        *
 *        *                                                          *
 *              differential equations. Specifically, the routine will   *
 *              perform motion generation computations for a 6 DOF   *
 *                                                                   *
 *              rigid body (r-bar, and v-bar tracking)               *
 *              for docking procedures. The target platform          *
 *              is a space station in circular orbit.     *          *
 *                                                        *          *
 *********************************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>
#include <signal.h>
#include <arpa/inet.h>
#include "sb.h"

#define MAXLINE 4096
#define MAX_TIMES 500

float init[12], newfunc[12], new[12], k1[12], k2[12], k3[12], k4[12];
float fcl[12][12], frcfunc[12][6], K[9][12], Kinv[12][9], M[9][6];
float u_bias[9], input[6], volts, frdof, force_volts;
float t1,t2,t3,t4,t5,t6,x_home,x,scale,scalex;
float finalout_1,collide_x,moved;
struct sockaddr_in cli_addr,serv_addr;
struct servent *sp;
char inbuf[MAXLINE],outbuf[MAXLINE];
int sockfd,op_mode,flag1,scale_button,n,collision;
int clilen, childpid, newsockfd,count;
```

```c
void open_spaceball();
void init_spaceball(Byte mode);
void write_spaceball();
int  read_raw_spaceball();
void read_spaceball();
void parse_spaceball();
void close_spaceball();
int read_standard ();
void parse_standard ();
void calculate_velocity();
FILE *out;

main(argc, argv)
int argc;
char *argv[];
{

/*
 * Declare the 'main' program's variables.
 */
  SpaceballData sd;
  char igripbuf[MAXLINE];
  char returnbuf[MAXLINE];
  int sockport;
  int sockdsc;
  int i, j, k, l, mm, ctrl_num;
  int stop_process, test;
  float fcn(float[], float[], int);
  float pi, facos(float), sum;
  float frcfix[12], output[6][12];
  float init[12], no_change, previous_t3;
  float u[9], M_fix[12], altitude;
  float finalout_2, finalout_3, delta;
  float finalout_4, finalout_5, finalout_6;
  float sensitize,collide_x;
  void r_bar();
  void v_bar();
  void motor_control();
  void scale_voltage();

        printf("The server for object is started.\n");

/* Open up file for data collection */
  out = fopen("velocity4.out","w");
```

```c
/*
 * Initialize all the variables pertinent to this program to
 * zero (0), to limit the errors which might occur in the
 * computational process.
 */
 for (i = 0; i <= 11; i++)
 {
   finalout_1 = 0.0;
   finalout_2 = 0.0;
   finalout_3 = 0.0;
   finalout_4 = 0.0;
   finalout_5 = 0.0;
   finalout_6 = 0.0;
   frcfix[i] = 0.0;
   init[i] = 0.0;
   new[i] = 0.0;
   newfunc[i] = 0.0;
   M_fix[i] = 0.0;
   k1[i] = 0.0;
   k2[i] = 0.0;
   k3[i] = 0.0;
   k4[i] = 0.0;
   u_bias[i] = 0.0;
   for (j = 0; j <= 11; j++)
   {
       fcl[i][j] = 0.0;
   }
 }
 for ( i = 0; i <= 5; i++)
 {
   input[i] = 0.0;
   for ( j = 0; j <= 11; j++)
   {
       frcfunc[j][i] = 0.0;
       output[i][j] = 0.0;
   }
 }
 for ( i = 0; i <= 8; i++ )
 {
   u[i] = 0.0;
   for ( j = 0; j <= 11; j++ )
     K[i][j] = 0.0;
   for ( j = 0; j <= 5; j++ )
     M[i][j] = 0.0;
```

```
        }
        pi = acos(-1.0);
        altitude = 0.0;
        stop_process = 1;
        op_mode = 0;
        volts = 0.0;
        scale = 1.0;
        sensitize = 1.0;
        scalex = 1.0;


/*IMPORTANT****IMPORTANT****IMPORTANT****IMPORTANT****IMPORT
ANT*
 *
 * Set the incremental change for the Runge-Kutta computations.
 * This incremental step is in seconds.  It can be changed at
 * the discretion of the user.
 */

        delta = 20.0;

/*
 * The following are the values for the appropriate output
 * vectors. This output matrix has a dimension of 6 x 12
 * and it is similar to the 'Hsys' of R. Lawrence's thesis, 1992.
 */

        output[0][0]  = 1000.0;
        output[1][2]  = 1000.0;
        output[2][4]  = 1000.0;
        output[3][6]  = 180.0/3.141592654;
        output[4][8]  = 180.0/3.141592654;
        output[5][10] = 180.0/3.141592654;

/*
 * The following is the initialization for this
 * program, SERVER_FROBMC.C, to communicate over the
 * network with OBBC_comm_hub.c
 */

        if ((sockfd = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP)) < 0)
            perror("server: cannot open stream socket");
        printf("client sck# %d\n", sockfd);

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = INADDR_ANY;
```

```c
serv_addr.sin_port = htons(4903);

printf("%d is the port\n",serv_addr.sin_port);
if(bind(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr))<0)
{
  perror("server:cannot bind local address");
  exit(0);
}
printf("binding OK!!!\n");
if(listen(sockfd,5)<0)
    perror("listen error");
printf("listen() OK!!!\n");


/*
 * Check the port socket, the one used for communication between
 * the FROBMC.GSL and this program to ensure the link
 * is properly operating.  If the port socket is not functioning
 * properly, send a message to "Standard Error" (stderr) output
 * and exit the program immediately.
 */

sockport = atoi(argv[1]);

if (argc != 2)
{
  fprintf(stderr,"Usage: %s <port>\n", argv[0]);
  exit(1);
}
if ( net_init_socket_serv( sockport, &sockdsc ) != 0 )
{
  fprintf(stderr,"Can't initialize server socket.\n");
  printf("the server socket is initialized.\n");
  exit(1);
}

/*
 * Read from FROBMC.GSL. the following:
 * The initial controller configuration, ctrl_num, the operation
 * mode (force ref., virtual force ref. etc.), the orbital
 * altitude of the space platform.
 * Then create the initial bias, 'stand-off' current using
 * stdoff_vel(altitude, ctrl_num).
 */
```

```c
    stop_process = net_readsocket( sockdsc, igripbuf );
    if ( stop_process == 0 )
    {
        sscanf(igripbuf,"%d %d %f", &ctrl_num, &op_mode, &altitude);
        printf("%s\n", igripbuf);
        fflush(stdout);
        stdoff_vel(altitude);
    }

/* Initialize target ultimate position of approach axis. This is
 * dictated by the separation distance and the geometry of the
 * space station.
 */
    x_home = -90.374;
    if (ctrl_num == 2)
        {
        x_home = 409.75;
        }

/* Assign correct tracking method coefficients to the
 * position controller input, system, pre-filter, and
 * gain matricies. These are provided by Richard Lawrence (21).
 */

    switch(ctrl_num)
    {
        case 2:  r_bar();
                break;
        case 3:  v_bar();
                break;
    }

/*
 * Read in the target orbit characteristics
 */

    stop_process = net_readsocket( sockdsc, igripbuf );
    if ( stop_process == 0 )
    {
        sscanf(igripbuf,"%f %f %f %f %f
%f",&input[0],&input[1],&input[2],&input[3],&input[4],&input[5]);
        printf("%s\n",igripbuf);
        fflush(stdout);
    }
/* Make the necessary transition from degrees to radians */
```

```c
for (i=3;i<=5;i++)
   input[i] = (input[i]*3.141592654)/180.0;


/* Open a path to the spaceball input port and initialize
 * the spaceball (see spaceball user's guide).  Initialization is
 * dictated by the sb.h source file.  Changes must be coded.
 * In particular, the mode or the argument for init_spaceball.
 */


open_spaceball("/dev/ttyd2",B19200);
init_spaceball(SB_STANDARD);


/* make initial handshake to kirk */

  clilen = sizeof(cli_addr);
  newsockfd = accept(sockfd,(struct sockaddr *) &cli_addr,&clilen);

  if((childpid=fork())<0 )
    perror("server: cannot fork.");

/* if handshake is made, pass the information. */
  else if(childpid == 0)
  {
       printf("Communicating\n");
       close(sockfd);
     volts=0;
     sprintf(outbuf,"%f\n",volts);
       n=strlen(outbuf);
     put_data(newsockfd,outbuf,n);
     /* printf("sent initial h/s.\n"); */

/*
 * Continue to loop until the whole process is "killed".  That is,
 * the simulation is killed by the DENEB/IGRIP simulation running
 * in the foreground or if bad communication between the two
 * programs occur.
 */

  while(( 1 ) && (stop_process ==0 ))
  {
/*
 * Read in spaceball controller information from the RS232 port for
 * the six values of the joints, three translational and
 * three rotational.  All these values can be continuously changed
```

```
 * These values must be converted from byte values (-127 to 128)
 * and the z values must be reversed to match the simulation.
 * NOTE: Spaceball x,y and z values do not correspond to the x,y
 * and z values of the simulation.
 */

 read_spaceball(&sd);

  frdof = 0;

/* Check for an adjustment (scaling) to the commanded input.
 */

scale_button = ((sd.button<<24)>>24);

switch(scale_button)
 {
        case 2:  /* reset scale and sensitivity */
             scale = 1.0;
              sensitize = 1.0;
              scalex=1.0;
              printf("input back to normal.\n");
              break;
        case 4:  /* adjust sensitivity to off-axes */
              sensitize = 0.1;
              printf("off-axes now less sensitive.\n");
              break;
        case 5:  /* scale input by one-half of off-axes  */
              scale = scale*0.5;
              printf("input of off axes halved.\n");
              break;
        case 6:  /* double the input of axis of interest */
              scalex=scalex*2;
              if (scalex>4)
               scalex=4;
              printf("input of axis of interest doubled.\n");
              break;
        case 8:  /* double the input of off axes  */
              scale = scale*2.0;
              printf("input of off axes doubled.\n");
              break;
        default:  /* do nothing */
              break;
 }
```

```c
/* printf("scale button is %f\n",scale_button); */

/*  Now read in any spaceball induced changes to the commanded
 *  input. Ignore axis of interest if flag1 conditions met.
 */

t1=((sd.xtrans<<24)>>24)/-128.0*scale*sensitize;
t2=((sd.ytrans<<24)>>24)/128.0*scale*sensitize;
t3=((sd.ztrans<<24)>>24)/-128.0*scalex;
if (flag1>=1 && t3>0)
  {
   t3=0.0;
  }
t4=(((sd.xrot<<24)>>24)/128.0)*(-3.141592654/180.0)*scale*sensitize;
t5=(((sd.yrot<<24)>>24)/128.0)*(3.141592654/180.0)*scale*sensitize;
t6=(((sd.zrot<<24)>>24)/128.0)*(-3.141592654/180.0)*scale*sensitize;

if (op_mode == 4)
  {
   if (t3 > 0)
     t3=1;
   if (t3 < 0)
     t3=-1;
  }

if (op_mode == 3 && t3 > 0 && force_volts != 20)
  {
  frdof = t3 - (force_volts/10.0);
  if (previous_t3 > t3)
   frdof = 0;
  input[0]=input[0] + frdof;
  previous_t3 = t3;
  }
else
  {
  input[0]=t3+input[0];
  previous_t3 = 0;
  }

  input[1]=t1+input[1];
  input[2]=t2+input[2];
  input[3]=t6+input[3];
  input[4]=t4+input[4];
  input[5]=t5+input[5];
/*printf("%f %f %f %f %f %f\n",t1,t2,t3,t4,t5,t6);
```

```c
    printf("%x %x %x\n",sd.xtrans,sd.ytrans,sd.ztrans);*/

/*
 * Create the new forcing array with the input variables
 * Use the equation:
 *
 *  init[12] = Gcl[12][6] * r[6][1] - Fcl[12][12] * x[12][1]
 *
 * The Gcl[12][6] matrix is from R. Lawrence's
 * thesis work.  In server_frobmc.c,
 * the Gcl[12][6] matrix assumes the responsibility of the
 * forcing function for the equations of motion and,
 * therefore, the matrix is re-assigned to the the matrix
 * frcfunc[12][6].
 *
 */

    for ( i = 0; i <= 11; i++)
    {
      sum = 0.0;
      for ( j = 0; j <= 5; j++)
        sum = sum + frcfunc[i][j]*input[j];
      frcfix[i] = sum;
    }

