THE DESIGN AND IMPLEMENTATION OF A THREE DEGREE OF FREEDOM FORCE OUTPUT JOYSTICK

by

Massimo Andrea Russo

B.S. in Mechanical Engineering
Massachusetts Institute of Technology (1988)

Submitted to the Department of
Mechanical Engineering
In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
in Mechanical Engineering

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1990

Signature of Author [illegible]
Department of Mechanical Engineering
May 11, 1990

Certified by [illegible]
Woodie C. Flowers
Thesis Supervisor

Accepted by [illegible]
Ain A. Sonin
Chairman, Department Committee on Graduate Studies

THE DESIGN AND IMPLEMENTATION OF A THREE DEGREE-OF-FREEDOM FORCE OUTPUT JOYSTICK

by

Massimo Andrea Russo

Submitted to the Department of Mechanical Engineering
on May 11, 1990 In Partial Fulfillment of the
Requirements For the Degree of Master of Science in
Mechanical Engineering

ABSTRACT

The advent of high resolution, physical model based computer graphics has left a gap in the design of input/output technology appropriate for interacting with such complex virtual environments. Since virtual worlds consist of physical models, it is appropriate to display the inherent force information necessary for the simulation to the user.

This thesis describes the design and implementation of a new type of force reflective joystick. It has three degrees-of-freedom that are actuated by both motors and brakes on each axis. A novel kinematic design allows all three axis to be uncoupled. Two of the degrees-of-freedom are actuated through an offset gimbal, and the third through a sleeved cable transmission. The workspace volume is one cubic foot, and the closed loop bandwidth is 60 Hz for the gimbal axes and 35 Hz for the third axis. The innovative use of brakes and motors allows hard objects to be simulated without the stability and related safety issues involved with high torque motors alone.

The joystick performance is measured by its ability to simulate certain basic physical elements. A hybrid motor/brake control strategy based on power considerations is proposed. Diffe.ent friction compensation and force control strategies are discussed and implemented.

Thesis Supervisor: Dr. Woodie C. Flowers
Title: Professor of Mechanical Engineering

THE DESIGN AND IMPLEMENTATION OF A THREE DEGREE-OF-FREEDOM FORCE OUTPUT JOYSTICK

by

Massimo Andrea Russo

Submitted to the Department of Mechanical Engineering
on May 11, 1990 In Partial Fulfillment of the
Requirements For the Degree of Master of Science in
Mechanical Engineering

ABSTRACT

The advent of high resolution, physical model based computer graphics has left a gap in the design of input/output technology appropriate for interacting with such complex virtual environments. Since virtual worlds consist of physical models, it is appropriate to display the inherent force information necessary for the simulation to the user.

This thesis describes the design and implementation of a new type of force reflective joystick. It has three degrees-of-freedom that are actuated by both motors and brakes on each axis. A novel kinematic design allows all three axis to be uncoupled. Two of the degrees-of-freedom are actuated through an offset gimbal, and the third through a sleeved cable transmission. The workspace volume is one cubic foot, and the closed loop bandwidth is 60 Hz for the gimbal axes and 35 Hz for the third axis. The innovative use of brakes and motors allows hard objects to be simulated without the stability and related safety issues involved with high torque motors alone.

The joystick performance is measured by its ability to simulate certain basic physical elements. A hybrid motor/brake control strategy based on power considerations is proposed. Different friction compensation and force control strategies are discussed and implemented.

Thesis Supervisor: Dr. Woodie C. Flowers
Title: Professor of Mechanical Engineering

Acknowledgements

There are many people that contributed to my experiences as a graduate student, but none taught me more than my advisor. Woodie, you always gave me the freedom to explore. Your enormous insight, intuition, and pure curiosity in the way things work are contagious, and make you a natural teacher. I can only hope that some of the way you look at the world has rubbed off on me. Thanks for all the good meetings and friendly support.

It all started with Megan. Thanks, I owe it all to you in the end. Scuba diving with you, Margaret, who would have thought that we would work together like this! I want to thank you for the great code we have been showing off since your are back at MIT. Marg, you are the backbone of all this stuff, and I wish you the best.

Yet, this project would never have flown without the continuing support of the Media Lab. Thank you, DZ for the room to do my thing, and the moral and financial trust that it would come out great in the end. I have enjoyed my time in the Snakepit, and I will miss all you guys.

Al, you were a great partner and are a good friend. It's a miracle that we managed to live and work and take classes together without killing each other! Sorry about all the sign errors, who knows where I would be without you catching them. Thanks, and all the luck !

Dov, your experience and help through the years allowed me to avoid many pitfalls - and maybe one day I'll forgive you for not having hired me as a UROP ! Stay in touch, and thanks for all the advice. Will, the Prototype lab was a great resource, and I thank you for letting us live there for six months !

Apple Computer, the Media Lab and the Concourse Program have supported me financially throughout my stay here, and I thank them for that. Concourse means much more to me than a few lines of thanks could ever express !

Bob, you are one tough cookie, but you taught me more than many a Professor. Thanks for all the help in the shop, and the advice on life in general !! Good luck in your quest of keeping students from hurting themselves on the machines.

Farla, we made it together, the first hurdle of many we will face. Life is best with you in it, and I cannot wait to meet our next adventure !

Finally, I want to thank my parents for all their love and support. I think that a parent's job is sometimes to give without expecting thanks, and a child's job is to expect without thanking. Well, for all those times, THANKS, and let's go sailing !

One last word: I'll be back !!

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

The tasks faced by humans gain daily in complexity and hazard, as the frontiers of space or deep sea exploration, of medical technologies, and of information systems are pushed forward. It is clear that many of these tasks must be either simulated for training, done from a distance, or more effectively presented to the human cognitive system. Teleoperation, Supervisory Control, and Virtual Environments, all defined below, offer extensions of the human's limited capabilities.

Visual and aural information displays are well developed, but input devices that are more advanced than the keyboard, position or rate joystick, and mouse, are not yet available or rely still only on the sense of vision. Interfaces that communicate through the sense of touch promise to have a wide field of applications: the investigation of human perception, the visualization of scientific data, the exploration of dangerous or remote environments, teleconferences and parallel multi-user design systems, and as handicapped aids. There are many instances where one would gain a clearer understanding of the task if one could feel what is seen. This thesis presents the design and implementation of a three degree of freedom touch interface.

1.2 Concept Introductions

Before proceeding with a more detailed discussion this thesis' objectives, some crucial concepts are defined. A teleoperator is a machine that extends a person's sensing and manipulating capability to a remote location, and includes sensors and devices to apply forces and manipulate objects in the distant environment. The term supervisory control implies that one or more human operators are continually programming and receiving information from a computer that interconnects through a teleoperator to the task environment. Various degrees of independence are possible in a supervisory control scenario, ranging from a direct master/slave teleoperator link to complete manipulator autonomy. A Virtual Environment is experienced by a person when sensory information produced only by a computer compels a feeling of presence in the computer generated surroundings.

Force, kinesthetic, tactile feedback or output are used interchangeably and describe forces generated by the interface device that are felt by the user. The term force feedback is also used in the chapter on control to specify a closed loop force control, the distinction will be clear from the context.

One of the main points of contention in the teleoperator and simulation fields are how telepresence is measured or whether, and if so, when, it contributes to performance. It is not always self-evident that the more telepresence, the better. Which characteristics of the teleoperator or virtual environment system contribute to presence, which are irrelevant, and which degrade it are important issues to explore [Sheridan]. To what extent is it necessary for the human operator to adapt to the teleoperation or virtual environment situation in order to experience a strong sense of presence and/or to function effectively, and how does force feedback aid in this ? These are some questions that future research needs to address, and an appropriate interface device to explore these issues is presented below.

1.3 The Role of Touch Feedback

The advent of high resolution, physical model based computer graphics has left a gap in the design of input/output technology appropriate for interacting with complex virtual world models. To develop communication through the most information rich sensory channel, visual and aural display technologies have improved dramatically. But, advances in stereographic displays and more proprioceptive input devices that contain higher degree of freedom position information, such as full hand or body gesture recognition, still rely only on the visual and aural senses. The user is not able to feel, grasp, touch, smell, or poke the objects that she interacts with and operates in a world of relative paralysis.

The human perception system can be modelled as shown in the block diagram below. Vision provides a wide-band information input to the human, but it is through the tactile sense and motor effects that the environment itself is changed interactively. When perception is undisturbed, the human sense of presence is achieved. To achieve a remote or virtual presence, the kinesthetic interface has to be implemented effectively. Problems arise when the habitual world model is disturbed by spatial inconsistencies, temporal asynchronies such as time delay, altered dynamics such as microgravity, and sensory deprivation. A person would not be able to discriminate between actual presence, telepresence, and virtual existence with sufficiently good technology that is clear of such disturbances. The conditions for sensory presence can be summarized as: the sensor must be localized in the virtual world, i.e. its position and orientation are obvious, the sensor must be causally independent of the objects in the virtual world, and the sensor must be capable of accessing the virtual world at will. These are the general requirements that should be met by the ultimate human-computer interaction device.

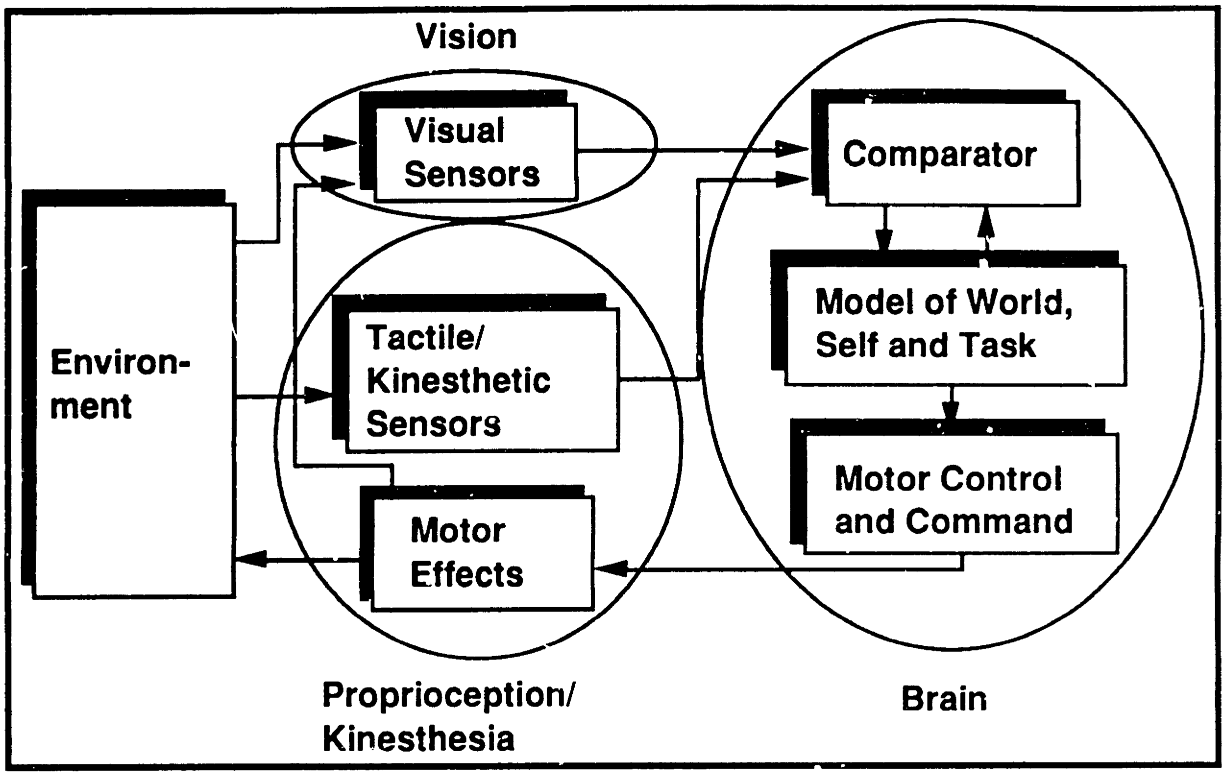

Vision
Visual Sensors
Comparator
Model of World, Self and Task
Environ-ment
Tactile/ Kinesthetic Sensors
Motor Control and Command
Motor Effects
Proprioception/ Kinesthesia
Brain

Figure 1.1: Human perception system

The importance of tactile and proprioceptive information in understanding three dimensional stereoscopic images has been stressed in work done as early as 1971 [Noll]. People perceive depth due to the geometry of their forward facing, separated eyes. Yet, all the senses are needed, in particular the sense of touch, to easily understand their subjective position within a three dimensional environment. For example, it is very difficult to determine whether a cursor has come into contact with an object in a stereoscopic graphic environment if the only assistance in performing such a task is the graphic display and the user's depth-perceptive abilities. If the device used to move in this environment locks once the cursor and object meet, the user is given the impression of actually having hit or touched the object. This added sense augments vision and allows for a more effective and information-filled communication. Various such devices have been under consideration. Yet, these devices are still limited in their ability to simulate an actual object because the only tactile information transmitted to the user is whether an object has been met, or because the device frequency response is too low for convincing simulations. How soft or hard an object is, how heavy, or in particular the high frequency content characteristics of the object, such as its surface texture, are not adequately communicated.

1.4 Previous Tactile Output Devices

Since virtual world computer languages are based on physical model descriptions, it is appropriate to output the inherent force information necessary for the simulation to the user. The technology of such kinesthetic force feedback and tactile displays is in its infancy. Most designs stem from the field of teleoperation, where master/slave manipulators have incorporated force feedback since the 1950's. For example, a simple cartesian coordinate drive with each axis driven by a leadscrew was developed in 1971 [Noll]. This device proved to be large, difficult to backdrive because of the leadscrew pitch, and not ergonomic since the drive axes would interfere with the user arm motion. A more effective design, adapted from a 6 d.o.f. master/slave manipulator pair, is the University of North Carolina GROPE arm [Brooks, Ming]. Again, though, the GROPE dynamics are limited in bandwidth (about 7Hz), address much larger, full arm scale movements, and are kinematically complex. A similar force reflecting device for master-slave manipulation, developed at Stanford [Salisbury, Becjzy], has been used to study the effect of force feedback in peg-in-hole insertion tasks. [Adelstein] devised a high resolution virtual environment system for the study of human arm tremor, and the joystick design in this thesis draws heavily on his experiences. At the University of Utah, a full seven degree of freedom arm and a hand master with force reflection controls a slave arm with an MIT/UTAH hand as an endeffector [Jacobsen]. An interesting different approach is the "touchy-feely" cable suspension design that provides six degree of freedom force and torque feedback in a very limited workspace. These devices tend to be expensive, complicated, dangerous, and usually too large for practical use as a general computer/user interface. Before one commits large resources to the construction of such machines, the appropriate design constraints and problem definitions for simulated environment interactions must be addressed. The problem can be summarized as:

How can the symbiosis between the sense of presence in the virtual environment be maximized without compromising the interaction task, under the constraints of the mechanical device limitations?

Research in tactile feedback interfaces will yield insights to the optimal human sensory feedback mix for a wide spectrum of control and interaction problems. [Smith] developed the initial concepts for the force output joystick presented in this thesis and performed studies of modelling simple physical elements on a one degree of freedom force feedback device. [Cherubino] explored the effect of force feedback on a task learning curve for a shuffleboard game.

A flexible research tool to explore the variety of possible force interactions has been constructed. It is designed to maximize the qualities for a convincing force output and yet is also a robust and easily reproducible product prototype.

1.5 Thesis Scope and Organization

The following thesis presents a three degree of freedom force output joystick that was developed as a joint project between the MIT Media Lab and the Mechanical Engineering Department. The author worked in conjunction with [Smith] and [Tadros] in the design of the presented system. [Tadros] cooperated with this author in the development of various control strategies, and implemented sensor and signal conditioning hardware. He also coded the low level control strategies presented below on a Macintosh II, characterized the mechanical hardware dynamic parameters, and benchmarked the achievable servo-loop rates.

The detail mechanical design process, implementation and control of a three degree of freedom force output joystick are the main topics of this thesis. The kinematic design allows all three axes to be uncoupled, so that the system inertia matrix is diagonal. Each axis of the device is controlled by both a motor and a magnetic particle brake. This combination of motors and brakes allows some objects with high resistive torque requirements to be simulated without the stability and related safety issues involved with high torque, energy storing motors alone. Each axis incorporates direct position, velocity, and applied force sensors. Different control strategies are discussed and implemented, with an emphasis on how virtual environment force information is transmitted to the joystick control processor.

Chapter two introduces the design issues, sets system specifications, and presents the detail mechanical hardware solutions. The following chapter discusses the kinematic and dynamic characteristics of the joystick linkage geometry and develops a friction model for a sleeved cable transmission. The strategies to control the joystick output forces and a stability analysis are then given in chapter four, which also includes a detailed analysis of a hybrid motor/brake actuator control and evaluates the joystick system as to its ability to simulate certain test objects. Chapter five briefly discusses two approaches to modelling dynamic objects efficiently on the joystick control processor. Chapter six presents a summary of applications currently in development and suggests avenues for future research.

Chapter 2

Design Issues

2.1 Overview of Design Specification

Force output has traditionally been employed at the Master end of Master/Slave teleoperated systems [Goertz]. These devices have concentrated on large, global movements, motivated by the scale of the teleoperated robot motion. Before the advent of high speed digital computers that can operate real-time servoloops, a direct analog slave sensor feedback signal controlled the master joint motors. The geometric configuration and physical dimensions of the controlled manipulator and of the master were the same. In many cases, however, geometric equality is not necessary and can even hinder a particular successful task completion. For example, in any manipulation that is significantly larger or smaller in scale, or that requires more accurate and precise motion than the human arm or wrist is capable of, a dissimilar master/slave geometry is appropriate [Hunter]. Recently, in the flexible, unspecified tasks that are possible with virtual environments, the goal is to design the force output device with transparent dynamics and a control system that can simulate the widest frequency band of forces possible.

The range of motion in a computer interaction is limited to a small surface or volume. As justification, one need only think of the popular "mouse" interface device movement. The design of a force output device for virtual environments should maximize the device fidelity under the general constraints of size, cost, limited computation speed, and safety. Table 2.1 presents a quick summary of the design constraints and requirements addressed in the force reflecting joystick.

Table 1: Design Categories:

- Engineering Development:
  - Design Simplicity:
    - Appearance
    - Limited Size
    - Robust
    - Versatile
    - Modular
    - Expandable
    - Safe
  - Difficulty of implementation
  - Technological base
  - Cost
- Controllability:
  - Interface Transparency:
    - Can be counterbalanced (Electronically or Mechanically)
    - Backdriveable
    - High DOF
    - Uncoupled axes
  - Low degree of computation:
    - Full state measurement
    - Integrated into system w/o control/graphic display interference
    - Limited control logic
    - High position resolution
    - Uncoupled axes
  - Closed or open loop control:
    - Full state measurement
  - Stability:
    - Stiff
    - High natural frequency
- Human-Device Interaction:
  - Secondary function control
  - Non-fatiguing
  - Safe
  - Human arm limitations never exceeded
  - Interface transparency
  - Variable force feedback ratio

2.1.1 Engineering Development

To maintain design simplicity and limit size without compromising device versatility, the designed joystick incorporates linear but no rotational force output as shown in figure 2.1. Many manual tasks in which force information is of utmost importance are completed with the use of tools: a surgeon has an assortment of scalpels, a carpenter chisels, a machinist a variety of instruments. The metaphor employed as a design guideline, which allows for a tractable problem within the design requirements, is a tool endpoint force simulator. The joystick endpoint can be upgraded to include a small three degree-of-freedom torque mechanism, so a full six degree-of-freedom force and torque output could be realized.

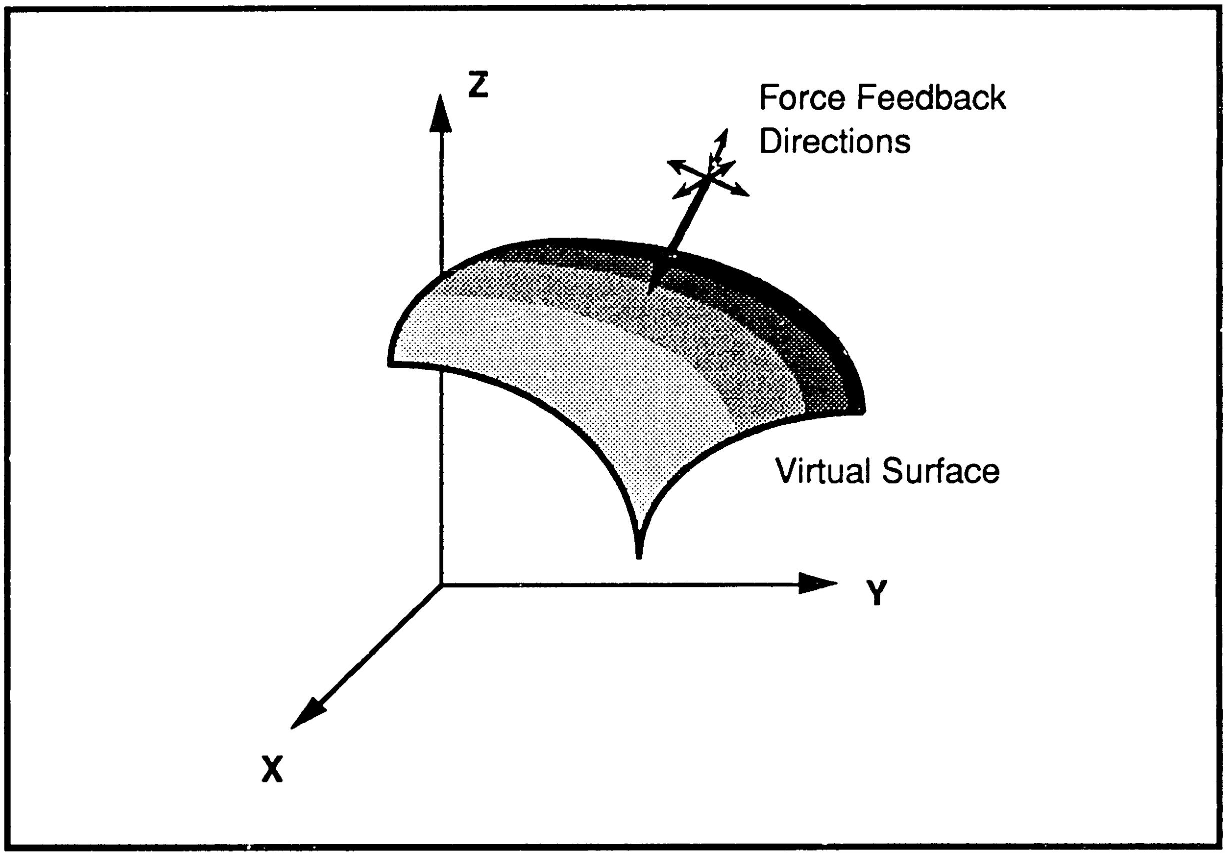


Figure 2.1: Feedback degrees of freedom

Any point contact can be modulated according to the constraint characteristics imposed by the simulation. The joystick endpoint is a single probe with which to explore the virtual computer space. How effective such an approach is in transmitting tactile information is a matter of ongoing research [Minsky, Steele]. Full, high resolution tactile displays have a large number of independent degrees of freedom, determined by the size and resolution of the display. Each tactile "pixel" must be independently controlled, so devices of this kind are difficult to implement. The state of the art are Braille machines for the blind, and it is an open question if the device resolution is sufficient for effective, more general representation tactile displays.

A fine scalpel is not grasped in the palm of the hand, but delicately between one's fingers, so the force magnitude is much lower than in full arm manipulation. However, because of the low inertia of the hand and the frequency sensitivity of skin tactile sensors, the bandwidth requirements are also more stringent than in full arm motion force feedback devices. The restriction to finer forces and a smaller workspace, which is appropriate for the limited travel of the human hand and wrist, also allows the joystick to be constrained in size since the axes brakes and motors can have lower torque outputs. The smaller motors also more easily meet safety constraints and the limitation to three degrees of freedom (d.o.f.) decreases the implementation difficulty and cost. The three d.o.f. are chosen so that the axes are perpendicular and all the actuators are attached to ground, thus eliminating cross coupling between the endpoint principal directions and between the actuator torques. More on this point is presented below in the Dynamics section. Rotary, no torque ripple, low inertia motors and proportional magnetic particle brakes actuate all three axes. The advantage of a hybrid motor/brake actuator is that, for the same power input, much larger dissipative torques can be generated by the brakes than by the motors alone. A full discussion of the brake characteristics is given in Chapter 4. Most of the joystick state sensors are readily available from existing technology, only a three axis linear load cell of sufficiently small size and mass was specially designed and constructed.

2.1.2 Controllability

The detailed mechanical design of the force output joystick is dominated by the control requirements of the simulation. To both impose the arbitrary simulation dynamics onto the joystick endpoint and minimize user fatigue when it acts as a passive position input device, the actual characteristic or plant dynamics should not interfere with the simulation. This interface transparency is from the perspective of the user interacting with the joystick.

So, one input to the system is the force exerted by the user on the endpoint. The simulation, however, controls the desired dynamics through the input signals to the motors and brakes. The control strategy needs to maintain the simulation dynamics under varying user force inputs, and under the changing coupling characteristics between the user's arm and the joystick plant dynamics. The simulation essentially modulates the brake and motor signals so as to match the endpoint impedance, the effective mass, stiffness, and damping characteristics, to that of the virtual dynamics.

The mechanical design maximizes the inherent device transparency and so minimizes the control effort needed to backdrive and control the joystick. Backdriveability is determined by two independent sources of resistive forces: an acceleration dependent component due to inertial effects, and a velocity dependent component due to frictional causes. Both types of resistive forces are minimized in the present design, though inertia is especially restricted given the bandwidth requirements. Because of closed loop force control stability issues discussed below, open loop backdriveability is obtained wherever possible. A direct drive between the brake/motor actuators and the joystick endpoint is necessary to minimize stiction, viscous friction and effective actuator inertia, which are all amplified by a transmission ratio (viscous friction as the ratio, inertia as the ratio squared). Direct drive in turn requires large motors for a significant torque. If the actuators are carried by the transmission linkage, then the difficulty is how to support their mass and also uncouple all reaction torques. All the motors and brakes are therefore fixed to ground so that their weight and reaction torques do not have to be compensated for. Since the brakes have a much higher torque output for a given size and power input, high torque requirements of the simulation will be met by the brakes, inherently passive, non energy storing and therefore safe torquers. Both backdriveability and output force are therefore maximized within the size and safety constraints.

Appropriate strategies for hybrid, independent proportional brake and motor control have not yet been researched. This novel combination promises to increase stability and safety bounds in actively controlled mechanisms without decreasing the closed loop device stiffness. As discussed in more detail below, the control of the brakes necessitates a direct measurement of the contact force between the endpoint and user. Both for this reason and to actively backdrive the shaft transmission, a loadcell 3-axis force sensor is implemented in the device.

The joystick endpoint position, velocity, and force are necessary to determine the characteristics of the interface dynamics. The control loop must sample the joystick states at a minimum of twice the joystick natural frequency, which translates to a minimum value of 120 Hz for a measured closed loop bandwidth of 60 Hz. In practice, the actual loop sampling frequency is, however, restricted by the computation cost of the simulated environment. Bandwidth limitations are discussed in chapter 4 below. All the joystick control operations are performed on a single processor in real time. The degree of computation lag in the control loop must therefore be minimized [Tadros]. To achieve this and to eliminate quantization and noise errors at low velocities introduced by digital differentiation, both motor/brake shaft position and velocity are measured directly. The endpoint position and velocity are found through a matrix position and Jacobian transformation respectively. The joystick linkage is such that all actuator induced endpoint velocities are perpendicular, and any actuator exerts a force in one of the endpoint principle directions. The system inertia matrix is therefore only diagonal, though time variant. A more detailed analysis of the joystick kinematics and Jacobian conditioning are given below in the Kinematics chapter, with the inertia matrix characteristics developed in the Dynamics section.

2.1.3 Human - Device Interaction

Ergonomic considerations have to be incorporated into the joystick design. The user's ability to activate a given function or best sense a simulated object, or function control, is one of many criteria in the device design. Force/torque decoupling and endpoint grasp configuration define how the user experiences the touch interface. Rather than a full palm or hand grasp, the joystick endpoint should emphasize a fine tool-like grab posture.

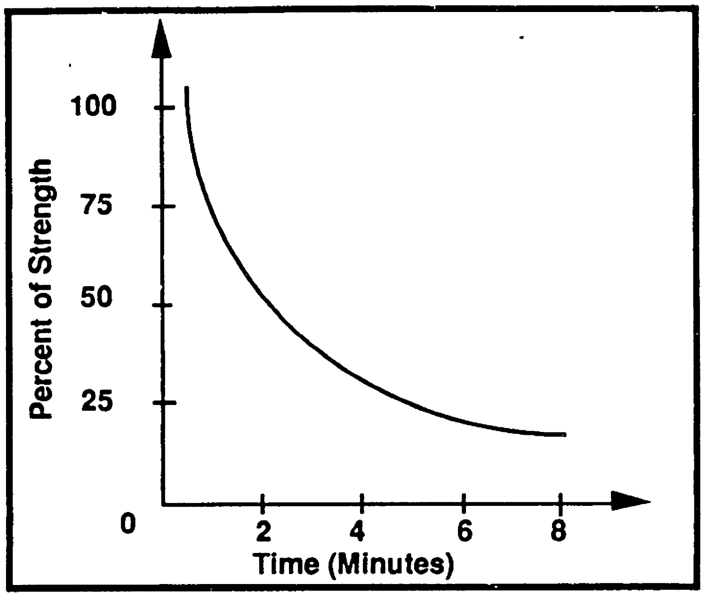


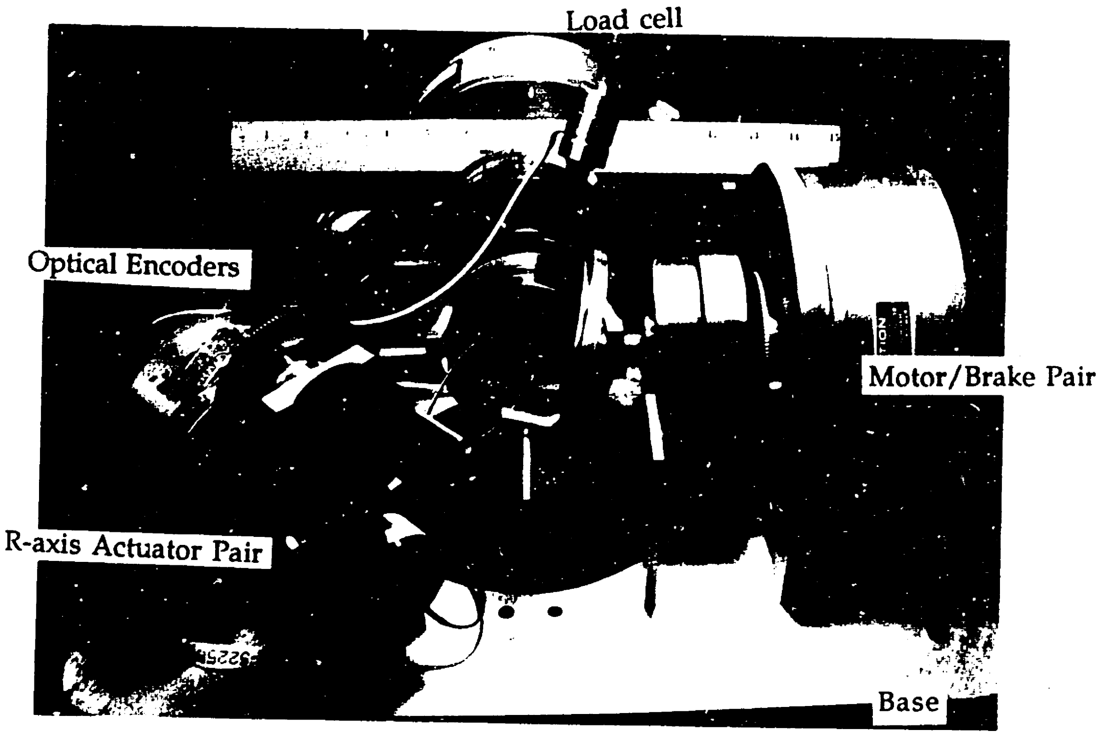
Figure 2.2: Endurance as a function of time and exerted force

The correct force feedback scaling should allow safe, fatigue free, yet proprioceptively rich information. One of the most important human limitations, endurance, is related to the magnitude of the muscular force and the time over which it must be exerted. Figure 2.2 illustrates that people can maintain their maximum effort only briefly, whereas they can exert a 25% force or less for an extended period [Brooks]. Since a simulation may require the user to exert a force over longer periods of time, the force should be well below the individual's maximum force limits. The maximum grasp force, the ability to generate torques, and the user's endurance are functions of the grasping technique, and range from 147 lbs to 13 lbs. For a light hand grasp, the just noticeable force difference is about 6-8% for a reference force ranging from 2 to 5 lbf. The related safety issues are obvious, as any motor that generates even 5 lbs at a lever arm of 18 inches is rather formidable, and can easily accelerate the light joystick endpoint to high velocities. However, for a high frequency response, the larger torque outputs are necessary to prevent motor amplifier saturation and deliver enough acceleration to follow the high frequency inputs. As a compromise, the joystick linkage inertia is minimized, and a high closed loop bandwidth is maintained with motors limited at 1.5 ftlbs of continuous stall torque. The stall torque corresponds to a maximum force level at the joystick endpoint that is well below the fatigue limit. In the designed joystick large resistive torques are generated by proportional brakes that provide large decelerations without the dangers of energy storing, high acceleration motors.

