To the Graduate Council,

I am submitting herewith a thesis written by Angela Michelle Gizelar entitled "Modeling and Control of a Hydraulically Actuated Flexible Robot Link of Variable Length." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

Rajiv Dubey, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

MODELING AND CONTROL OF A HYDRAULICALLY ACTUATED FLEXIBLE ROBOT LINK OF VARIABLE LENGTH

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Angela Michelle Gizelar

December 1995

Acknowledgements

I would like to thank Dr. Reid Kress for helping me settle on a topic and for his patience during my research. I would like to thank my major professor, Dr. Rajiv Dubey, and the other committee members, Dr. William Hamel and Dr. Grzegorz Kawiecki for their helpful suggestions. I would also like to thank my family and friends for their assistance, both emotional and financial, over the past three years. This research has been supported by Lockheed Martin Energy Systems, Inc. contract number MARTIN MARPO11X-SN603V.

Abstract

Through research in and production of radioactive materials the Department of Energy (DOE) has been generating toxic waste that was commonly stored in large, underground storage tanks. Long-reach manipulators (LRMs) will be used to remove waste from the tanks. LRMs exhibit flexible behavior and present control problems that are different from those seen with conventional manipulators. Because of the large payload requirement for removing tank waste, LRMs will use hydraulic rather than electric actuators. To explore the behavior of LRMs the Flexible Beam Test Bed (FBTB) was constructed at Pacific Northwest Laboratory (PNL). This thesis describes research performed on a simulation of the PNL FBTB at Oak Ridge National Laboratory (ORNL). The simulation includes dynamic properties of the FBTB, a PD controller with robust notch filter for control of the base actuator, and a model of the hydraulic actuator. The beam was rotated by the base actuator while being either held at a fixed length or varied in length. For each of these tests the controller was tuned for two different configurations. The results of the simulation experiments are presented in plots of actuator position versus time and actuator torque versus time. Rise time, settling time, and overshoot were calculated for each move. These performance characteristics were used to find the best control scheme for a given beam configuration.

Table of Contents

Table of Contents, cont.

List of Figures

Chapter 1

Introduction

This first chapter will give the background of the tank waste problem that motivates research into long-reach manipulators (LRMs). A review of literature concerning LRMs and related topics is presented, as well as a description of the remaining chapters of this thesis.

1.1 Background

Since its inception one of the primary missions of the Department of Energy (DOE) has been the production of and research in radioactive materials. Each DOE facility working with radioactive materials generated by-products that were also radioactive. One common disposal method for these by-products was storage in large underground storage tanks or above-ground silos. This disposal method was intended to be temporary and the tanks were designed to last for 20 to 50 years. The older storage tanks are of single-shell steel construction. Many of these storage tanks have been in use in excess of their expected life cycles and have begun to leak. Current remediation efforts are concentrated on the waste in these tanks and especially the tanks that have already developed leaks [Bills, 1994].

The method of remediation is constrained by the size of the storage tanks, the available access into the tanks, and the variety of waste within the tanks. The tanks range up to 80 feet in diameter and 35 feet in height. Access to the inside of the tanks is available through a limited number of holes in the dome-shaped tops. These holes range from 12 to 42 inches in diameter. The waste inside the tanks is in various forms, including liquid, sludge, crust, and hardpan. Some of the waste can be easily removed, either directly or by being sluiced with water and pumped out. Other waste has dried sufficiently hard that removal must be facilitated by a high pressure water jet. Because of the large size of the tanks and the probable difficulty of extracting the waste, a long-reach, high-capacity manipulator will be used for the remediation task [Jansen,1991a].

A long-reach manipulator presents problems that are different from a typical robotic manipulator. To explore these problems, an experimental test bed was constructed at the Pacific Northwest Laboratory (PNL). This test bed consists of a long, flexible beam with a hydraulic actuator at one end and a Schilling manipulator at the other end, as shown in Figure 1. The actuator rotates the beam parallel to the floor. The end of the beam with the Schilling manipulator is supported by an air bearing. A graphical simulation of the test bed was written to test different control schemes at ORNL.