/*
 * With the equation to obtain the vector array of currents,
 * given as:
 *
 *    u[8] = M[9][6] * input[6][1] - K[9][12] * x[6][1]
 *
 * From page 4-11 and 4-12, of R. Lawrence's thesis, the u[9]
 * array contains eight (8) current variables and one (1)
 * thruster value.  Therefore, one of parts on the right-hand
 * side of the above equation,
 *
 *            M[9][6] * input[6][1]
 *
 * can be multiplied now, while the remaining part
 *
 *            K[9][12] * x[6][1]
 *
 * can only be derived at the completion of one full loop of
 * this routine because that is when the x[6][1]
 * values are obtained.
```

```
*/

    for (i = 0; i <= 8; i++)
    {
      sum = 0.0;
      for (j = 0; j <= 5; j++)
        sum = sum + M[i][j]*input[j];
      M_fix[i] = sum;
    }

/*
 * Loop which obtains all K1[i] values.
 */

    for (i = 0; i <= 11; i++)
    {
      for (j = 0; j <= 11; j++)
          new[j] = fcl[i][j];
      k1[i] = fcn(new, init, 11) + frcfix[i];
    }

/*
 * Loop which obtains all K2[i] values.
 */

    for (i = 0; i <= 11; i++)
    {
        for (j = 0; j <= 11; j++)
        {
          new[j] = fcl[i][j];
          newfunc[j] = init[j] + ((delta/2.0) * k1[j]);
        }
        k2[i] = fcn(newfunc, new, 11) + frcfix[i];
    }

/*
 * Loop which obtains all K3[i] values.
 */

    for (i = 0; i <= 11; i++)
    {
        for (j = 0; j <= 11; j++)
        {
          new[j] = fcl[i][j];
          newfunc[j] = init[j] + ((delta/2.0) * k2[j]);
```

```
        }
        k3[i] = fcn(newfunc, new, 11) + frcfix[i];
    }
```

```
/*
 * Loop which obtains all K4[i] values.
 */
```

```
    for (i = 0; i <= 11; i++)
    {
        for (j = 0; j <= 11; j++)
        {
            new[j] = fcl[i][j];
            newfunc[j] = init[j] + delta * k3[j];
        }
        k4[i] = fcn(newfunc, new, 11) + frcfix[i];
    }
```

```
/*
 * The following loop solves each incremental value for the the
 * sought after x[6][1] array from the equation:
 *
 *   x(DOT)[12][1] = M_fix[12][1] + Fcl[12][12] * x(*)[12][1]
 *
 *   or, as will be shown below:
 *
 *           x(*)[12][1] = init[12][1]
 *
 *   And to derive the desired ouput array, y[6][1]
 *
 *       y[6][1] = output[6][12] * x(*)[12][1],
 */
```

```
    for (i = 0; i <= 11; i++)
        init[i] = init[i] + (delta/6.0)*(k1[i] + 2.0*k2[i] + 2.0*k3[i] + k4[i]);
```

```
/*
 * The following will obtain the 'u' vector array, which
 * consists of eight current values and one thruster
 * value.  The bias current, obtained in the sub-
 * routine, stdoff_vel(altitude, crtl_num), is also added on in
 * all 9 cases. ( 9 cases = 8 currents + 1 thruster )
 *
 *     u[9][1] = M_fix[9][1] - K[9][12] * x(*)[12][1]
 */
```

```
    for (i = 0; i <= 8; i++)
    {
      sum = 0.0;
        for (j = 0; j <= 11; j++)
          sum = sum + K[i][j]*init[j];
      u[i] = M_fix[i] - sum + u_bias[i];
    }
```

```
/*
* The following statements will find the appropriate
* output array to move the OBJECT, in the DENEB/IGRIP
* simulation, accordingly.  The below statements performed
* to the equation
*
*       y[6][1] = output[6][12] * x(*)[12][1]
*/
```

```
    for (i = 0; i <= 11; i++)
    {
      if ( output[0][i] != 0.0 )
          finalout_1 = output[0][i] * init[i];
        if ( output[1][i] != 0.0 )
          finalout_2 = output[1][i] * init[i];
        if ( output[2][i] != 0.0 )
          finalout_3 = output[2][i] * init[i];
        if ( output[3][i] != 0.0 )
          finalout_4 = output[3][i] * init[i];
        if ( output[4][i] != 0.0 )
          finalout_5 = output[4][i] * init[i];
        if ( output[5][i] != 0.0 )
          finalout_6 = output[5][i] * init[i];
    }
```

```
/*
* Send the data back to frobmc.gsl.
*/
```

```
        sprintf(returnbuf,"%f %f %f %f %f %f", finalout_1, finalout_2, finalout_3,
finalout_4, finalout_5, finalout_6);
```

```
/*
* Check to make sure the data made it back to frobmc.gsl
*/
```

```c
    if ( net_writesocket( sockdsc, returnbuf ) != 0 )
    {
      printf("Write failed, server aborting...\n");
      break;
    }
```

/* The igripbuf contains environment information from
 * the frobmc.gsl program. This information must be read
 * and further manipulated before being relayed to the
 * connect.c program on kirk
 */

```c
    if (net_readsocket(sockdsc,igripbuf) != 0)
    {
        printf("Read from IGRIP failed, IGRIP server aborting..\n");
        net_close_socket(sockdsc);
        break;
    }
    sscanf(igripbuf, "%f %d", &force_volts,&collision);
    fflush(stdout);
    printf("force volts = %f\n",force_volts);
```

/* Data must be evaluated to perform neccessary motor control.
 */

```c
  motor_control();
  printf("volts = %f\n",volts);
```

/* send motor control voltage to OBBC_comm_hub.c on kirk
 * a handshake from kirk must be received first.
 */

```c
  get_data(newsockfd,inbuf,MAXLINE);

  sprintf(outbuf,"%f\n",volts);
  n=strlen(outbuf);
  put_data(newsockfd,outbuf,n);
```

/* calculate velocity */

```c
  calculate_velocity(finalout_1,delta);
```

/* reset motor voltage */

```c
  volts = 0.0;
```

```c
      }
    }
    printf("out of else loop,program terminated.\n");
    l++;
    if (l == 2) fclose(out);
    /*net_close_socket( sockdsc );
    exit( 0 );*/
}
```

```
/***************************************************************
 *
 * Subroutine fcn() which operates in a loop and to obtains a
 * result returns to each of the seperate K[i] value loops.
 *
 ***************************************************************/
```

```c
float fcn( NEW1, NEW2, T)
float NEW1[], NEW2[];
int T;
{
  float value;
  int m;
  value = 0;
  for (m = 0; m <= T; m++)
  {
    value = value + (NEW1[m] * NEW2[m]);
  }
  return value;
}
```

```
/***************************************************************
 *
 * The following subroutine will determine the difference in
 * velocities for two space vehicles.  Then it will adjust the
 * objects bias current appropriately so that the two will be
 * able to stay parallel with each other in the same orbital
 * plane.
 *
 ***************************************************************/
```

```c
void stdoff_vel(orbit)
float orbit;
{
  int q, r, tolerance;
```

```
float sum, tol_check[12], sub_input[6], sub_frcfix[12];
float subM_fix[12], fabs(float);
float sub_init[12], sub_newfunc[12], sub_new[12];
float u_biaslast[12], sub_frcfunc[12], sub_delta;
for (q = 0; q <= 5; q++)
  sub_input[q] = 0.0;
for (r = 0; r <= 11; r++)
{
  sub_init[r] = 0.0;
  u_biaslast[r] = 0.0;
  sub_newfunc[r] = 0.0;
  sub_new[r] = 0.0;
  sub_frcfunc[r] = 0.0;
  tol_check[r] = 0.0;
}

/*
 * Set the "tolerance" flag low to enable the R-K routine to
 * sufficiently compute the values necessary for the calculation.
 */

tolerance = 0;
sub_delta = 20.0;
sub_input[0] = (orbit - 400.0) * 1000.0;
if ( abs(sub_input[0]) < 1.0 )
{
  for (q = 0; q <= 8; q++)
    u_bias[q] = 0.0;
  tolerance = 1;
  printf("This is where sub_input < 1.0.\n");
  fflush(stdout);
}

/*
 * Create the new forcing array for the inputted variables
 * Multiply the input array, the input[i] values, which is held
 * in a 6 x 1 matrix, and pre-multiply it with the
 * frcfunc[i][j] values, which are held in a 12 x 6 matrix, to
 * develop a new 12 x 1 matrix.  For the bias current
 * level, this vector array will only have to be computed
 * once because the values of 'input[q]' are static, or fixed.
 */

for (q = 0; q <= 11; q++)
{
```

```
    sum = 0.0;
    for (r = 0; r <= 5; r++)
      sum = sum + frcfunc[q][r]*sub_input[r];
    sub_frcfix[q] = sum;
  }


/*
 * With the equation to obtain the vector array of currents,
 * given as:
 *
 *     u[i] = M[i][j]*input[j][1] - K[i][k]*x[k][1]
 *
 * Find the new M[i][j] matrix now and put the newly obtained
 * values back into M[i][j].  This vector array will only have
 * to be completed once since both the values of M[q][r] and
 * input[r] are static, or fixed.
 */

    for (q = 0; q <= 8; q++)
    {
      sum = 0.0;
      for (r = 0; r <= 5; r++)
        sum = sum + M[q][r]*sub_input[r];
      subM_fix[q] = sum;
    }


/*
 * Loop until all values in the 'u' vector array, the eight
 * bias currents and the thruster value meet the tolerance
 * desired by the programmer.
 */

  while ( tolerance == 0 )
  {

/*
 * Loop which obtains all K1[q] values.
 */

    for (q = 0; q <= 11; q++)
    {
      for (r = 0; r <= 11; r++)
        sub_new[r] = fcl[q][r];
      k1[q] = fcn(sub_new, sub_init, 11) + sub_frcfix[q];
    }
```

```c
/*
 * Loop which obtains all K2[q] values.
 */

    for (q = 0; q <= 11; q++)
    {
      for (r = 0; r <= 11; r++)
      {
        sub_new[r] = fcl[q][r];
        sub_newfunc[r] = sub_init[r] + ((sub_delta/2.0) * k1[r]);
      }
      k2[q] = fcn(sub_newfunc, sub_new, 11) + sub_frcfix[q];
    }

/*
 * Loop which obtains all K3[q] values.
 */

    for (q = 0; q <= 11; q++)
    {
      for (r = 0; r <= 11; r++)
      {
        sub_new[r] = fcl[q][r];
        sub_newfunc[r] = sub_init[r] + ((sub_delta/2.0) * k2[r]);
      }
      k3[q] = fcn(sub_newfunc, sub_new, 11) + sub_frcfix[q];
    }

/*
 * Loop which obtains all K4[i] values.
 */

    for (q = 0; q <= 11; q++)
    {
      for (r = 0; r <= 11; r++)
      {
        sub_new[r] = fcl[q][r];
        sub_newfunc[r] = sub_init[r] + sub_delta * k3[r];
      }
      k4[q] = fcn(sub_newfunc, sub_new, 11) + sub_frcfix[q];
    }

/*
 * Find the incremental values of new_init and then compare them
```

```
 * with the last values of 'new_initlast' to ensure all the
 * values will meet the tolerance specification.
 */

    for (q = 0; q <= 11; q++)
       sub_init[q] = sub_init[q] + (sub_delta/6.0)*(k1[q] + 2.0*k2[q] + 2.0*k3[q] +
k4[q]);
    for (q = 0; q <= 8; q++)
    {
      sum = 0.0;
      for (r = 0; r <= 11; r++)
         sum = sum + K[q][r]*sub_init[r];
      u_bias[q] = -1.0*(subM_fix[q] - sum);
      tol_check[q] = u_bias[q] - u_biaslast[q];
      if (tol_check[q] < 0.0 )
         tol_check[q] = -tol_check[q];
    }

    if ( tol_check[0] < 0.000001 )
    {
      if ( tol_check[1] < 0.000001 )
      {
         if ( tol_check[2] < 0.000001 )
         {
           if ( tol_check[3] < 0.000001 )
           {
              if ( tol_check[4] < 0.000001 )
              {
                if ( tol_check[5] < 0.000001 )
                {
                   if ( tol_check[6] < 0.000001 )
              {
                   if ( tol_check[7] < 0.000001 )
                   {
                     if ( tol_check[8] < 0.000001 )
                     {
                     for (q =0 ; q <= 5; q++)
                {
                        tolerance = 1;
                        }
                     }
                   }
                }
              }
           }
         }
      }
    }
```