2.2 Detail Mechanical Design

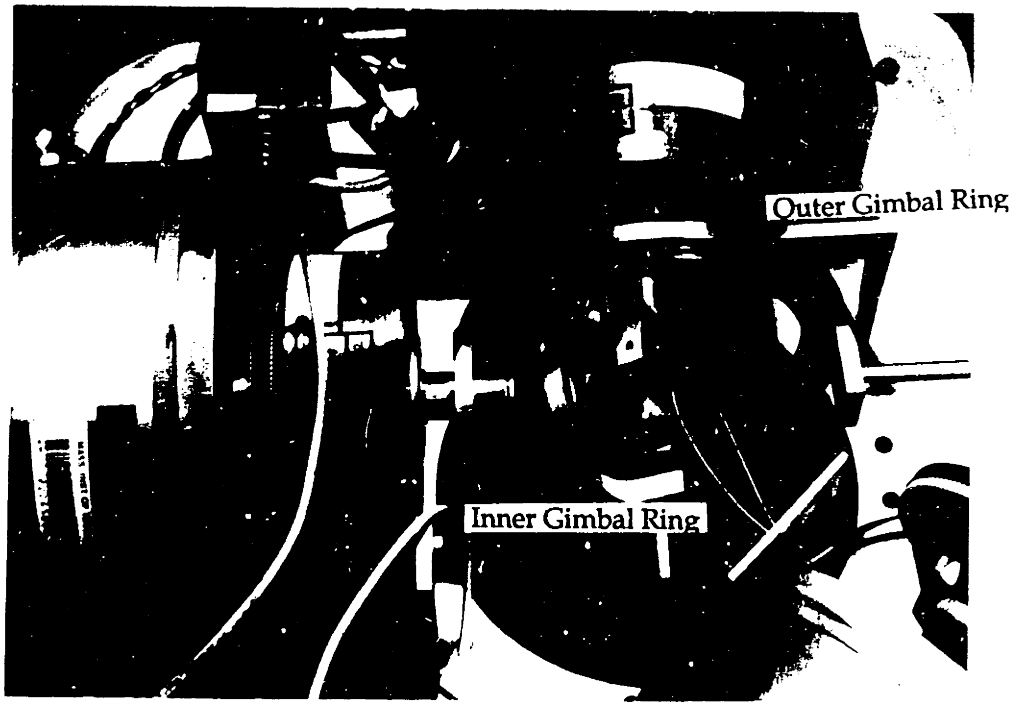
2.2.1 Gimbal linkage characteristics

The main determinant of the joystick geometry as shown in figure 2.3 proved to be the requirement of three independent, fixed to ground actuators for the three perpendicular endpoint directions and the limited size constraint . The two planar direction axes are controlled through a gimbal linkage. All three axes intersect in the middle of the gimbal pair. The gimbal bearings are displaced so as to allow the joystick shaft to pass through the gimbal center. Each gimbal ring is cantilevered off of a motor/brake pair shaft. The motor and brake that form an actuator pair are connected end to end, so they both drive the same shaft. A gimbal ring consists of a thin ball bearing ("Kaydon" Reali-Slim), and an outer and inner race support ring. The outer race ring is supported on both sides by an offset that allows the ring to be slipped onto the drive shaft and fixed with a set screw. For the third axis motion, the inner support rings have similar offsets with sintered bronze bearings that allow the joystick shaft to reciprocate through the gimbal center. Opposite the motor/brake driveshafts, there is a bearing support block that distributes the vertical reaction loads caused by the joystick shaft motion and the endpoint forces. These loads would otherwise appear as a pure moment across the outer bearing ring supported by the motor/brake shaft alone.



a)

Figure 2.3: a) Joystick Mechanical System, b) Gimbal Linkage Detail



b)

Figure 2.3: a) Joystick Mechanical System, b) Gimbal Linkage Detail

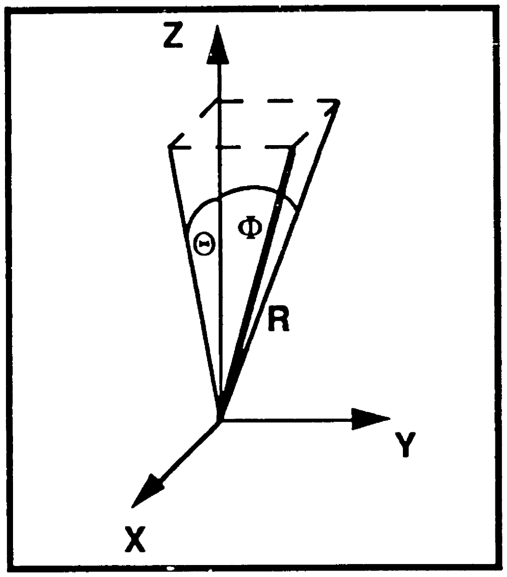


Figure 2.4: Coordinate Definitions

The gimbal geometry constrains the shaft endpoint to move on a spherical surface. As shown in figure 2.4, the measured rotation angles θ and Φ are the projection of the gimbal ring rotations into the XZ and YZ planes defined by the inertial cartesian frame centered at the gimbal rings and shaft axes intersection. The joystick shaft orientation is fixed relative to the inner bearing. The shaft remains in the plane defined by the inner bearing, so any force applied along a direction tangential to and in the plane of the bearing will not have any torque components across the bearing outer support ring. The two bearings rotate relative to each other and do not intersect at right angles throughout the workspace. The joystick shaft must therefore retain a rotational degree of freedom relative to the outer bearing. This extra degree of freedom is provided by the same bronze bushings that allow the linear shaft motion. The gimbal geometry is uncoupled in that an angular velocity about the θ, Φ axes will produce a respective endpoint motion in a plane parallel to the XZ, YZ inertial coordinate planes. The geometric transformations necessary for controlling the joystick are discussed in more detail below.

2.2.2 Shaft Axis Transmission

The shaft linear motion is actuated by a rotary motor/brake pair. A transmission is therefore necessary to transduce from rotary to linear motion along the shaft axis. Initially, a leadscrew drive system suspended in the gimbal center was suggested [Smith]. The high possible gear ratio and therefore small actuator size, and the simplicity of the transmission represent distinct advantages over other transmission types. However, a leadscrew is not passively backdriveable if the screw pitch angle is less than the friction cone angle, so the transmission ratio is limited by the backdriveability requirement. The actuators can thus not be arbitrarily small. The added inertia of the motor and brake is the main disadvantage of this configuration, since this would have to be carried by the gimbal linkage drive motors. Further, the kinematic complexity of controlling the endpoint forces is increased by coupling of the motor reaction torques to the gimbal rings. Primarily for these reasons a leadscrew transmission was not implemented.

Next, a series of cable/pulley transmissions were considered that have the distinct advantages of a high stiffness to weight ratio and that the actuator could be fixed at a remote location, the joystick base. All reaction torques are therefore transmitted to ground and do not couple into the other axes. A pulley transmission also has a high efficiency, so viscous and Coulomb friction effects are minimized. The cable runs in a closed loop from the shaft end, over a series of idler pulleys, around the motor drive shaft, and back to the opposite end of the joystick shaft, as shown in figure 2.5. Because of cable pretension, the cable always remains in a tension state during operation.

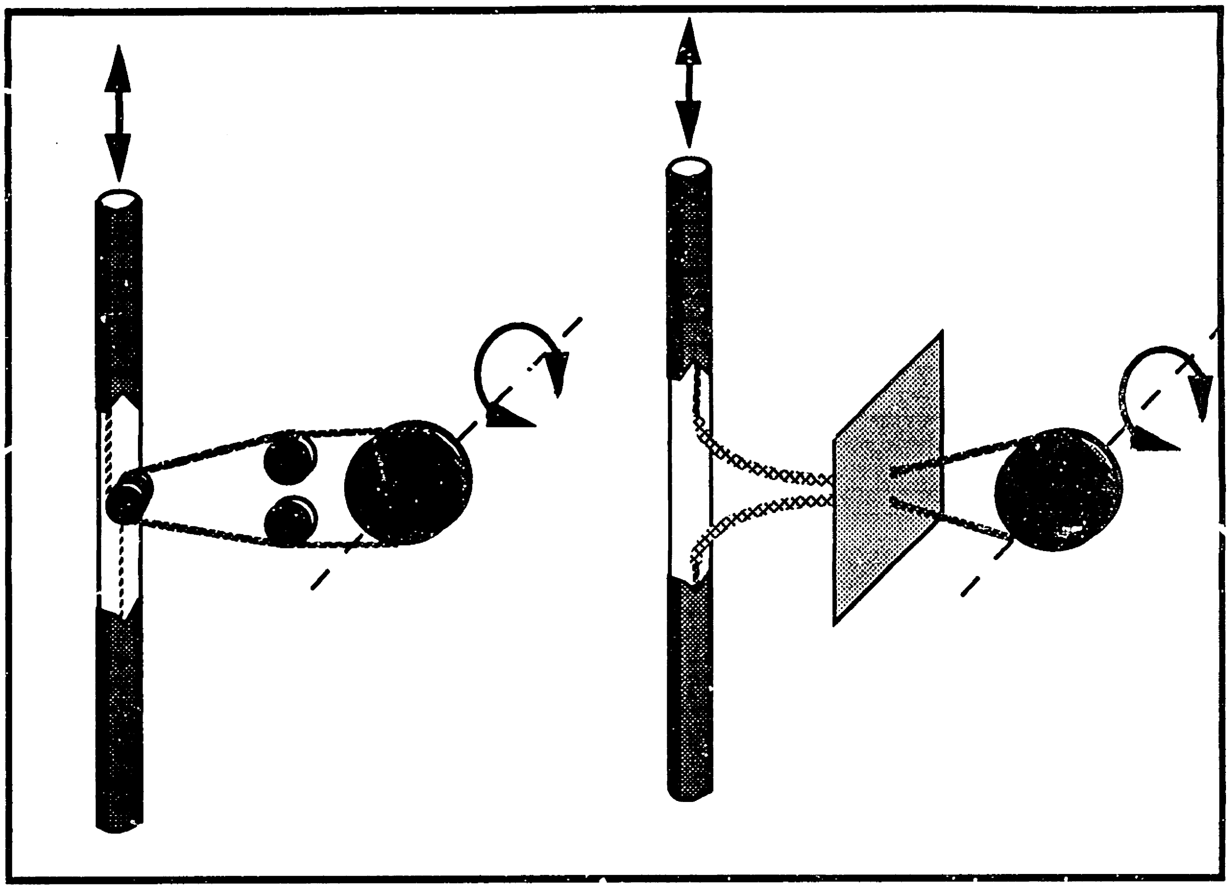


Figure 2.5: Cable Transmission Concepts

However, a detailed analysis of the gimbal geometry and cable length variations as the shaft rotates about the gimbal center reveal that the shaft must reciprocate as the cable winds around the center pulley as seen by inspection of figure 2.5. This added kinematic complexity, together with the fact that the minimum cable pulley diameter is limited by the bending radius the cable can withstand without fatiguing too quickly, proved to be the major difficulty in implementing such a compact cable transmission.

To keep the high stiffness to weight ratio advantage of a cable transmission and maintain the cable bending radius at a maximum, transmission efficiency was compromised by introducing a frictional cable sleeve to guide the cable out of the gimbal center to the actuator shaft. Independent of the gimbai position, the cable length is fixed and does not wrap around any idler pulleys causing a coupling to the shaft linear motion. The maximum cable bending radius is determined by the distance from the sleeve attachment point in the gimbal to the attachment at the motor/brake shaft, which can be chosen at an arbitrary distance. The limiting drive shaft radius the cable must pass over is given solely by the transmission ratio which relates the desired output force to the actuator output torque. The cable frictional losses are a direct function of the sleeve angle and the pretension, and this relation is developed in section 3.2.2.1. The added friction can be compensated for by closing a force control loop along the joystick shaft direction, as detailed below.

The cable is guided along the shaft in a rectangular slot and is fixed at both ends by a retaining ring. A turnbuckle at one end of the shaft allows careful pretension adjustments. For a desired force output range, the pretension can be adjusted so as to minimize sleeve frictional losses but prevent any slippage on the motor/brake drive shaft. The sleeve is held aligned with the shaft slot by a brass terminal fixed to the inner gimbal offset through which the joystick shaft passes. The brass terminal also prevents any rotational shaft motion relative to the inner gimbal. The implementation of the sleeve cable transmission is much simpler than a complex pulley system. Figure 2.6 shows a detailed view of the actual cable transmission.

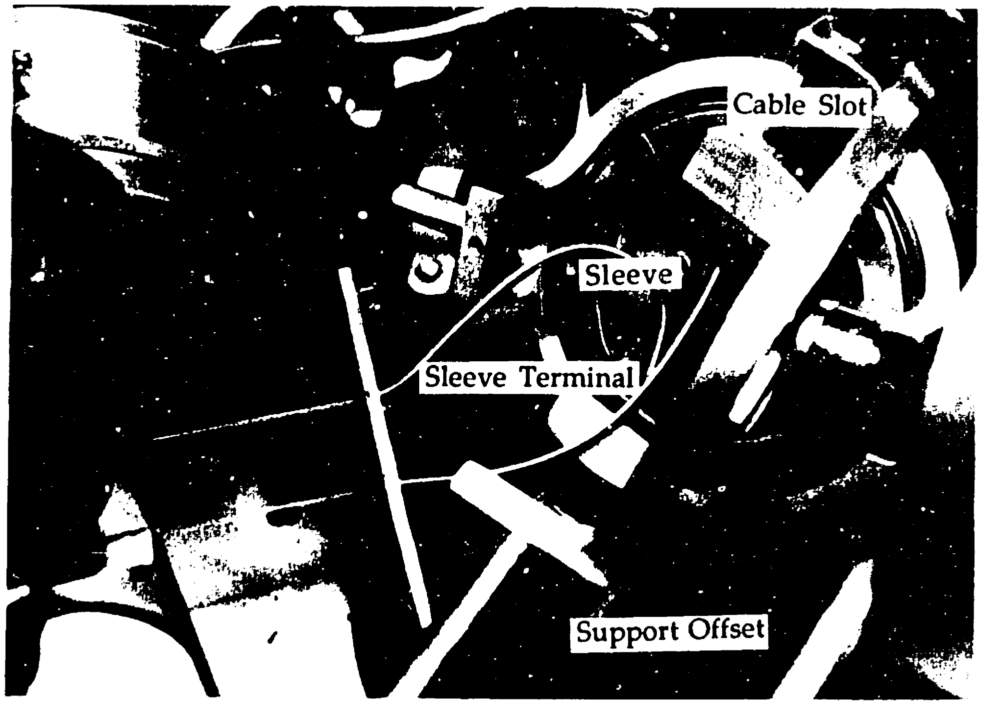


Figure 2.6: Cable Transmission detail

The shaft must reciprocate and rotate relative to the outer gimbal. However, a ball bearing could not minimize friction because of the cable slot running along the length of the shaft. The relative rotation would not allow the bearing balls to remain in contact with the shaft at all times, but cause them to fall into the cable slot. To maintain simplicity and because of the proportionally large friction introduced by the cable/sleeve transmission, sintered bronze bearings are used instead. Cable sleeve supports attached to the inner bearing allow the sleeve to maintain an alignment with the shaft slot and rotationally fix the shaft relative to the inner bearing .

2.2.3 Actuator Selection

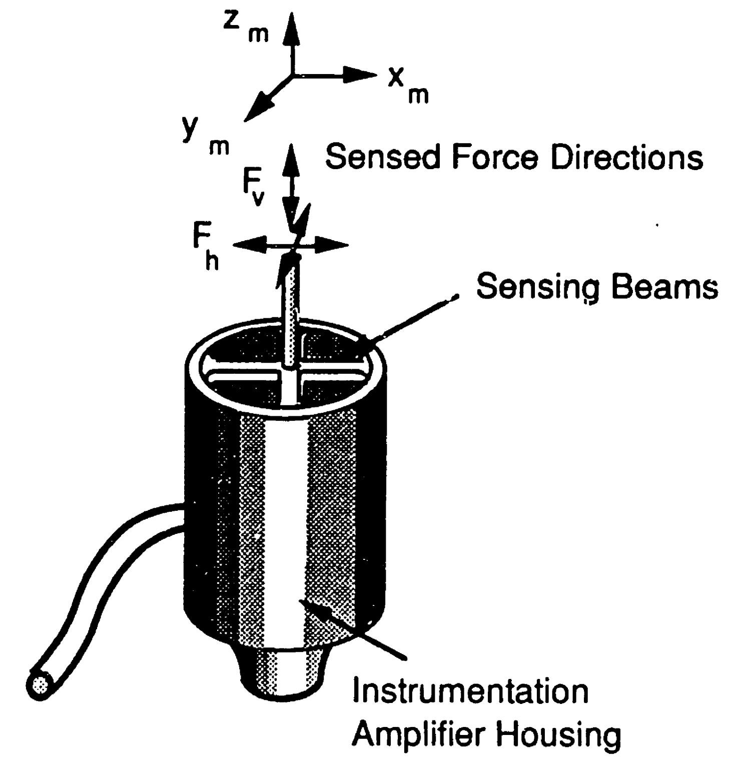
In the high bandwidth simulations the joystick is specified to achieve, it is essential that the actuator bandwidth be virtually flat across the frequencies of interest. Further, output noise such as commutator torque ripple has to be minimized. The only motors available that meet the bandwidth requirements with no torque variation for a given command current were printed "pancake" armature disk DC brush motors (PMI Servodisc U-series). The motor stator consists of permanent magnets, and the printed armature disk rotates between magnet pairs. The absence of rotating iron eliminates cogging or "preferred" armature positions under the magnetic poles during low speed operation. In addition, the many conductors (165 on motors used) on the commutating surface provide a smooth path for the brushes, minimizing torque ripple. Low armature inertia enables these motors to achieve a mechanical time constant of 8.8 msec or a bandwidth of 120 Hz. The axial length is compact in comparison to conventionally wound motors and the absence of iron in the armature reduces the motor weight considerably. The only disadvantage of these motors is the relatively large housing diameter for the given torque output, and the high price. The chosen motors also include an integral tachometer. The motor stall torque output is 1.45 ftlbf, which is considered adequate for smaller scale force feedback.

The incorporated brakes are current controlled magnetic particle brakes that consist of an armature rotating in a fluid in which ferromagnetic particles are suspended. The current induced magnetic field causes ferromagnetic particles to precipitate onto the armature plate, increasing its effective surface area and generating a proportional increase in fluid mechanical resistance. The torque output of the brakes is 5 ftlbf for one tenth the maximum power input of the motors. The mechanical time constant is 12 msec, of the same order as the motor time constant. Hysterisis effects due to residual magnetic fields also need to be compensated for [Tadros].

2.2.4 Sensor Designs

2.2.4.1 Load cell

A three degree of freedom loadcell has been designed for closed loop force control to backdrive the frictional cable drive and for active control of the brakes. As shown in figure 2.7, the loadcell consists of a bending stem that applies loads to a beam cross, where each cross leg is a separate bending beam. The bending strains in the four beams can be directly correlated to the force applied at the endpoint of the sensor. A full strain gage Wheatstone bridge on each cross leg senses the bending strains without the need of thermal compensation or a bridge resistance balance potentiometer, and also maximizes the loadcell sensitivity. The reference frame fixed to the loadcell endpoint is as shown in the figure. Loads in the x_m or y_m directions generate pure opposite bending moments in the corresponding cross beam pair. For a load in the z_m direction, all four beams will sense the same bending moment. The different load configurations are shown in figure 2.7.



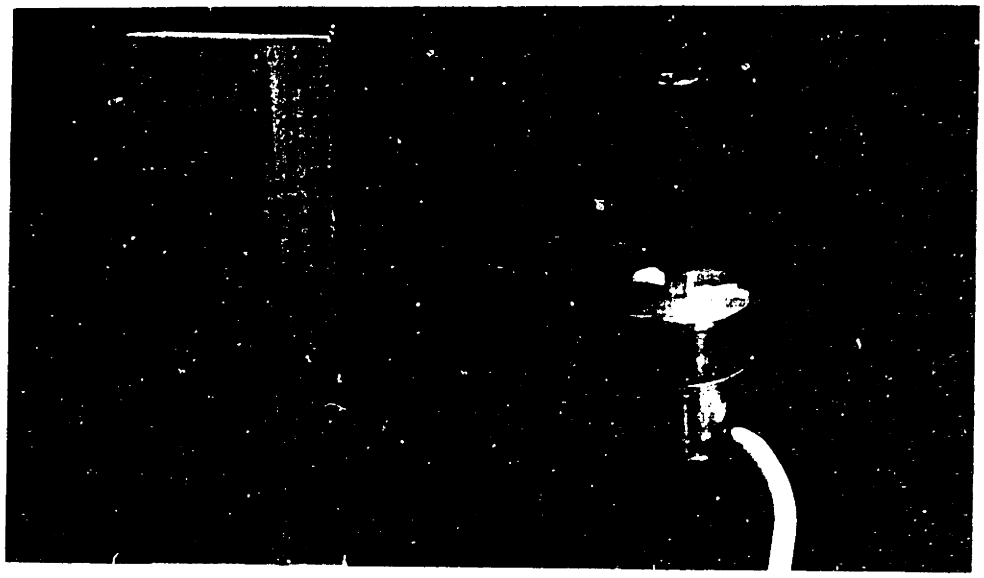


Figure 2.7: a) Loadcell sensor design

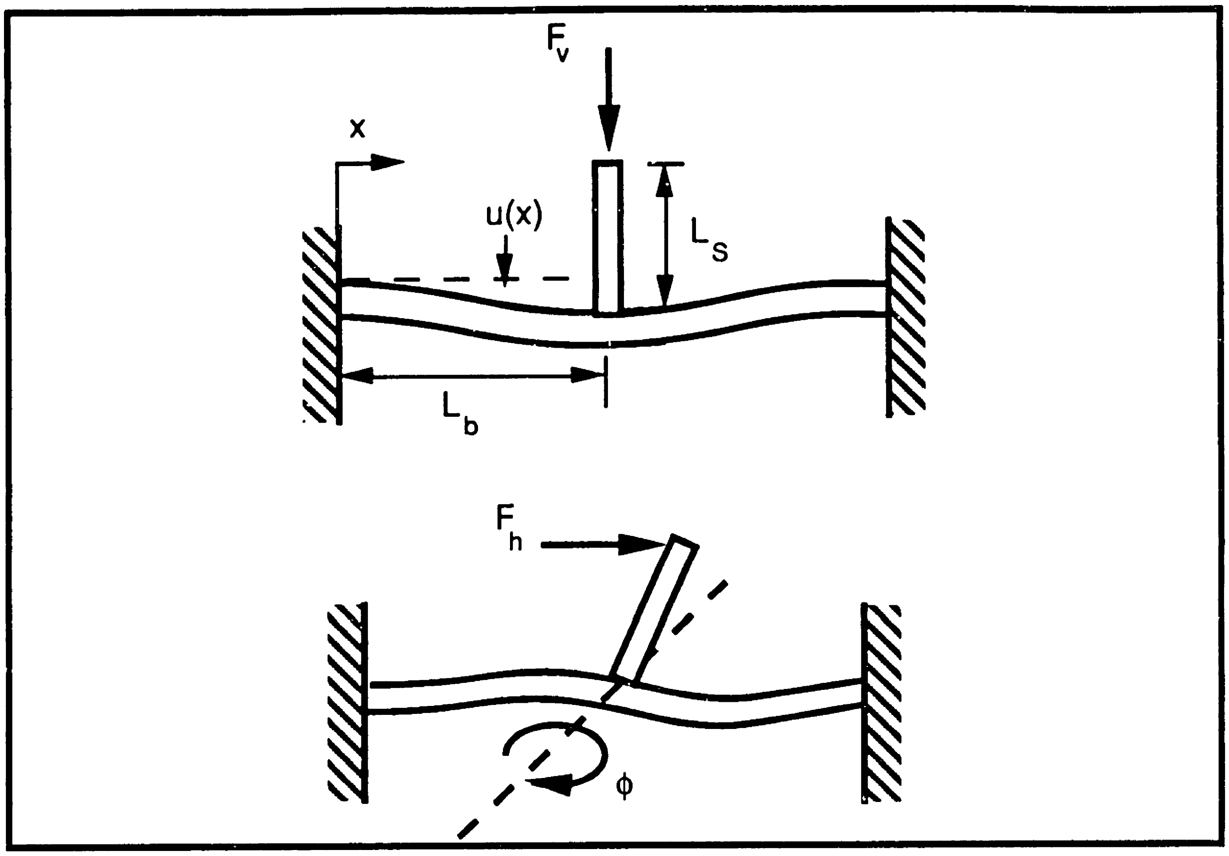


Figure 2.7: b) Load Configurations

For bending in the x_m and y_m horizontal directions, the boundary conditions are simply:

$$u=0,\ \frac{du}{dx}=0 \qquad \text{at } x=0$$

$$u=0,\ M_b = EI\frac{d^2u}{dx^2} \quad \text{at } x=L_b \tag{2.1}$$

where E is the Young's modulus of the loadcell material and I is the sensing beam moment of inertia.

The displacement and moment distributions are are found by solving the Bernoulli beam bending equation:

$$M(x) = \frac{3}{2}\frac{M_b}{L_b}x - \frac{M_b}{2}$$

$$u(x) = \frac{1}{4}\frac{M_b}{EIL_b}x^3 - \frac{M_b}{4\,EI}x^2 \tag{2.2}$$

The maximum bending moment is at the loadcell cross center, so the greatest sensitivity is achieved by locating the sensing bridges close to the loadcell stem. For a simple Bernoulli beam, the bending strain ε is then:

$$\varepsilon = \frac{M_b}{2\,EI}h \tag{2.3}$$

where h is the beam cross section height.

The unknown bending moment M_b for a given load cell stem length L_s and applied force F_h is given by equilibrium. The torsional bending stiffness of the non-sensing cross beams cannot be neglected. For a rectangular cross section the torsional stiffness K_ϕ is given by:

$$K_\phi = \frac{M_\phi}{\phi} = .33\,\frac{b\,h^3}{L_b}G\left[1 - 0.63\,\frac{h}{b}\left(1 - \frac{h^4}{12\,b^4}\right)\right] \tag{2.4}$$

where G is the shear modulus of elasticity, and ϕ is the torsional angle of deflection. From the bending analysis above, the torsional angle of deflection is related to the bending displacement of the sensing beams by:

$$\phi = \frac{d\,u}{d\,x}\Big|_{x=L_b} \tag{2.5}$$

so

$$\phi = \frac{1}{4}\frac{M_b\,L_b}{EI}$$

Given a stem length of L_s, and a horizontally applied force F_h, moment equilibrium yields:

$$F_h L_s = 2\,M_b + 2\,M_\phi$$
$$= 2\,M_b + 2\,K_\phi\left(\frac{1}{4}\,\frac{M_b L_b}{EI}\right) \qquad (2.6)$$

solving for the bending moment alone:

$$M_b = \frac{F_h L_s}{2 + \frac{1}{2}\,\frac{K_\phi L_b}{EI}} \qquad (2.7)$$

and the bending strain in equation (2.3) above is uniquely determined.

For vertical loading F_v along the stem axis, the moment distribution through any of the four sensing beams is:

$$M_b = \frac{F_v}{8}\,x - \frac{F_v}{16}\,L_b \qquad (2.8)$$

and the strain is then:

$$\varepsilon = \frac{F_v L_b h}{32\,EI} \qquad (2.9)$$

The loadcell material is 2024 T4 Aluminum alloy because of its high resistance to ductile deformation, which limits any mechanical hysteresis effects and maintains the linearity of the loadcell signal. Given the Aluminum shear and Young's moduli, the chosen sensing gage geometry, a maximum sensed force of 10 lbf, and a maximum elastic strain of .1% to limit hysteresis effects, the beam height is fixed at .125 in. From the bending strains that induce a change in the gage resistances, the applied load vector can be resolved.

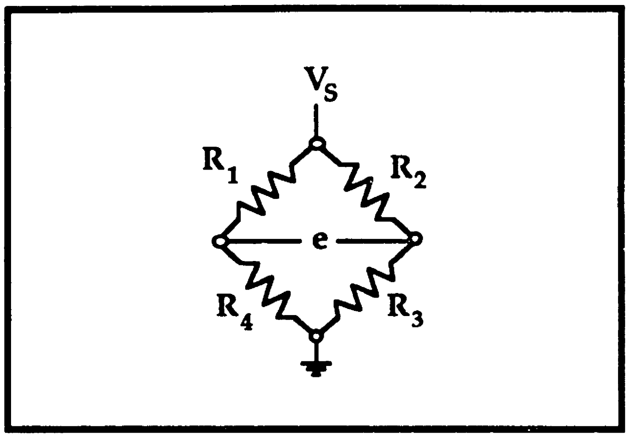


Figure 2.8 Wheatstone Bridge Resistance Definitions

For a general Wheatstone bridge as shown in figure 2.8, the output voltage e is given by:

$$e = V_S \frac{R_1R_3 - R_4R_2}{(R_1 + R_2)(R_3 + R_4)} \tag{2.10}$$

The output voltage sensitivity to changes in the gage resistances is then found by taking the corresponding partial derivatives:

$$\Delta e = \frac{R_2\,R_1}{(R_1 + R_2)^2}\left(\frac{\Delta R_1}{R_1} - \frac{\Delta R_2}{R_2}\right)V_S + \frac{R_3\,R_4}{(R_3 + R_4)^2}\left(\frac{\Delta R_3}{R_3} - \frac{\Delta R_4}{R_4}\right)V_S \tag{2.11}$$

Since the gage resistances are all the same (350 Ohm), the above term can be regrouped to:

$$\Delta e = \frac{1}{4}\left(\frac{\Delta R_1}{R} - \frac{\Delta R_2}{R} + \frac{\Delta R_3}{R} - \frac{\Delta R_4}{R}\right)V_S \tag{2.12}$$

To maximize the output voltage for a given bending strain, the resistance changes should be configured so that:

$$\Delta R_1 = -\Delta R_2 = \Delta R_3 = -\Delta R_4 = \Delta R \tag{2.13}$$

The resistance change is in turn related to the applied strain by the gage factor GF:

$$\varepsilon_n = \frac{\Delta R_n}{R_n}\frac{1}{GF} \tag{2.14}$$

so that the output voltage is then simply related to the bending strain:

$$\Delta e = V_s\, \varepsilon\, GF \tag{2.15}$$

Since the output voltage range should correspond to the A/D converter range of ± 10V with a maximum measured force of 5 lbf corresponding to the highest brake torque output, the necessary bridge amplifier gain is 1000 V/V for a gage factor of 2.