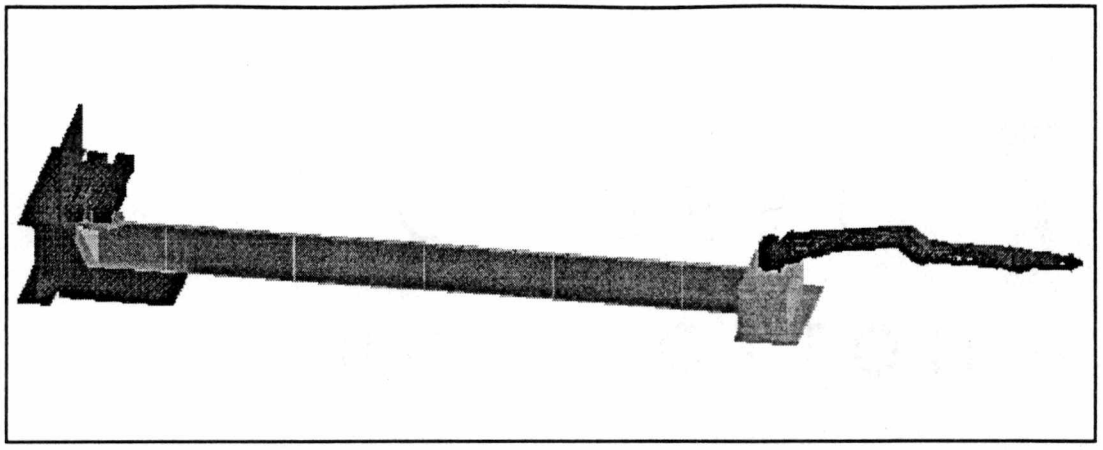


Figure 1: Flexible Beam Test Bed

The graphical simulation was written using TeleGRIP software and run primarily on a Silicon Graphics Onyx workstation. The simulation includes the dynamic properties of each element in the test bed as well as the controller for the actuator. The simulation is used to test different actuator controllers because implementation of controllers on the actual hardware can be time consuming, especially if it is necessary to travel a long distance to the location of the test bed. Also, if the controller being tested causes some unexpected behavior in the manipulator, it is preferable to discover this on a simulation rather than on the hardware. A simulation also offers the opportunity to vary the parameters of the system, such as geometry or dynamic properties, and see the effects of these changes. Besides controller modelling, a simulation can be used for motion planning and operator training.

1.2 Literature Survey

The scope of this thesis includes several topics that have been covered extensively in the literature. The papers surveyed here fall into three categories: long-reach/flexible manipulator papers, hydraulic system papers, and variable length beam papers.

1.2.1 Long-Reach/Flexible Manipulator Papers

The problem of tank waste retrieval has generated interest in manipulators with a large aspect ratio, or long-reach manipulators (LRMs). Several publications from ORNL have dealt with possible designs of LRMs, the problems unique to LRMs, and methods of preventing vibration in LRMs. Some of these papers are summarized here, with an emphasis on avoiding structural vibration.

The design of a long-reach manipulator is discussed in [Kwon, 1993]. Several kinematic configurations are examined, and the addition of an extra roll degree of freedom (DOF) gives several advantages including damping. The mixed configurations between elbow and SCARA will allow the LRM to be configured to better control the vibration in any direction. The different kinematic configurations will produce varying natural frequencies. The highest natural frequency is provided by the telescopic sequential entry type at 1 to 2 Hz. The lowest natural frequency is provided by the folded entry type at 1 to 1.5 Hz. A brief study of the dynamics of an actuator of the first pitch joint of a LRM was conducted. Based on linearized models, the lowest open-loop natural frequency is about 0.5 Hz, which comes from the joint compliance due to the compressibility of the hydraulic fluid. Utilizing position, velocity, and differential

pressure feedback, the closed loop actuator bandwidth could be increased to about 2 Hz. The practical actuator-load bandwidth is likely to be 0.5 to 1 Hz, which is less than that obtained with a small-signal, linear analysis.

The problem of a long-reach manipulator is also discussed in [Jansen, 1991a]. The problem of a long-reach manipulator is much different than that of a conventional manipulator. Energy storage for the long-reach manipulator is distributive, and partial rather than ordinary differential equations are necessary for the analysis. A long manipulator will have a very low structural frequency and therefore a low bandwidth. Expressions are developed for beam deflection and natural frequency. Because the long manipulator will have some vibration damping is an important topic, and experimental work in this area is reviewed. The requirements for the design of the manipulator and its controller will be solidified when the task description is more complete.

The design of a controller for a long-reach, flexible manipulator is discussed in [Kwon, 1994] and [Kwon, 1995]. In both papers the PNL flexible beam test bed is described along with its MICA control system.