```c
                    }
                }
            }
        }
        if ( tolerance == 0 )
        {
          for (q = 0; q <= 11; q++)
            u_biaslast[q] = u_bias[q];
        }
      }
    }
}


/****************************************************************
 *
 * Purpose: The purpose if this subroutine is to re-assign the
 *          values of the matrices to enable the usage of a
 *          controller to operate in the R-bar tracking approach.
 *
 ****************************************************************/

void r_bar()
{

/*
 * The following are the coefficients for the K matrix, the
 * controller gain matrix, for R-Bar tracking. These numbers
 * were generaterd by MATLAB (Lawrence,Bridgman).
 */

  K[0][0]  = -1.9372844e+01;
  K[0][1]  =  1.1382179e+03;
  K[0][2]  = -1.0484363e+01;
  K[0][3]  = -1.2627757e+04;
  K[0][4]  =  4.8058444e-03;
  K[0][5]  =  2.6076772e-01;
  K[0][6]  =  1.9799792e+04;
  K[0][7]  =  6.5975727e+06;
  K[0][8]  = -8.0401281e+03;
  K[0][9]  = -6.3170347e+05;
  K[0][10] = -8.9295885e+03;
  K[0][11] = -2.3974952e+06;

  K[1][0]  = -1.9372920e+01;
  K[1][1]  =  1.1383228e+03;
  K[1][2]  = -1.0484639e+01;
```

```
K[1][3]  = -1.2627702e+04;
K[1][4]  =  4.7943422e-03;
K[1][5]  =  2.6058334e-01;
K[1][6]  = -1.9799731e+04;
K[1][7]  = -6.5975459e+06;
K[1][8]  =  8.0401275e+03;
K[1][9]  =  6.3170358e+05;
K[1][10] =  8.9295654e+03;
K[1][11] =  2.3974857e+06;

K[2][0]  =  2.1350014e+01;
K[2][1]  =  1.3290093e+04;
K[2][2]  = -1.9259708e+01;
K[2][3]  = -7.0571294e+03;
K[2][4]  = -8.3645144e-02;
K[2][5]  = -3.6078841e+00;
K[2][6]  = -1.6592817e+04;
K[2][7]  =  1.6285092e+05;
K[2][8]  =  5.7744003e+02;
K[2][9]  =  4.6901470e+04;
K[2][10] = -5.6121438e+04;
K[2][11] = -2.8910389e+06;

K[3][0]  =  2.1349998e+01;
K[3][1]  =  1.3290139e+04;
K[3][2]  = -1.9259816e+01;
K[3][3]  = -7.0571196e+03;
K[3][4]  = -8.3644301e-02;
K[3][5]  = -3.6078704e+00;
K[3][6]  =  1.6592063e+04;
K[3][7]  = -1.6316309e+05;
K[3][8]  = -5.7743319e+02;
K[3][9]  = -4.6902710e+04;
K[3][10] =  5.6121730e+04;
K[3][11] =  2.8911501e+06;

K[4][0]  = -7.7490634e+00;
K[4][1]  =  4.5518412e+02;
K[4][2]  = -4.1934744e+00;
K[4][3]  = -5.0511578e+03;
K[4][4]  =  1.9192968e-03;
K[4][5]  =  1.0424961e-01;
K[4][6]  =  4.8473696e+04;
K[4][7]  =  1.5742645e+07;
K[4][8]  = -3.2478818e+02;
```

```
K[4][9]  =  5.8681483e+04;
K[4][10] = -1.7758985e+04;
K[4][11] = -5.5497698e+06;


K[5][0]  = -7.7492422e+00;
K[5][1]  =  4.5543218e+02;
K[5][2]  = -4.1941265e+00;
K[5][3]  = -5.0510259e+03;
K[5][4]  =  1.9207779e-03;
K[5][5]  =  1.0429081e-01;
K[5][6]  = -4.8473672e+04;
K[5][7]  = -1.5742634e+07;
K[5][8]  =  3.2478792e+02;
K[5][9]  = -5.8681439e+04;
K[5][10] =  1.7758976e+04;
K[5][11] =  5.5497660e+06;


K[6][0]  =  1.3774246e+00;
K[6][1]  =  8.5741921e+02;
K[6][2]  = -1.2425465e+00;
K[6][3]  = -4.5530252e+02;
K[6][4]  = -5.4403149e-03;
K[6][5]  = -2.3349659e-01;
K[6][6]  =  3.2213237e+03;
K[6][7]  =  8.8134193e+05;
K[6][8]  =  5.7155433e+04;
K[6][9]  =  4.7502364e+06;
K[6][10] = -7.6466147e+02;
K[6][11] = -3.2563935e+05;


K[7][0]  =  1.3774149e+00;
K[7][1]  =  8.5743450e+02;
K[7][2]  = -1.2425841e+00;
K[7][3]  = -4.5529419e+02;
K[7][4]  = -5.3525526e-03;
K[7][5]  = -2.3203597e-01;
K[7][6]  = -3.2213723e+03;
K[7][7]  = -8.8136207e+05;
K[7][8]  = -5.7155432e+04;
K[7][9]  = -4.7502365e+06;
K[7][10] =  7.6468029e+02;
K[7][11] =  3.2564653e+05;


K[8][0]  =  1.2111042e-01;
K[8][1]  =  7.1257672e+01;
```

```
K[8][2]  = -9.8619627e-02;
K[8][3]  = -3.3139152e+01;
K[8][4]  =  3.1574096e+01;
K[8][5]  =  1.5467371e+03;
K[8][6]  = -9.5481011e-03;
K[8][7]  = -3.2811691e+00;
K[8][8]  = -3.7105341e-02;
K[8][9]  = -7.7442906e+00;
K[8][10] =  2.4333163e-03;
K[8][11] =  1.1794325e+00;

/*
 * The following are the coefficients for the M matrix, the pre-
 * filter matrix, for R-bar tracking.
 * These numbers were generated by MATLAB (Lawrence,Bridgman)
 */

M[0][0] = -1.9372844e+01;
M[0][1] = -1.0484557e+01;
M[0][2] =  4.7994661e-03;
M[0][3] =  2.1123605e+04;
M[0][4] = -8.0977716e+03;
M[0][5] = -7.6148644e+03;

M[1][0] = -1.9372920e+01;
M[1][1] = -1.0484445e+01;
M[1][2] =  4.8007206e-03;
M[1][3] = -2.1123544e+04;
M[1][4] =  8.0977710e+03;
M[1][5] =  7.6148414e+03;

M[2][0] =  1.2182571e+01;
M[2][1] = -1.9259736e+01;
M[2][2] = -8.3644978e-02;
M[2][3] = -1.6627255e+04;
M[2][4] =  5.7893960e+02;
M[2][5] = -5.5930583e+04;

M[3][0] =  1.2182555e+01;
M[3][1] = -1.9259788e+01;
M[3][2] = -8.3644467e-02;
M[3][3] =  1.6626501e+04;
M[3][4] = -5.7893276e+02;
M[3][5] =  5.5930875e+04;
```

```
M[4][0] =  -7.7490634e+00;
M[4][1] =  -4.1939361e+00;
M[4][2] =   1.9220176e-03;
M[4][3] =   5.1638699e+04;
M[4][4] =  -3.0019900e+02;
M[4][5] =  -1.4629438e+04;

M[5][0] =  -7.7492422e+00;
M[5][1] =  -4.1936648e+00;
M[5][2] =   1.9180571e-03;
M[5][3] =  -5.1638675e+04;
M[5][4] =   3.0019874e+02;
M[5][5] =   1.4629429e+04;

M[6][0] =   7.8597663e-01;
M[6][1] =  -1.2425733e+00;
M[6][2] =  -5.3858544e-03;
M[6][3] =   3.4046851e+03;
M[6][4] =   5.7647611e+04;
M[6][5] =  -5.8255894e+02;

M[7][0] =   7.8596693e-01;
M[7][1] =  -1.2425573e+00;
M[7][2] =  -5.4070131e-03;
M[7][3] =  -3.4047338e+03;
M[7][4] =  -5.7647610e+04;
M[7][5] =   5.8257776e+02;

M[8][0] =   7.1462518e-02;
M[8][1] =  -9.8619627e-02;
M[8][2] =   3.1622591e+01;
M[8][3] =  -9.5481011e-03;
M[8][4] =  -3.7105341e-02;
M[8][5] =   2.4333163e-03;

/*
 * The following are the coefficients for the closed loop
 * matrix needed for R-bar tracking. Generated by MATLAB.
 */

fcl[0][1]  =  1.0;

fcl[1][0]  = -5.1051910e-06;
fcl[1][1]  = -5.5693194e-03;
fcl[1][2]  =  8.0709427e-06;
```

```
fcl[1][3]  =  5.2205767e-03;
fcl[1][4]  =  3.5051926e-08;
fcl[1][5]  =  1.5119071e-06;
fcl[1][6]  =  1.5802786e-07;
fcl[1][7]  =  6.5408553e-05;
fcl[1][8]  = -1.4335932e-09;
fcl[1][9]  =  2.5981651e-07;
fcl[1][10] = -6.1117026e-08;
fcl[1][11] = -2.3302007e-05;

fcl[2][3]  =  1.0;

fcl[3][0]  = -1.5126204e-06;
fcl[3][1]  = -2.1743624e-03;
fcl[3][2]  = -8.1862214e-07;
fcl[3][3]  = -9.8596385e-04;
fcl[3][4]  =  3.7478776e-10;
fcl[3][5]  =  2.0353354e-08;
fcl[3][6]  =  2.3765060e-09;
fcl[3][7]  =  1.0449128e-06;
fcl[3][8]  = -2.5176599e-11;
fcl[3][9]  =  4.3071479e-09;
fcl[3][10] = -9.0131231e-10;
fcl[3][11] = -3.6980522e-07;

fcl[4][5]  =  1.0;

fcl[5][0]  =  2.2754180e-08;
fcl[5][1]  =  9.0926835e-06;
fcl[5][2]  = -5.9408860e-08;
fcl[5][3]  = -2.4888944e-05;
fcl[5][4]  = -8.3504213e-04;
fcl[5][5]  = -4.0843854e-02;
fcl[5][6]  =  1.9793661e-07;
fcl[5][7]  =  6.4205930e-05;
fcl[5][8]  =  9.8030032e-07;
fcl[5][9]  =  2.0440757e-04;
fcl[5][10] = -4.3297856e-08;
fcl[5][11] = -2.3151249e-05;

fcl[6][7]  =  1.0;

fcl[7][0]  =  1.3678761e-14;
fcl[7][1]  = -8.2696450e-11;
fcl[7][2]  =  1.8553312e-13;
```

```
fcl[7][3]  = -4.8641306e-12;
fcl[7][4]  = -2.0444772e-15;
fcl[7][5]  = -3.0428091e-14;
fcl[7][6]  = -1.0752985e-04;
fcl[7][7]  = -4.3338970e-03;
fcl[7][8]  =  3.2474382e-06;
fcl[7][9]  =  2.7426953e-04;
fcl[7][10] = -2.9926523e-04;
fcl[7][11] = -1.3879672e-02;

fcl[8][9]  =  1.0;

fcl[9][0]  =  1.8386265e-15;
fcl[9][1]  =  1.9163855e-12;
fcl[9][2]  =  1.2998572e-15;
fcl[9][3]  =  1.6394579e-12;
fcl[9][4]  = -5.3121648e-14;
fcl[9][5]  =  3.7007982e-12;
fcl[9][6]  = -2.3849997e-06;
fcl[9][7]  = -4.1956024e-04;
fcl[9][8]  = -2.9269167e-04;
fcl[9][9]  = -2.4093633e-02;
fcl[9][10] = -2.3515522e-06;
fcl[9][11] = -3.2041876e-05;

fcl[10][11] =  1.0;

fcl[11][0]  =  1.3811916e-13;
fcl[11][1]  = -1.2208962e-10;
fcl[11][2]  =  3.7644795e-13;
fcl[11][3]  = -8.6974114e-11;
fcl[11][4]  = -4.1910707e-15;
fcl[11][5]  = -6.8744864e-14;
fcl[11][6]  = -2.4813087e-04;
fcl[11][7]  =  5.1646815e-03;
fcl[11][8]  =  5.6838123e-06;
fcl[11][9]  =  4.7023891e-04;
fcl[11][10] = -8.6626763e-04;
fcl[11][11] = -4.5582467e-02;