To calibrate the four voltage output channels of the loadcell to the applied endpoint forces, the voltages are measured and read into a 4x1 vector. The corresponding forces are also represented as 3x1 vectors. After m measurements, a projection matrix H (4x3) relates the measured voltage matrix (4xm) to the applied force matrix F (3xm):

$$\begin{aligned} &V = HF \\ &H^T V = H^T H F \\ &(H^T H)^{-1} H^T V = C\, V = F \end{aligned} \tag{2.16}$$

The applied forces are related to the measured voltages by the pseudo-inverse $C=(H^T H)^{-1} H^T$. Because of the one redundant voltage signal, the signal to noise ratio is improved by finding the optimal calibration matrix H that minimizes the measurement error by projecting the voltage measurements into the 3-dimensional force space:

$$\begin{aligned} &V F^T = H F F^T \\ &V F^T (F F^T)^{-1} = H \end{aligned} \tag{2.17}$$

The joystick endpoint includes a balljoint to decouple the controlled linear forces from any applied torques. The necessary code to find the calibration matrix C is given in Appendix A2.

2.2.4.2 Position and Velocity Sensors

The joystick endpoint position is measured from incremental optical encoders located at the motor/brake drive shafts. The encoders have 4096 counts for a full revolution, and are initialized at the beginning of a control routine. Since the gimbal linkage allows only a range of 90 degrees of motion, the encoders are geared through a belt with a ratio of 4:1 so that the full gimbal motion corresponds to a full encoder rotation. No time delay is added by any analog to digital conversions, because encoders give a direct digital position measurement of the drive shaft. The joystick shaft axis position is similarly measured by a 1:3 geared encoder. From the measured drive shaft positions, the joystick endpoint position is found through the kinematic relations developed in the next chapter.

The endpoint velocity is extrapolated from the joint velocities measured by sensors located also at the actuator drive shafts. The drive motors include integral tachometers that provide an analog voltage proportional to the shaft rotational velocity. The tachometers have low ripple noise because they are based on the same printed disk, high conductor technology as the motor armature. They are therefore ideally suited to measure low velocities with high signal to noise ratios. The tachometer signals had to nevertheless be heavily filtered to reduce coupling noise introduced by the PWM motor amplifiers 20 kHz operating frequency. A more detailed description of the filter and tachometer calibration procedure can be found in [Tadros]. The endpoint velocity could be measured by differentiating the encoder position signals, but at low velocities differentiation results in a noisy velocity signal due to encoder quantization. The added computational burden introduces a longer delay than the analog to digital conversion required for the tachometer signal measurement. The relation between the drive shaft and endpoint velocities is given by the Jacobian described in the next chapter.

2.2.5 Aesthetic and Safety Considerations

From a human factors standpoint, a force reflective virtual tool must communicate clearly its function and correct use without affecting the simulation. The form of the mechanical drive and linkage should not distract from the task. The joystick height, endpoint, and general appearance ought to be inviting and natural to use, not configured as a machine tool that the user will be preoccupied with. The overall size and form must be small, simple and clean. In general, the user should forget during a particular simulation that the actual joystick does not correspond directly both spatially and in form with what is seen on the display screen, so that a complete identification with the simulation can occur.

The joystick base must be be easily positioned at the user's discretion, yet also be stable enough to react against the forces generated by the motors and brakes without deflecting. The portability of the joystick is enhanced by a modular configuration, separating I/O electronics from power amplifiers and the actual mechanical hardware. In the designed device, the base mass and support footprint are adequate to react against the generated torques, so no mechanical restraint is necessary.

The joystick cover provides a safety shield between the user and the mechanical and power electronic hardware, and also provides a convenient support the user may use as an arm brace to limit fatigue while executing wrist-scale motions. As further precautions, an enable/disable switch is hardwired at the motor amplifier console control panel, within easy reach of the user. The tachometers are also monitored within the software control loop as a continuous safety check. All motors are disabled if the velocity signal reaches above a specified threshold level. The complete joystick system, with base, amplifiers and cover, is shown in figure 2.9.

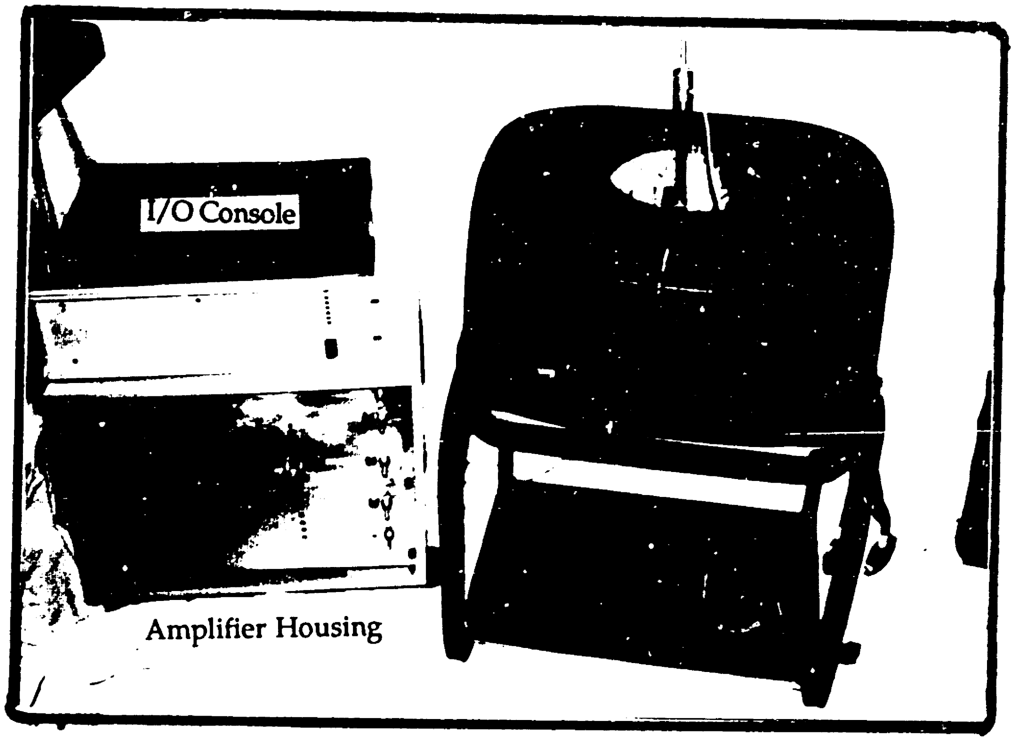


Figure 2.9: Complete Joystick System

Chapter 3

Kinematic and Dynamic Analysis

3.1 Kinematic Analysis

3.1.1 Coordinate transformations

For the joystick linkage coordinate geometry shown in figure 3.1, the coordinate transformation equations are given below. The gimbal geometry is a closed kinematic chain that constrains the endpoint to move on a spherical surface. The location of the endpoint on the sphere is fixed by two generalized coordinates that are rotation angles relative to any two axes intersecting in the sphere center, not necessarily at right angles. In a traditional spherical coordinate system, a point on a spherical surface is fixed by two consecutive rotation angles, first about the z relative to the x-axis and then about an intermediate axis x' relative to the z-axis. If the rotations are represented by circles inscribed on the sphere surface, then these circles will always be perpendicular. The measured generalized coordinates in the joystick are rotation angles about two fixed and pependicular axes, the X and Y axes of the inertial coordinate frame fixed in the gimbal center. The gimbal rings that locate the endpoint of the joystick on the spherical surface do not intersect at right angles except for zero rotation. The sphere radius corresponds to the measured R coordinate value.

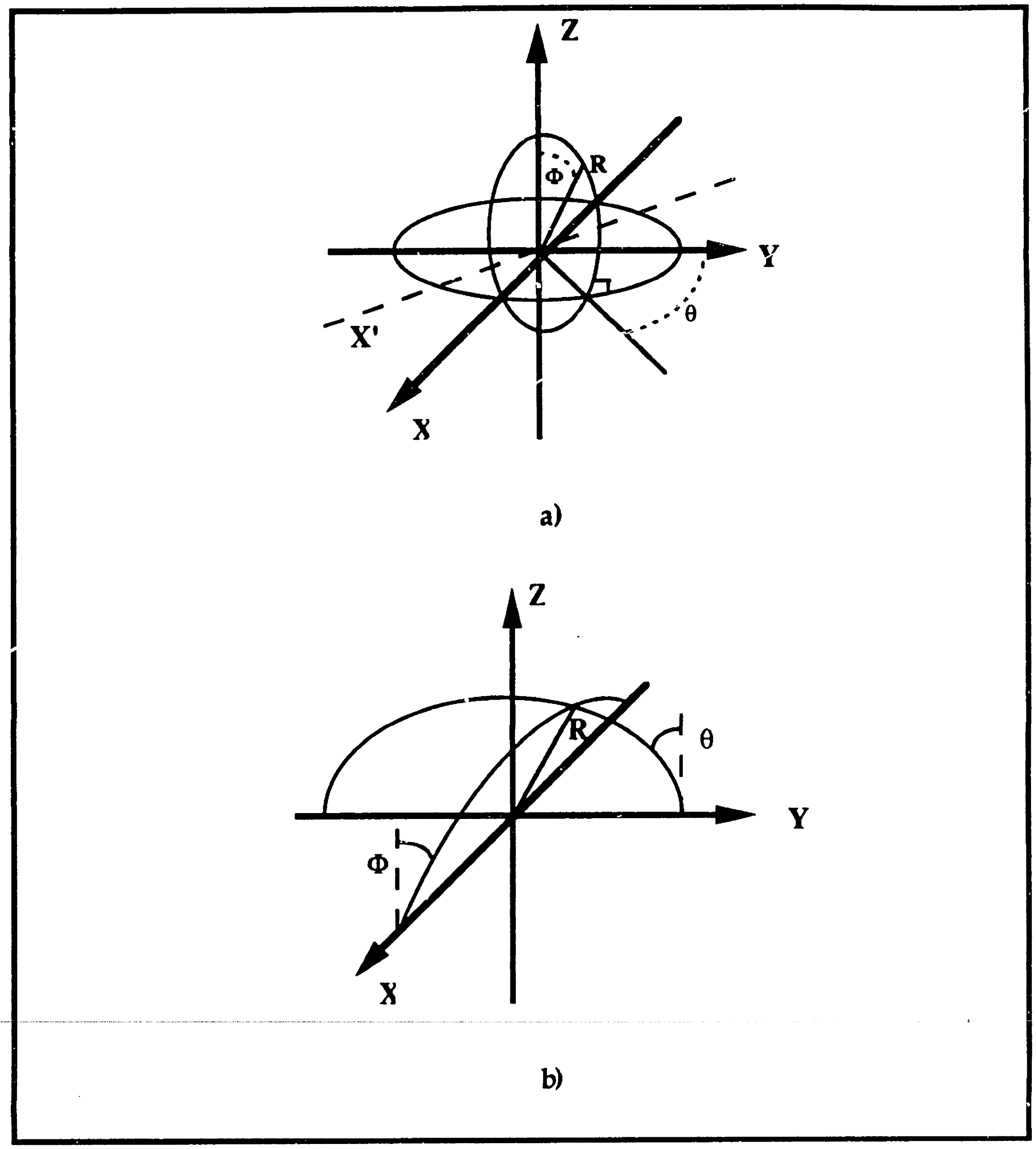


Figure 3.1: a) Spherical Coordinate Geometry, b) Measured Joystick Coordinates

The joystick endpoint components in the inertial frame are denoted by R_X, R_Y, and R_Z. The relationship between the measured joystick coordinates and the cartesian inertial frame are then:

$$R=\sqrt{R_X^2+R_Y^2+R_Z^2}$$
$$\tan\Phi=\frac{-R_Y}{R_Z}$$
$$\tan\theta=\frac{R_X}{R_Z} \tag{3.1}$$

A simulated environment geometry is most naturally specified in a cartesian frame. For the control strategies described below, the forces generated by the environment simulator are given in cartesian space. If an impedance control approach is used to generate the necessary forces, or for simple screen display purposes, it is the joystick endpoint location in cartesian space that is of interest. Inverting the above equations to solve for the inertial frame coordinates yields:

$$R_X=\frac{R\tan\theta}{\sqrt{\left(\tan^2\Phi+\sec^2\theta\right)}}$$
$$R_Y=-\frac{R\tan\Phi}{\sqrt{\left(\tan^2\Phi+\sec^2\theta\right)}}$$
$$R_Z=\frac{R}{\sqrt{\left(\tan^2\Phi+\sec^2\theta\right)}} \tag{3.2}$$

or in terms of sines and cosines only:

$$R_X=\frac{R\sin\theta\cos\Phi}{\sqrt{1-\sin^2\theta\sin^2\Phi}}$$
$$R_Y=-\frac{R\sin\Phi\cos\theta}{\sqrt{1-\sin^2\theta\sin^2\Phi}}$$
$$R_Z=\frac{R\cos\theta\cos\Phi}{\sqrt{1-\sin^2\theta\sin^2\Phi}} \tag{3.3}$$

Since the above components describe the position vector of the endpoint in cartesian space, the magnitude of the vector is the measured R coordinate value. Therefore, the unit vector pointing along the joystick shaft direction is simply:

$$\vec{n}_s = \frac{\sin\theta\cos\Phi}{\sqrt{1-\sin^2\theta\sin^2\Phi}}\vec{i} - \frac{\sin\Phi\cos\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}}\vec{j} + \frac{\cos\theta\cos\Phi}{\sqrt{1-\sin^2\theta\sin^2\Phi}}\vec{k} \qquad (3.4)$$

and describes the orientation of the joystick shaft in the inertial frame. Since the three-axis loadcell is attached to the joystick endpoint, the measured force coordinates are aligned with the joystick shaft. If the applied force in the cartesian frame is desired, the measured force components have to be rotated into the inertial frame. The three sensed coordinates are denoted by:

$$\vec{F}_m = \begin{bmatrix} F_{mx} \\ F_{my} \\ F_{mz} \end{bmatrix}$$

The three unit vectors that describe the measured force directions have to be specified from measured joystick shaft coordinates. The z-component of the sensed force vector is always aligned in the joystick shaft direction, thus along the shaft unit vector:

$$\vec{n}_{mz} = \vec{n}_s \qquad (3.5)$$

Because the shaft orientation is fixed relative to the inner gimbal ring, the plane that is defined by the inner gimbal has a unit vector that is always perpendicular to the shaft orientation vector and is given by:

$$\vec{n}_{my} = \cos\Phi\vec{j} + \sin\Phi\vec{k} \qquad (3.6)$$

The third measured force component direction is therefore simply given by the cross product of the previous two:

$$\vec{n}_{mx} = \vec{n}_{my} \times \vec{n}_{mz} \tag{3.7}$$

which yields:

$$\vec{n}_{mx} = \vec{i}\,\frac{\cos\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} + \vec{j}\,\frac{\sin\theta\sin\Phi\cos\Phi}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} - \vec{k}\,\frac{\cos^2\Phi\sin\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} \tag{3.8}$$

A transformation matrix T that rotates inertial cartesian forces into loadcell measured states is defined as:

$$\vec{F}_m = T\vec{F} \tag{3.9}$$

and is given by:

$$T = \begin{bmatrix} \frac{\cos\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} & \frac{\sin\theta\sin\Phi\cos\Phi}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} & -\frac{\cos^2\Phi\sin\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} \\ 0 & \cos\Phi & \sin\Phi \\ \frac{\sin\theta\cos\Phi}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} & -\frac{\sin\Phi\cos\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} & \frac{\cos\theta\cos\Phi}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} \end{bmatrix} \tag{3.10}$$

Note that the transformation matrix represents a pure rotation. The transformation matrix is therefore orthonormal, and the inverse is equal to the transpose of the transformation matrix. The matrix transformation from the measured loadcell to the inertial cartesian forces is then simply:

$$\vec{F} = T^{-1}\vec{F}_m = T^t\vec{F}_m \tag{3.11}$$

The relation between the measured or inertial force components and the actuator torques needs to be developed in order to control the endpoint forces.

The actuator torques and endpoint forces are not, however, related through a simple rotation matrix. The necessary transformation matrix is instead developed in the next section.

3.1.2 Jacobian characteristics

The differential displacement relationship between two coordinate frames is given by the Jacobian, a matrix of partial derivatives. The Jacobian is simply a compact description of applying the chain rule to relate derivatives in one set of coordinates to those in another. If the endpoint position vector components are as specified in the previous section, then the Jacobian is simply:

$$J = \begin{bmatrix} \frac{\partial}{\partial \theta} R_X & \frac{\partial}{\partial \Phi} R_X & \frac{\partial}{\partial \Psi} R_X \\ \frac{\partial}{\partial \theta} R_Y & \frac{\partial}{\partial \Phi} R_Y & \frac{\partial}{\partial \Psi} R_Y \\ \frac{\partial}{\partial \theta} R_Z & \frac{\partial}{\partial \Phi} R_Z & \frac{\partial}{\partial \Psi} R_Z \end{bmatrix}$$

where Θ, Φ and Ψ are the drive shaft positions measured by the optical encoder sensors. The R coordinate that appears in the previous section is simply given in terms of the measured driveshaft angular displacement by:

$$R = r\Psi \tag{3.12}$$

where r is the driveshaft radius of the cable transmission. Evaluating the Jacobian yields:

$$J=\begin{bmatrix} \dfrac{r\Psi\cos\theta\cos\Phi}{\left(-\sin^2\theta\sin^2\Phi+1\right)^{3/2}} & -\dfrac{r\Psi\cos^2\theta\sin\theta\sin\Phi}{\left(-\sin^2\theta\sin^2\Phi+1\right)^{3/2}} & \dfrac{r\sin\theta\cos\Phi}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} \\ \dfrac{r\Psi\cos^2\Phi\sin\theta\sin\Phi}{\left(-\sin^2\theta\sin^2\Phi+1\right)^{3/2}} & -\dfrac{r\Psi\cos\theta\cos\Phi}{\left(-\sin^2\theta\sin^2\Phi+1\right)^{3/2}} & -\dfrac{r\sin\Phi\cos\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} \\ -\dfrac{r\Psi\cos^3\Phi\sin\theta}{\left(-\sin^2\theta\sin^2\Phi+1\right)^{3/2}} & -\dfrac{r\Psi\cos^3\theta\sin\Phi}{\left(-\sin^2\theta\sin^2\Phi+1\right)^{3/2}} & \dfrac{r\cos\theta\cos\Phi}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} \end{bmatrix} \tag{3.13}$$

If the derivatives are with respect to time, it is clear from its definition that the Jacobian relates the endpoint velocity in cartesian space to the measured velocities in the drive joint coordinates:

$$\vec{v}=J\begin{bmatrix}\dot{\theta}\\ \dot{\Phi}\\ \dot{\Psi}\end{bmatrix} \tag{3.14}$$

The quality of the workspace that is reachable by the joystick is quantified by the number of singularities over the operating range of joint values. A singularity is by definition a point where the rank of the Jacobian matrix decreases by one, so that a given velocity direction cannot be reached by any actuator commands. Because the Jacobian columns are linearly dependent in a singular configuration, the full X,Y, Z space of the inertial frame cannot be reached by linear combinations of the Jacobian columns. In the dual case, a particular desired force in the inertial frame cannot be reached by any joint torque commands. A matrix that is not of full rank will have a determinant of zero, so a singularity is given by joint values that make the determinant of the Jacobian vanish.

The Jacobian determinant in this case is given by:

$$|J|=-\frac{r^3\Psi^2\cos\theta\cos\Phi}{\left(-\sin^2\theta\sin^2\Phi+1\right)^{3/2}} \tag{3.15}$$

The only values for which the Jacobian will vanish are $\theta=+/-\ \pi/2$, $\Phi=+/-\ \pi/2$, or both. This is when either gimbal ring is rotated by $+/-\ \pi/2$ radians, so that motion of the other gimbal ring will only cause a relative rotation about the joystick shaft bushings, and not produce any tangential force component. This case is obvious by inspection of the gimbal geometry. Since the joint angle space lies within $+/-\ \pi/4$, this singularity is never reached. There are no other singularities over the specified force feedback space. From a purely kinematic standpoint therefore, any cartesian inertial force can be generated. The relation between the endpoint forces and the actuator torques, however, still needs to be developed.

How the actuator torques and endpoint forces in Cartesian coordinates are related is shown by applying the principle of virtual work. For virtual displacements in cartesian space δx, δy, and δz, and in the measured joint space $\delta\theta$, $\delta\Phi$, $\delta\Psi$ the total virtual work W is given by the dot products:

$$W=\vec{\tau}^T\begin{bmatrix}\delta\theta\\ \delta\Phi\\ \delta\Psi\end{bmatrix}=\vec{F}^T\begin{bmatrix}\delta x\\ \delta y\\ \delta z\end{bmatrix} \tag{3.16}$$

substituting the Jacobian relation for differential displacements:

$$\vec{\tau}^T\begin{bmatrix}\delta\theta\\ \delta\Phi\\ \delta\Psi\end{bmatrix}=\vec{F}^T J\begin{bmatrix}\delta\theta\\ \delta\Phi\\ \delta\Psi\end{bmatrix} \tag{3.17}$$

and simplifying:

$$\vec{\tau}^T=\vec{F}^T J \tag{3.18}$$

or:

$$\hat{\tau}=J^T F$$

The Jacobian then relates both velocities and forces between the inertial cartesian frame and the actuator coordinates.

An important feature of the gimbal geometry is its uncoupled nature, which becomes apparent in the inverse of the Jacobian:

$$J^{-1}=\begin{bmatrix} \frac{A\cos\theta}{r\Psi\cos\Phi} & 0 & -\frac{A\sin\theta}{r\Psi\cos\Phi} \\ 0 & -\frac{A\cos\Phi}{r\Psi\cos\theta} & -\frac{A\sin\Phi}{r\Psi\cos\theta} \\ \frac{\sin\theta\cos\Phi}{Ar} & -\frac{\sin\Phi\cos\theta}{Ar} & \frac{\cos\theta\cos\Phi}{Ar} \end{bmatrix} \tag{3.19}$$

where:

$$A=\sqrt{-\sin^2\theta\sin^2\Phi+1}$$

Neglecting the shaft translational degree of freedom, the zero off-diagonal terms express the fact that a velocity in the inertial X- or Y-direction will cause a rotation about only the θ or Φ axes respectively. Again, the dual case is a that each actuator torque only generates a force in one of the planar XY inertial frame directions. It is this decoupled nature that is the main advantage of the gimbal linkage, as it allows a direct control of cartesian forces with minimal computation cost devoted to kinematics.

To relate the actuator torques to the forces measured in the endpoint loadcell frame, the rotation matrix described above is substituted in the cartesian force to joint torque relation:

$$\hat{\tau}=J^t\hat{F}=J^tT^t\hat{F}_m \tag{3.20}$$

or evaluating the matrix products:

$$\vec{\tau}=\begin{bmatrix} -\dfrac{r\Psi\cos\Phi}{-\sin^2\theta\sin^2\Phi+1} & 0 & 0 \\ -\dfrac{r\Psi\sin\theta\sin\Phi\cos\theta}{-\sin^2\theta\sin^2\Phi+1} & -\dfrac{r\Psi\cos\theta}{\sqrt{-\sin^2\theta\sin^2\Phi+1}} & 0 \\ 0 & 0 & r \end{bmatrix}\vec{F}_m \tag{3.21}$$

Again, the sparsity of the transformation matrix from measured forces to actuator torques maintains a simple control strategy with limited computation cost. Note, however, that in the loadcell frame the forces and torques are not completely uncoupled because of the one off-diagonal term. This term expresses the relative rotation of the gimbal rings, which do not remain orthogonal throughout the joystick range of motion. Because the loadcell is fixed relative to the shaft and the inner gimbal ring as described above, the relative ring rotation manifests itself in the transformation matrix as a coupling term.

For completeness and because it is used in the control routines described below, the inverse transformation is also presented.

$$\vec{F}_m=\left(J^tT^t\right)^{-1}\vec{\tau}=\begin{bmatrix} -\dfrac{A^2}{r\Psi\cos\Phi} & 0 & 0 \\ \dfrac{A\sin\theta\sin\Phi}{r\Psi\cos\Phi} & -\dfrac{A}{r\Psi\cos\theta} & 0 \\ 0 & 0 & \dfrac{1}{r} \end{bmatrix}\vec{\tau} \tag{3.22}$$

where A is again as defined above. The kinematic relationships above fully describe the drive linkage geometry and transformations between the various coordinate frames of interest. However, to better understand the dynamic behavior of the joystick, a lumped parameter model is presented in the next section. The open-loop, acceleration dependent component of backdriveability is determined by the joystick inertia as a function of gimbal configuration. Further, the cable transmission dynamics are of interest for the closed-loop friction compensation that minimizes the velocity and inertial force components along the reciprocating shaft.

3.2 Joystick Dynamics

3.2.1 Gimbal Linkage Inertias

An analytical model for the dynamics of the three-axis joystick is developed below. The model parameters are as shown in Figure 3.2. The inertial frame is fixed in the gimbal center. The joystick end position is found within this inertial coordinate system by a transformation from the measured Θ, Φ, and $R=r\Psi$, coordinates, as described in the previous section. Because the gimbal geometry essentially represents a direct drive, the only dynamic parameter of interest is the configuration dependent inertia as seen by the actuator motors and brakes.

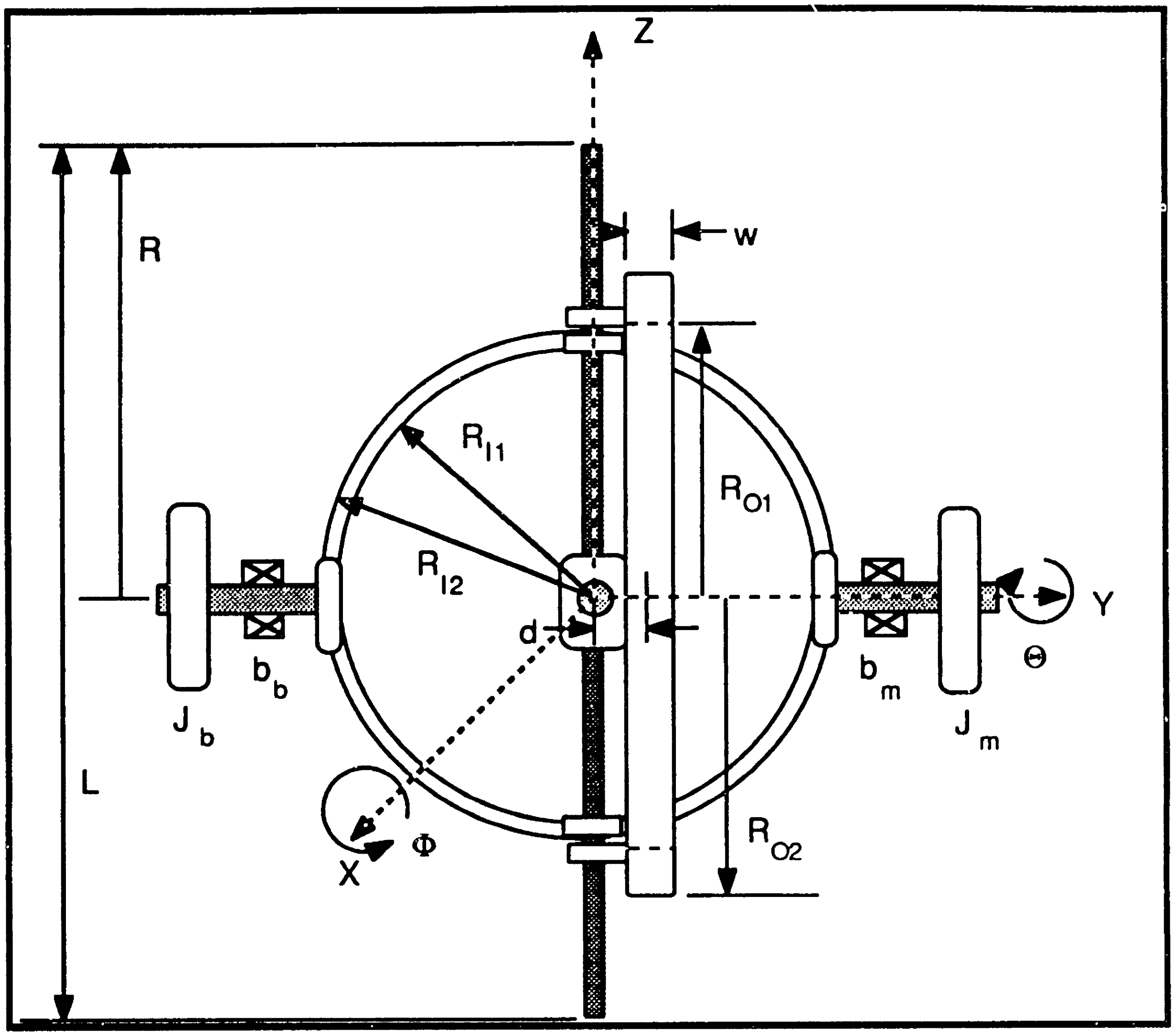


Figure 3.2: Joystick Model geometry

The inertia tensors for the gimbal bearings and the joystick shaft are found by applying Laplace's equation. The lumped parameter variable names and their meaning are given in table 1.