/*
 * The forcing function vector of the R-bar tracking procedure.
 */

frcfunc[1][0]  =  5.1051910e-06;
```

```
frcfunc[3][0]  =  1.5126204e-06;
frcfunc[5][0]  = -2.2754180e-08;
frcfunc[7][0]  = -1.3678761e-14;
frcfunc[9][0]  = -1.8386266e-15;
frcfunc[11][0] = -1.3811916e-13;

frcfunc[1][1]  = -8.0709427e-06;
frcfunc[3][1]  =  8.1862214e-07;
frcfunc[5][1]  =  5.9408860e-08;
frcfunc[7][1]  = -1.8553312e-13;
frcfunc[9][1]  = -1.2998570e-15;
frcfunc[11][1] = -3.7644795e-13;

frcfunc[1][2]  = -3.5051926e-08;
frcfunc[3][2]  = -3.7478776e-10;
frcfunc[5][2]  =  8.3504213e-04;
frcfunc[7][2]  =  2.0444795e-15;
frcfunc[9][2]  =  5.3121680e-14;
frcfunc[11][2] =  4.1910758e-15;

frcfunc[1][3]  = -1.5802786e-07;
frcfunc[3][3]  = -2.3765060e-09;
frcfunc[5][3]  = -1.9793661e-07;
frcfunc[7][3]  =  1.0752985e-04;
frcfunc[9][3]  =  2.3849997e-06;
frcfunc[11][3] =  2.4813087e-04;

frcfunc[1][4]  =  1.4335932e-09;
frcfunc[3][4]  =  2.5176599e-11;
frcfunc[5][4]  = -9.8030032e-07;
frcfunc[7][4]  = -3.2474382e-06;
frcfunc[9][4]  =  2.9269167e-04;
frcfunc[11][4] = -5.6838123e-06;

frcfunc[1][5]  =  6.1117026e-08;
frcfunc[3][5]  =  9.0131231e-10;
frcfunc[5][5]  =  4.3297856e-08;
frcfunc[7][5]  =  2.9926523e-04;
frcfunc[9][5]  =  2.3515522e-06;
frcfunc[11][5] =  8.6626763e-04;
}


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

```
* Purpose: The purpose if this subroutine is to re-assign the
*          values of the matrices to enable the graphics
*          simulation to depict V-bar tracking maneuevers.
*
*****************************************************************/

void v_bar()
{

/*
* The following are the coefficients for the K matrix (contains
* the controller gain coefficients), needed by V-Bar tracking.
*/

    K[0][0]  = -3.6612135e+01;
    K[0][1]  = -1.3904802e+04;
    K[0][2]  =  5.1325587e+00;
    K[0][3]  = -1.0949306e+04;
    K[0][4]  =  8.3798158e-03;
    K[0][5]  =  5.9216850e-01;
    K[0][6]  =  2.6305463e+01;
    K[0][7]  =  5.9199334e+03;
    K[0][8]  = -1.8968539e+03;
    K[0][9]  = -1.6171560e+04;
    K[0][10] = -1.8019238e+04;
    K[0][11] = -1.6660839e+06;

    K[1][0]  = -3.6612122e+01;
    K[1][1]  = -1.3904757e+04;
    K[1][2]  =  5.1326473e+00;
    K[1][3]  = -1.0949301e+04;
    K[1][4]  =  8.3862481e-03;
    K[1][5]  =  5.9261237e-01;
    K[1][6]  = -2.6300618e+01;
    K[1][7]  = -5.9197905e+03;
    K[1][8]  =  1.8968117e+03;
    K[1][9]  =  1.6322903e+04;
    K[1][10] =  1.8019244e+04;
    K[1][11] =  1.6660671e+06;

    K[2][0]  =  1.3096827e+01;
    K[2][1]  =  6.1718000e+03;
    K[2][2]  = -4.5054323e+00;
    K[2][3]  =  2.2426954e+03;
    K[2][4]  = -4.2521850e-03;
```

```
K[2][5]  = -3.4124974e-01;
K[2][6]  = -7.2112769e+03;
K[2][7]  = -6.0229524e+05;
K[2][8]  =  9.6384832e+02;
K[2][9]  =  8.4922898e+05;
K[2][10] = -5.2040471e+02;
K[2][11] = -1.3382572e+05;

K[3][0]  =  1.3096872e+01;
K[3][1]  =  6.1718330e+03;
K[3][2]  = -4.5054138e+00;
K[3][3]  =  2.2427073e+03;
K[3][4]  = -4.1720655e-03;
K[3][5]  = -3.3794867e-01;
K[3][6]  =  7.2112753e+03;
K[3][7]  =  6.0229520e+05;
K[3][8]  = -9.6402456e+02;
K[3][9]  = -8.4929048e+05;
K[3][10] =  5.2042519e+02;
K[3][11] =  1.3383270e+05;

K[4][0]  = -2.3620767e+00;
K[4][1]  = -8.9708690e+02;
K[4][2]  =  3.3112876e-01;
K[4][3]  = -7.0640921e+02;
K[4][4]  =  5.2476146e-04;
K[4][5]  =  3.7588427e-02;
K[4][6]  =  1.6979459e+04;
K[4][7]  =  1.4148367e+06;
K[4][8]  =  6.9926058e+01;
K[4][9]  =  4.1536603e+05;
K[4][10] = -1.6119563e+02;
K[4][11] = -5.2096581e+04;

K[5][0]  = -2.3620689e+00;
K[5][1]  = -8.9707819e+02;
K[5][2]  =  3.3114260e-01;
K[5][3]  = -7.0640415e+02;
K[5][4]  =  5.5692008e-04;
K[5][5]  =  3.8849049e-02;
K[5][6]  = -1.6979459e+04;
K[5][7]  = -1.4148367e+06;
K[5][8]  = -6.9928781e+01;
K[5][9]  = -4.1535627e+05;
K[5][10] =  1.6119602e+02;
```

```
K[5][11] =   5.2095498e+04;

K[6][0]  =   5.2385689e+00;
K[6][1]  =   2.4685985e+03;
K[6][2]  =  -1.8022490e+00;
K[6][3]  =   8.9703003e+02;
K[6][4]  =  -2.0229852e-03;
K[6][5]  =  -1.4965554e-01;
K[6][6]  =   2.2402244e+02;
K[6][7]  =   7.8175511e+03;
K[6][8]  =   7.5309034e+03;
K[6][9]  =   7.4136323e+06;
K[6][10] =  -8.6934831e+02;
K[6][11] =  -8.1372833e+05;

K[7][0]  =   5.2389108e+00;
K[7][1]  =   2.4688547e+03;
K[7][2]  =  -1.8020894e+00;
K[7][3]  =   8.9713105e+02;
K[7][4]  =  -1.3467150e-03;
K[7][5]  =  -1.2202382e-01;
K[7][6]  =  -2.2402310e+02;
K[7][7]  =  -7.8175666e+03;
K[7][8]  =  -7.5309739e+03;
K[7][9]  =  -7.4136569e+06;
K[7][10] =   8.6935651e+02;
K[7][11] =   8.1373112e+05;

K[8][0]  =   7.5763769e-02;
K[8][1]  =   2.9838519e+01;
K[8][2]  =  -1.3043747e-02;
K[8][3]  =   2.1240033e+01;
K[8][4]  =   9.9516137e+00;
K[8][5]  =   8.6823629e+02;
K[8][6]  =  -7.9539094e-02;
K[8][7]  =  -2.3193669e+00;
K[8][8]  =  -1.3096549e+00;
K[8][9]  =  -2.7011420e+03;
K[8][10] =   1.7483581e-01;
K[8][11] =   2.9749725e+02;

/*
 * The following coefficients are for the M matrix, the pre-
 * filter matrix.  These values were generated by MATLAB.
 * V-Bar tracking.
```

```
*/

    M[0][0] = -3.6612135e+01;
    M[0][1] =  5.1326009e+00;
    M[0][2] =  8.3737002e-03;
    M[0][3] =  2.5909729e+01;
    M[0][4] = -1.9521226e+03;
    M[0][5] = -1.8304848e+04;

    M[1][0] = -3.6612122e+01;
    M[1][1] =  5.1326052e+00;
    M[1][2] =  8.3923637e-03;
    M[1][3] = -2.5904884e+01;
    M[1][4] =  1.9520804e+03;
    M[1][5] =  1.8304854e+04;

    M[2][0] = -3.6105526e+01;
    M[2][1] = -4.5054050e+00;
    M[2][2] = -4.0171014e-03;
    M[2][3] = -7.1960649e+03;
    M[2][4] =  3.0883771e+03;
    M[2][5] = -7.0489404e+02;

    M[3][0] = -3.6105481e+01;
    M[3][1] = -4.5054410e+00;
    M[3][2] = -4.4071491e-03;
    M[3][3] =  7.1960632e+03;
    M[3][4] = -3.0885533e+03;
    M[3][5] =  7.0491452e+02;

    M[4][0] = -2.3620767e+00;
    M[4][1] =  3.3114037e-01;
    M[4][2] =  6.2504192e-04;
    M[4][3] =  1.6943090e+04;
    M[4][4] =  9.7619402e+02;
    M[4][5] = -2.3989392e+02;

    M[5][0] = -2.3620689e+00;
    M[5][1] =  3.3113099e-01;
    M[5][2] =  4.5663961e-04;
    M[5][3] = -1.6943090e+04;
    M[5][4] = -9.7619675e+02;
    M[5][5] =  2.3989431e+02;

    M[6][0] = -1.4442372e+01;
```

```
M[6][1] =  -1.8020246e+00;
M[6][2] =  -1.5768961e-05;
M[6][2] =   2.2191543e+02;
M[6][3] =   2.5670786e+04;
M[6][4] =  -2.3900313e+03;

M[7][0] =  -1.4442030e+01;
M[7][1] =  -1.8023138e+00;
M[7][2] =  -3.3539312e-03;
M[7][3] =  -2.2191608e+02;
M[7][4] =  -2.5670856e+04;
M[7][5] =   2.3900394e+03;

M[8][0] =   2.6115867e-02;
M[8][1] =  -1.3043747e-02;
M[8][2] =   1.0000109e+01;
M[8][3] =  -7.9539094e-02;
M[8][4] =  -1.3096549e+00;
M[8][5] =   1.7483581e-01;

/*
 * The following are the coefficients for the closed loop matrix
 * generated by MATLAB (Lawrence,Bridgman) for V-Bar tracking.
 */

fcl[0][1]  =  1.0;

fcl[1][0]  =  2.8190912e-06;
fcl[1][1]  = -4.8189089e-04;
fcl[1][2]  =  3.5178012e-07;
fcl[1][3]  =  2.0881292e-03;
fcl[1][4]  =  3.2887964e-10;
fcl[1][5]  =  2.6515657e-08;
fcl[1][6]  =  6.4125679e-11;
fcl[1][7]  =  1.5155439e-09;
fcl[1][8]  =  6.8801932e-09;
fcl[1][9]  =  2.4008197e-06;
fcl[1][10] = -7.9943551e-10;
fcl[1][11] = -2.7268703e-07;

fcl[2][3]  =  1.0;

fcl[3][0]  = -1.5342577e-05;
fcl[3][1]  = -8.0901361e-03;
fcl[3][2]  =  2.1508544e-06;
```

```
fcl[3][3]  = -4.5883849e-03;
fcl[3][4]  =  3.5129702e-09;
fcl[3][5]  =  2.4824550e-07;
fcl[3][6]  =  1.0151231e-09;
fcl[3][7]  =  2.9957644e-08;
fcl[3][8]  = -8.8419494e-09;
fcl[3][9]  =  3.1710672e-05;
fcl[3][10] =  1.2707592e-09;
fcl[3][11] = -3.5152392e-06;

fcl[4][5]  =  1.0;

fcl[5][0]  =  4.8422408e-08;
fcl[5][1]  =  2.2197206e-05;
fcl[5][2]  = -1.3945275e-08;
fcl[5][3]  =  7.3203772e-06;
fcl[5][4]  = -2.6406463e-04;
fcl[5][5]  = -2.2926794e-02;
fcl[5][6]  =  2.1001850e-06;
fcl[5][7]  =  6.1241658e-05;
fcl[5][8]  =  3.4581546e-05;
fcl[5][9]  =  7.1322364e-02;
fcl[5][10] = -4.6166045e-06;
fcl[5][11] = -7.8552673e-03;

fcl[6][7]  =  1.0;

fcl[7][0]  = -8.0046854e-14;
fcl[7][1]  = -3.8390332e-11;
fcl[7][2]  =  4.2680318e-14;
fcl[7][3]  = -3.4225792e-13;
fcl[7][4]  = -1.4483390e-14;
fcl[7][5]  = -1.0586828e-12;
fcl[7][6]  = -2.8722782e-04;
fcl[7][7]  = -2.3993513e-02;
fcl[7][8]  =  4.8978833e-06;
fcl[7][9]  =  9.1323594e-04;
fcl[7][10] = -8.7260906e-07;
fcl[7][11] = -7.1646026e-05;

fcl[8][9]  =  1.0;

fcl[9][0]  =  3.7317838e-14;
fcl[9][1]  =  2.0159832e-12;
fcl[9][2]  = -4.2131677e-14;
```

```
fcl[9][3]  =  1.0032060e-11;
fcl[9][4]  = -4.6428223e-13;
fcl[9][5]  =  3.4128175e-12;
fcl[9][6]  = -2.2684248e-08;
fcl[9][7]  = -2.5126452e-05;
fcl[9][8]  = -9.7419189e-06;
fcl[9][9]  = -2.0259801e-03;
fcl[9][10] = -2.5681076e-05;
fcl[9][11] = -2.1763998e-03;

fcl[10][11] =  1.0;

fcl[11][0]  = -8.8295896e-14;
fcl[11][1]  = -5.1984722e-11;
fcl[11][2]  = -2.1924174e-14;
fcl[11][3]  = -3.4937462e-11;
fcl[11][4]  = -5.5774712e-14;
fcl[11][5]  = -3.4692950e-12;
fcl[11][6]  = -2.1126336e-06;
fcl[11][7]  = -1.2768635e-04;
fcl[11][8]  = -2.4516140e-05;
fcl[11][9]  =  7.6569121e-05;
fcl[11][10] = -2.3999590e-04;
fcl[11][11] = -2.1867746e-02;

/*
 * The forcing function vector for the V-bar tracking procedure.
 */

frcfunc[1][0]  = -2.8190912e-06;
frcfunc[3][0]  =  1.5342577e-05;
frcfunc[5][0]  = -4.8422408e-08;
frcfunc[7][0]  =  8.0046854e-14;
frcfunc[9][0]  = -3.7317838e-14;
frcfunc[11][0] =  8.8295896e-14;

frcfunc[1][1]  = -3.5178012e-07;
frcfunc[3][1]  = -2.1508544e-06;
frcfunc[5][1]  =  1.3945275e-08;
frcfunc[7][1]  = -4.2680318e-14;
frcfunc[9][1]  =  4.2131677e-14;
frcfunc[11][1] =  2.1924175e-14;

frcfunc[1][2]  = -3.2887964e-10;
frcfunc[3][2]  = -3.5129702e-09;
```

```
        frcfunc[5][2]  =  2.6406463e-04;
        frcfunc[7][2]  =  1.4483415e-14;
        frcfunc[9][2]  =  4.6428224e-13;
        frcfunc[11][2] =  5.5774730e-14;

        frcfunc[1][3]  = -6.4125680e-11;
        frcfunc[3][3]  = -1.0151231e-09;
        frcfunc[5][3]  = -2.1001850e-06;
        frcfunc[7][3]  =  2.8722782e-04;
        frcfunc[9][3]  =  2.2684248e-08;
        frcfunc[11][3] =  2.1126336e-06;

        frcfunc[1][4]  = -6.8801932e-09;
        frcfunc[3][4]  =  8.8419494e-09;
        frcfunc[5][4]  = -3.4581546e-05;
        frcfunc[7][4]  = -4.8978833e-06;
        frcfunc[9][4]  =  9.7419189e-06;
        frcfunc[11][4] =  2.4516140e-05;

        frcfunc[1][5]  =  7.9943551e-10;
        frcfunc[3][5]  = -1.2707592e-09;
        frcfunc[5][5]  =  4.6166045e-06;
        frcfunc[7][5]  =  8.7260906e-07;
        frcfunc[9][5]  =  2.5681076e-05;
        frcfunc[11][5] =  2.3999590e-04;

}


/**************************************************************
 * This subroutine performs all remaining analysis for       *
 * motion of the actuator on the manual controller.          *
 **************************************************************/

void motor_control()

{
 /* motor move */
 collide_x = x_home;

 switch(op_mode)
  {
        case 1:  /* simple mode */
                volts=0;
                break;
        case 2:  /* force reflection mode */
```

```
case 3:  /* virtual force reflection mode */
          if (collision == 1)
          {
          flag1++;
          if (flag1 == 1)
             collide_x = finalout_1/1000.0;
          }
          else flag1=0;

          volts = (t3/scalex)*10.0+force_volts;

          if (input[0] > collide_x)
           input[0] = collide_x;
          if (volts >= 0)
           volts = 0.015;
          scale_voltage();
          if (t3<=0)
           volts=0;
          break;
case 4:  /* electronic funneling mode */
          if (force_volts == 0 && t3 >= 0)
          {
          flag1=1;
          input[0]=finalout_1/1000.0;
          }
          else
           flag1=0;
          if (t3>0)
           {
           volts=0.025*force_volts;
           break;
           }
          if (t3<0)
           {
           volts=-0.035;
           break;
           }
          else volts=0;

  }
}
```

```
/******************************************************************
* This subroutine scales the voltages to necessary values for input to the servo amplifier.
******************************************************************/
```

```c
void scale_voltage()

{
  if (volts > 0)
   volts=volts*0.0050+0.025;
  if (volts < 0)
   volts=volts*0.0055-0.035;
}

/*****************************************************************
 * This subroutine calculates closure rate during simulation.
 *
 ****************************************************************/

void calculate_velocity(displaced,timer)

float displaced,timer;

{
float velocity,elapsed;

printf("in the velocity routine.\n");
count++;
if (count == 1)
  moved = finalout_1;
if (count == 3)
 {
  velocity = ((displaced-moved)/(timer*3))/1000.0;
  fprintf(out,"%f\n",velocity);
  printf("the velocity is: %f\n",velocity);
  count=0;
 }
}
```

Appendix C

C.1 Graphics Simulation Software: Flowchart
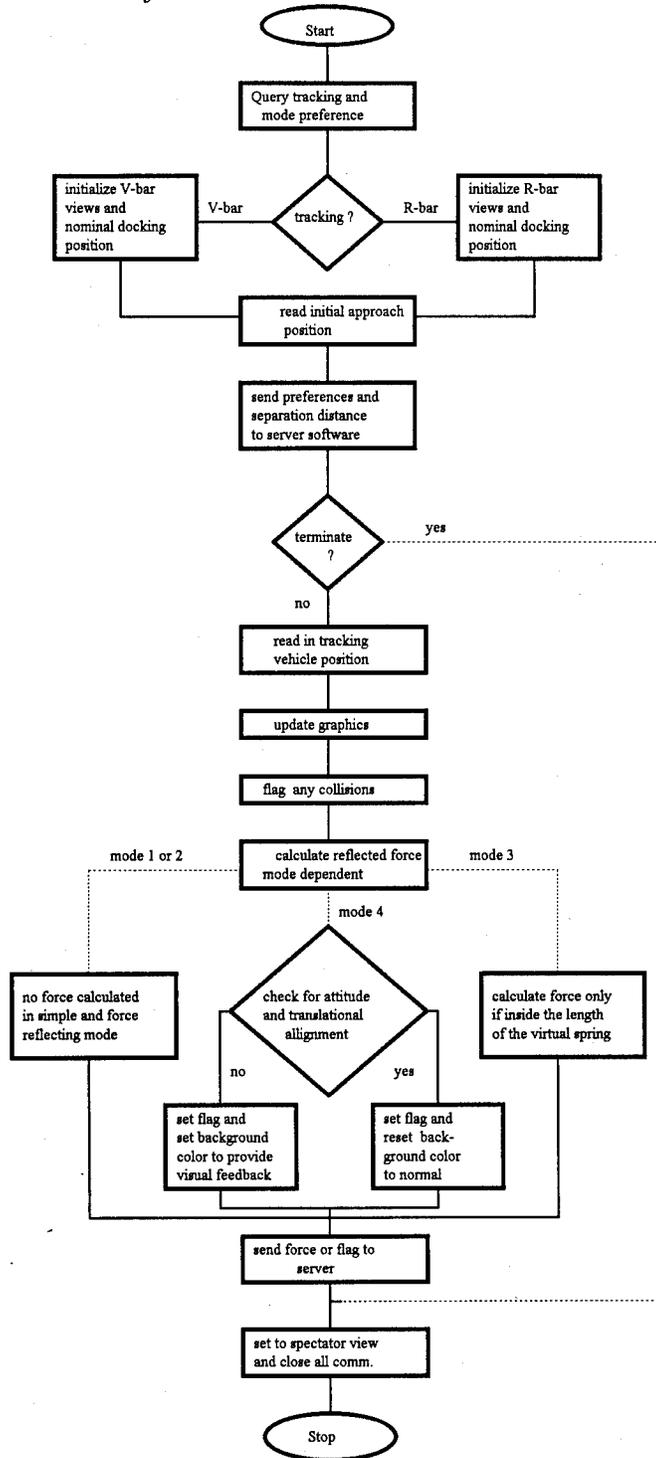
Figure C-1 Frobmc.gsl Flow Chart

```
------------------------------------------------------------------------
-- Paul Woznick, Jul 94
-- Adapted from program asset, T. Bridgman.
--
-- frobmc.gsl must begin with the 'PROGRAM' name followed
-- by the name of the device being programmed in a WORKCELL.
-- The device OBJECT will be assigned to this program.
-- A separate server/number crunching program must be
-- running before the simulation starts.  To enable the 'C'
-- routine, goto the /usr/deneb/igrip.4d/giftware directory and
-- type server_frobmc 2074.  With this, the two
-- can communicate with each other through the address location
-- 2074.  Ensure OBBC_comm_hub.c is all ready running on KIRK.
--
------------------------------------------------------------------------


Program object


------------------------------------------------------------------------
--
-- Declaration of the variables that will be used in this program.
-- 'VAR' variables are ones declared frobmc.gsl while
-- the 'CLI_VAR' variables are used when a command like
-- CLI("    ") is called.
--
------------------------------------------------------------------------


VAR

    pick, track_num, i:  INTEGER
    view_num:            INTEGER
    preference:          INTEGER
    opmode:              INTEGER
    loop:                INTEGER
    coll_checker:        INTEGER
    flagger:             INTEGER
    X, Y:                REAL
    orbit:               REAL
    ext_jt:              REAL
    volts,virt_dist:     REAL
    force_ref_axis_dist: REAL
    xf,xo,yf,yo,zf,zo:   REAL
```

```
    yawf,yawo,rollf:    REAL
    rollo,pitchf,pitcho: REAL
    z_check,y_check:    REAL
    yaw_check:          REAL
    funnel_radius:      REAL
    pitch_check:              REAL
    roll_check:         REAL
    collide_x:          REAL
    jt_in:              ARRAY[6] of REAL
    jt_out:             ARRAY[6] of REAL
    p1, p2:             POSITION

CLI_VAR

    stand_off:          REAL
    temp_sep:           REAL

Begin


    ----------------------------------------------------------------
    --
    -- Below, the user will be promted for the type of control
    -- employed to dock the OBJECT with a space station.
    --
    ----------------------------------------------------------------

    write("Before the simulation begins, choose from the ", cr)
    write("the tracking options listed below.", cr)
    write(cr)
    DELAY 1000
    write("The choices are:", cr)
    write(cr)
    write(" 1. V-BAR tracking.", cr)
    write(" 2. R-BAR tracking.", cr)
    write(cr)
    read_kbd( 'Enter a tracking preference', preference )
    write(cr)
    write("You must also choose the mode of operation of ", cr)
    write("the controller.  The choices are:", cr)
    write(cr)
    write(" 1. Simple 6DOF control.", cr)
    write(" 2. 6 DOF control with force reflection.", cr)
    write(" 3. 6 DOF control with virtual force reflection.", cr)
    write(" 4. 6 DOF control with Electronic funneling.", cr)
    read_kbd( 'Enter operation mode', opmode )
```

```
write(cls)
if (opmode > 4) then
  opmode = 1
endif
```