Table 3.1: System Parameters

J_S, m_S	:	Joystick Shaft Inertia, mass
J_{RI1}, m_{RI1}	:	Inner Bearing Inner Ring Inertia, mass
J_{RI2}, m_{RI2}	:	Inner Bearing Outer Ring Inertia, mass
J_{RO1}, m_{RO1}	:	Outer Bearing Inner Ring Inertia, mass
J_{RO2}, m_{RO2}	:	Outer Bearing Outer Ring Inertia, mass

Because kinetic energy is a scalar value, the contribution of each model parameter can be evaluated separately. For the shaft kinetic energy T_S one finds:

$$T_S = \frac{1}{2}J_S\left(\cos\Phi\frac{d\theta}{dt}\right)^2 + \frac{1}{2}J_S\left(\cos\theta\frac{d\Phi}{dt}\right)^2 + \frac{1}{2}\left(m_s + m_{LC}\right)\left(\frac{dR}{dt}\right)^2 \tag{3.23}$$

where m_{LC} is the included load cell mass. The shaft inertia is:

$$J_S = \frac{m_S L^2}{12} + m_S R^2 + m_{CL}\left(\frac{L}{2} - R\right)^2 \tag{3.24}$$

by the Parallel Axis Theorem and with R=0 at the center of the joystick shaft. The kinetic energy is found similarly for each of the gimbal rings. For the outer gimbal ring:

$$T_{BO} = \frac{1}{2}m_{RO1}R_{O1}^2\left(\frac{d\theta}{dt}\right)^2\cos^2\Phi + \frac{1}{2}J_{O1}\left(\frac{d\Phi}{dt}\right)^2 + \frac{1}{2}J_{O2}\left(\frac{d\Phi}{dt}\right)^2 \tag{3.25}$$

where again the individual inertias are:

$$J_{O1} = \frac{1}{2}m_{RO1}R_{O1}^2 + \frac{1}{3}w^2 m_{RO1}$$

$$J_{O2} = \frac{1}{2}m_{RO2}R_{O2}^2 + \frac{1}{3}w^2 m_{RO2}$$

Similarly for the inner gimbal ring:

$$T_{BI}=\frac{1}{2}m_{RI1}R_{I1}^2\left(\frac{d\Phi}{dt}\right)^2\cos^2\theta+\frac{1}{2}J_{I1}\left(\frac{d\theta}{dt}\right)^2+\frac{1}{2}J_{I2}\left(\frac{d\theta}{dt}\right)^2 \qquad (3.26)$$

and again the inertias:

$$J_{I1}=\frac{1}{2}m_{RI1}R_{I1}^2+\frac{1}{3}w^2m_{RI1}$$

and

$$J_{I2}=\frac{1}{2}m_{RI2}R_{I2}^2+\frac{1}{3}w^2\,in_{RI2}$$

The total kinetic energy can then be written in terms of the joint velocities and the joystick inertia matrix as:

$$T_{tot}=T_S+T_{BI}+T_{BO}=\frac{1}{2}\begin{bmatrix}\frac{d\theta}{dt} & \frac{d\Phi}{dt} & \frac{d\Psi}{dt}\end{bmatrix}\begin{bmatrix}H_{11} & 0 & 0\\ 0 & H_{22} & 0\\ 0 & 0 & H_{33}\end{bmatrix}\begin{bmatrix}\frac{d\theta}{dt}\\ \frac{d\Phi}{dt}\\ \frac{d\Psi}{dt}\end{bmatrix} \qquad (3.27)$$

where the diagonal terms are:

$$H_{11}=\left(J_S+m_sr^2\Psi^2+m_{CL}\left(\frac{L}{2}-r\Psi\right)^2\right)\cos^2\Phi+J_{I1}+J_{I2}+\frac{1}{2}m_{RO1}R_{O1}^2\cos^2\Phi$$

$$H_{22}=\left(J_S+m_sr^2\Psi^2+m_{CL}\left(\frac{L}{2}-r\Psi\right)^2\right)\cos^2\theta+J_{O1}+J_{O2}+\frac{1}{2}m_{RI1}R_{I1}^2\cos^2\theta$$

$$H_{33}=\left(m_S+m_{LC}\right)r^2$$

The inertia matrix is purely diagonal, a consequence of the uncoupled actuator coordinates. For the purpose of modelling the joystick plant dynamics, each of the gimbal axes can be considered a pure inertia, if any compliances are neglected. In the shaft motion case, the H_{33} inertia term is just the shaft and loadcell mass reflected through the motor drive shaft radius. However, as described below, the cable transmission cannot be modelled as a pure inertia because of the significant cable and gimbal ring compliance, which limit the stability of the closed loop friction compensation control. A model that describes the friction characteristics of the cable/sleeve interface and that incorporates the lumped transmission compliance is presented in the next section.

3.2.2 Cable Transmission Model

3.2.2.1 Friction Model

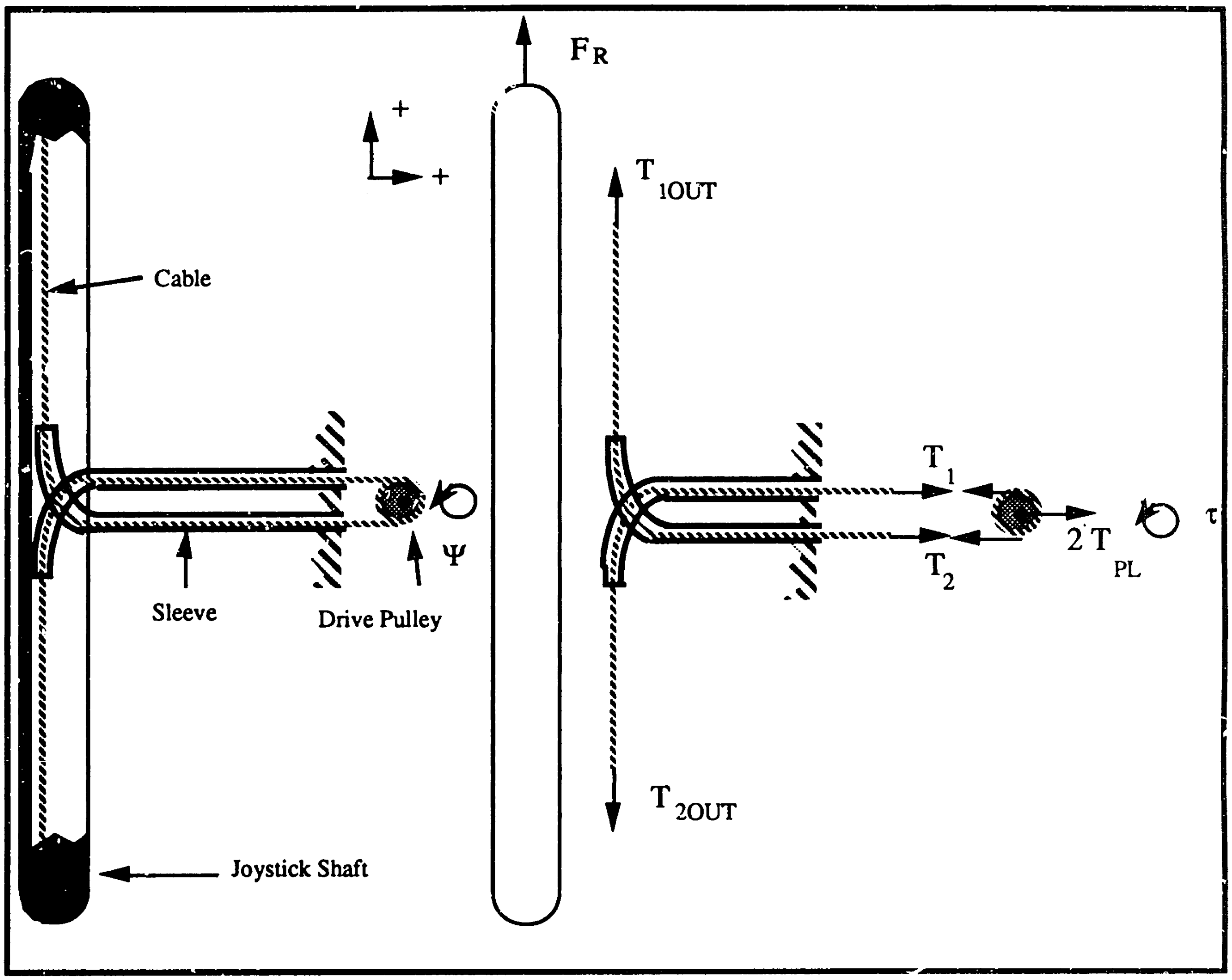


Figure 3.3: Cable transmission

The cable drive transmission, schematically shown in figure 3.3, transduces the linear shaft mass into an equivalent rotary inertia. T_1 and T_2 are the tensions in the cable on each side of the drive pulley, μ is the coefficient of friction between the cable and the drive pulley surface, β is the wrap angle of the cable around the pulley and T_{PL} is the cable pre-load tension.The force component along the shaft axis, F_R, is then simply given by the difference in the cable tensions:

$$F_R = T_2 - T_1 = \frac{\tau}{r} \tag{3.28}$$

where τ is the drive torque applied by the brake and motor, and r is the drive pulley radius. The maximum radial force along the joystick shaft axis that can be generated is determined by the capstan law, which gives the largest possible tension difference for zero slip. With a given pre-load, the ratio of tensions for zero slip is:

$$\frac{T_1}{T_2} = \frac{T_{PL} - \frac{\tau}{2r}}{T_{PL} + \frac{\tau}{2\,r}} = e^{\mu\beta} \tag{3.29}$$

so the maximum obtainable force is a direct function of the wrap angle. Equations (3.28) and (3.29) determine the necessary cable pre-load for a given coefficient of friction and wrap angle.

Because of the cable sleeve, frictional losses are introduced which cannot be neglected in the dynamics model. Information supplied by the cable manufacturer suggests that these losses depend only on the amount of bend in the sleeve. The loss factor LF is defined as the ratio between input and output cable force through a sleeve. The data given in figure 3.4 suggests that this loss factor is independent of the cable tension pre-load. Simple experiments show that this is an unrealistic assumption. Further, the frictional losses are due to stiction between the cable and sleeve walls. To model this behavior more realistically, the capstan law gives the static force needed to initiate movement and the manufacturer data is used for the dynamic frictional losses. With an angle α denoting the degrees of bend in the sleeve, and a coefficient of friction ν between the sleeve wall and cable, the ratio of input to output cable tensions for each sleeve are given by:

$$\frac{T_{1OUT}}{T_{PL} - \frac{\tau}{2\,r}} = e^{\nu\alpha} \tag{3.30}$$

and

$$\frac{T_{PL} + \frac{\tau}{2\ r}}{T_{2OUT}} = e^{\nu\alpha} \tag{3.31}$$

The optimal selection of a pre-load tension then maximizes the transmitted force but maintains the frictional losses at a minimum. The actual transmitted force F_R is:

$$\begin{aligned} F_R &= T_{2OUT} - T_{1OUT} = \frac{T_2}{e^{\nu\alpha}} - T_1 e^{\nu\alpha} \\ &= T_{PL}\left(\frac{1}{e^{\nu\alpha}} - e^{\nu\alpha}\right) + \frac{\tau}{2\ r}\left(\frac{1}{e^{\nu\alpha}} + e^{\nu\alpha}\right) \end{aligned} \tag{3.32}$$

If the pre-load Coulomb friction effects are neglected, the loss factor LF as defined by the cable manufacturer is then given by:

$$LF = \frac{F_R}{\frac{\tau}{2r}} = \left(\frac{1}{e^{\nu\alpha}} + e^{\nu\alpha}\right) \tag{3.33}$$

The comparison of the manufacturer supplied input load factor data[1] to the capstan law solution is given in figure 3.4.

When $\tau=0$, the Coulomb force is the force necessary to cause slippage in the cable sleeve, and therefore motion of the joystick shaft. Similarly, for zero output force and neglecting any dynamic effects, the necessary torque to cause motion is:

$$\tau_c = \frac{e^{\nu\alpha} - e^{-\nu\alpha}}{e^{-\nu\alpha} + e^{\nu\alpha}}\, 2\ r\ T_{PL} \tag{3.34}$$

this torque is fed forward in the friction compensation control loop described in the next section.

[1]from SAVA Industries, miniature and small cables catalog

The capstan model accurately describes the frictional force generated in the cable and sleeve and is therefore a useful relation to optimize the pre-load and number of drive pulley windings for the shaft cable drive.

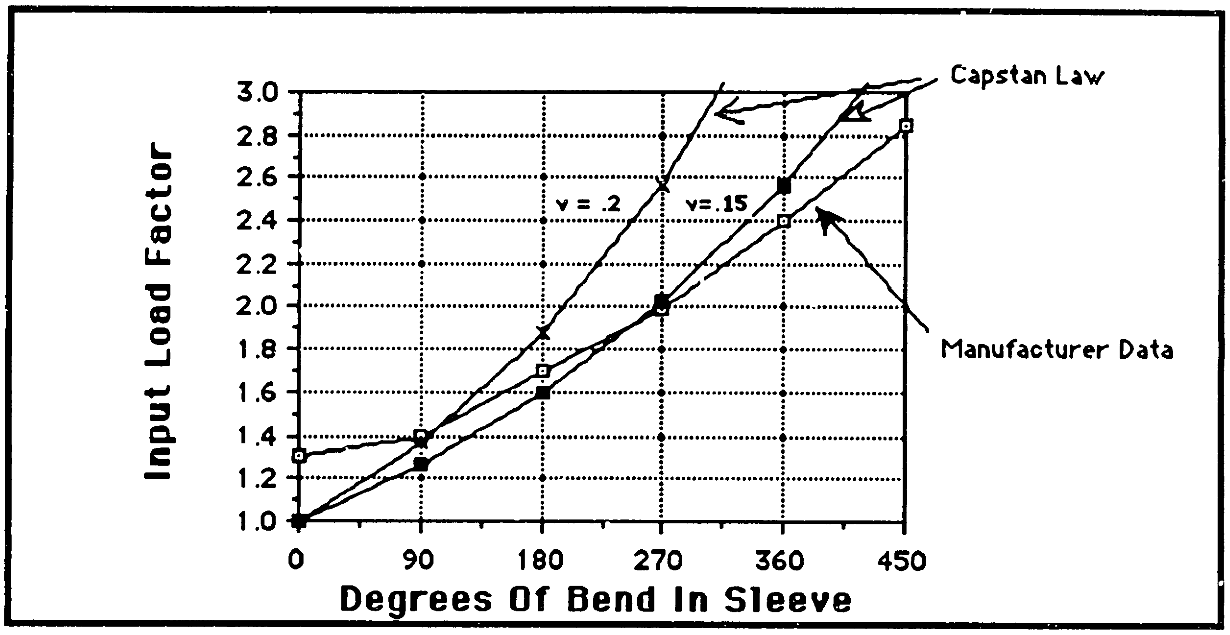


Figure 3.4: Input Load Factor Comparison

3.2.2.2 Transmission Dynamics

Since the R- coordinate value is not measured directly, but instead the motor drive shaft rotation Ψ, the shaft dynamics are related to this coordinate. The dominant dynamic terms, other than the stiction described above, will be the shaft inertia J_S and cable elasticity k. A force balance between the equivalent linear shaft mass and cable stiffness yields:

$$m_S \ddot{R} + k R = F_R \tag{3.35}$$

If the friction in the sleeve is neglected, the dynamics can be found for the measured Ψ coordinate by substituting $r\Psi$ for R and τ / r for the applied force F_R:

$$m_S\, r^2\, \ddot{\Psi} + k\, r^2\, \Psi = \tau \tag{3.36}$$

If Couloınb friction is not neglecteᵈ and F_d is the magnitude of the damping force with:

$$F_d = T_{PL}\,\left(\frac{1}{e^{\nu\alpha}} - e^{\nu\alpha}\right) \tag{3.37}$$

then equation (3.36) can be written in the form:

$$m_S\, r^2\, \ddot{\Psi} + F_d\, r\, \mathrm{sgn}(\dot{\Psi}) + k\, r^2 \Psi = \frac{\tau}{2\, r}\left(\frac{1}{e^{\nu\alpha}} + e^{\nu\alpha}\right) \tag{3.38}$$

where the symbol "sgn" denotes sign of and represents a function having the value of +1 if the argument is positive and -1 if the argument is negative. Equation (3.38) is nonlinear, but in the unforced $\tau = 0$ case, it can be separated into two linear equations :

$$m_s r^2\, \ddot{\Psi} + k\, r^2\, \Psi = -F_d\, r \quad \text{for } \dot{\Psi} > 0$$
$$m_s r^2\, \ddot{\Psi} + k\, r^2\, \Psi = F_d\, r \quad \text{for } \dot{\Psi} < 0 \tag{3.39}$$

The damping forces are passive in nature, and equation (3.39) represents forced vibrations. It follows that for Coulomb damping the decay is linear with time, as opposed to the exponential decay for viscous damping. The motion stops abruptly when the displacement at the end of a given half-cycle is not sufficiently large for the restoring force to overcome the static friction.

The solution of equation (3.39) can be found for one time interval at a time[1], depending on the sign of $\dot{\Psi}$. If the system is started from rest at an initial position Ψ_0 where the restoring force $kr^2\, \Psi_0$ is large enough to overcome the Coulomb friction, the solution for the first half cycle is:

---

[1] **sec also: Meirovitch, Leonard <u>Elements of Vibration Analysis,</u> Mc Graw Hill, pp31-33**

$$\Psi(t) = (\Psi_0 - \frac{F_d}{kr}) \cos \omega_n t + \frac{F_d}{kr} \quad \text{with } \omega_n^2 = \frac{k}{m_s} \tag{3.40}$$

which is valid until time t_1 when the velocity is once again zero. This time is found by differentiating the displacement solution:

$$\dot{\Psi}(t) = -\omega_n (\Psi_0 - \frac{F_d}{kr}) \sin \omega_n t \tag{3.41}$$

to be $t_1 = \frac{\pi}{\omega_n}$. At this time the displacement is:

$$\Psi(t_1) = -(\Psi_0 - 2\frac{F_d}{kr}) \tag{3.42}$$

A repeat solution of the above differential equation for $t > t_1$, now with $\dot{\Psi}(t)$ less than zero, and initial conditions $\Psi(t_1)$, and $\dot{\Psi}(t) = 0$, yields:

$$\Psi(t) = (\Psi_0 - 3\frac{F_d}{kr}) \cos \omega_n t - \frac{F_d}{kr} \tag{3.43}$$

This solution is valid until $t_2 = \frac{2\pi}{\omega_n}$, when the velocity $\dot{\Psi}(t_2)$ is zero once again. As can be seen from these solutions, the displacement consists of a linearly decaying oscillatory term and an alternating offset. The average value over a half cycle will alternate between $\frac{F_d}{kr}$ and $-\frac{F_d}{kr}$, and after each half cycle the displacement amplitude is reduced by $2\frac{F_d}{kr}$. This behavior will continue for n cycles until the static displacement magnitude is not enough to cause the spring restoring force to overcome the static friction:

$$|(\Psi_0 - n\frac{F_d}{kr})|\, k < F_d \tag{3.44}$$

after which the system remains at rest.

This analysis explains why simple proportional control cannot compensate for undesired Coulomb friction effects in position or force feedback control loops. If, for example, position is the variable of interest and is to be maintained at a reference value, a pure proportional feedback gain K_p will only cause a decrease in the position error to $\frac{F_d}{kr + K_p}$. For stability reasons discussed below, however, the proportional gain cannot be increased indefinitely, so the position error will be bounded by the critical gain that causes the system to be marginally stable. Since force control against a compliant environment is equivalent to position control, the same argument will hold true for force control. Integral feedback will cause instability sooner because of the added time delay introduced by integrator windup. A different force control approach that is described in the next chapter is implemented in the cable transmission for these reasons.

Chapter 4

Device Control

4.1 Force Control

The previous chapter developed in some detail the kinematic and dynamic characteristics of the joystick system. These are of principal importance in the control of the forces felt at the endpoint, which ultimately define the effectiveness of the joystick in force interactive tasks. This chapter will examine in more detail how a given endpoint force is achieved. It is assumed that the simulated environment information, i.e. the input to the joystick control system, is a stream of force commands. The more general problem of how to generate these forces given a particular environment model composition, are presented in the next chapter.

The advantage of uncoupled actuator axes that generate corresponding forces in the measured loadcell frame directions is that the measured forces can be used to directly close a loop around the actuator torques, without the need of any decoupling computation. The desired endpoint command forces in the environment cartesian frame are therefore first transformed into the loadcell frame. For small angular variations, the matrix relationship in equation (3.21) relating the input torques to measured forces can be approximated simply by:

$$\begin{aligned} \tau_\theta &= - r\,\Psi \cos\Phi\, F_{mx} \\ \tau_\Phi &= - r\,\Psi \cos\theta\, F_{my} \\ \tau_\Psi &= r\, F_{mz} \end{aligned} \tag{4.1}$$

Because the gimbal linkage has little inherent friction, it can be easily backdriven passively. The forces in the F_{mx} and F_{my} directions are therefore simply commanded open loop. Unfortunately, however, the large friction present in the cable drive has to be compensated for actively both for backdriveability and force control.

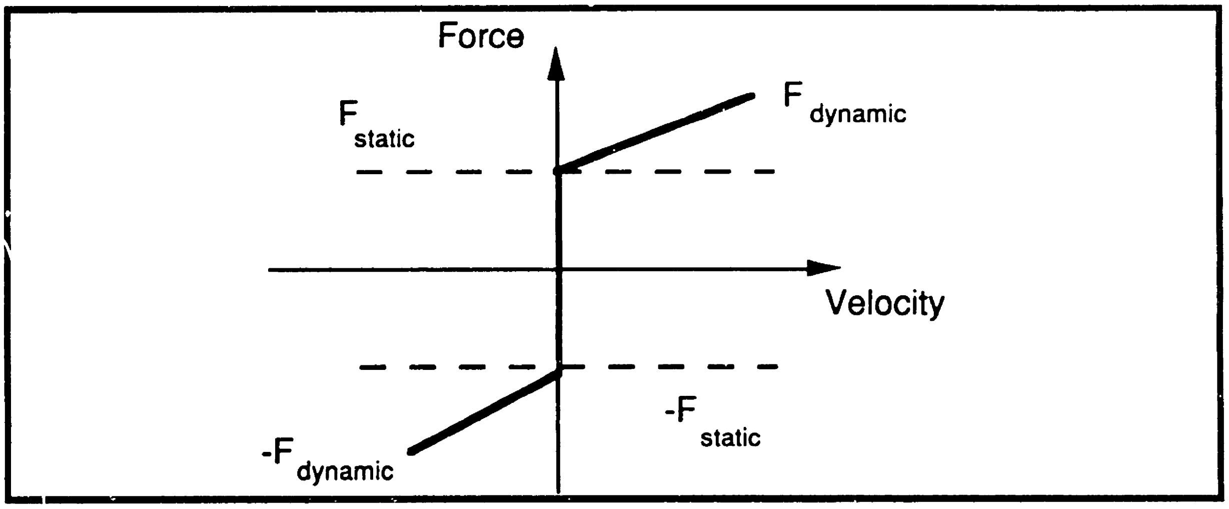


Figure 4.1: Transmission Friction Model

The friction model of the transmission includes a Coulomb friction static term and a viscous dynamic term as shown in figure 4.1. In the compensation control loop, the static friction level is compensated for through a feedforward term. A proportional feedback loop then eliminates any dynamic friction components. The static friction levels, including offsets due to gravitational effects, are updated on each joystick initialization. The motor torque command is ramped up until motion occurs. The command value at which motion is first sensed is then simply stored. When a small positive or negative force is sensed at the endpoint, this static friction value is inverted and sent to the shaft axis motor. The fed forward friction model also includes a small deadband to eliminate loadcell noise induced oscillations when the sensed force is zero. A feedback loop then sums to the feedforward command a term proportional to the error between the sensed and commanded signal. A complete block diagram of the control system is shown in figure 4.2. As a measure of backdriveability, the user effort necessary to move the joystick along the R-axis is plotted in figure 4.3 for the un- and compensated case. [Tadros].

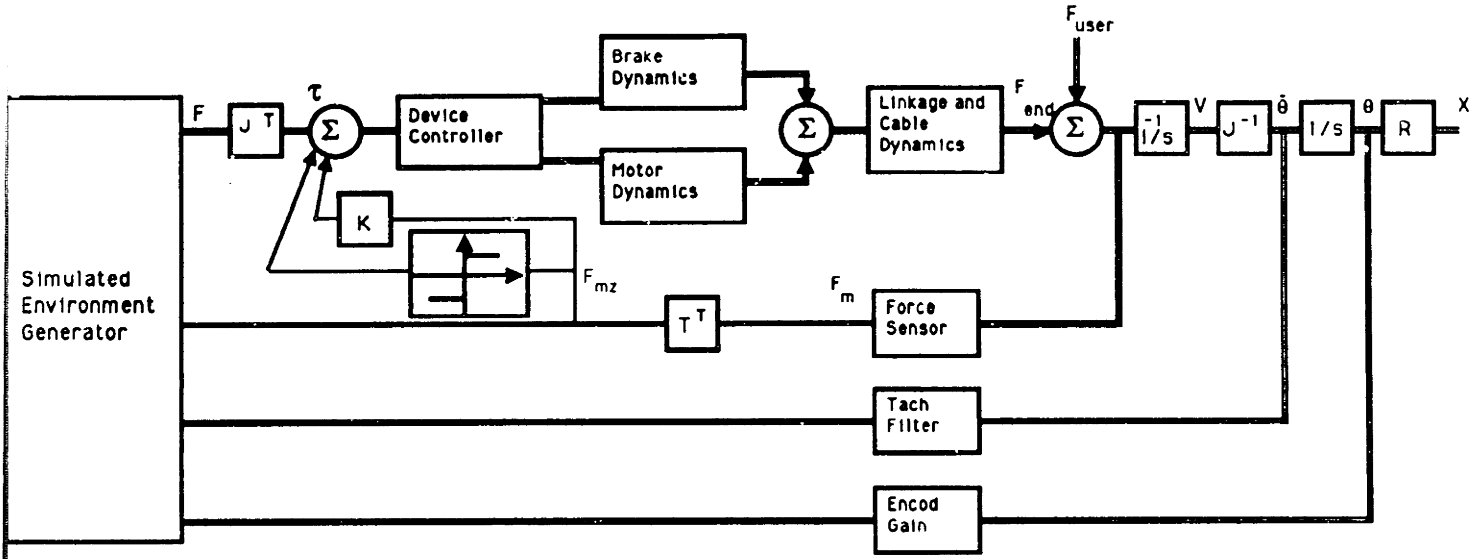


Figure 4.2: System Block Diagram

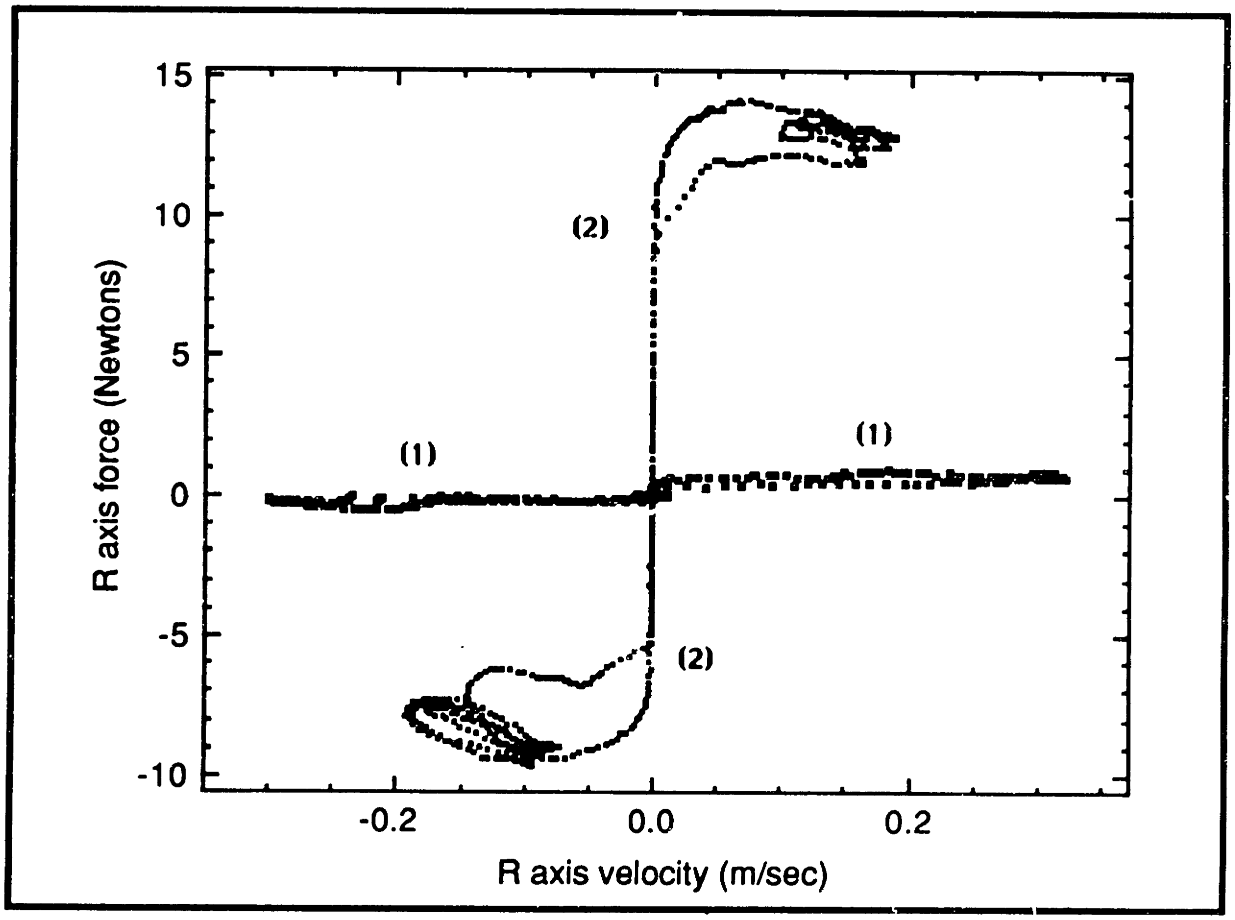


Figure 4.3 : Cable Transmission Friction. (1) Compensated (2) Uncompensated

Traditionally, in position tracking problems, the effects of Coulomb friction are removed by superimposing a high frequency dither signal to the command input. The dither breaks the static friction and causes the output position to track the input with less steady state error. This approach to friction compensation cannot be used in the joystick because the high frequency signal would appear as noise in the force simulation and degrade the overall performance. Further, the high frequency input signal can cause the force feedback transmission control loop to go unstable because of phase lag in the transmission dynamics introduced at that frequency.

4.2 Model Stability Issues

4.2.1 Closed Loop Force Control

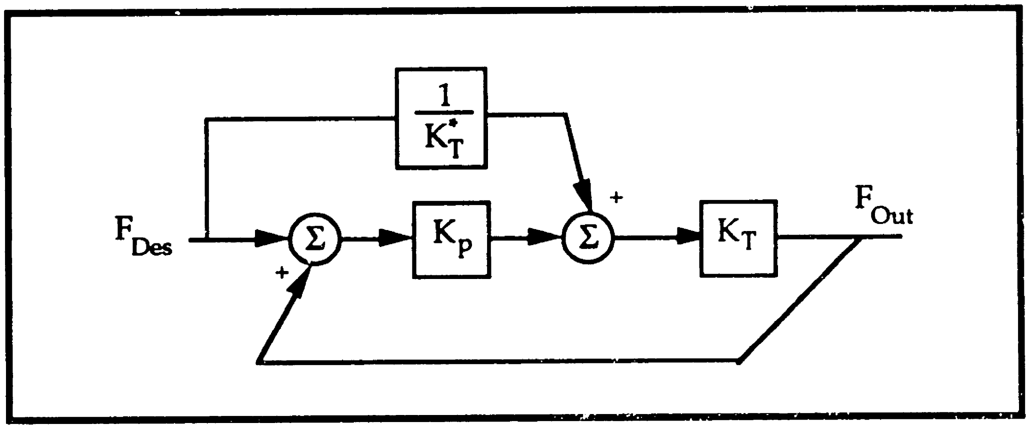


Figure 4.4: General Force Controller with Feedforward Term

The force felt by the user is the principle state of interest in force output devices. In the limit of a perfectly controlled system, the actual joystick dynamics should be completely masked by the controller, so that the felt impedance at the joystick endpoint is that required by the simulated environment. An accurate model of the true joystick dynamics is therefore essential. As a first approximation and in the absence of any transmission dynamics, the joystick plant can essentially be represented by a configuration dependent inertia as described in the previous section. If the endpoint force can be perfectly measured, that is the force sensor frequency response is essentially flat within the bandwidth of the mechanical joystick system, then a force control loop around a perfect transmission with a ratio of K_t will yield a zeroth order system. The block diagram in figure 4.4 shows a general force servo configuration with a feedforward term that includes an estimated transmission ratio K_t^* and a proportional force gain K_p. The resulting closed loop transfer function is:

$$F_{Out} = F_{Des} \frac{\frac{K_t}{K_t^*} + K_p K_t}{1 + K_p K_t} \tag{4.2}$$

As K_p approaches infinity, the errors due to imprecisions in the estimated transmission gain model tend to zero and the desired output force equals the input.

Neglecting any transmission and human hand dynamics, the coupled joystick - user system should remain stable for any force feedback gains. When, however, this force loop is closed on the actual joystick, an oscillatory instability appears on all three axes at a certain stiffness between the joystick endpoint and the contact surface, normally the user's hand. By stiffening the wrist and arm joints while grasping the endpoint, the instability is excited and occurs at 35 Hz for the cable transmission and 60 Hz for the two gimbal axes. Decreasing the force gain just changes the onset of the instability as the stiffness is increased. Is this instability caused by the added dynamics of the human arm alone ?

4.2.2 Cable Transmission Instability

The control problem is essentially how to match the endpoint joystick impedance to that of the simulated environment while the joystick plant is coupled to the highly varying impedance of the human arm system. Recent work has shown that human arm stiffness can vary from 2 N/m to 800 N/m, the wrist mass from .2 kg for tangential motion to 2 kg for radial motion, and a joint viscosity of 3 Nsec/m for tangential and 15 Nsec/m for radial motion [Hogan]. The joystick dynamics must remain stable for this wide range of impedance setpoints.

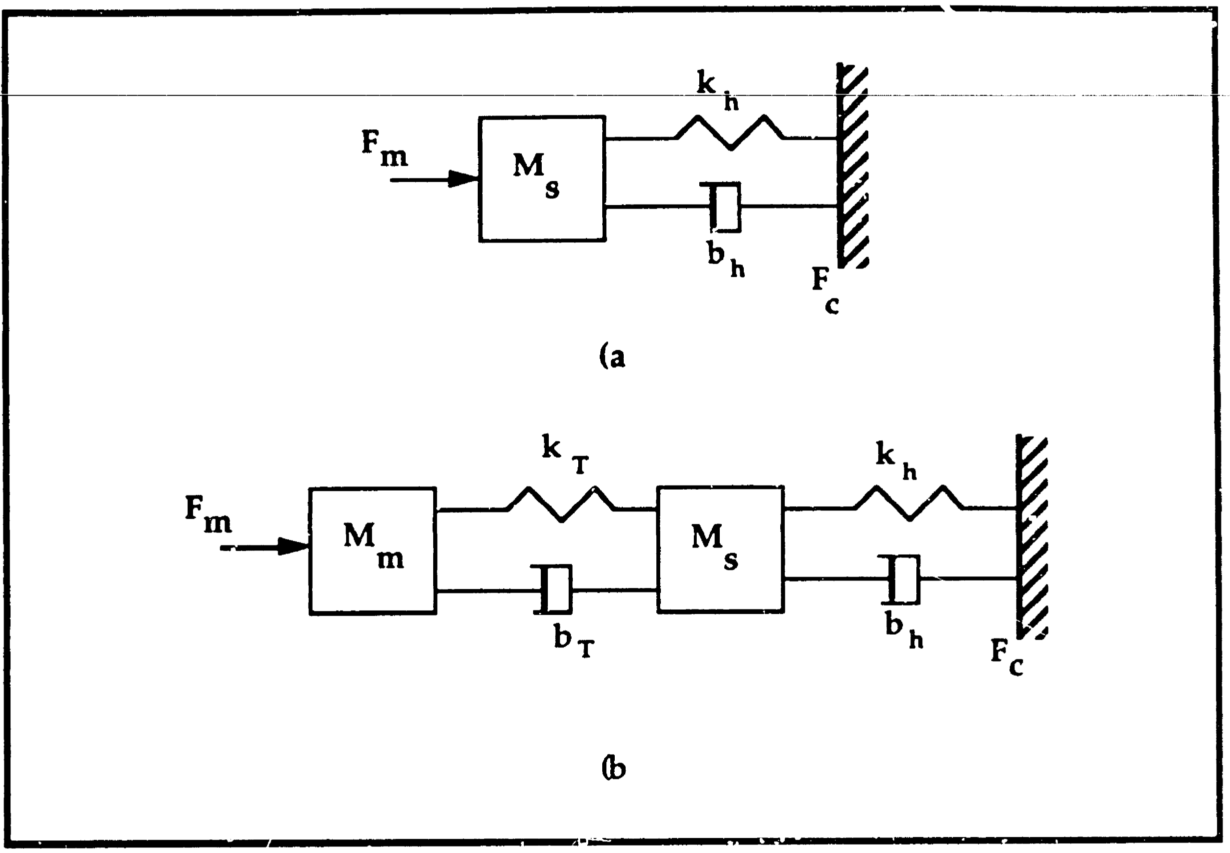


Figure 4.5: Transmission and User Hand Lumped Parameter Model: a) Transmission as Simple Inertia, b) Including Transmission Stiffness and Damping

The hand stiffness and damping is included in the lumped parameter model shown in figure 4.5 that shows the dynamically interacting joystick and user hand system. From the open loop transfer function and associated root locus in figure 4.6:

$$\frac{F_c}{F_m} = \frac{k_h + s\, b_h}{M_s\, s^2 + b_h\, s + k_h} \tag{4.3}$$

it is clear that this model cannot exhibit the stability characteristics observed, and should again not display unstable behavior.

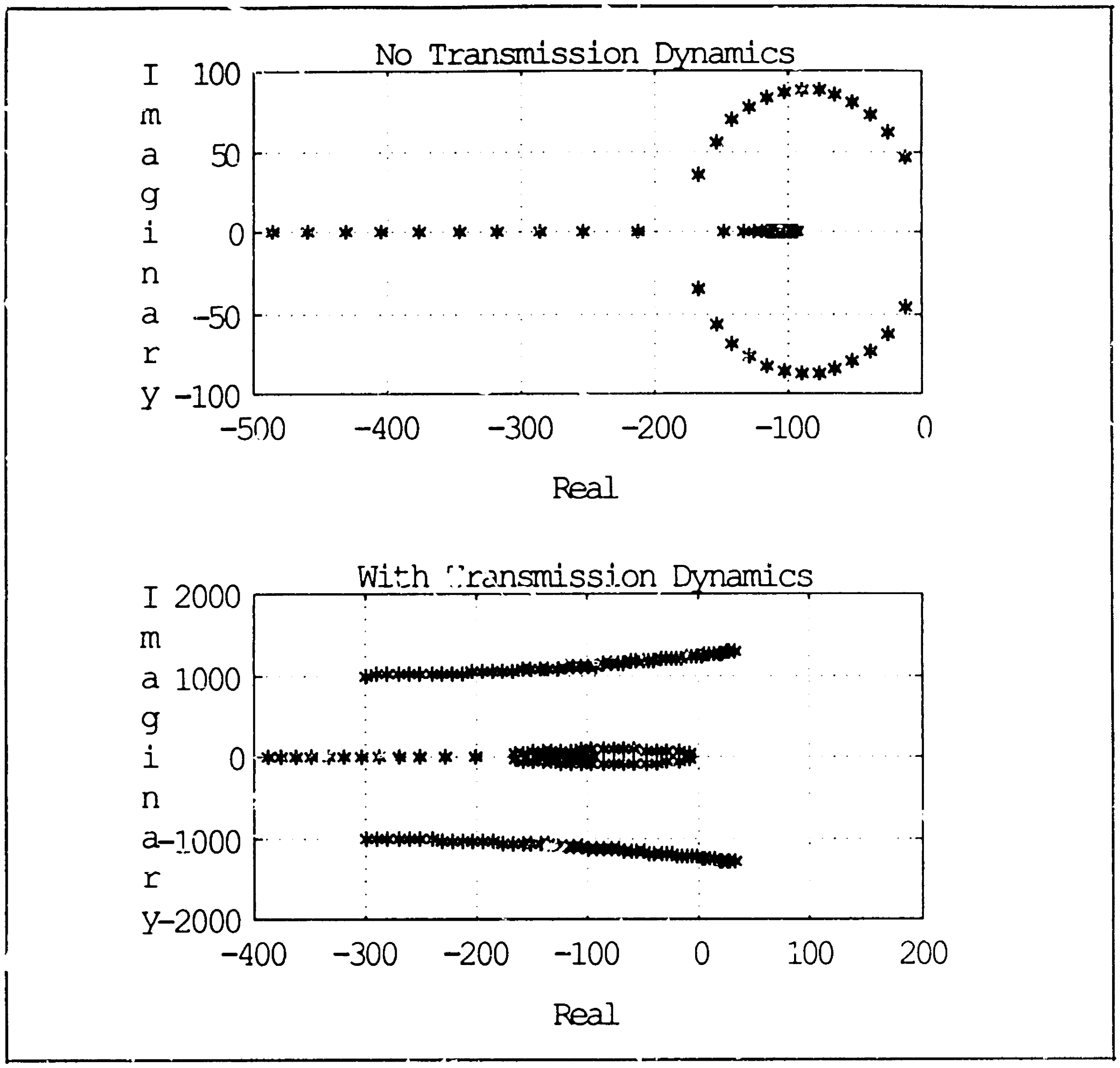


Figure 4.6: Model Root Loci

If, however, some force sensor and transmission compliance $1/k_t$ is introduced into the model, with viscous damping b_t to account for dissipative effects through the transmission, the associated closed loop transfer function can exhibit unstable behavior for certain gains as shown from the root locus in figure 4.6 of the transfer function:

$$\frac{F_c}{F_m} = \frac{b_h b_t s^2 + (b_t k_h + b_h k_t) s + k_h k_t}{M_m M_s s^4 + (b_t M_s + b_h M_m + b_t M_m) s^3 + (k_t M_s + b_h b_t + k_t M_m + k_h M_m) s^2 + (b_h k_t + b_t k_h) s + k_h k_t} \tag{4.4}$$

The motor and brake inertias are reflected through the transmission ratio to an equivalent lumped mass M_m. For the friction level in the transmission, increasing torque commands were sent to the servomotors and the output shaft velocity was measured for each torque input value. After Coulomb friction effects and a gravity offset were subtracted, the found viscous damping component was 45 Nsec/m. If the cable is assumed to always be in a tension state, this is thought to be the primary contribution to the transmission compliance. With a stiffness value from the suppliers of 35280 N/(% elongation), the actual length of cable used translates to a transmission stiffness of k_t = 82340 N/m. These values together with a hand contact damping of 8 Nsec/m and stiffness of 700 N/m, give a numerical transfer function whose root locus is seen in figure 4.6. It is clear that this feedback system will exhibit unstable oscillatory behavior for certain gain values. The critical gain depends primarily on the transmission and contact stiffnesses. If one substitutes $s=j\omega$ where $j=\sqrt{-1}$, then the open loop bandwidth is found by solving for the real roots of the characteristic equation. The smaller of the two roots is the first breakpoint frequency and will yield the open loop system bandwidth:

$$\omega_{1/2} = \sqrt{\frac{k_t M_s + b_h b_t + M_m(k_t + k_h)}{2M_m M_s} +/- \sqrt{\left(\frac{k_t M_s + b_h b_t + M_m(k_t + k_h)}{2M_m M_s}\right)^2 - \frac{k_h k_t}{M_m M_s}}} \tag{4.5}$$

For the parameter values that generated the above root loci, the smaller of the two roots is ω_1 = 41 rad/sec, which corresponds to the frequency of the inner, smaller complex pair. However, it is clear from the root locus that in the closed loop case the lower frequency mode is always stable, it is the higher transmission dynamics that lead to unstable behavior for higher gains. The root locus predicts that the instability frequency should occur at ω_{crit} = 1200 rad/sec, or 190 Hz. However, figure 4.7 shows the frequency spectrum of the instability, and the cable axis instability frequency is appreciably lower at 35 Hz. This is because the primary compliance is not in the pretensioned cable, but due to a sleeve and gimbal ring flexing. Further, because of the loadcell filter cutoff frequency is at 130 Hz, the added lag causes a lower instability frequency than predicted by the model. As a recommendation for future design changes, the gimbal ring should be supported more firmly and any added filter lags in the feedback loop should be avoided to raise the instability frequency and increase the closed loop force bandwidth.

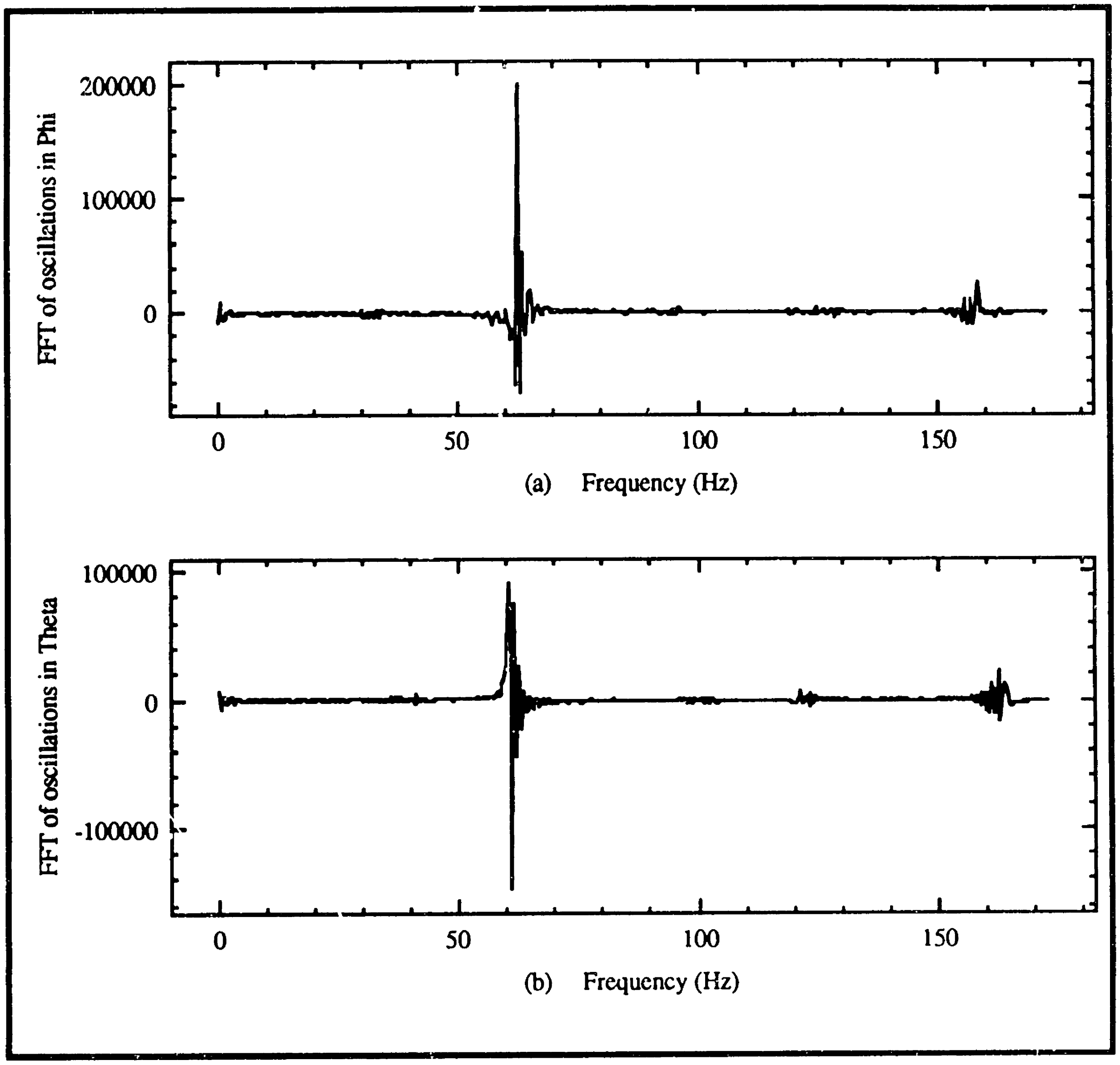


Figure 4.7 : Frequency Spectra of Contact Instability for Three Joint Axes : a) Φ axis, b) Θ axis

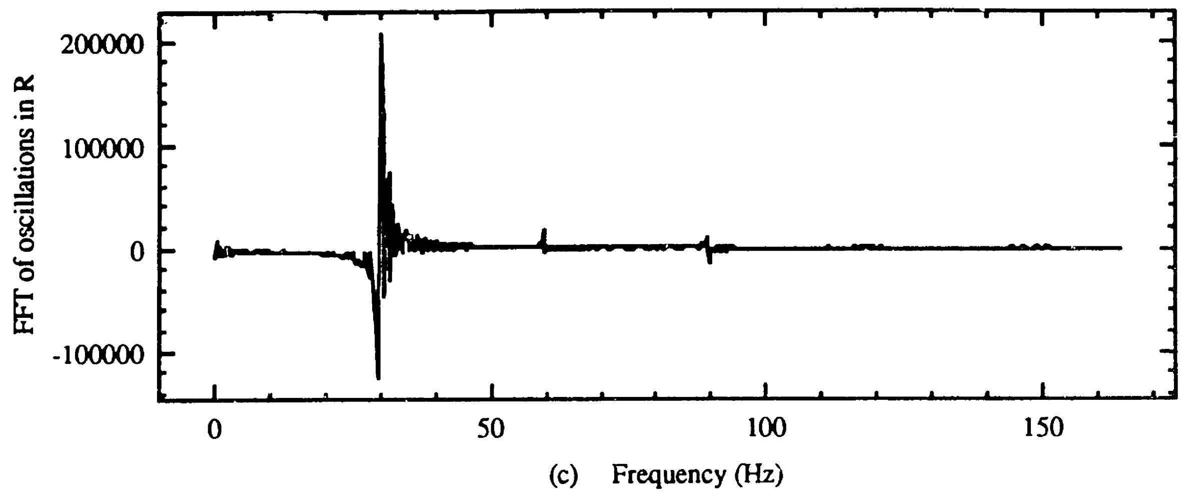


Figure 4.7 : Frequency Spectra of Contact Instability for Three Joint Axes ; c) R axis

This stability analysis does not consider any of the dry friction nonlinearity in the cable transmission. In the actual transmission, the dry friction that is present is Coulomb friction, no difference in static and dynamic friction levels were measured that would indicate stiction. Coulomb friction causes the closed loop force control system to enter into a limit cycle for feedback gains that are unstable in the linear lumped parameter model. The stability bounds are actually increased by Coulomb friction [Townsend], so the linear model analysis yields a conservative critical gain. In the transmission force control feedback system, the actual value of the static friction is measured and its inverse is fed forward as described in the previous section.

4.3 Hybrid Motor/Brake Control

4.3.1 Open Loop Brake Control

As described in the actuator section 2.2.3, the magnetic particle brakes are current controlled resistive torque sources. In the open-loop case, i.e. if a constant current is sent to the brakes, they will clamp at a proportional, constant torque. Therefore, to cause any brake shaft rotation, a torque must be applied to the brake shaft to overcome the brake clamping torque. The appropriate model for the open-loop brake characteristics is a Coulomb friction element, with a static torque value τ_b proportional to the input current i_b, as shown in figure 4.8.

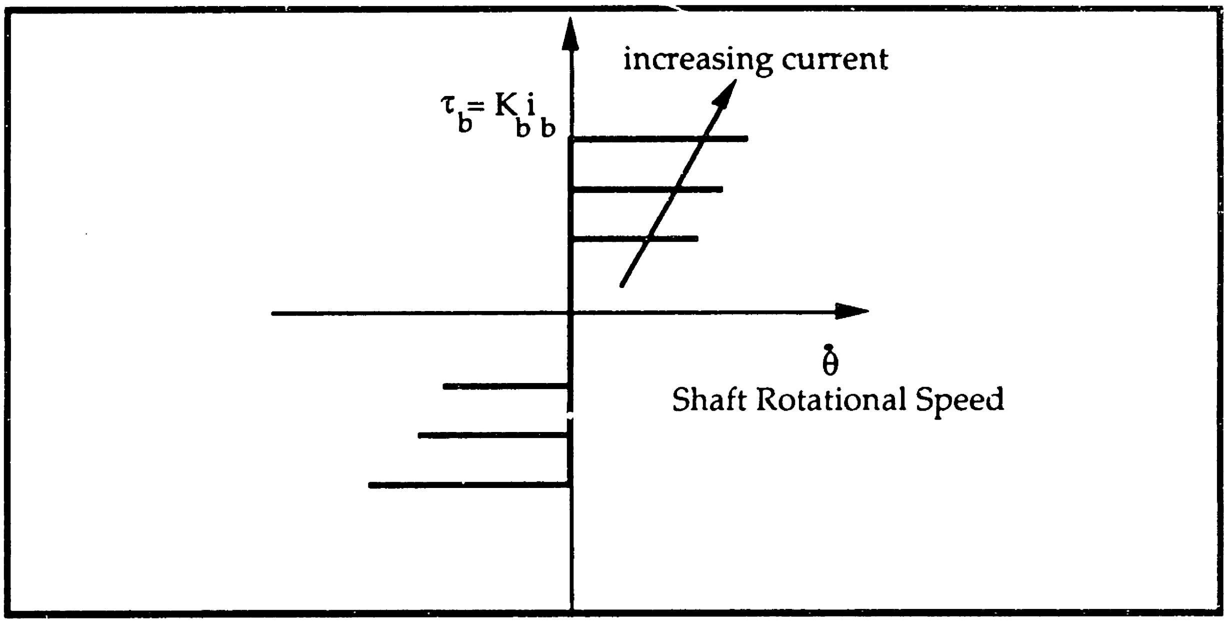


Figure 4.8: Open Loop Brake Torque Characteristics

There are a variety of interesting force effects that can be generated by the brakes in an open-loop configuration. For simplicity's sake the analysis is limited to two dimensions, say the XY plane as viewed from the top of the gimbal linkage. If the brakes are set at a given torque value, a square loading area centered at the joystick endpoint is maintained a shown in figure 4.9. This set of vectors represents all resistive load thresholds that have to be overcome to initiate joystick movement. The loads can be modulated freely by varying the brake command signals. If one of the brake torques is exceeded, then rotation of that brake's shaft will occur and cause motion in only one direction as indicated in the figure.

Any inelastic, dissipative material can be simulated effectively. If, for example, a clay-like material is simulated, the brakes can provide the resistive forces that model the clay deformation. If the material exhibits any restoring force at all, this can be maintained by the motors alone.

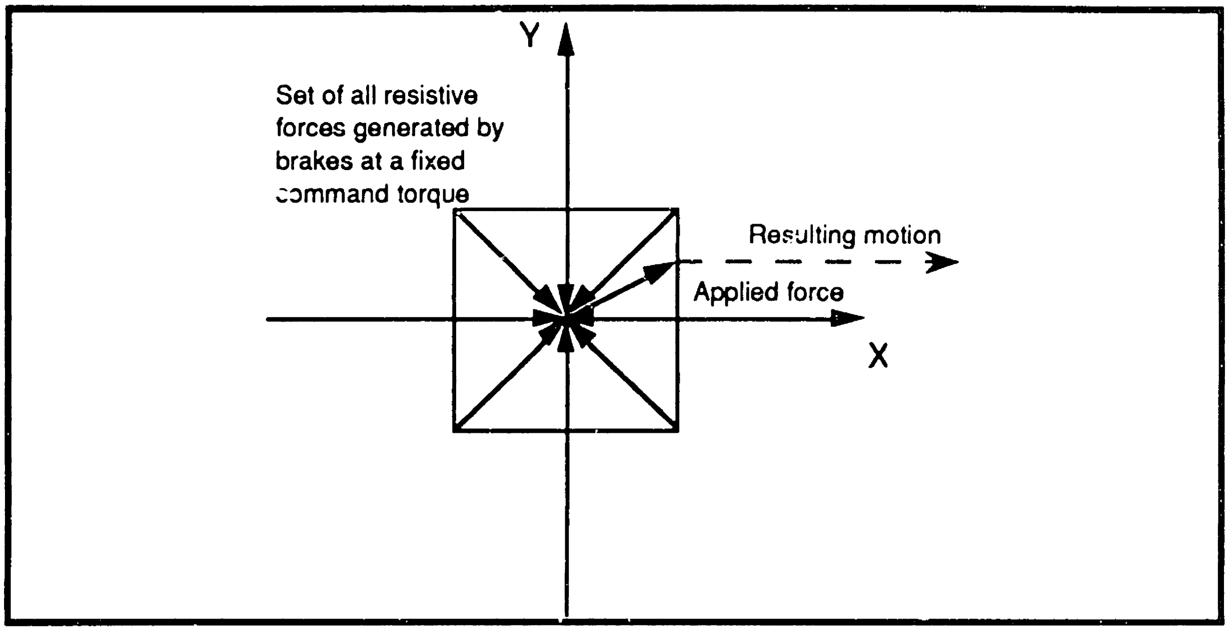

Y
Set of all resistive forces generated by brakes at a fixed command torque
Resulting motion
Applied force
X

Figure 4.9: Brake resistive force loads:

However, the direction of the force applied by the user at the endpoint has to be an input to the brake control algorithm. For example, a wall is simulated along one of the brake axes. If the joystick and wall locations are in coincidence, the brake must be turned on proportional to the applied force as long as the user pushes in a direction towards the wall in order to simulate a pure normal force. As soon as the user decides to apply a force away from the wall, the brake should deactivate to not constrain this direction of motion. What happens, however, if the constraint is not aligned with the brake axes, but at some angle ?

For constrained motion tasks, the brakes cannot be the only actuators. If the constraint is uni-directional, for example in a frictionless wall contact, then the force component along the wall must be zero while perpendicular to the wall the normal force is active. Because of the Coulomb friction behavior of the brakes, the generated constraint forces at the endpoint of the joystick have components in all directions, as discussed above. The motors then have to actuate against the brakes to add an opposite force component perpendicular to the constraint where zero resistance to motion should be felt. But, two actuators cannot control more than two degrees of freedom, so any motor force that would cancel a brake-applied force in a fixed direction will cancel all brake effects. The promise of the brakes is to modulate them to dissipate large energy amounts, for example in an impact situation, and then fold over the steady state control to the servo motors.

The issue at hand is how to control the magnetic particle brakes together with the servomotors to generate a force output at the joystick endpoint. First, the basic physical elements that can or cannot be simulated with the magnetic particle brakes are considered. The feedback of various measured brake shaft states may generate dynamic behaviors that model the principle linear elements: a mass, damper, and spring. However, since the brakes are purely dissipative devices, no energy storing elements can be simulated. The power input P_{in} to the brakes must therefore always be positive.

$$P_{in} = \tau_b \dot{\theta} > 0 \tag{4.6}$$

The simplest element to simulate is a viscous damper. If a velocity loop is closed around the brake, so that the relationship between brake current command and brake shaft velocity $\dot{\theta}$ is a constant K_v, then the effective viscous damping b_b is:

$$b_b = K_b K_v$$

$$\text{since } \tau_b = K_b K_v \dot{\theta}. \tag{4.7}$$

Because all three brakes are uncoupled and independently controlled, the effective viscous damping at the endpoint does not have to be isotropic, but can be modulated to generate an endpoint damping ellipsoid, with principle directions aligned along the uncoupled loadcell frame axes.

If instead the shaft position is fed back, then the brake output will match that of a torsional spring under certain conditions. As stated above, the power to the brake must always be positive. The output will match that of a torsional spring only in the regions where work is done on the spring, i.e. the power flow is into the element. The effective "spring constant" k_b for a position feedback gain K_p can then be expressed by:

$$k_b = K_b K_p \operatorname{sgn}(\dot{\theta})$$

$$\text{so that } \tau_b = K_b K_p \theta \operatorname{sgn}(\dot{\theta}). \tag{4.8}$$

Finally, if brake shaft acceleration is used as the fed back state, then an inertial element is simulated as long as the power flow is into the element. The torque relation now is:

$$\tau_b = K_b Ka \ddot{\theta} \operatorname{sgn}(\dot{\theta}) \tag{4.9}$$

where K_a is the acceleration feedback gain. Figure 4.10 summarizes the result for all the state feedback cases. Assuming all gains are unity, the figure is a plot of shaft torque that must be applied to the brake in order to generate a pure sinusoidal shaft motion. The shaded areas represent regions of positive power and so correspond to the brake torque output τ_b. Only in the viscous damper case is the brake torque a pure sinusoid because energy is always dissipated. The discontinuities in the position or acceleration feedback situations are also caused by the positive power requirement. The 90 degree phase shift between position or acceleration and velocity introduce the discontinuous jump in brake torque. A simple hybrid motor/brake control strategy monitors the power requirements of the simulation and activates the brakes whenever a dissipative or energy storage element is simulated with a positive power, that is work is done on the element. The release of energy, or negative power flow, then is accomplished by the motor torque commands, according to the constitutive equation of the modelled element.

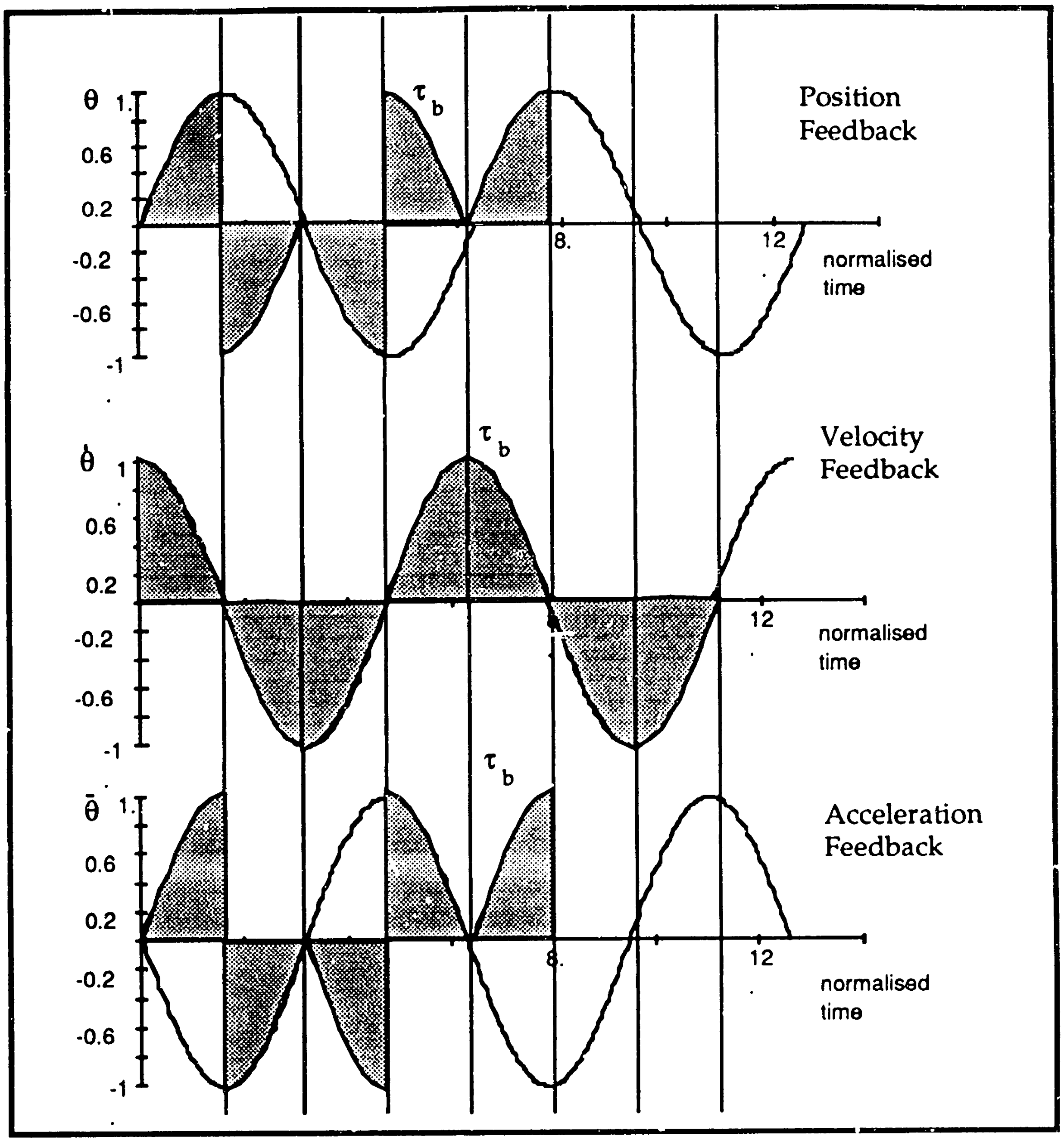


Figure 4.10: Brake Shaft Torque Input for Sinusoidal Position Output Under Various State Feedback

4.4 Joystick System Evaluation

The most difficult part in measuring a particular force output device's performance is how to quantify the notion of what "feels good". The design presented in this thesis approached the problem by simply aiming for the highest achievable bandwidth and signal to noise ratio. [Jex], however, suggests some rules-of-thumb for good force feedback simulation. The vernacular version of these benchmarks are that a good simulation of a:

- pure inertia should "feel like a stick of balsa wood"
- hard stop should "feel like a brick wall"
- coulomb friction should "feel like sliding a refrigerator magnet"
- centering detente should " yield an audible 'klunk' when traversed"

The results presented in this section are characteristic force plots measured by the sensor located at the joystick endpoint. The lever arm from the joystick endpoint to the motor/brake drive shaft is 20 cm. Equivalent torques at the actuators are then simply found by multiplying the measured force values by this lever arm.