--
-- Open up a port for communication with server_frobmc.c.
-- For this simulation, the communications port
-- is assigned unit number 3. Therefore, all writing
-- to and reading from server_frobmc.c necessary for this
-- simulation is done through unit #3.
--

```
open client 'hardy:2074' for update as 3
```

--
-- Set up the space station view and allocate appropriate
-- V-bar or R-bar tracking matricies.
--

```
view_station()
```

--
-- Initialize docking position to match camera view at
-- separation of 500m.
--

```
SWITCH preference

CASE 1:   xf=-90374
          yf=512640
          zf=-4406
              collide_x = xf
CASE 2:   xf=409700
          yf=12873
          zf=-5039
              collide_x = xf

ENDSWITCH
```

```
    yawf=0
    pitchf=0
    rollf=0



-----------------------------------------------------------------
--
-- After either V-Bar tracking or R-Bar tracking is picked have
-- the command of the program goto the 'position()' sub-routine
-- to prompt the user to input the six (6) needed values for
-- "automated" flight.  The six (6) values are three (3) trans-
-- lational and three (3) rotational.  Recommended values are:
-- x=400, y= 12, z= -5, xrot=yrot=zrot=0.
--
-----------------------------------------------------------------


rendevous_position()


-----------------------------------------------------------------
--
-- Write to server_frobmc.c the pertinent values user preference
-- and simulation execution.  The values sent to
-- server_frobmc.c are the tracking preference, the operation
-- mode (ie. virtual force reflection) and station altitude.
-- Then send the object orbital characteristics after rendevous.
--
-----------------------------------------------------------------

    write #3, ( track_num,' ', opmode,' ', temp_sep, cr )
    DELAY 10
    write #3, ( jt_out[0],' ', jt_out[1], ' ', jt_out[2], ' ', jt_out[3], ' ', jt_out[4], ' ', jt_out[5],
cr )


-----------------------------------------------------------------
--
-- Turn on collision queue if in a force reflection mode.
--
-----------------------------------------------------------------

 if (opmode > 1) then
  ADD 'freedom','asset' TO QUEUE
  SET COLLISION CHECKS ON
 endif


-----------------------------------------------------------------
```

```
--
-- Initialize zero force f/b if in simple DOF mode.
--
------------------------------------------------------------------

  if (opmode == 1) then
    volts = 0.0
    coll_checker = 0
  endif
------------------------------------------------------------------
  --
  -- This simulation will keep operating until the user presses
  -- both the middle and the right mouse buttons simultaneously.
  --
------------------------------------------------------------------

  while (pick <> 3) do
    pick = MOUSE_BUTTON( X, Y )


------------------------------------------------------------------
  --
  -- Read from the sub-routine the up-dated values so the
  -- object can move accordingly.  The values read in are
  -- the three (3) translational movements, jt_in[0] thru
  -- jt_in[2] and the three (3) rotational movements, jt_in[3]
  -- thru jt_in[5].
  --
------------------------------------------------------------------


    read ( #3, jt_in[0], jt_in[1], jt_in[2], jt_in[3], jt_in[4], jt_in[5] )


------------------------------------------------------------------
  --
  -- Use the MOVE JOINT _ TO 'value' IMMEDIATE command to
  -- move each of the six (6) joints of the object.
  --
------------------------------------------------------------------

    if (coll_checker == 1 OR jt_in[0] > collide_x) then
    jt_in[0] = collide_x
    endif

    MOVE JOINT 1 TO jt_in[0] IMMEDIATE
    MOVE JOINT 2 TO jt_in[1] IMMEDIATE
```

```
        MOVE JOINT 3 TO jt_in[2] IMMEDIATE
        MOVE JOINT 4 TO jt_in[3] IMMEDIATE
        MOVE JOINT 5 TO jt_in[4] IMMEDIATE
        MOVE JOINT 6 TO jt_in[5] IMMEDIATE
        SIM_UPDATE
```

--
-- Determine the voltage necessary to provide the appropriate
-- force reflection to the manual controller. Send
-- this information to server_frobmc.c
--

```
   if (opmode > 1) then
     check_position()
     collide_checker()
     calculate_force2volts()
   endif

-- write(volts,' ',coll_checker, cr)
   write #3, ( volts,' ',coll_checker, cr )
```

--
-- The program is terminated when the value 'pick' = 3.
--

```
   endwhile
```

--
-- Close the port address so this
-- address doesn't 'hang' or go into a 'dead-lock'.
--

```
   close #3
```

--
-- Set the 'MULTI VIEWS' from the two (2) screen, horizontally
-- split window to the full, unsplit picture window. Also
-- set the view to a good spectator view by using the
-- selected view, 'spec_canada.'

```
--
-----------------------------------------------------------

   CLI("MULTI VIEWS 0")
   CLI("SET VIEW TO 'spec_canada'")


-----------------------------------------------------------
--
-- End the main program with the 'END' command following by
-- name of the main program, in this case 'object'.
--
-----------------------------------------------------------


END object


-----------------------------------------------------------
-----------------------------------------------------------
-----------------------------------------------------------

--
-- All the procedures start here.
--
-----------------------------------------------------------
-----------------------------------------------------------


-----------------------------------------------------------

--
-- The following sub-routine is for tracking with a space station
--
-----------------------------------------------------------


Procedure view_station()
Begin

-----------------------------------------------------------
--

-- Set view_num to '1' for viewing space station.
-- and track_num to '2' for R-Bar controller configuration
-- or track_num to 3 for V-bar tracking
--
-----------------------------------------------------------


   view_num = 1

-----------------------------------------------------------
--
```

```
-- Prompt the user for the separation between the space
-- station and the object.  This depends on tracking
-- configuration.
--
------------------------------------------------------------
  if (preference == 2) then
    track_num = 2
    write( cls )
    write("You have selected R-BAR tracking. ", cr)
    write("At the prompt below,", cr)
    write("indicate the separation between the ", cr)
    write("space station and the object (in meters).", cr)
    DELAY 5000
    write("Remember that R-bar tracking dictates an ", cr)
    write("approach to the target along the target's ", cr)
    write("position vector, r.  A recommended ", cr)
    write("safe minimum separation is 100 ", cr)
    write("meters, therefore acceptable values ", cr)
    write("of separation should be greater than ", cr)
    write("100 meters", cr)
    read_kbd( 'Enter the separation, in meters', stand_off )
    temp_sep = 400000 + stand_off
    temp_sep = temp_sep/1000
    DELAY 500
    write( cls )
    ext_jt = 0
  endif

  if (preference == 1) then
    track_num = 3
    write( cls )
    write("You have selected V-BAR tracking. ", cr)
    write("At the prompt below,", cr)
    write("indicate the separation between the ", cr)
    write("space station and the object (in meters).", cr)
    DELAY 5000
    write("Remember that V-bar tracking dictates an ", cr)
    write("approach to the target along the target's ", cr)
    write("orbit (east/west).  A recommended ", cr)
    write("safe minimum separation is 100 ", cr)
    write("meters, therefore acceptable values ", cr)
    write("of separation should be greater than ", cr)
    write("100 meters", cr)
    read_kbd( 'Enter the separation, in meters', stand_off )
    temp_sep = 400
```

```
      DELAY 500
      write( cls )
--     ext_jt = 90
   endif


------------------------------------------------------------
--
-- convert standoff distance to milimeters
--
------------------------------------------------------------

      stand_off = stand_off * 1000
      DELAY 500


------------------------------------------------------------
--
-- Retrieve the space station from the /usr/deneb/Usrlib/DEVICES
-- directory and translate it to the standoff distance specified
-- by the user.
--
------------------------------------------------------------
   if (track_num == 2) then
     CLI("GET DEVICE '/usr/deneb/Usrlib/DEVICES/freedom'")
     DELAY 100
     CLI("ACTIVATE 'freedom'")
     CLI("TRANSLATE DEVICE freedom TO stand_off, 0, 0")
     DELAY 500
     CLI("DEACTIVATE 'freedom'")
     DELAY 500
   endif
   if (track_num == 3) then
     CLI("GET DEVICE '/usr/deneb/Usrlib/DEVICES/freedom'")
     DELAY 100
     CLI("ACTIVATE 'freedom'")
     CLI("TRANSLATE DEVICE freedom TO 0, stand_off, 0")
     DELAY 500
     CLI("ROTATE DEVICE freedom TO 0, 0, 90")
     DELAY 100
     CLI("DEACTIVATE 'freedom'")
     DELAY 500
   endif


------------------------------------------------------------
--
-- Move the object to position (0, 0, 0, 0, 0, 0) so that
```

-- the simulation can begin from the original reference point.

--

```
    MOVE JOINT 1 TO jt_in[0] IMMEDIATE
    MOVE JOINT 2 TO jt_in[1] IMMEDIATE
    MOVE JOINT 3 TO jt_in[2] IMMEDIATE
    MOVE JOINT 4 TO jt_in[3] IMMEDIATE
    MOVE JOINT 5 TO jt_in[4] IMMEDIATE
    MOVE JOINT 6 TO jt_in[5] IMMEDIATE
    DELAY 500
```

--

-- Set split screen views to the space station viewing.

--

```
    CLI("MULTI VIEWS 6")
    CLI("ACTIVATE VIEW 1")
    CLI("SET VIEW TO 'ssf_canada'")
     if (track_num == 3) then
    CLI("SET VIEW TO 'vbar_object'")
     endif
    CLI("SCALE WORLD TO 20000")
    CLI("ACTIVATE VIEW 2")
    CLI("ATTACH EYE TO TAG p1")
    CLI("SCALE WORLD TO 20000")
End
```

--

-- This subroutine prompts for the initial desired position
-- command.