Of these criteria, the inertia simulation is the most difficult to achieve. The need for a clean acceleration signal that is fed back to the motor actuators is a difficult requirement because of the large noise levels associated with such a measurement. If the plant dynamics are a pure inertia, then the force signal measured at the endpoint is a signal proportional to the acceleration, and can be used to reduce or increase system inertia. A 50% reduction in apparent inertia is theoretically possible before instability occurs [Colgate]. However, because o. the low inertial levels involved in the joystick mechanism compared to viscous damping terms, it is difficult to measure a reduction in the inertia. Force feedback increases or reduces all resistive forces, and it is difficult to separate the inertial terms alone. An accelerometer would aia in inertia control since the inertial force terms are directly proportional to the acceleration alone. Such a sensor was however not included in the design, and a digital differentiation of the measured velocity signals was not attempted.

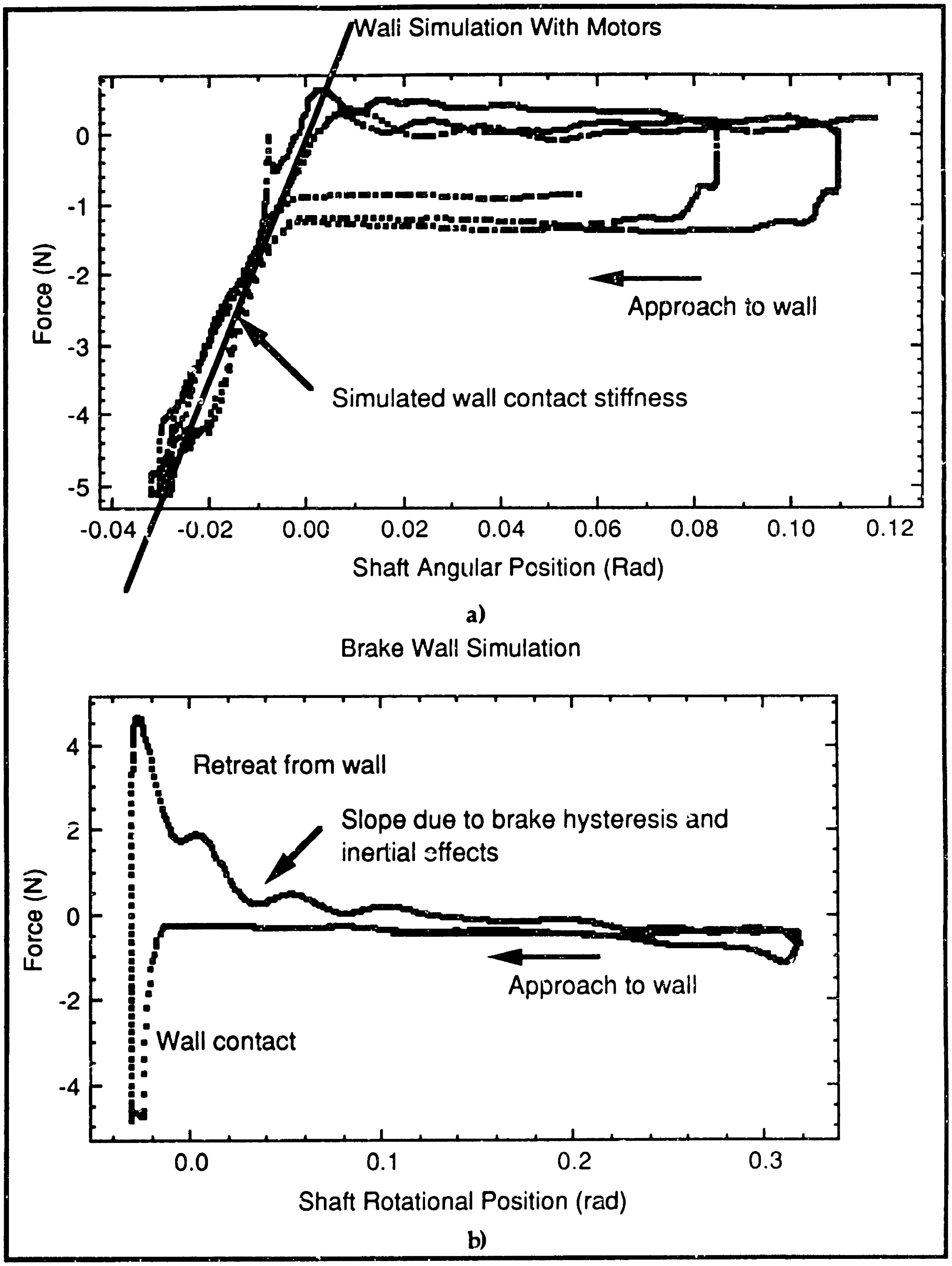


Figure 4.11: Wall simulation using a) motors, b) brakes

The "brick wall" effect is simulated well in the present system through the use of both the motors and magnetic particle brakes. Both cases are given in figure 4.11. In the brake wall simulation, the brakes are commanded to the maximum torque value when the joystick endpoint is at the wall position <u>and</u> the user is pushing into the wall direction. The instance the user applies a zero or positive force, the brakes are commanded to zero torque and the joystick endpoint is free to move. The positive force in the graph corresponds to the force necessary to overcome the torque hysteresis in the brake due to residual magnetic effects. The motor simulation models the wall as a high stiffness element and the effective wall stiffness is clear from the characteristic force/position plot. To best simulate a wall, however, a hybrid control scheme is necessary that uses the brakes for the large contact forces and then employs the motor to overcome the residual magnetic effects when the user wants to move in a direction away from the wall. Such a simulation would combine the desirable effects of each actuator and yield a force characteristic closer to the ideal model.

Because of the open loop torque characteristic of the brakes, Coulomb friction is trivial to simulate. A torque command to the brakes will set the effective coefficient of friction as described in the brake model above. Figure 4.12 is a characteristic plot of the open loop brake behavior and closely matches that of a Coulomb element.

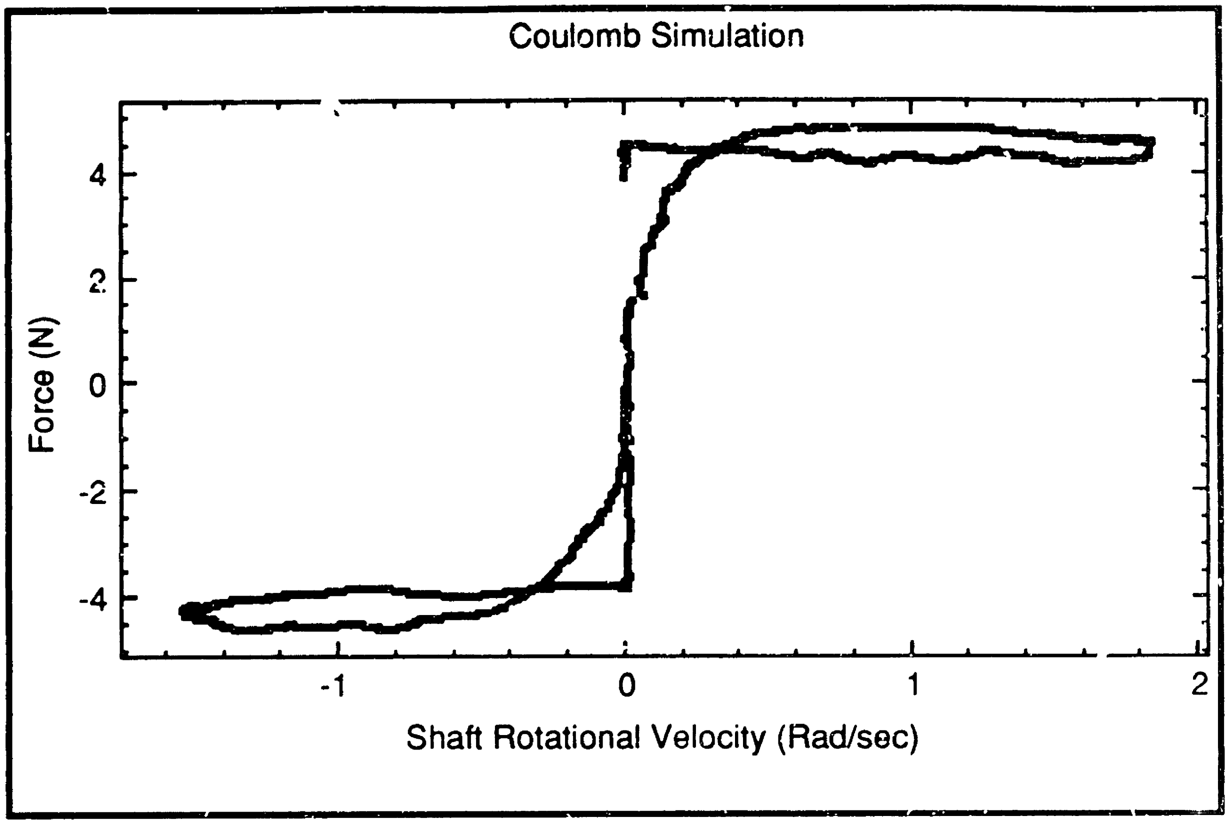


Figure 4.12: Coulomb friction element model

The detente effect is easily modelled as a strong potential well with a repulsive force around the periphery, and can be simply modelled with two spring elements. A characteristic spring behavior is modelled with the motors alone in figure 4.13. The hysteresis loop is due to slight Coulomb friction in the gimbal.

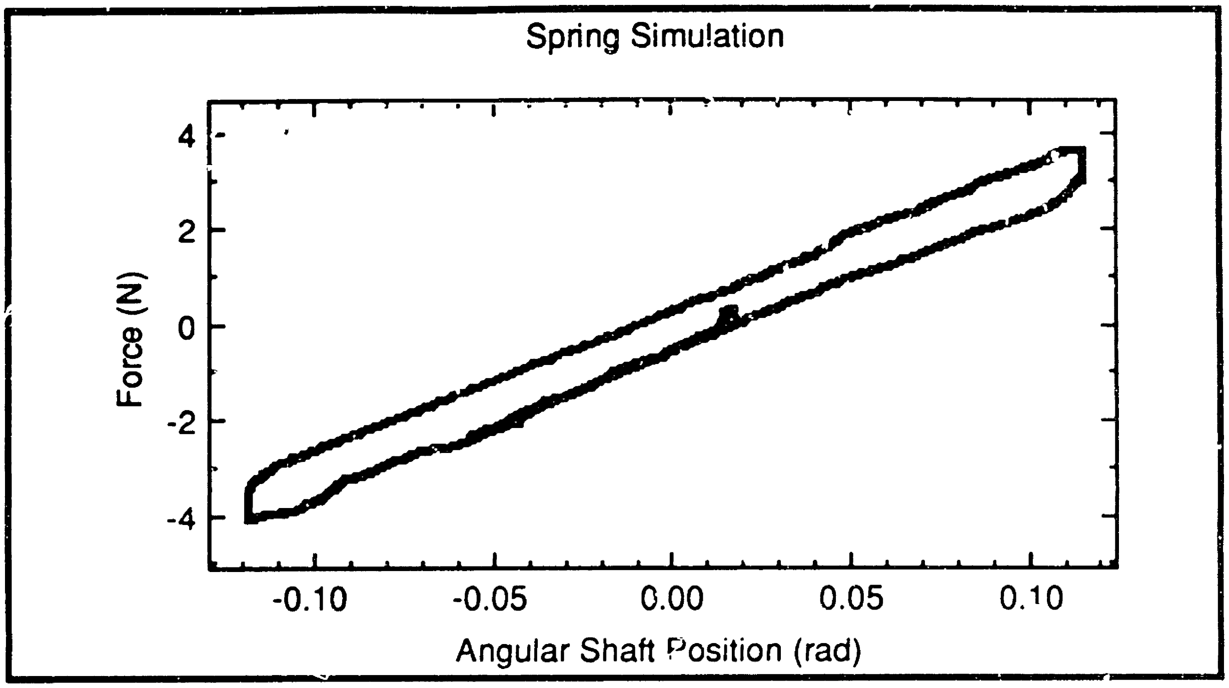


Figure 4.13: Spring element model

Finally, a comparison of the viscous damping simulation characteristic curves is given in figure 4.14 for the motors and brakes as actuators. Because of the higher torque output of the brakes, a larger damping coefficient is achieved in the brake simulation. The inflection point at zero velocity is a direct measure of the Coulomb friction present in the system and lies in the same one Newton range that appears in the spring simulation hysteresis loop above.

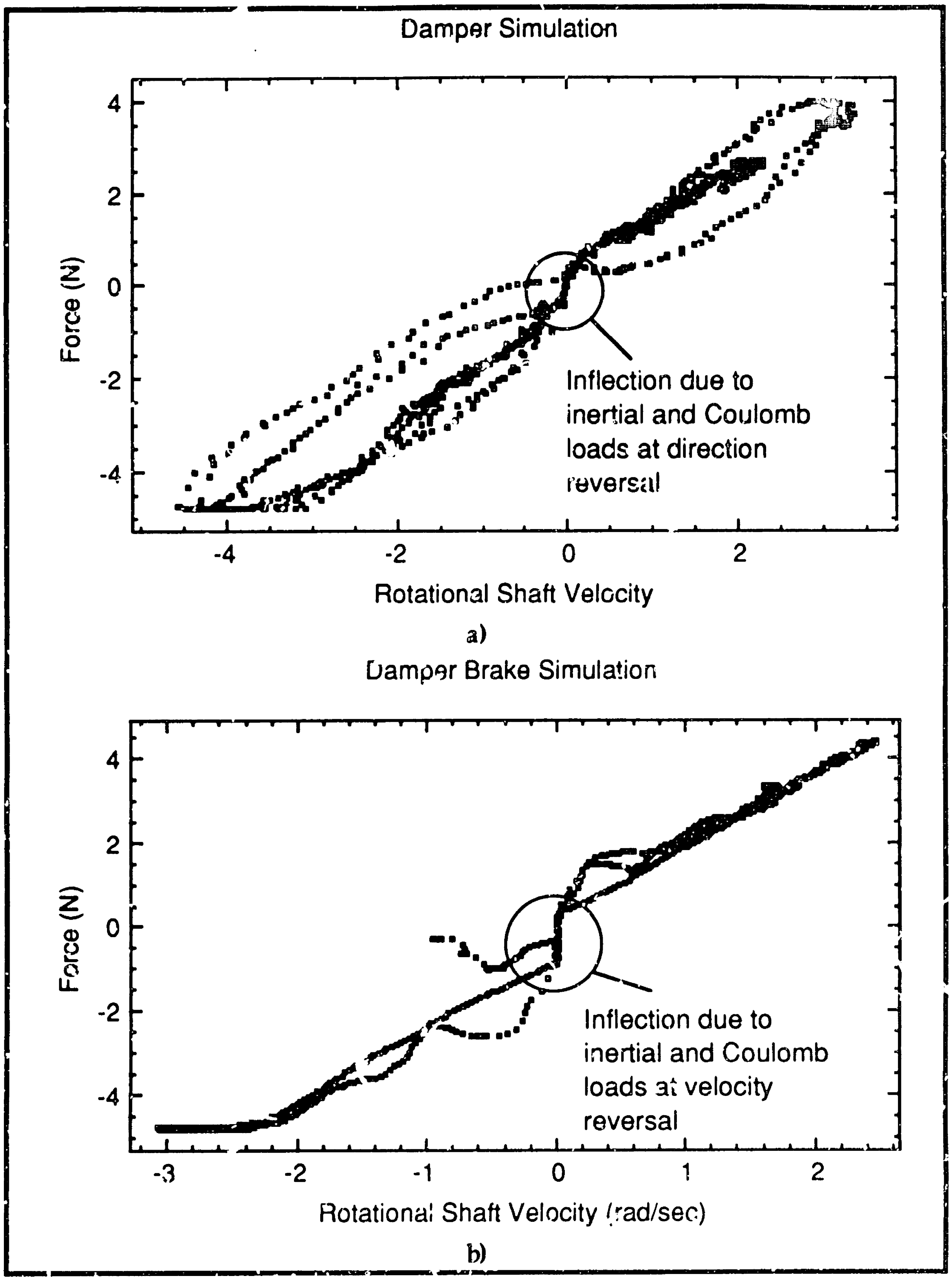


Figure 4.14: Viscous damping simulation: a) motors, b) brakes

Chapter 5

Simulated Force Generation

The essential difference between a graphic visual display and force output is that the graphics calculations do not have to occur in real time for the user to understand the sequence of images. These can be pre-computed and then played back, or they can operate at significant delays and only test the patience of the user In force output, however, the joystick control operates in real time, and computation delays manifest themselves as mechanical instabilities. The main cause of time delays, the graphic visual display calculations, have to be separated from the physical environment model. Rather than compute the complete simulation on the environment host computer and then send force commands to a control routine on the joystick processor, a compact representation of the environment dynamics is updated on the joystick processor which is forced to run in real time (loop rates greater than 120 Hz). The graphics display and simulated environment model are implemented on the environment host computer and do not necessarily run in real time. The environment model controller is a real time algorithm that models the interaction dynamics and resides in the joystick control processor. The joystick controller is the collection of transformations and servoloop codes that take as an input a desired joystick endpoint force and command the actuators accordingly. Two possible approaches to representing object and surface characteristics are suggested in this chapter.

A distinction is made between static and dynamic objects. Static objects are considered as the environment background, they do not move or change interactively in a simulation. For example, static objects would be the virtual workspace boundaries and the rigid objects attached to ground. The characteristics of these constraints are known and constant. The force output information can be stored in the real-time environment simulation for each point on the static objects before the actual dynamic simulation begins. Dynamically varying object boundaries and characteristics are much more difficult to implement and are of primary interest for the remainder of this chapter.

5.1 Force Information Coding Strategies

The control of the joystick plant dynamics to achieve a particular endpoint force is well understood and implemented, as presented in the previous chapter. The force commands from the environment simulation are assumed to be available and updated continuously. However, in practice this is not the case due to computation delays in the environment graphics visual display and dynamics calculations. The delay varies with the population of objects, their complexity measured in number of polygons per displayed object, and how many objects are moving. So, the updates of force inputs to the joystick controller occur at varying time intervals. The frequency that is achieved by the present BOLIO environment system on Hewlett Packard 835 machines ranges from 40 Hz for relatively simple graphic objects with no motion to 7 Hz if multiple complex objects that are moving have to be displayed. If the force commands to the joystick system are simply held between updates, the joystick inputs will be a series of step inputs rather than a continuous force function. Simple force commands from the graphics and environment host computer are therefore not adequate, and a different approach that implements a partial environment model at the real time control level is necessary. Two early approaches that have been partially implemented on the current system are described in the next sections.

5.1.1 Object Impedance Approach

Besides providing discrete inputs to the joystick system, the computation delays also effectively sample the simulated environment. The mechanical bandwidth of the environment dynamics that can be simulated is therefore limited by the sample frequency. If the simulated environment has a natural frequency that is more than twice the computer loop update rate, then aliasing will degrade the simulation output. As a simple example, if the joystick endpoint is to simulate a mass-spring-damper system with a natural frequency of 14 Hz and the simulation computation loop runs at 7 Hz, the input to the joystick controller will be a square wave, far from the desired dynamics.

The force output will be continuous if, instead, the joystick processor implements a cartesian impedance control with a simulated spring reference position, stiffness, damping constant, and mass as the endpoint impedance parameters. The force inputs are not varied directly by the host computer, but a servo loop tries to maintain the joystick endpoint at the current reference position. Any position disturbance will generate a restoring force proportional to the current servo loop gains, which are updated to match the desired endpoint impedance. With this impedance controller, the simple spring example mentioned above would be simulated by a position servo loop with a gain that corresponds to the desired spring constant. The graphic display is still updated at 7 Hz, but this delay does not enter into the joystick force control. Figure 5.1 shows the distinction between these two approaches.

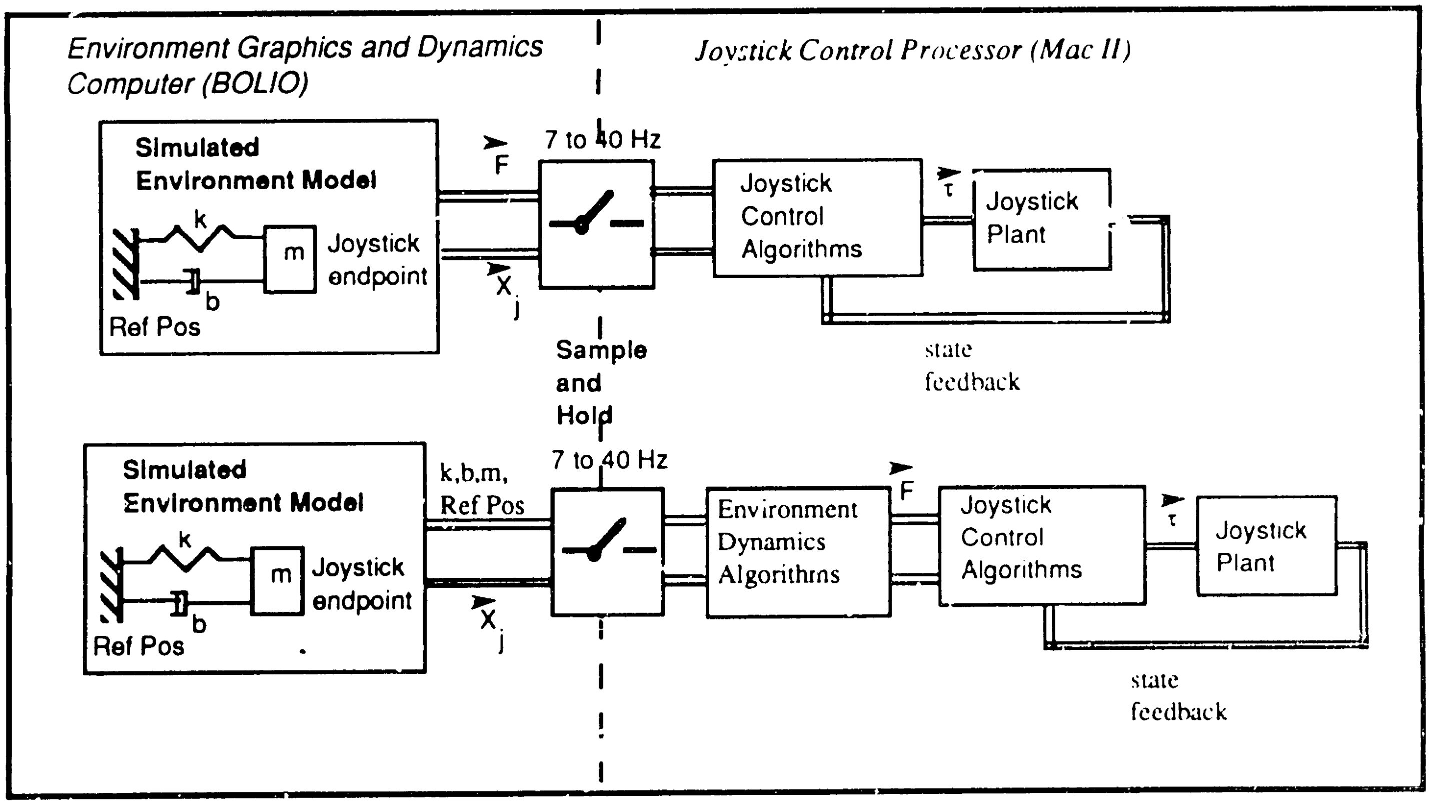

Environment Graphics and Dynamics Computer (BOLIO)
Joystick Control Processor (Mac II)
Simulated Environment Model
k
m
Joystick endpoint
b
Ref Pos
F
7 to 40 Hz
Joystick Control Algorithms
τ
Joystick Plant
X j
Sample and Hold
state feedback
Simulated Environment Model
k,b,m, Ref Pos
7 to 40 Hz
Environment Dynamics Algorithms
F
Joystick Control Algorithms
τ
Joystick Plant
X j
state feedback

Figure 5.1: Separation of Graphics Computation and Joystick Control

The simple one dimensional spring model can be easily expanded to the case of more complex objects. Objects within an environment can be described in terms of a set of geometric primitives: points, lines, planes, ellipsoids, parallelpipeds, finite cylinders, and cones, for example. The virtual surface in figure 2.1, can be discretized into a collection of local tangent planes. The object's physical characteristics are then given in terms of its centroid parameters such as mass and moment of inertia tensor, and the surface contact stiffness and damping characteristics. The graphics computation determines the closest tangent plane to the present joystick endpoint position through a simple minimization procedure. The position vector $\bar{x}_p$, and unit normal vector $\bar{n}_p$ specify the plane's location and orientation is space and are passed together with the impedances in the normal and tangential directions to the environment controller located in the joystick real-time control code. For a joystick position vector $\bar{x}_j$, the scalar distance to the plane is simply:

$$|\bar{x}_j - \bar{x}_p| \, \bar{n}_p^T = d \tag{5.1}$$

If d > 0, then the joystick endpoint is above and not in contact with the plane, and the impedance is zero since the endpoint is not constrained in any direction. For d < 0, the endpoint is in contact with the plane and a force $\bar{F}_n$, in the normal direction is commanded so that:

$$\bar{F}_n = \bar{n}_p \, |\{ \bar{n}_p (\bar{x}_j - \bar{x}_p)^T K_n + \bar{n}_p (\dot{\bar{x}}_j - \dot{\bar{x}}_p)^T b_n + \bar{n}_p \, (\ddot{\bar{x}}_j - \ddot{\bar{x}}_p)^T m_n \}| \tag{5.2}$$

This force assumes that the virtual surface has a particular compliance. However, the loadcell yields the actual force vector applied by the user. This information can be used to model an essentially rigid contact. If the endpoint is in contact with the surface, the command force should equal the component of the measured force $\bar{F}_m$ in the plane normal direction. A rigid surface can therefore be modeled even more simply once contact has been established by:

$$\bar{F}_n = \left\{ \bar{F}_m \, \bar{n}_p^T \right\} \bar{n}_p \tag{5.3}$$

For a frictionless contact, the tangential directions are unconstrained. If, however, a force term is included in the tangential $\bar{t}_p$ direction, then the direction is given by:

$$\bar{t}_p = \frac{(\bar{x}_j - \bar{x}_p) - d\,\bar{n}_p}{|\,(\bar{x}_j - \bar{x}_p) - d\,\bar{n}_p\,|} \tag{5.4}$$

The tangential impedance then yields forces for a specified deviation from the surface reference position in the tangential direction. The total force command $\bar{F}_{in}$ is then simply the sum of the tangential and normal contact forces. The endpoint force follows this input according to the force control strategy described in the previous chapter.

5.1.2 Energy Function Representation

The advantage of describing an object by its collection of tangent planes is that the objects geometric complexity is limited only by the available computation speed. The main difficulty is the determination of which local plane lies closest to the joystick endpoint. This minimization has to be done during each graphics update cycle to determine which plane normal, position vectors and impedances have to be sent to the joystick environment controller. If, however, the object primitives are a sum of polynomial scalar function surface descriptions, i.e. ellipsoids, parallelpiped, cones and cylinders, then a computationally less expensive approach is possible.

Each object k has an associated potential energy function $U_k(\bar{x})$. Since energy is a scalar function, the total potential field for the environment will just be given by the sum of all the object energy functions. The surface normal contact force for any object is then simply found from the gradient of the potential function:

$$\bar{F}_n = -\text{grad}\left[\sum_{k=1}^{n} U_k(\bar{x})\right] \tag{5.5}$$

Any non-conservative force terms such as viscous damping have to be summed to the normal force separately.

The problem now reduces to finding a suitable potential function for objects of interest. It can be chosen to have a limited influence close to the surface, or be of a high enough order so that it quickly falls to zero away from the object. An object described by an analytic equation $f_k(x,y,z)=c$ may have as a potential function:

$$U_k(\bar{x}) = \begin{cases} \frac{1}{n} K_n (f_k(\bar{x}) - c)^n & \text{if } f_k(\bar{x}) < c \\ 0 & \text{if } f_k(\bar{x}) > c \end{cases} \tag{5.6}$$

where n is the order of the potential function and K_n is the surface normal stiffness. A power n higher than two will not yield a linear force versus displacement function for the contact stiffness. A simple control algorithm would substitute the current joystick endpoint coordinates in each object equation and test for whether the joystick is inside (contacting) the object, in which case the potential function is active. Instead of a number of minimizations as in the planar object description above, only one function has to be evaluated for each object. Also, the resolution of the planar description increases the computational burden for determining the closest plane, but the energy approach requires only one simple substitution into the object function independent of the spatial resolution.

Another potential field candidate that provides a repulsive force close to an object boundary is an inverse square field suggested by [Khatib]:

$$U_k(\bar{x}) = \begin{cases} \frac{1}{n} K_n \left(\frac{1}{f_k(\bar{x})} - \frac{1}{f_k(\bar{x}_0)} \right)^n & \text{if } f_k(\bar{x}) < f_k(\bar{x}_0) \\ 0 & \text{if } f_k(\bar{x}) > f_k(\bar{x}_0) \end{cases} \tag{5.7}$$

On the surface, $U_k(\bar{x})$ tends to infinity, and at $\bar{x}_0$ the function is zero, beyond which there is no force. The surface normal contact force is then computed at any given point in the joystick workspace by a center difference approximation of the gradient:

$$\bar{F}_n = \left\{ \begin{array}{c} \dfrac{U(x_j+h, y_j, z_j) - U(x_j-h, y_j, z_j)}{2h} \\ \dfrac{U(x_j, y_j+h, z_j) - U(x_j, y_j-h, z_j)}{2h} \\ \dfrac{U(x_j, y_j, z_j+h) - U(x_j, y_j, z_j-h)}{2h} \end{array} \right\} \tag{5.7}$$

The disadvantage of the energy approach is that the object primitives are limited in the complexity of surfaces they can represent. The planar surface description can describe any object surface at a desired resolution, though the computational cost quickly becomes prohibitive. Coding strategies for virtual environment dynamic information are a matter of ongoing research in the computer graphics and robotics fields. Surface texture mappings [Minsky], real-time dynamic computation, parallel algorithms, and finite element method approaches are all considered elsewhere [Media Lab CGA].

Chapter 6

Conclusions

6.1 Summary

Force feedback display technology draws from many diverse fields that range from neurophysiology to robotics. There are, however, a set of design guidelines and results from experience that are crucial. The first is to distinguish between force and tactile output. The latter talks to the fingertips where the former involves full hand and wrist or arm motion, depending on the workspace size. The bandwidth requirements for each of these output modalities vary over orders of magnitude (10Hz - 1000 Hz), and a single device cannot address them both. Second, there exists a strong symbiosis between all the senses, and even a simple force transducer will provide a convincing simulation with the appropriate visual and aural cues. Third, because one seldomly uses full arm motion in a precise task, it is more appropriate to display to the hand rather than the arm, and the device must adapt to the user's style. Fourth, there still is a shortage of computer speed to meet the real-time requirements of the dynamic simulation, and servo loop rates are a more stringent constraint that visual update rates. Finally, the most difficult part is modelling the virtual environment, and it is along these lines that present research is aimed.

There are also many issues from a mechanical perspective that have to be addressed. Though instabilities in force control loops caused by non-collocated sensors and actuators are well understood, there are no solutions on how to improve the closed loop device bandwidth since any transmission element will have high frequency dynamics that cause unstable behavior. One improvement, direct drive, avoids transmission problems, but the limited electromechanical power to weight ratio requires impractical motor sizes. Further, compensation for undesired friction, inertia, and viscous drag determines the system backdriveability and ultimate bandwidth. Sensor technologies, in particular miniature multi-axis force transducers, leave much room for improvement. These are all avenues of current research [Jacobsen, Salisbury].

6.2 Future Applications

The best measure of the force output fidelity, it's "feel", is difficult to quantify. Some basic criteria and the system's performance in meeting these are given in Chapter 4. However, a more interesting simulation to evaluate a person's impressions of a force simulation is the two dimensional virtual environment "Sandpaper" code developed at the University of North Carolina by [Minsky and Steele]. The low level force command and position measurement subroutines were adapted to communicate with the present joystick design. The simulation models a three dimensional static object, the seminal computer graphics teapot, and also a dynamic "Bolo" system, in effect two masses attached by springs to the joystick endpoint cursor. The teapot model shown in figure 6.1 uses a depth map to generate a gradient field of the surface. The two dimensional joystick command forces are then proportional to the position dependent local slope. The user thus receives a sense of the teapot surface contour. For the "Bolo" simulation, the stiffness, mass, and viscosity parameters can be varied within the simulation. The forces that are transmitted to the user are computed in real-time and depend on the displacements and accelerations of the two masses relative to the joystick endpoint cursor. Both of these simple effects generate convincing simulations.

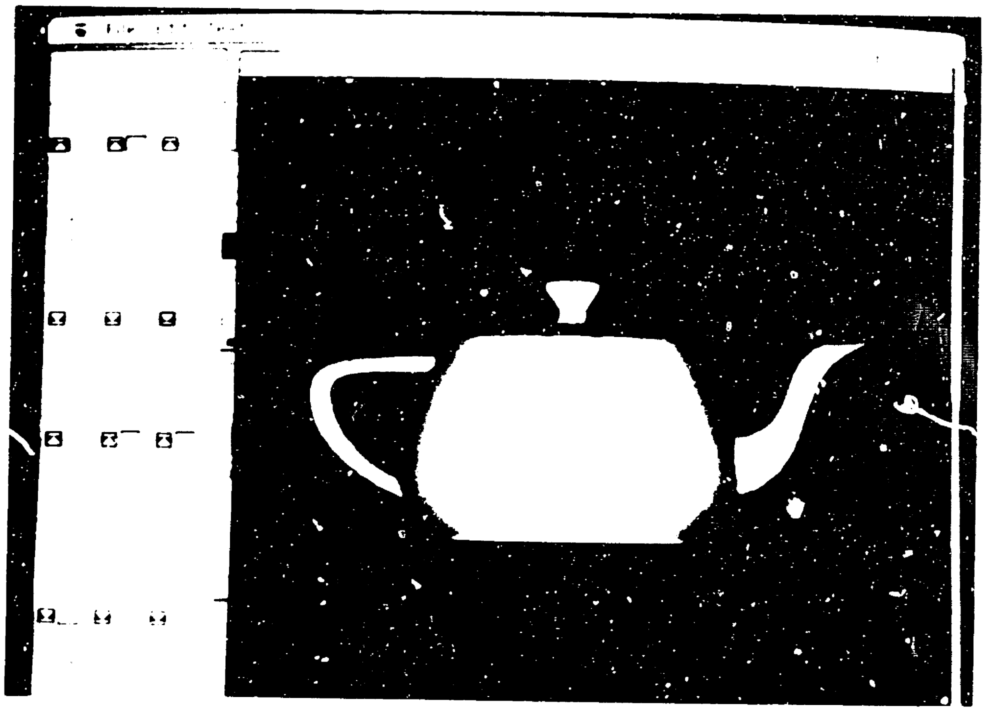


Figure 6.1: [illegible] pot Simulated Environment [Steele and Minsky]

Present work is aimed at connecting the joystick system to the three dimensional modelling environment in Bolio [S. Drucker, S. Pieper, Media Lab Computer Graphics and Animation Group]. The strategies described in chapter 5 are currently being implemented to generate full three dimensional object dynamics simulations. A communication link between the Bolio system and the MIT Artificial Intelligence Laboratory allows the remote control of the Whole Arm Manipulator (WAM) robot [Townsend, Salisbury]. The WAM is controlled through a model of the WAM kinematics that is displayed and computed in real-time in the Bolio system. The force reflecting joystick will act as a general master that displays the endpoint WAM forces to the user [Drucker]. Virtual planes or objects can be used to constrain the joystick endpoint and therefore also the slave WAM arm to remain within certain regions of the workspace. Bolio can generate a model environment of the task and filter certain information before it is displayed to the user. Finally, [Pieper] is working on a full anatomical leg model that includes muscle, tendon, and bone dynamics. It will be a natural extension to use this model as a base for surgical simulations that use the force feedback joystick as a virtual scalpel. The dynamics of the surgical simulation can then be felt by the trainee. These are various applications that are current topics of investigation at the MIT Media Lab.

The successful implementation of a high bandwidth force output device involves issues that span many disciplines. The number of applications for a tool that allows people to feel what is seen on the computer's screen are limitless. The device presented in this thesis represents a good compromise between the many conflicting requirements. Simple physical elements were modelled as a benchmark test of the system's performance and to evaluate different approaches to hybrid motor and brake control. The successful fusion of current graphics simulation technology with the force output joystick is a valuable tool to study task planning models, robotic teleoperator control issues, and virtual environment simulations. Perhaps the next generation of doctors, pilots, astronauts, manufacturers, deep sea explorers, and kids at the local arcade will reach out and touch some...... thing.

Bibliography

Adelstein, B.D. A Virtual Environment System for the Study of Human Arm Tremor, PhD thesis, Department of Mechanical Engineering, MIT, Cambridge, MA 1989

An, C.H., and Hollerbach, J.M. Dynamic Stability Issues in Force Control of Manipulators, In Proceedings of the IEEE International Conference on Robotics and Control, 1987

Batter, J.J. and Brooks Jr. , F.P. GROPE-1: A computer display to the sense of feel. In Information Processing 1971, Proceedings of the IFIP Congress 71, Ljuljana, Yugoslavia, 1971

Bejczy, A.K. and Handlykken, M. Experimental Results with a six-degree-of-freedom hand-controller. In Proceedings of the 7th Annual Conference on Manual Control, 1981

Bejczy, A.K. and Salisbury, Jr., J.K. , Kinesthetic coupling between operator and remote manipulator. In Proceedings of the International Computer Technology Conference of ASME, San Francisco, CA 1980.

Bliss, J.C., Hill, J.W. and Wilber, B.M., Tactile Perception Studies Related to Teleoperator Systems, NASA CR-1775 Report, Stanford Research Institute, Menlo Park, CA 1971.

Brooks, Jr., F.P. The computer "scientist" as toolsmith - studies in interactive computer graphics. In B. Gilchrist, editor, Information Processing 1977, Proceedings of the IFIP Congress 77, Toronto, 1977.

Brooks, T.L. and Bejczy, A.K., Hand controllers for teleoperation, a state-of-the-art technology survey and evaluation, Jet Propulsion Laboratory, Pasadena, CA , 1985

Cherubino, C., BSc thesis Department of Computer Science, MIT, 1988

Colgate, J.E. The Control of Dynamically Interacting Systems. PhD thesis, Department of Mechanical Engineering, MIT, 1988

Eppinger S.D. and Seering W.P., Understanding bandwidth limitations in robot force control. In Proceedings of the 1987 IEEE International Conference on Robotics and Automation, Raleigh, NC 1987.

Fisher, S., Telepresence master glove controller for dextrous robotic end-effectors, In Proceedings of the SPIE Cambridge Symposium on Optical and Optoelectronic Engineering, Cambridge, MA, 1986.

Geyer, K.E., and Wilson, K.R., Computing with Feeling, In Proceedings of IEEE 1975 Conference on Computer Graphics, 1975.

Goertz R.C., et al., The ANL Mark E4A Electric Master-Slave Manipulator. In Proceedings of the 14th Conference of Remote Systems Technology, 1966.

Hogan, N., Controlling Impedance at the Man/Machine Interface, In Proceedings of IEEE Robotics and Automation Conference 89, Scottsdale, AR 1989

Hogan, N., Impedance control: An approach to manipulation: Part I - Theory, ASME Journal of Dynamic Systems, Measurement and Control, 1985.

Hogan, N., Impedance control: An approach to manipulation: Part II - Implementation. ASME Journal of Dynamic Systems, Measurement and Control, 1985.

Hogan, N., Impedance control: An approach to manipulation: Part III - Applications. ASME Journal of Dynamic Systems, Measurement and Control, 1985.

Hunter, I.W., Lafontaine, S., Nielsen, P.M.F., Hunter, P.J., and Hollerbach, J.M.. A tele-microrobot for manipulation and dynamic mechanical testing of single living cells. In Proceedings of the IEEE Micro Electro Mechanical Systems Conference, February 1989.

Jex, H.R. Some Criteria for Teleoperator Virtual Environments, In Human-Machine Interfaces for Teleoperators and Virtual Environments, Santa Barbara, CA, 1990

Khatib, O., Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, In The International Journal of Robotics Research, Vol. 5 No. 1, 1986

Maxwell, S.M., PhD thesis in progress, Department of Mechanical Engineering, MIT, Cambridge, MA 1990.

Ming, O., Minsky, M., Behensky, M., Brooks Jr., F.P., Creating an Illusion of Feel: Force Display, In Proceedings of 1989 ACM SIGGRAPH Conference, Boston, MA, 1989.

Ming, O., Pique, M., Hughes, J., Srinivasan, N., and Brooks Jr., F.P. Using a manipulator for force display in molecular docking. In Proceedings of the 1988 IEEE International Conference on Robotics and Automation, Philadelphia, PA, 1988.

Minsky, M., and Steele, O., Sandpaper: An Environment for Feeling and Seeing Texture and Dynamics, In Proceedings of the 1990 Symposium on Interactive 3D Graphics,

Minsky, M., PhD thesis in Progress, MIT Media Laboratory, Cambridge, MA 1990

Mussa-Ivaldi, F.A., Hogan, N., and Bizzi, E., Neural, mechanical, and geometric factors subserving arm posture in humans. Journal of Neuroscience, October, 1985.

Noll, A.M., Man-Machine Tactile Communication, PhD thesis, Department of Electrical Engineering, Polytechnic Institute of Brooklyn, Brooklyn, NY,1971.

Russo, M.A., A Control Analysis of a Hybrid Motor-Brake System for Tactile Simulations, Department of Mechanical Engineering, MIT, Cambridge, MA, 1988.

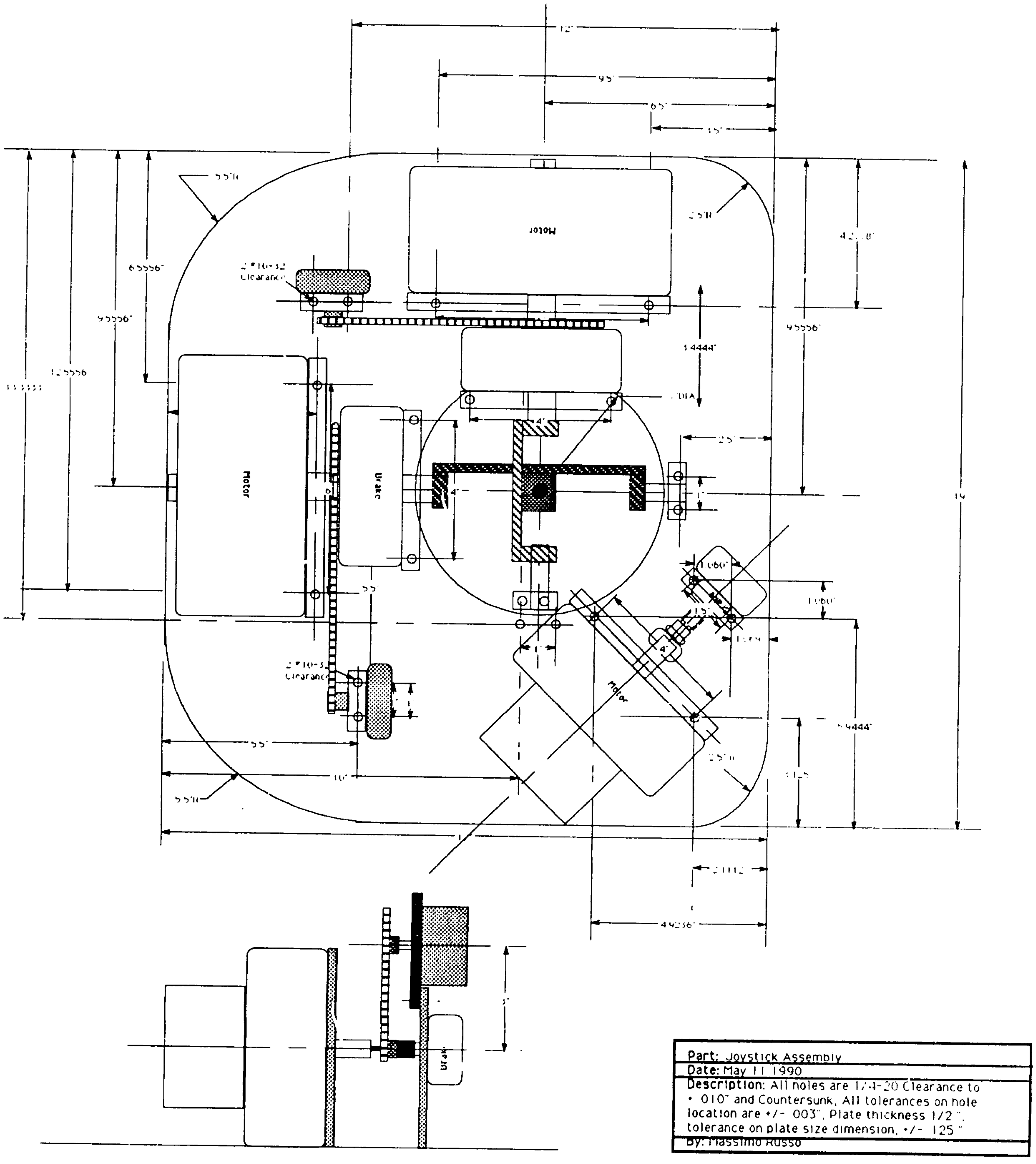
Smith, M.J., Tactile Interface for Three-Dimensional Computer-Simulated Environments: Experimentation and the Design of a Brake-Motor Device, Masters thesis, Department of Mechanical Engineering, MIT, Cambridge, MA, 1988

Tadros, A.H., Control System Design for a Three Degree of Freedom Virtual Environment Simulator Using Motor/Brake Pair Actuators, Masters thesis, Department of Mechanical Engineering, MIT, Cambridge, MA 1990

Townsend, W. T., The Effect of Transmission Design on the Performance of Force-Controlled Manipulators, PhD thesis, Department of Mechanical Engineering, MIT, 1988

Appendix 1

Joystick Assembly Drawing



Part: Joystick Assembly
Date: May 11 1990
Description: All holes are 1/4-20 Clearance to + .010" and Countersunk, All tolerances on hole location are +/- .003", Plate thickness 1/2", tolerance on plate size dimension, +/- .125"
By: Massimo Russo

Appendix 2

Loadcell Calibration Code

```
/****************************************************************************/
/**************    LOADCELL CALIBRATION ROUTINE *****************/
/****************************************************************************/


#include <stdio.h>
#define _MC68881_
#include <Math.h>

/* Read in initial voltage values from laod cell three axis strain gage channels and the x-axis
potentiometer (analog position) channel are used to read the four voltages */

main()
{
int vinit1, vinit2, vinit3, vinit4,v[4][20], n,m,i,j,k,v1,v2,v3,v4;
float h[4][3],ht[3][4],f[3][20], hth[3][3],invhth[3][3],C[3][4],ftinvfft[3][3],transpose[3][3],det;
float ftrans[20][3],fft[3][3],invfft[3][3], cof[3][3];
int x,y,z,fx,fy,fz;
FILE *calibration_matrix;

if((calibration_matrix = fopen("calibration_matrix", "w"))==NULL)   /*open file for
calibration matrix*/
{printf("Can't open file 'calibration_matrix' \n");}


/*   INITIALISATION of motors and zero force loadcell values */

initports();

for(i=0;i<3;i++){
        setaxis(i);
        putmotordac(2048);
}
updatedacs();
reset_time();
wait_for_time(250);

vinit1=0;
for(j=0;j<10;j++){
        setaxis(0);
        setmux(0);
        delay(10);
        vinit1 +=getadc();
}
vinit1 /=10;

vinit2=0;
for(j=0;j<10;j++){
        setaxis(0);
        setmux(2);
        delay(10);
        vinit2 +=getadc();
}
vinit2 /=10;
```

```
vinit3=0;
for(j=0;j<10;j++){
        setaxis(1);
        setmux(0);
        delay(10);
        vinit3 +=getadc();
}
vinit3 /=10;

vinit4=0;
for(j=0;j<10;j++){
        setaxis(2);
        setmux(0);
        delay(10);
        vinit4 +=getadc();
}
vinit4 /=10;

/* START OF CALIBRATION ROUTINE
Collect measurements */

printf("how many calibration force measurements will you want ?\n");
scanf("%d",&m);

for (i=0;i<m;i++)
{
printf("force measurement #: %d\n",i);
printf("Input force components:\n");
printf("x=");
scanf("%d",&x);

printf("y=");
scanf("%d",&y);

printf("z=");
scanf("%d",&z);

/* load force values into force array */

f[0][i]=x;
f[1][i]=y;
f[2][i]=z;

printf("fx= %d fy=%d fz=%d \n",x,y,z);

/* Read in loadcell voltages */

 printf(" reading in loadcell voltages: \n");


setaxis(0);
setmux(0);
delay(10);
v1=getadc()-vinit1;

setmux(2);
delay(10);
v2=getadc()-vinit2;

setaxis(1);
setmux(0);
delay(10);
v3=getadc()-vinit3;

setaxis(2);
setmux(0);
delay(10);
v4=getadc()-vinit4;



printf("v1= %d v2=%d v3=%d v4=%d \n",v1,v2,v3,v4);

/* store voltages in voltage array */

v[0][i]=v1;
v[1][i]=v2;
v[2][i]=v3;
v[3][i]=v4;

}

printf(" Done taking calibration data.\n");

/* Acknowledge measurement matrices */

printf(" inputted force array=\n");
for  (i=0;i<3;i++)
        {
        for (j=0; j<m; j++)

        printf("%5f  ", f[i][j]);
        printf("\n");
        }
printf("inputted voltage array=\n");
for  (i=0;i<4;i++)
        {
        for (j=0; j<m; j++)

        printf("%d  ", v[i][j]);
        printf("\n");
        }
printf("\nPress mouse button when ready to see arrays\n");
while(!Button());

/* for V(4xm)=h(4x3)*f(3xm), h is found from h=ff'inv(ff') , so */
```

```
/*find transpose of f, ftrans*/

for  (i=0;i<3;i++)
        for (j=0;j<m;j++)
                ftrans[j][i] = f[i][j];

printf(" Ftrans=\n");
for  (i=0;i<m;i++)
        {
        for (j=0; j<3; j++)

        printf("%5f    ", ftrans[i][j]);
        printf("\n");
        }

printf("\nPress mouse button when ready to see arrays\n");
while(!Button());

/* multiply f*ftrans */
for  (i=0;i<3;i++)
        for (j=0;j<3;j++)
                fft[i][j]=0;


for  (i=0;i<3;i++)
        for (j=0;j<3;j++)
                for  (k=0;k<m;k++)
                fft[i][j]=fft[i][j]+f[i][k]*ftrans[k][j];

printf(" F*Ftrans=\n");
for  (i=0;i<3;i++)
        {
        for (j=0; j<3; j++)

        printf("%5f     ", fft[i][j]);
        printf("\n");
        }

/* these are the  matrix manipulation routines for finding a 3*3 matrix inverse */


/* find matrix of cofactors */
for  (i=0;i<3;i++)
        for (j=0;j<3;j++)
                cof[i][j]=0;


cof[0][0] = fft[1][1]*fft[2][2]-fft[1][2]*fft[2][1];
cof[0][1] = -fft[1][0]*fft[2][2]+fft[1][2]*fft[2][0];
cof[0][2] = fft[1][0]*fft[2][1]-fft[1][1]*fft[2][0];
cof[1][0] = -fft[0][1]*fft[2][2]+fft[0][2]*fft[2][1];
cof[1][1] = fft[0][0]*fft[2][2]-fft[0][2]*fft[2][0];
cof[1][2] = -fft[0][0]*fft[2][1]+fft[0][1]*fft[2][0];
```

```
cof[2][0] = fft[0][1]*fft[1][2]-fft[0][2]*fft[1][1];
cof[2][1] = -fft[0][0]*fft[1][2]+fft[0][2]*fft[1][0];
cof[2][2] = fft[0][0]*fft[1][1]-fft[0][1]*fft[1][0];

/* find the determinant */

det=fft[0][0]*cof[0][0]+fft[0][1]*cof[0][1]+fft[0][2]*cof[0][2];

printf("%5f",det);

/* find transpose of matrix of cofactors */

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                transpose[i][j] = cof[j][i];

/* find matrix inverse */

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                invfft[i][j]=transpose[i][j]/det;


printf(" inv(F*Ftrans)=\n");
for (i=0;i<3;i++)
        {
        for (j=0; j<3; j++)

        printf("%5f     ", invfft[i][j]);
        printf("\n");
        }

/* multiply by f' */
for (i=0;i<m;i++)
        for (j=0;j<3;j++)
                ftinvfft[i][j]=0;

for (i=0;i<m;i++)
        for (j=0;j<3;j++)
                for (k=0;k<3;k++)
                ftinvfft[i][j]=ftinvfft[i][j]+ftrans[i][k]*invfft[k][j];

/* multiply by v */

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                h[i][j]=0;

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                for (k=0;k<m;k++)
                h[i][j]=h[i][j]+v[i][k]*ftinvfft[k][j];

printf(" H=\n");
```

```
for (i=0;i<3;i++)
        {
        for (j=0; j<3; j++)

        printf("%5f     ", h[i][j]);
        printf("\n");
        }



/* finally find calibration matrix: */

/* find h' :*/

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                ht[i][j] = h[j][i];

/* multiply h by ht */

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                hth[i][j]=0;

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                for (k=0;k<3;k++)
                hth[i][j]=hth[i][j]+ht[i][k]*h[k][j];


/* find inverse of hth */


/* find matrix of cofactors */

cof[0][0] = hth[1][1]*hth[2][2]-hth[1][2]*hth[2][1];
cof[0][1] = -hth[1][0]*hth[2][2]+hth[1][2]*hth[2][0];
cof[0][2] = hth[1][0]*hth[2][1]-hth[1][1]*hth[2][0];
cof[1][0] = -hth[0][1]*hth[2][2]+hth[0][2]*hth[2][1];
cof[1][1] = hth[0][0]*hth[2][2]-hth[0][2]*hth[2][0];
cof[1][2] = -hth[0][0]*hth[2][1]+hth[0][1]*hth[2][0];
cof[2][0] = hth[0][1]*hth[1][2]-hth[0][2]*hth[1][1];
cof[2][1] = -hth[0][0]*hth[1][2]+hth[0][2]*hth[1][0];
cof[2][2] = hth[0][0]*hth[1][1]-hth[0][1]*hth[1][0];

/* find the determinant */

det=hth[0][0]*cof[0][0]+hth[0][1]*cof[0][1]+hth[0][2]*cof[0][2];

/* find transpose of matrix of cofactors */

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                transpose[i][j] = cof[j][i];
```

```
/* find matrix inverse */

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                invhth[i][j]=transpose[i][j]/det;

/* multiply by ht to find the calibration matrix C */
for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                C[i][j]=0;

for (i=0;i<3;i++)
        for (j=0;j<3;j++)
                for (k=0;k<3;k++)
                C[i][j]=C[i][j]+invhth[i][k]*ht[k][j];

/* print calibration matrix */
printf("\n");
printf("Calibration matrix :\n");
for (i=0;i<3;i++){
                for (j=0; j<3; j++){
                        printf("%5f  ", C[i][j]);
                        fprintf(calibration_matrix, "%5f  ", C[i][j]);
                }
        printf("\n");
        fprintf(calibration_matrix, "\n");
        }
printf("This calibration matrix was also printed to the file 'calibration_matrix'\n");
fclose(calibration_matrix);

/**********************************************************************Now   for   an
input joint torque, the decoupled endpoint force will be calculated.
**********************************************************************/

        /* zero all motor torques */
for(i=0;i<3;i++){
        setaxis(i);
        putmotordac(2048);
}
updatedacs();

printf("Calibration matrix :\n");
for (i=0;i<3;i++){
                for (j=0; j<4; j++){
                        printf("C[%d][%d]=",i,j);
                        scanf("%f",&C[i][j]);
                }
        printf("\n");
        fprintf(calibration_matrix, "\n");
        }

printf("Press mouse button to re-initialize the load cell.\n");
while(!Button());
```

```
        /* initialize the load cell values again. */
vinit1=0;
for(j=0;j<10;j++){
        setaxis(0);
        setmux(0);
        delay(10);
        vinit1 +=getadc();
}
vinit1 /=10;

vinit2=0;
for(j=0;j<10;j++){
        setaxis(0);
        setmux(2);
        delay(10);
        vinit2 +=getadc();
}
vinit2 /=10;

vinit3=0;
for(j=0;j<10;j++){
        setaxis(1);
        setmux(0);
        delay(10);
        vinit3 +=getadc();
}
vinit3 /=10;

vinit4=0;
for(j=0;j<10;j++){
        setaxis(2);
        setmux(0);
        delay(10);
        vinit4 +=getadc();
}
vinit4 /=10;


        /* Keep reading in desired joint torque values and outputting the decoupled endpoint
forces. */

while(1)
{
setaxis(0);
setmux(0);
delay(10);
v1=getadc()-vinit1;

setmux(2);
delay(10);
v2=getadc()-vinit2;

setaxis(1);
setmux(0);
delay(10);
v3=getadc()-vinit3;

setaxis(2);
setmux(0);
delay(10);
v4=getadc()-vinit4;

fx=C[0][0]*v1+C[0][1]*v2+C[0][2]*v3;
fy=C[1][0]*v1+C[1][1]*v2+C[1][2]*v3;
fz=C[2][0]*v1+C[2][1]*v2+C[2][2]*v3;

printf("fx= %d fy=%d fz=%d \n",fx,fy,fz);
}

}
```

Appendix 3

Joystick Control Routines

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

                              JOYSTICK CONTROL CODE

                                       by

                           Alfred Tadros and Massimo Russo

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


#include <stdio.h>
#define _MC68881_
#include <Math.h>
#include <storage.h>
#include <MacTypes.h>
#include <SerialDvr.h>
#include <MacProtos.h>

/*#define MALLOC_DEBUG 1          */

#define SLOTBASE     0xc00000
#define PARPORT0     SLOTBASE
#define PORTA        ((char *) PARPORT0)
#define PORTB        ((char *) (PARPORT0 + 0x4) )
#define PORTC        ((char *) (PARPORT0 + 0x8) )
#define CONTROL ((char *) (PARPORT0 + 0xc) )


            /********** variable list **********/
int Load_init [3], init_vel[3];
int R_old, z_over, R_temp;
int frictionup_val, frictiondown_val;
int angle_index, phi_angle_index, theta_angle_index;
float k_brake_dac[3];              /* Coefficients to go from D/A units to brake torque */
float k_motor_dac[3];              /* Coefficients to go from D/A units to motor torque */
float f_sensor[3], c[3][3];        /* load cell values and its coefficient matrix */
float angle_pos[3], xyz[3];                /* theta, phi, R and x, y, z position values */
float angle_vel[3], end_vel[3];    /* theta, phi, R and x, y, z velocity values */
float tach_gain[3];                              /* tach gains set in init_tachs(); */
float tan_func[4096];              /* tan_func is a 4096 element table of tangent values */
float *sin_func, *cos_func;        /* sin_func and cos_func are trig tables using malloc */
float J[3][3], J_inv[3][3];                /* elements of the Jacobian and its inverse */
float force[3], fsens_calc[3];   /* forces in the XYZ global frame and those trasformed to xyz */
float torque[3], DtoA_torque[3];
float torq_desi[3], torq_meas[3];  /* desired and actual(load cell) motor torques */
float brake_torque[3], motor_torque[3]; /* commanded motor and brake torques in Nm */

/* system initialization routine */

initjs()
{
        int i;
```

```
	asm
	{
			move.b			#0x82, CONTROL		/* A=out, B=in, Clo=out,
Chi=out*/
			move.b			#0xff, PORTA	/* Everything off (all signals are
active low) */
			move.b			#0xff, PORTB
			move.b			#0xdf, PORTC	/* -S/H normally low to allow
sampling */
	}
			for(i=0; i<3; i++) {			/*With offset DACs, 2048 is used to set*/
				setaxis(i);					/*all analog outputs to zero
volts*/
				putbrakedac(3900);			/* 000H = 0000 => -10volts	*/
				putmotordac(2048);			/* 800H = 2048 => 0volts */
				updatedacs();				/* FFFH = 4096 => 10volts */
						}
			init_load_cell();
			init_tachs();
			resetencoders();

/****** Scaling to go from desired motor torques to the needed D/A units for the amps. *****/

k_motor_dac[0] = 1149.6285;	/* these constants are the conversion from [Nm] of motor shaft*/
k_motor_dac[1] = 1149.6285;	/* torques (i.e. in joint coordinates) to D/A units for input */
k_motor_dac[2] = 2319.1163;	/* to the amplifiers. So these coefficients are in units of*/
k_brake_dac[0] = 501.563;	/* [(D/A)/Nm] with full scale from 0 to 4095 D/A units */
k_brake_dac[1] = 501.563;
k_brake_dac[2] = 3399.209;

			for (i=0; i<3; i++) {	/* initialize brake and motor torque commands to 0 */
				brake_torque[i]=0;
				motor_torque[i]=0;
			}

			z_over=0;			/* flag to act as an index for the R axis encoder */

			J_inv[0][1] = 0;	/* These elements are exactly zero hence initialized as such */
			J_inv[1][0] = 0;

			sin_func = (float *)malloc(4096 * sizeof(float));
			if (sin_func == NULL)
			{	fprintf(stderr, "yer hosed\n");
				fflush(stderr);
				ExitToShell();
			}
			cos_func = (float *)malloc(4096 * sizeof(float));
			if (cos_func == NULL)
			{	fprintf(stderr, "yer hosed on cos_func\n");
				fflush(stderr);
				ExitToShell();
			}
			/****************************************************************/
			for(angle_index=0;angle_index<4096;angle_index++)
```