--

```
Procedure rendevous_position()

Begin
    write("The following prompts below will ask ", cr)
    write("for a series of attributes about ", cr)
    write("the rendevous orbit.  Requested are ", cr)
    write("three position and three attitude values.", cr)
    read_kbd( 'Enter an attainable x (m)', jt_out[0] )
```

```
    read_kbd( 'Enter an attainable y (m)', jt_out[1] )
    read_kbd( 'Enter an attainable z (m)', jt_out[2] )
    read_kbd( 'Enter an attainable x_rot (deg)', jt_out[3] )
    read_kbd( 'Enter an attainable y_rot (deg)', jt_out[4] )
    read_kbd( 'Enter an attainable z_rot (deg)', jt_out[5] )
    write( cls )

End


---------------------------------------------------------------
--
-- This procedure makes joint values from the simulation
-- available for manipulation by program procedures.
--
---------------------------------------------------------------
Procedure check_position()

Begin

    UNPOS('asset',xo,yo,zo,yawo,pitcho,rollo)

End


---------------------------------------------------------------
--
-- This procedure initiates the creation of the forces
-- of the object/environment interaction
--
---------------------------------------------------------------
Procedure calculate_force2volts()

Begin

 force_ref_axis_dist = xf-xo

 SWITCH opmode

  CASE 2:  volts=0
        if(force_ref_axis_dist <= 0)then
        volts = force_ref_axis_dist/1000.0
         if (volts < -10) then
         volts = -10
          endif
        return
        endif
```

```
CASE 3:  virtual_force2volts()
        return

CASE 4:  funnel_force2volts()
        return

ENDSWITCH

End
```

```
-----------------------------------------------------------
--
--  This procedure creates the virtual spring environment
--  and the associated forces involved with it's interaction.
--
-----------------------------------------------------------
Procedure virtual_force2volts()

Begin

  virt_dist = (100000.0 - force_ref_axis_dist)/1000.0
  if (virt_dist <= 0) then
    return
  endif

  volts = (virt_dist^2)/1000

  if (volts > 10.0) then
   volts = 10.0
  endif
 volts=-volts

End
```

```
-----------------------------------------------------------
--
--  This procedure checks for a collision between the space
--  station and the tracking vehicle.
--
-----------------------------------------------------------
Procedure collide_checker()

Begin
```

```
 coll_checker = DEV_COLLISIONS('freedom','asset')

 if (coll_checker == 1) then
  check_position()
  collide_x = xo
 endif

End


----------------------------------------------------------------
--
-- This procedure creates the hypercone environment and
-- the associated forces involved with it's interaction.
--
----------------------------------------------------------------
Procedure funnel_force2volts()

Begin

 volts = 1.0
 if (coll_checker == 1)then
  volts=0
 endif
 flagger = 0

 if ((force_ref_axis_dist/1000) <= 25 AND force_ref_axis_dist > 0) then

  funnel_radius = 0.25*force_ref_axis_dist
  z_check = abs(zf-zo)-funnel_radius
  y_check = abs(yf-yo)-funnel_radius
  yaw_check = abs(yawo) - funnel_radius/1000.0
  pitch_check = abs(pitcho) - funnel_radius/1000.0
  roll_check = abs(rollo) - funnel_radius/1000.0

  if (y_check>=0 OR z_check>=0) then
   volts = 0.0
  endif

  if (yaw_check >= 0) then
    flagger=flagger+1
    volts=0
  endif
  if (pitch_check >= 0) then
    flagger=flagger+2
    volts=0
```

```
  endif
  if (roll_check >= 0) then
    flagger=flagger+4
    volts=0
  endif


SWITCH flagger

 CASE 1:
      CLI("SET BACK COLOR TO 0.01,0.99,0.99")

 CASE 2:
      CLI("SET BACK COLOR TO 0.99,0.99,0.01")

 CASE 3:
      CLI("SET BACK COLOR TO 0.01,0.99,0.99")
      CLI("SET BACK COLOR TO 0.99,0.99,0.01")

 CASE 4:
      CLI("SET BACK COLOR TO 0.99,0.01,0.99")

 CASE 5:
      CLI("SET BACK COLOR TO 0.01,0.99,0.99")
      CLI("SET BACK COLOR TO 0.99,0.01,0.99")

 CASE 6:
      CLI("SET BACK COLOR TO 0.99,0.99,0.01")
      CLI("SET BACK COLOR TO 0.99,0.01,0.99")

 CASE 7:
      CLI("SET BACK COLOR TO 0.01,0.99,0.99")
      CLI("SET BACK COLOR TO 0.99,0.99,0.01")
      CLI("SET BACK COLOR TO 0.99,0.01,0.99")

 DEFAULT:
      CLI("SET BACK COLOR TO 0.001,0.001,0.001")

ENDSWITCH

 endif

End
```

Appendix D

D.1 Supporting Software Code Listing: Communication Hub

```c
/*********************************************************************
**
*
* OBBC_comm_hub is the ethernet communication hub for the OBBC demonstration
*  It is modified by Paul Woznick from connect.c, written by Tom Deeter.
* Code was extracted from Dave Doaks (AFIT) and Matthew Gertz (CMU) programs
*
*********************************************************************
*/
#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>
#include <signal.h>
#include <chimera.h>
#include <enet.h>
#include <string.h>

#define WFC_JOINTS      1
#define WFC_QUIT   0
#define MAXLINE 4096

int sockfd;

/**********************************************
*
* Function clean_up will gracefully disconnect
* machines.
**********************************************/
void clean_up()
{
  close(sockfd);
  shutdown();
```

```
  exit(0);
}

/*************************************************
 *
 * Function put_data will send the data to the remote
 * non chim machine.
 *
 *************************************************/

put_data(sd,buf,nbytes)
register int sd;
register char *buf;
register int nbytes;
{
    int nleft,nwritten;

    nleft = nbytes;

    while(nleft > 0)
    {
      nwritten = write(sd,buf,nleft);
      if(nwritten <= 0)
        return nwritten;
      nleft -= nwritten;
      buf   += nwritten;
    }
    return nbytes-nleft;
}

/********************************************************
 *
 * Function get_data will read the data from the remote
 * non chim machine.
 *
 ********************************************************/

int get_data(sd,buf,maxlen)
register int sd;
register char *buf;
register int maxlen;
        {
        int n,rc;
        char c;
```

```
        for(n=0;n<maxlen;n++){
                if((rc=read(sd,&c,1))==1)
                        {
                        *buf++ = c;
                        if(c == '\n')
                        break;
                        }
                else if( rc==0)
                        {
                        if(n==1) return  0; /* no data*/
                        else break;
                        }
                else
                        return -1;
        }
        *buf = 0;
        return n;
        }


main(argc,argv)
int argc;
char *argv[];
{
   struct sockaddr_in cli_addr;
   struct servent *sp;
   struct hostent *hp;
   float volts, pi;
   char *pname = argv[0];
   int  n, i, count;
   char inbuffer[MAXLINE],outbuff[MAXLINE];
   ENET  *chimenet;
   char  chimbuffer[4096];
   int   loop,type,size;


if (argc<2)
{
  printf("Usage: %s remote_host_name\n",argv[0]);
  exit(0);
}

while (1)
{
```

```
/*********************************************************
 *
 * Connect to chimera using enetCreate.  Do this first then
 * start fr.c on chimera.
 *
 ********************************************************/

printf("Creating server tsok on CHIMERA\n");
printf("Start CHIMERA NEW communication program\n");
chimenet = enetCreate("tsok",0);
printf("Connected...\n");

/*********************************************************
 *
 * Connect to remote non chim machine using standard unix (TCP Protocal)
 * calls.
 *
 ********************************************************/

signal(SIGINT,clean_up);

bzero(( char *)&cli_addr,sizeof(cli_addr));
if ((hp = gethostbyname(argv[1])) == NULL)
{
  perror("unknown host");
  exit(0);
}
bcopy(hp->h_addr,(char*)&cli_addr.sin_addr,hp->h_length);

cli_addr.sin_family = hp->h_addrtype;
cli_addr.sin_port = htons(4903);

printf("%d  is the family\n",cli_addr.sin_family);
printf("%d  is the port \n",cli_addr.sin_port);
if((sockfd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP)) < 0)
{
  perror("client: cannot open stream socket");
  exit(1);
}
if(connect(sockfd,(struct sockaddr *) &cli_addr,sizeof(cli_addr)) < 0)
{
  printf("error #: %d\n", errno);
  perror("client: cannot connect to server");
  exit(1);
}
```

```c
  printf("Connected to %s \n",argv[1]);

do
{
  size = sizeof(chimbuffer);
  type = 1;

/*  get data from hardy  */

  /*  printf("getting data from hardy\n"); */
    get_data(sockfd,outbuff,MAXLINE);
  /*  printf("got data\n"); */
    n = strlen(outbuff);
  /*  printf(" data to chim is %s\n",outbuff); */

  /* send data to chimera */

    enetSend(chimenet,WFC_JOINTS,n,outbuff,0);

  /* recieve handshake */

  /*  printf("receiving hand shake from chim\n"); */
    enetReceive(chimenet, &type, &size, chimbuffer, 0);

  /*  printf("got handshake: %s\n",chimbuffer);*/

/* create handshake for hardy */

    sprintf(inbuffer,"\n");
      n = strlen(inbuffer);

  /* send handshake to hardy*/

  /*  printf("sending handshake to hardy\n"); */
    put_data(sockfd,inbuffer,n);
  }
  while (type != WFC_QUIT);
  printf("Destroying server...\n");
  enetDestroy(chimenet);
  }
}
```

D.2 Supporting Software Code Listing: R/T Microproccessor Communication Link

```
/*****************************************************************
 *
 * This program modified for OBBC-microprocessor communication by Paul Woznick

 * Original code provided by Tom Deeter, 1994

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

#include <chimera.h>
#include <sbs.h>
#include <enet.h>

#define WFC_ACK    2

void poke_volts();

main()
{

  char connection[80],inbuff[4096],outbuffer[4096];
  int no_conn,type,size;
  int loop,n;
  float volts,hex_volts;
  unsigned short out_volt;
  ENET *enet;

printf("starting prog\n");
  no_conn=0;
  strcpy(connection,"tsok@kirk");

  /* Attach to an enet port if appropriate        */
printf("trying to attach\n");
    enet = enetAttach(connection, ENET_RETRIES(2));
printf("attached\n");
    poke_volts(0x07ff);
printf("attached2\n");

while(1)
 {

/*  get data from kirk, connect  */

  enetReceive(enet, &type, &size, inbuff,0);
/* kprintf("Received %s\n",inbuff); */
```

```c
/* assign data to a buffer */

  sscanf(inbuff,"%f",&volts);
  /* kprintf("%f\n",volts); */

/* convert float volts to hexadecimal and send to D/A   */
  hex_volts = 4095.0-((4096.5/5.0)*(volts+2.5));
  out_volt=(unsigned short) hex_volts;
 /* kprintf("just converted volts to hex\n"); */
  poke_volts(out_volt);

/*  create handshake buffer */

  sprintf(outbuffer,"test");
  n=strlen(outbuffer);

/*  send handshake to kirk   */

/* kprintf("send to kirk %s\n",outbuffer); */
  enetSend(enet, WFC_ACK, n, outbuffer,0);
  }
}

void poke_volts(volt)
unsigned short volt;
{
  unsigned short *D_A_addr;
  short i;
  D_A_addr = ( unsigned short *) 0xFAFFF900;
  /*printf("volt= %x\n",volt);*/

  /* send all channels same value */
  *(D_A_addr + 0) = (unsigned short)volt;
}
```