```
                {
                        tan_func[angle_index]=tan((angle_index-2048)*0.0003835);
                        sin_func[angle_index]=sin((angle_index-2048)*0.0003835);
                        cos_func[angle_index]=cos((angle_index-2048)*0.0003835);
#ifdef MALLOC_DEBUG
                        printf("tan[%d]=<%f>\t",angle_index, tan_func[angle_index]);
                        printf("sin[%d]=<%f>\t",angle_index, sin_func[angle_index]);
                        printf("cos[%d]=%f\n", angle_index, cos_func[angle_index]);
#endif
                }
                /******************************************************************/

                return;
}
/*****************************************************************************/

/***************************************************************************
                FUNCTION init_load_cell(         )
****************************************************************************/

init_load_cell()

{
int i,j;

                /* calibration coefficients for the three degree of freedom load cell*/
                /* These coefficients are used in get_forces() to resolve the forces */
                /* into three orthogonal axes all measured in "GRAMS" in earth's gravity.*/
                /* each is divided by 101.94 to get the forces "f_sense" into Newtons */

        c[0][0]= 0.138272 / 101.94;
        c[0][1]= -0.113362 / 101.94;
        c[0][2]= -0.018448 / 101.94;
        c[1][0]= -0.145929 / 101.94;
        c[1][1]= -0.174824 / 101.94;
        c[1][2]= 0.245045 / 101.94;
        c[2][0]= -1.453367 / 101.94;
        c[2][1]= -1.574125 / 101.94;
        c[2][2]= 0.018980 / 101.94;

for(i=0;i<3;i++) Load_init[i]=0;

for(i=0;i<2;i++)
        {
                for(j=0;j<10;j++)
                {
                        setaxis(i);
                        setmux(0);        /* strain gauge channel i+1 from strain gauge input */
                        delay(10);
                        Load_init[i+1] += getadc();
                }
                Load_init[i+1] /=10;

                if(i<1)
                {
                        for(j=0;j<10;j++)
                        {
                        setaxis(i);
                        setmux(2);      /* strain gauge channel 1 from X pot input */
                        delay(10);
                        Load_init[i] += getadc();
                        }
                        Load_init[i] /=10;
                }

        }
        return;
}
/************************************************************************************
This function initialises the tachometer voltage values   FUNCTION INIT_TACHS()

************************************************************************************/
init_tachs()

{
int i;
for (i=0; i<3; i++)
{
        setaxis(i);
        setmux(3);                              /* Get tachometer measurements*/
        delay(10);
        init_vel[i] = getadc();
}
        tach_gain[0]= -127.4;           /* tach gains used to convert to rad/sec for theta and
phi */
        tach_gain[1]= -125.7;           /* Phi is the positive rotation about X axis */
        tach_gain[2]= -1296.51;                 /* tach gain used to convert to m/sec for R
axis*/
                                                                /* the change in sign is to give
correct direction */
return;
}
/****************************************************************
*                       FUNCTION SAFETY()
*  This function acts as a software watchdog and monitors the
*   drive shaft velocities, everything is shut down if they reach
*  above a certain level
*
*****************************************************************/

safety()
{
int i;
gettach();
if (abs(angle_vel[0])>8.0 || abs(angle_vel[1])>8.0 || abs(angle_vel[2])>0.7143)
{
        for (i=0;i<3;i++)
        {
                setaxis(i);
                putmotordac(2048);
                putbrakedac(0);
        }
        updatedacs();

        printf("theta_vel= %f     phi_vel= %f     R_vel= %f   ",angle_vel[0],angle_vel[1],
angle_vel[2]);
        printf(" That's MACH 5 in joystick years \n\n");

        printf("\n\n                    Velocities above threshold!");
        printf("\n\n            Press the mouse button to get back to action. \n\n");
        while(!Button());
        for (i=0;i<3;i++)
        {
                setaxis(i);
                putbrakedac(3900);
        }
        updatedacs();

        return;
}
        return;

}




/*****************************************************************************************
***
*                                               FUNCTION    GET_TIME()

*       This function simply reads in the time from the real time clock.  The clock is
*       automatically reset upon returning from this function. The value returned (s) can
*       hold a value between 0 and 255. (i.e. This is an 8 bit clock) 1 LSB = 76.2 microsec.
/*****************************************************************************************
***/

 get_time()
{
        int s, tempa;
        tempa = *PORTA;
        *PORTA = tempa | 0x30;                                  /* Sets "axis"
to 3 for reading switches */
        *PORTA = *PORTA & 0xbf;                                 /* enable write
of switch buffer */
        s = *PORTB & 0xff;                                      /* read in time
from counters */
        *PORTA = tempa;                                             /*
Restore "axis" and clear counters */
        return(s);
}
```

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

/*******************************************************************************
*                        FUNCTION WAIT_FOR_TIME(PERIOD)
*               This subroutine can be used to have a constant loop rate for control code.
*               Just call the function with the desired period which is a value form 0 to 255
*               and the function will return after that time is up. The period is multiplied by
*               76.2 microseconds to get the real time value. The clock is reset to 0 on return.
*       NOTE: If this function is called when the clock value is greater than PERIOD,
*       then the negative of the clock value is returned to signify a time overflow.
*******************************************************************************/
wait_for_time(period)
int period;
{
        int s, tempa;
        s=0;
        tempa = *PORTA;
        *PORTA = tempa | 0x30;                          /* Sets "axis" to 3 for
reading switches */
        *PORTA = *PORTA & 0xbf;                         /* enable write of
switch buffer */
        while(s<period)                                         /* wait for the
period to end */
                s = *PORTB & 0xff;                      /* read in time from
counters */
        if(s>period)                                    /* if time is past the
desired period value*/
                s=-s;                                           /*  then return
the negative of the actual time*/
        *PORTA = tempa;                                         /* Restore
"axis" and clear counters */
        return(s);
}
/*******************************************************************************
*****/

/*******************************************************************************
*               FUNCTION   RESET_TIME()
*               This function resets the real time 8 bit clock. No value is returned but the
*               clock is activated upon reset. Hence on return the clock is at zero and
*               counting, with 1 LSB = 76.2 microseconds
*******************************************************************************/

        /* resets counter chips 74LS393 to all zeros and starts counting */
reset_time()
{
        int tempa;
        tempa = *PORTA;
        *PORTA = tempa | 0x30;                                  /* Sets "axis"
to 3 for reading switches */
        *PORTA = *PORTA & 0xbf;                                 /* A/D read
low */
```

```
        *PORTA = tempa;                                                    /*
Restore "axis" and clear counters */
        return;
}


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

/*****************************************************************************************
*
*                       FUNCTION    set_brakes()
*                       This function sets the brakes to the commanded torque values.
*
*****************************************************************************************/

set_brakes()
{
int value[3],i;
                for (i=0;i<3;i++) {
                        value[i]=3900-brake_torque[i]*k_brake_dac[i];
                        if(value[i]>4095)
                                value[i]=4095;
                        if(value[i]<0)
                                value[i]=0;
                        setaxis(i);
                        putbrakedac(value[i]);
                updatedacs();
        }
        return;
}

/*****************************************************************************************
*                       FUNCTION    set_motors()
*                       This function sets the motors to the commanded torque values.
*****************************************************************************************/

set_motors()
{
int value[3],i;
                for (i=0;i<3;i++) {
                        value[i]=2048+motor_torque[i]*k_motor_dac[i];
                        if(value[i]>4095)
                                value[i]=4095;
                        if(value[i]<0)
                                value[i]=0;
                        setaxis(i);
                        putmotordac(value[i]);
                updatedacs();
        }
        printf("z-motor value = %f",value[2]);
        return;
}
```

```
/*****************************************************************************************
*               FUNCTION get_forces()
*               Three orthogonal forces in the loadcell x, y, z frame are now in Load[]
*****************************************************************************************/
get_forces()

{
        int i,j, Load[3];

        for(i=0;i<3;i++)f_sensor[i]=0;   /* initialize the force values to zero */

        for(i=0;i<2;i++)
        {

                setaxis(i);
                setmux(0);                /* strain gauge channel i+1 from strain gauge input */
                delay(10);
                Load[i+1] = getadc()-Load_init[i+1];

                if(i<1)
                {
                        setaxis(i);
                        setmux(2);                /* strain gauge channel 1 from X pot input */
                        delay(10);
                        Load[i] = getadc()-Load_init[i];
                }
        }

        for(i=0;i<3;i++){
                for(j=0;j<3;j++) f_sensor[i] += Load[j]*c[i][j]; /* get loadcell forces in Newtons */
                }
                /************ The above matrix multiplication does the following *******/
                /*      f_sensor[0] =  Load[0]*c[0][0]+Load[1]*c[0][1]+Load[2]*c[0][2];  */
                /*      f_sensor[1] =  Load[0]*c[1][0]+Load[1]*c[1][1]+Load[2]*c[1][2];  */
                /*      f_sensor[2] = -(Load[0]*c[2][0]+Load[1]*c[2][1]+Load[2]*c[2][2]); */
                /*  the negative in the last equation is taken care of in the coef.s  */

        /***********************************************************************/
return;
}

/*****************************************************************************************
*               FUNCTION fsens_to_fglobal() in the inertial frame
*       This function uses the globaly defined trig function tables cos_func[] and sin_func[]
*       The result of this function is the Inertial XYZ force values in Newtons in the
*       array force[3], where X froce = force[0], Y force = force[1], and Z force = force[2].
*       This uses the fact that the measured force equals the transformation matrix (T) times
*       the forces in the inertial frame XYZ coordinates. (i.e. f=TF or F= (T transpose)f )
*       T transpose equals T inverse cause T is an orthonormal matrix.
*       NOTE: You need to call JOINTPOS(); and GET_FORCES(); before you call this *
*       function.
*****************************************************************************************/
fsens_to_fglobal()
```

```
{
float  sin_phi,cos_phi,sin_theta,cos_theta;
float denom, numer_ct, numer_stcp, term, term4;

        sin_theta=sin_func[theta_angle_index];
        cos_theta=cos_func[theta_angle_index];
        sin_phi=sin_func[phi_angle_index];
        cos_phi=cos_func[phi_angle_index];

        term4 = sin_theta*sin_phi;
        denom=pow(1-term4*term4,0.5);
        numer_ct=cos_theta/denom;
        numer_stcp=sin_theta*cos_phi/denom;
        term = numer_stcp*f_sensor[0] - numer_ct*f_sensor[2];

        force[0] = numer_ct*f_sensor[0] + numer_stcp*f_sensor[2];
        force[1] = cos_phi*f_sensor[1] + term*sin_phi;
        force[2] = sin_phi*f_sensor[1] - term*cos_phi;

                /* Now force[] are the forces in the inertial XYZ coordinate frame */
return;
}

/*****************************************************************************************
*****
*               FUNCTION fglobal_to_fsens()
*       This function uses the forces in the inertial (global) XYZ coordinates frame and
*       calculates the forces in the load cell (xyz) coordinate frame. This can be used
*       for closing a control loop around the endpoint forces and joint torques after the
*       desired environment forces in global XYZ coordinates are transfromed to the desired
*       forces in the load cell (xyz) coordinate frame.   The torques can then be calculated
*       using FMEAS_TO_TORQS(). This is done using the transformation matrix T in the fro
*       f=TF or (forces in sensor frame are equal to transformation matrix time forces in
*       the global XYZ frame.
*       NOTE: You need to call JOINTPOS(); and GET_FORCES(); before you call this
        function.
*****************************************************************************************/
fglobal_to_fsens()
{
float  sin_phi,cos_phi,sin_theta,cos_theta;
float denom, numer_ct, numer_stcp, term, term4;

        sin_theta=sin_func[theta_angle_index];
        cos_theta=cos_func[theta_angle_index];
        sin_phi=sin_func[phi_angle_index];
        cos_phi=cos_func[phi_angle_index];

        term4 = sin_theta*sin_phi;
        denom=pow(1-term4*term4,0.5);
        numer_ct=cos_theta/denom;
        numer_stcp=sin_theta*cos_phi/denom;
        term = sin_phi*force[1] - cos_phi*force[2];

        fsens_calc[0] = force[0]*numer_ct + term*numer_stcp;
        fsens_calc[1] = force[1]*cos_phi + force[2]*sin_phi;
        fsens_calc[2] = force[0]*numer_stcp - term*numer_ct;


        /* Now fsens_calc[] are the calculated forces in the load cell coordinate frame. */

return;
}

/***********************************************************************************
*
*               FUNCTION fmeas_to_torqs()
*       This function uses the measured forces in the load cell coordinate frame and the
*       Jacobian and Transformation matrix to calculated the effective motor torques.
*       The equation it uses is motor (torques) = (J transpose)*(T transpose)*(f measured).
*       So th'_ function can be used to compare the desired motor torques with the actual
*       motor torques. The actual torques calculated here are held in the array torq_meas[].
*       NOTE: You need to call JOINTPOS(); and GET_FORCES(); before you call this
*       function.
***********************************************************************************/
fmeas_to_torqs()
{
float  sin_phi,cos_phi,sin_theta,cos_theta;
float denom, denom_2, term0, term4;

        sin_theta=sin_func[theta_angle_index];
        cos_theta=cos_func[theta_angle_index];
        sin_phi=sin_func[phi_angle_index];
        cos_phi=cos_func[phi_angle_index];

        term4 = sin_theta*sin_phi;
        denom_2 = 1-term4*term4;
        denom = pow(denom_2,0.5);
        term0 = -angle_pos[2]*f_sensor[0]/denom_2;

        torq_meas[0] = cos_phi*term0;
        torq_meas[1] = (term4*term0 - angle_pos[2]*f_sensor[1]/denom)*cos_theta;
        torq_meas[2] = 0.009525 * f_sensor[2];    /*cable wrapped around 0.009525m motor shaft
                                                    radius*/

                /* Now force[] are the forces in the inertial XYZ coordinate frame */


return;
}

/***********************************************************************************
*                               FUNCTION backdrive()
*                This function performs only the backdrive of the frictional cable drive.
*                Use command_motor() to set a specific torque value at the motors
***********************************************************************************/
backdrive()
{
int z_mot;
```

```
z_mot=2048;
get_forces();
        z_mot +=f_sensor[2]*150.0;
        if(f_sensor[2]>0)
                z_mot+=160;
        if(f_sensor[2]<0)
                z_mot-=100;

        if(z_mot>4095)
                z_mot=4095;
        if(z_mot<0)
                z_mot=0;

        setaxis(2);
        putmotordac(z_mot);
        updatedacs();
        return;
        }


/*************************************************************************************
                                        JOINTPOS()

                Subroutine which finds the joint positions, angles in
                radians [radians], shaft in [meters] at a given joystick configuration.
*************************************************************************************/
jointpos()

{
                /***** THETA position measured in radians *****/
setaxis(0);                             /* Theta is defined as positive rotation about y axis */
angle_pos[0]=getencoder();
theta_angle_index = 4095 - angle_pos[0]; /* to make positive theta in the right direction */
if (angle_pos[0]>2000){
        theta_angle_index += 2048;
        angle_pos[0]=-(angle_pos[0]-4096)*0.0003835;  /* (PI/2)/4096 gives the encoder
                                                         position resolution */
        }
else{
        theta_angle_index -= 2048;
        angle_pos[0]= -angle_pos[0]*0.0003835;        /* (PI/2)/4096 gives the encoder
                                                         position resolution */
        }
                /***** PHI position measured in radians *****/
setaxis(1);
angle_pos[1]=phi_angle_index=getencoder();  /* Phi is defined as positive rotation about x
                                               axis */
if (angle_pos[1]>2000){
        phi_angle_index = 6143 - phi_angle_index;           /* index for trig. function look
                                                               up table */
        angle_pos[1]=(angle_pos[1]-4096)* -0.0003835; /* (PI/2)/4096 gives the encoder
                                                         position resolution */
        }
```

```
else{
        phi_angle_index = 2048 - phi_angle_index;          /* index for trig. function look u
                                                            up table */
        angle_pos[1]=angle_pos[1]* -0.0003835;          /* (PI/2)/4096 gives the encoder
                                                         position resolution */
        }

        /****** The full physical range of shaft, R axis, is used ******/
        /****************       R is in units of meters      ****************/
setaxis(2);
R_old=R_temp;
angle_pos[2]=R_temp=getencoder();
if(angle_pos[2]-R_old>2000) z_over=0;          /* check for overflow */
if(angle_pos[2]-R_old<-2000) z_over=4096;     /* check for underflow */
angle_pos[2]=((6483-(angle_pos[2]+z_over))*0.0000470)+0.1651;
                        /* The + 0.1651 meters at the end of the above equation is for     */
                        /* half the gimble distance since the shaft can not move all the */
                        /* way to the center of rotation.(i.e. R is at least 0.1651 meters) */
return;
}




/************************************************************************************
                                TRANSFORM()
            Subroutine which calculates the Jacobian J, Jacobian inverse, and coordinate
            transform at a given joystick configuration.
                    NOTE: You need to run jointpos() before you call transform()
************************************************************************************/
transform()

        {

        float count, a;
        float tan_theta, tan_phi, tan_theta_2, tan_phi_2, tan_theta_2_1, tan_phi_2_1;
        float denom_1, numer_1, numer_2;
        int nloops, k;
                tan_theta = tan_func[theta_angle_index];     /* tangent of theta angle
                                                                located */
                tan_phi = tan_func[phi_angle_index]; /* tangent of phi angle located */
                tan_theta_2=tan_theta*tan_theta;
                tan_phi_2=tan_phi*tan_phi;
                tan_theta_2_1 = tan_theta_2 + 1;
                tan_phi_2_1 = tan_phi_2 + 1;

                a=tan_theta_2 + tan_phi_2_1;
                denom_1 = pow(a, -0.5);
                numer_1 = (angle_pos[2]*denom_1)/a;
                numer_2 = 1/(angle_pos[2]*denom_1);  /*careful when angle_pos[2] R=0 this is
                                                        infinite*/

                /* the following are the elements in the Jacobian matrix (3X3) */
                J[0][0] = numer_1 * tan_theta_2_1 * tan_phi_2_1;
                J[0][2] = tan_theta * denom_1;
                J[1][1] = -J[0][0];
                J[1][2] = -tan_phi * denom_1;
                J[2][0] = -tan_theta * tan_theta_2_1 * numer_1;
                J[2][1] = -tan_phi * tan_phi_2_1 * numer_1;
                J[2][2] = denom_1;
                J[0][1] = J[2][1] * tan_theta;
                J[1][0] = -J[2][0] * tan_phi;


                /* the following are the coordinate transformations to the X, Y, Z plane */

                xyz[0] = tan_theta/numer_2;     /* J_13 * R if J_13 is calculated already */
                xyz[1] = -tan_phi/numer_2;      /* J_23 * R if J_23 is calculated already */
                xyz[2] = 1/numer_2;             /* J_33 * R if J_33 is calculated already */

                return;
        }


/*************************************************************************************
                                ENDPOINT_VEL() ***
                Subroutine which calculates the endpoint velocity of the joystick for a set of
                desired actuator velocities at a given joystick configuration.
*************************************************************************************/
endpoint_vel()
{
int i,j;

        jointpos();             /* Get the joint positions for use in the Jacobian */
        transform();    /* Calculate the elements of the Jacobian */
        gettach();              /* Read in the joint velocities */

                /* now the endpoint velocities can be calculated */
        for(i=0;i<3;i++){
                end_vel[i]=0;           /* zero the velocity value before loading the new value
*/
                for(j=0;j<3;j++)
                        end_vel[i] += J[i][j] * angle_vel[j];
                }
        /************************* The above does the following********************/
        /*end_vel[0] = J[0][0] * angle_vel[0] + J[0][1] * angle_vel[1] + J[0][2] * angle_vel[2];*/
        /*end_vel[1] = J[1][0] * angle_vel[0] + J[1][1] * angle_vel[1] + J[1][2] * angle_vel[2];*/
        /*end_vel[2] = J[2][0] * angle_vel[0] + J[2][1] * angle_vel[1] + J[2][2] * angle_vel[2];*/
        /***************************************************************************/

        return;

}
/*************************************************************************************
                                JOINT_TORQUES() ***
                Subroutine which calculates the actuator torques of the joystick for a
                desired endpoint force at a given joystick configuration.
```

```
*************************************************************************************/
joint_torques()
        {
int i,j;

        jointpos();                /* Get the joint positions for use in the Jacobian */
        transform();    /* Calculate the elements of the Jacobian */

                /* now the endpoint velocities can be calculated */
        for(i=0;i<3;i++){
                torque[i]=0;      /* zero the velocity value before loading the new value */
                for(j=0;j<3;j++)
                        torque[i] += J[j][i] * force[j];
                }
        /************************ The above does the following   *******************/

        /*      torque[0] = J[0][0] * force[0] + J[1][0] * force[1] + J[2][0] * force[2];    */
        /*      torque[1] = J[0][1] * force[0] + J[1][1] * force[1] + J[2][1] * force[2];    */
        /*      torque[2] = J[0][2] * force[0] + J[1][2] * force[1] + J[2][2] * force[2];    */
        /*************************************************************************/

        return;
}

/*************************************************************************************
*               FUNCTION GETTACH()
*               function that reads in the actuator shaft rotational velocities from the
*               tachometers
*************************************************************************************/
gettach()
{
int i, time;

for(i=0;i<3;i++){
        setaxis(i);                             /* axis 0 for x, 1 for y, 2 for R */
        setmux(3);                              /* Get tachometer */
        delay(10);                              /* S/H and MUX settling rate is max 25
microseconds */
        angle_vel[i] = (getadc()-init_vel[i])/tach_gain[i];
}                                               /* theta and phi in rad/sec  R in m/sec */


return;
}

/*************************************************************************************
*                               FRICTION()
*       Coulomb firction measurement routine
*************************************************************************************/

friction()
{
int i;
frictionup_val=0;
```

```
frictiondown_val=0;

gettach();
while (angle_vel[2]>-0.1)
{
frictiondown_val++;
setaxis(2);
putmotordac(2048-frictiondown_val);
updatedacs();
gettach();
}
        setaxis(2);
        putmotordac(2048);
        updatedacs();
/*      printf("frictiondown= %d  vel=%f \n",frictiondown_val, angle_vel[2]);*/
        gettach();
        for (i=1;i<=1000;i++);
        while (angle_vel[2]<0.1){
                frictionup_val++;
                setaxis(2);
                putmotordac(2048+frictionup_val);
                updatedacs();
                gettach();
        }
        putmotordac(2048);
        updatedacs();
        /*printf("frictionup= %d  vel=%f",frictionup_val, angle_vel[2]);*/

}
/*****************************************************************************
*                          command_motor()
*       Force servo to command force for transmission axis, open loop on other axes
*****************************************************************************/

command_motor()
{
        int motor[3],i,Kf;

        Kf=2;

        for (i=0;i<2;i++) {
                motor[i] = (int)(torque[i] * k_motor_dac[i]) + 2048;
                setaxis(i);
                putmotordac(motor[i]);

                if(motor[i]>4095)
                motor[i]=4095;
        if(motor[i]<0)
                motor[i]=0;

        updatedacs();
        }
        get_forces();
        motor[2] = (int)((torque[2] * k_motor_dac[2]) * (Kf+1) + Kf*100. * f_sensor[2]) + 2048;

        if(motor[2]>2048)
                        motor[2]+=150;
                if(motor[2]<2048)
                        motor[2]-=100;

        if(motor[2]>4095)
                motor[2]=4095;
        if(motor[2]<0)
                motor[2]=0;

                setaxis(2);
                putmotordac(motor[2]);

                updatedacs();

}


/************************************************************************************
*                       I/O FUNCTIONS
************************************************************************************/
/************ axis x=0 y=1 and z=2   axis = 3 is for various control functions ********/
setaxis(axis)
int axis;
{
        asm
        {
                move.w                  axis,D0
                and.b           #0x03,D0
                asl.b           #4,D0
                and.b           #0xcf,PORTA
                add.b           D0,PORTA                /* Preserve other bits on port A */
        }
}
/***********************************************************************************/

delay(i)
int i;
{
        while(i--)  ;
        }
/*********************************************************************************/
/****   Pick one of the analog inputs ***********************************************/

setmux(inp)
int inp;
{
        asm
        {
                move.w          inp,D1
                move.b          PORTA,D0
                and.b   #0xf8,D0
```

```
            and.b   #0x07,D1
            add.w   D1,D0
            move.b          D0,PORTA                /* Preserve other bits on port A */
      }
}
/**************************************************************************************/

getadc()
{
int val;
      asm
      {
                  ori.b          #0x20,PORTC   /* -S/H high to hold input value */
                  andi.b         #0xb7,PORTA   /* -A/D CS low & A0 low for 12 bit
                                               conversion */
                  andi.b         #0x7f,PORTC   /* R/-C low to start conversion */
                  ori.b          #0x80,PORTC   /* Return R/-C high */

                  move.w                #60,D0
      @loop:      sub.w          #1,D0         /* Delay > 25 uS */
                  bne                   @loop

                  andi.b         #0xb7,PORTA   /* -A/D read low & A0 low to read
MSB */

                  move.b                PORTB,D0
                  and.w          #0xff,D0
                  asl.w          #4,D0

                  ori.b          #0x08,PORTA   /* A0 high to read LSB */

                  move.b                PORTB,D1
                  and.w          #0xff,D1
                  asr.w          #4,D1

                  add.w          D1,D0

                  ori.b          #0x40,PORTA            /* Restore -A/D read high */
                  and.b          #0xdf,PORTC            /* -S/H low to track input */
                  move.w                D0,val
      }
return(val);
}
/**************************************************************************************/

getencoder()
{
int      val;
      asm
      {
            and.b          #0x77,PORTA   /* Quad SEL input & RD input low (enables
                                         MSB onto bus) */

            move.b                PORTB,D0
```

```
                and.w           #0xff,D0
                asl.w           #8,D0                   /* Read MSB */

                ori.b           #0x08,PORTA/* Quad SEL input high (enables LSB onto bus) */
                move.b                  PORTB,D1
                and.w           #0xff,D1
                add.w           D1,D0

                ori.b           #0x88,PORTA             /* Quad SEL high and RD high */
                move.w                  D0,val
        }
        return(val);
}
/*****************************************************************************************/

resetencoders()
{
        asm
        {
                move.b                  PORTA,D0
                ori.b           #0x30,PORTA             /* Sets "axis" to 3 for resetting
quadrature */
                and.b           #0x7f,PORTA             /* Strobe quadrature read low */
                move.b                  D0,PORTA                /* Restore axis & quadrature
read high */
        }
}
/*****************************************************************************************/
updatedacs()
{
        asm
        {
                move.b                  PORTA,D0
                ori.b           #0x30,PORTA                             /* Sets "axis" to 3 for
updating dacs */
                and.b           #0xaf,PORTC                             /* Strobe D/A update
and write low */
                ori.b           #0x50,PORTC                             /* Strobe D/A update
and write high */
                move.b                  D0,PORTA                                /* Restore axis
value */
        }
}
/*****************************************************************************************/


putbrakedac(value)
int value;
{
asm
{
        move.b                  PORTA,D0
        move.b                  PORTC,D1
        move.w                  value,D2
        move.w          D2,D3                   /*D3 is a safe copy of VALUE*/
        and.b       #0xf8,D0/* Hold on to old bit values (note: probably not all 1's) */
        and.b       #0xf0,D1
        ori.b       #0x50,D1/* Hold on to old bit values (note: PROBABLY all 1's) */

        /*      LOW NIBBLE  FOR DAC */

        move.b          D0,PORTA                        /* Set D/A address 0 */
        and.b       #0xf,D2                             /* Low nibble */
        add.w       D1,D2                       /*D2 is tempc1*/
        move.b          D2,PORTC
        move.w          D2,D4                           /*D4 is a safe copy of tempc1*/
        and.b       #0xef,D2                    /* Strobe D/A write low */
        move.b          D2,PORTC
        ori.b       #0x10,D4                    /* Strobe D/A write high */
        move.b          D4,PORTC
        asr.w       #4,D3                       /* Shift to middle nibble */

        /*      MIDDLE NIBBLE FOR DAC */

        add.b       #1,D0                       /* Set D/A address 1 */
        move.b          D0,PORTA
        move.w          D3,D2                           /*D2 is a copy of VALUE*/
        and.b       #0x0f,D2                    /* Middle nibble */
        add.w       D1,D2
        move.b          D2,PORTC
        move.w          D2,D4
        and.b       #0xef,D2                    /* Strobe D/A write low */
        move.b          D2,PORTC
        ori.b       #0x10,D4                    /* Strobe D/A write high */
        move.b          D4,PORTC
        asr.w       #4,D3                       /* Shift to high nibble */

        /*      HIGH NIBBLE FOR DAC */

        add.b       #1,D0                       /* Set D/A address 2 */
        move.b          D0,PORTA
        move.w          D3,D2                           /*D2 is a copy of VALUE*/
        and.b       #0x0f,D2                    /* High nibble */
        add.w       D1,D2
        move.b          D2,PORTC
        move.w          D2,D4
        and.b       #0xef,D2                    /* Strobe D/A write low */
        move.b          D2,PORTC
        ori.b       #0x10,D4                    /* Strobe D/A write high */
        move.b          D4,PORTC
}
}
/**********************************************************************************/


putmotordac(value)
int value;
{
```

```
asm
{
        move.b                  PORTA,D0
        move.b                  PORTC,D1
        move.w                  value,D2
        move.w                  D2,D3                              /*D3 is a safe copy of value*/
        and.b           #0xf8,D0                /* Hold on to old bit values (note: not all 1's) */
        add.b           #0x04,D0
        and.b           #0xf0,D1/* Hold on to old bit values (note: PROBABLY all 1's) */

        /*      LOW NIBBLE  FOR DAC */

        move.b                  D0,PORTA                           /* Set D/A address 0 */
        and.b           #0xf,D2                                    /* Low nibble */
        add.w           D1,D2                          /*D2 is tempc1*/
        move.b                  D2,PORTC
        move.w                  D2,D4                              /*D4 is a safe copy of tempc1*/
        and.b           #0xef,D2                       /* Strobe D/A write low */
        move.b                  D2,PORTC
        ori.b           #0x10,D4                       /* Strobe D/A write high */
        move.b                  D4,PORTC
        asr.w           #4,D3                          /* Shift to middle nibble */

        /*      MIDDLE NIBBLE FOR DAC */

        add.b           #1,D0                   /* Set D/A address 1 */
        move.b                  D0,PORTA
        move.w                  D3,D2
        and.b           #0x0f,D2                /* Middle nibble */
        add.w           D1,D2
        move.b                  D2,PORTC
        move.w                  D2,D4
        and.b           #0xef,D2                /* Strobe D/A write low */
        move.b                  D2,PORTC
        ori.b           #0x10,D4                /* Strobe D/A write high */
        move.b                  D4,PORTC
        asr.w           #4,D3                   /* Shift to high nibble */

        /*      HIGH NIBBLE FOR DAC */

        add.w           #1,D0                   /* Set D/A address 2 */
        move.b                  D0,PORTA
        move.w                  D3,D2
        and.b           #0xf,D2                            /* High nibble */
        add.w           D1,D2
        move.b                  D2,PORTC
        move.w                  D2,D4
        and.b           #0xef,D2                /* Strobe D/A write low */
        move.b                  D2,PORTC
        ori.b           #0x10,D4                /* Strobe D/A write high */
        move.b                  D4,PORTC
}
}
```