D.3 Supporting Software Code Listing: Spaceball Communication

```c
/***********************************************************
 * The following subroutine opens a port to the spaceball.  All Spaceball
 * subroutines are based on Bob Filers thesis work, converted to C and
 * modified to communicated with a Silicon Graphics Indigo Iris workstation.
 ***********************************************************/

void open_spaceball(ttyport, speed)
```

```c
String ttyport;
int speed;
  {
    struct termio tty;
    int status;

    if ((SB_fd = open(ttyport, O_RDWR | O_NDELAY)) == -1)
        {
                perror("Cannot open Spaceball");
                exit(1);
        }

    /*  Get the current port parameters */

    status = ioctl(SB_fd, TCGETA, &tty);
    if (status == -1) {
                fprintf (stderr,"Error in save ioctl call\n\n");
    }

    /*
     *  Set the port up for:
     *          Hang up on last close
     *          eight bits
     *          local line
     *          enable receiver
     *          enable signals
     *          canonical input
     *          user specified baud rate
     */
/*
 * These flags set up the port for "raw" input.  This allows us to
 * grab whatever input is present in the read queue regardless of
 * whether the device is done sending a full packet or not.
 */

                tty.c_cflag = HUPCL | CS8 | CLOCAL | CREAD | speed;
                tty.c_iflag = IGNBRK;
                tty.c_lflag = 0;
                tty.c_oflag = 0;
                tty.c_cc[VMIN] = 0;
                tty.c_cc[VTIME] = 0;

                status = ioctl (SB_fd, TCSETAF, &tty);
                if (status == -1) {
                        fprintf (stderr,"Error in set ioctl call\n\n");
```

```
                }
        }

/*********************************************************
 *  The following subroutine initializes the spaceball to          *
 *  one of three modes: voltage, force or standard.  See           *
 *  user's guide.  NOTE: The mode will be determined by the                *
 *  programmer (when the call is made).  The simulation            *
 *  user will not be able to dictate a particular mode             *
 *********************************************************/

void init_spaceball(Byte mode)
  {
    if ((mode == SB_VOLTAGE) || (mode == SB_FORCE) || (mode ==
SB_STANDARD))
                SB_mode = mode;
        else {
                SB_mode = SB_FORCE;
                perror ("Incorrect mode request in init call\n");
        }
  }


/*************************************************************
 *  The following subroutine reads in the raw spaceball data             *
 *  from the RS232 port and stores it in buf.  This is called          *
 *  only when the spaceball is in the force and voltage modes.           *
 *************************************************************/

int read_raw_spaceball(Byte *buf,int len)

  {
    int count = 0;

        /* flush the input queue to get the most recent data */

        ioctl(SB_fd, TCFLSH, (struct termio *)0);

        while(count != len)
                count = read(SB_fd, buf, len);
        buf[len - 1] = NULL;  /* strip CR/LF */
        return(strlen((const char* )buf));
  }


/*******************************************************
 *  The following subroutine is needed for the standard            *
```

```
 *    operation mode.                                                    *
 ************************************************************/

void write_spaceball(Byte *buf,int len)

  {
    if (write(SB_fd, buf, len) != len)
        {
                perror("Short write to Spaceball");
                exit(1);
        }
  }


/**************************************************************
 *  The following subroutine reads in input from the spaceball        *
 *  only if it is operating in the standard mode. It will store       *
 *  the incoming data in buf.                                   *
 ************************************************************/

int read_standard(Byte *buf,int len)

  {
    Byte enq[2];
        int count = 0;
        int times = 0;

        enq[0] = 5;     /* ASCII ENQ character */
        write_spaceball(enq,1);

#ifdef ONYX
        sginap ((long)1);
#endif

        while ((count != len) && (times < MAX_TIMES)) {
                count = read(SB_fd, buf, len);
                times++;
        }

        if (times == 500) return (-1);
                else return(strlen((const char *)buf));
  }

/**************************************************************
 *  This subroutine formats incoming data.               *
 ************************************************************/
```

```c
void parse_spaceball(SpaceballData *sd, Byte *buf)

{
        short button;
        sscanf((const char *)buf, "%hd %hd %hd %hd %hd %hd %hd",
                        &(sd->xtrans), &(sd->ytrans), &(sd->ztrans),
                        &(sd->xrot), &(sd->yrot), &(sd->zrot),
                        &button);
        switch (SB_mode)
        {
        case SB_FORCE:
                sd->button = ABS(button);
                break;
        case SB_VOLTAGE:
                sd->button = 8 - (short)log10((double)button);
                break;
        }
}

/************************************************************
 * This subroutine formats incoming data when in standard
 * mode.                                                   *
 ***********************************************************/

void parse_standard (SpaceballData *sd, Byte *buf)
{
        if (buf[3] != 37) {
                perror ("Incorrect standard read\n");
        } else {
                sd->xtrans = buf[4];
                sd->ytrans = buf[5];
                sd->ztrans = buf[6];
                sd->xrot   = buf[7];
                sd->yrot   = buf[8];
                sd->zrot   = buf[9];
                sd->button = ABS(buf[10]);
        }
}

/************************************************************
 * This subroutine is the one called by the main program to    *
 * read in spaceball data.  The data is stored in sd.          *
 ***********************************************************/
```

```c
void read_spaceball(SpaceballData *sd)

{
  Byte buf[SB_DATA_SIZE + 1];
        short oldbutton = 0;
        switch(SB_mode)
        {
                case SB_FORCE:
                        read_raw_spaceball(buf, SB_FORCE_SIZE);
                        parse_spaceball(sd, buf);
                        break;
                case SB_VOLTAGE:
                        read_raw_spaceball(buf, SB_VOLTAGE_SIZE);
                        parse_spaceball(sd, buf);
                        break;
                case SB_STANDARD:
                        if ((read_standard (buf, SB_STANDARD_SIZE)) == -1) {
                                sd->xtrans = sd->ytrans = sd->ztrans = 0;
                                sd->xrot = sd->yrot = sd->zrot = 0;
                                sd->button = 0;
                        } else
                                parse_standard (sd, buf);
                        break;
        }

        /* De-bounce buttons */

        if ((sd->button > 0) && (sd->button != oldbutton))
                oldbutton = sd->button;
        else
        {
                oldbutton = 0;
                sd->button = 0;
        }
}
```

Appendix E

E.1 Operating Instructions: Preparation

1. Ensure that the CHIMERA operated microprocessor is powered on.

2. Login to Kirk and start CHIMERA by typing **chim** <enter> at a command prompt.

3. To insure that the D/A board is cleared of any voltages type **ex clean** <enter> at the chim control prompt. **CAUTION**: This step must be completed successfully before proceeding to step 4.

4. Connect both plugs of the hand controller into standard AC outlets.

5. Toggle on switch for the CIS DIM 6. Watch for all indicator lights on the keypad to flash. This indicates proper power up of the Dim 6.

6. Select TRA button on the Dim 6 keypad.

7. Ensure that the serial cable from the Dim 6 is connected to serial port 1 on the Silicon Graphics workstation.

8. Sign on the Silicon Graphics Hardy Workstation (pwoznick home directory) and create 4 shells with the shell tool. Make the 4 shells as short as possible and about 4 inches wide. Drag shells to each corner of the display. The two windows on top will be your local shells and the two on the bottom will be your remote shells.

9. In the upper left window at the prompt type:
    cd deneb <enter>

10. In the upper right window at the prompt type:
    cd /usr/deneb/igrip.4d/giftware/woznick <enter>

11. In the lower left window at the prompt type:
    rlogin kirk <enter>
    cd /Thesis/frobmc/comm <enter>

12. Repeat step 9 for the lower right window.

13. Return the cursor to the lower left window and at the prompt type:
    chim <enter>
    A chim control prompt should appear.

14. In the upper left window type:

/usr/deneb/igrip.4d/igrip <enter>

This will create a ghost shell. Drag and click at the center of the monitor. Make this window as large as possible without covering the other 4 window. This is the IGRIP workcell.

E.2 Operating Instructions: Execution

1. With the mouse, select the **SYS** button from the tool bar displayed along the top of the IGRIP workcell. Now select the **FILE** button from the vertical tool bar located at the extreme right of the workcell. Next select the **APPEND** button from the lower portion of the vertical tool bar. A pop up window will open. Use the mouse to select **Usrlib**. The message: File - Usrlib - selected
                        Config file appended

2. Now return the horizontal tool bar along the top of the IGRIP window and select **LAYOUT**. Now select Retrieve Workcell from the vertical tool bar. Pick from a resulting pop up window the library:
/home/pwoznick/usr/deneb/Usrlib/WORKCELLS
Now pick **object**. Wait about 90 seconds for the program to be retrieved.

3. A message window will appear. Use the mouse to drag and click the window to the top center of the IGRIP window, just below the horizontal tool bar.

4. Now select **MOTION** from the horizontal tool bar. Then select **Simulate** from the upper portion of the vertical tool bar.

5. Move the cursor to the upper right 'local' window. At the prompt type:
                        server_frobmc 2074 <enter>
The message: object server started
and: Listening!?

6. Move the cursor to the IGRIP window and select **Run** from the upper middle of the vertical tool bar. A pop up window will appear. Select **done** from the upper right of this window. The graphics simulation will begin. A series of pop up windows will prompt the user. Follow these instructions.

7. The first prompt will ask for a tracking method preference. Using the SGI keyboard enter the number 1 or 2. 1 is V-bar and 2 is R-bar. Default is R-bar.

8. The second prompt will ask the user for the OBBC controller feedback preference. Using the SGI keyboard, enter numbers 1, 2, 3, or 4. 1 is baseline. 2 is force reflection. 3 is virtual force reflection and 4 is electronic funneling. Default is baseline mode.

9. The third prompt will be for a separation or standoff distance. Type **500** <enter>. This can be anything but to ensure safe separation, this is the recommended separation. Graphics will update after <enter>.

10. The final set of prompts will ask for initial desired position commands. This can be used to get the tracking vehicle in the direct proximity of the target vehicle. Suggested values for good demonstration and ensuring all programmed camera views. Enter the values depending on mode and tracking method from the table below:

Input	Modes 1, 2 & 3		Mode 4	
	R-bar	V-bar	R-bar	V-bar
x	400	-115	400	-112
y	10	510	12.873	512.64
z	-5	-5	-5.039	-4.406
xrot	0	0	0	0
yrot	0	0	0	0
zrot	0	0	0	0

Table E-1. Initial Input

11. Upon entering the zrot value the display will switch to a split screen display. The cockpit view will be on the lower screen and a top view of the space station will be on the upper screen.

12. Move the cursor to the lower right 'remote' window. At the prompt type:
> **OBBC hardy** <enter>

A message prompting the initiation of the comm link on the microprocessor.

13. Move the cursor to the lower left 'remote' window and at the chim control prompt type: **ex OBBC1** <enter>

A series of messages should appear in both remote windows indicating the necessary connections have been made. The simulation should begin in IGRIP at this time.

14. Use the hand controller to complete the approach. Control is completely natural. Do to the controller knob what you would like done to the tracking vehicle.

15. It may be helpful to use the TRA or ROT and/or DOM buttons on the keypad of the controller. These functions are described in chapter 3. Additionally, the keypad has 8 function keys. The functions programmed with each of these keys are described in Chapter 4.

16. Remember to keep the cursor in the IGRIP window to keep the simulation running. Also, keep an eye on the upper right local window for any communication messages and other information. IGRIP is an interactive graphics environment. Therefore tool buttons

located on the bottom horizontal tool bar as well as the buttons on the current vertical tool bar are available to enhance the simulation demonstration. Particularly useful buttons are the **magnify** and the **jnt vals** buttons. Magnify can be used to inspect the mating of the spacecraft and joint values function will keep you updated on the tracking vehicles position.

17. To end the simulation, ensure the mouse cursor is in the main IGRIP window and then simultaneously press the right and middle mouse buttons. The simulation will end with a spectator view of the tracking and target vehicle. Disconnect both controller electrical plugs.

Appendix F

F.1 CIS Dimension 6 Communication Parameters The Dim 6 has dip switch selectable communication parameters, baud rate and software protocol. The baud rate can be from 300 to 19200 bps and is set at 19200 bps for this thesis. The software protocol can be set to ASCII voltages, ASCII force, ASCII standard, or special protocol. For this thesis the protocol is ASCII standard. Using this protocol, an imparted force on the spherical knob is converted to a hexadecimal value between -127 and 128. Table F-1 shows the dip switch setings of the spaceball compatable with the OBBC prototype software.

Dip Switch	1	2	6	7	8
State	on	off	on	on	on

Table F-1. Spaceball Dipswitch Settings

Vita

Captain Paul Woznicki was born on 31 March 1967, in Long Island, New York. He graduated from West Milford Township High School in West Milford, New Jersey and attended the U.S. Air Force Academy, graduating with a Bachelor of Science in Astronautical Engineering in May 1989. Upon graduation, he received his regular commission in the USAF and served his first tour of duty at Falcon AFB, Colorado. He began as a planner analyst in the 1st Space Operations Squadron (1 SOPS) where he performed routine commanding and state of health analysis of the Defense Support Program and Defense Meteorogical Satellite Program satellites. He was then handpicked to help form the 1 SOPS' first satellite engineering group and went on to become the squadron's lead engineer for the Defense Support Program satellite constellation. He entered the School of Engineering, Air Force Institute of Technology, in June of 1993. His next assignment is as a Foreign Communication Satellite Analyst at the National Air Intelligence Center, Wright-Patterson AFB, Ohio.

Permanent Address:

1515 Lynlee Dr.

Bellbrook, OH 45305-1134

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Telepresent Spacecraft Docking With Object-Based Bilateral Control (OBBC)	

6. AUTHOR(S)

Paul Woznick, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Air Force Institute of Technology, WPAFB, OH 45433-6583	AFIT/GA/ENY/94D-9

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution unlimited	

13. ABSTRACT (Maximum 200 words)

The concept of object-based control is extended to the field of teleoperation, specifically to accomplish the task of spacecraft docking via a bilateral manual controller. An object-based controller with bilateral feedback controls the motion of the grasped object, not the trajectory of the manipulator. For this reason it can be designed with feedback that is intricately linked with the task kinematics. The benefits derived from anthropomorphicity and force feedback are possible without kinematically/geometrically similar master-slave systems, complex calibration and joint mapping schemes, or expensive, high degree-of-freedom force reflection. Object-based control is ideal for low-level telerobotic interfaces.

A hand controller and a spacecraft docking simulation are designed and constructed to demonstrate object-based bilateral control. The dominant task objective in spacecraft docking is the approach to a target vehicle along a single axis of motion. Several methods of bilateral feedback linked with this dominant objective are proposed in addition to simple force reflection. One method involves virtual forces and another utilizes velocity reflection. Each method, practical only with object-based control, enhance the man-machine interface by providing a heuristic method of control.

14. SUBJECT TERMS			15. NUMBER OF PAGES
Robotics, telerobotics, object-based control, bilateral control, spacecraft docking			122
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to *stay within the lines* to meet *optical scanning requirements*.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.