In [Kwon, 1995] filtering methods for controlling vibration in flexible beams are examined. The flexible beam is modeled using the assumed mode method. The types of filtering examined are an impulse shaping filter, a robust notch filter, a feedforward simulation filter, and a fuzzy shaping method. The impulse shaping filter requires knowledge of the frequency and damping ratio of the dominant vibration mode, and introduces zeros where the system's dominant closed-loop poles exist. The robust notch filter requires knowledge of only the frequency of the dominant vibration mode, and adds

zeros at the damped resonant frequency and introduces critically damped poles at the low-pass filter natural frequency. Both the impulse shaping filter and robust notch filter give good performance but have a tracking delay problem for slow systems. The feedforward simulation filter offers the higher bandwidth of end-position feedback with the stability of conventional proportional-derivative joint feedback. A feedforward torque loop is added to improve tracking. A robust notch filter is used to generate a joint trajectory. This method gives very good tracking performance with high computational overhead. The fuzzy shaping method can be used when the dynamics of the system are not precisely known. The joint trajectory is modified by fuzzy rules using the acceleration profile. The modified joint trajectory is a combination of the output of the fuzzy shaping method and the original trajectory. The modified joint trajectory greatly improves the performance of the proportional-derivative controller alone.

In [Kwon, 1994], the details of Kwon's controller are discussed. Different filtering methods are discussed and the choice of the feedforward simulation filter is explained. The experimental results showed good tracking of joint trajectory and end-position trajectory. The feedforward simulation approach offers excellent tracking performance at the expense of knowledge of the flexible dynamics. For cases when the manipulator will vary its configuration or payload, it is suggested that the shaping filter be adapted by the use of a real-time fast Fourier transform.

The simulation of the flexible beam test bed is discussed in [Bills, 1994]. The creation of the simulation in IGRIP is described along with the dynamic properties assigned to the flexible beam. The controller is implemented through the shared library

and the PID gains are tuned to reflect the system's servo bandwidth and the damping ratio. A robust notch filter and torque feedforward loop are used to improve stability and tracking. The default PID controller embedded in IGRIP is compared to the improved PID controller with a shaping filter. Using the default PID controller the simulation took 70 seconds to reach its desired position and there was significant overshoot and oscillation. Using the improved PID controller the simulation took 20 seconds to reach its desired position and there was little overshoot.

The papers presented in this section stress the importance of considering damping and vibration control in the design of a LRM and its controller.

1.2.2 Hydraulic System Papers

The subjects of the papers covered here include the design of hydraulic systems for various applications, and modeling methods for hydraulic systems. The hydraulic equations used in this thesis are taken from [Merritt, 1967].

The design of a hydraulic system is covered in [Salcudean, 1994]. A motion simulator in the form of an inverted Stewart platform suspended from the ceiling is described. The inverse dynamics of the simulator are used to determine the geometry and specify the actuators that would satisfy given stroke, velocity, and acceleration requirements. Rexroth proportional valves were used instead of servo valves because of cost. These valves give nearly linear response and 35 Hz bandwidth at full signal. An actuator with a 1.5 inch bore is chosen because it provides the needed force over an acceptable range of velocities without buckling. Teflon seals are used on the cylinders to prevent the operator from

feeling any "stick-slip" motion. A lag-type controller with a pre-filter is used for each cylinder. The control rate is 200 Hz. Several safety precautions were designed into the system, including fail-safe isolation valves between the pilots and the main-stage valve spools.

[Sepehri, 1994] also presents the design of a hydraulic system. A hydraulic control system is applied to the resolved-mode control of an excavator. The control scheme consists of a load compensator, a flow constrainer, and a closed-loop controller. The load compensation algorithm selects the correct valve orifice, pump to actuator or actuator to tank, to control based on the direction of the net force on the manipulator relative to the desired velocity. The load compensator was tested for single link control in simulation and experiment with a control rate of 50 Hz. The results were satisfactory. The flow constrainer algorithm determines the valve flow to the actuators for multiple link velocity control. It uses the total pump supply and actuator requirements to maintain the desired end-effector trajectory. The flow constrainer was tested in simulation and experiment with a control rate of 25 Hz. The maximum error, due to drift, was less than four percent. The closed-loop control is included to minimize the effects of unmodeled dynamics, friction, and other uncertainties. The combination of the load compensator and flow constrainer, called the Decoupled-Flow Compensator, is able to produce satisfactory results without feedback control.

The development of a hydraulic model is covered in [McLain, 1989]. The hydraulic system consists of a single-stage, four-way suspension valve coupled to a linear piston actuator. The equations that model the dynamic response of the system are derived from

a bond graph model. The third-order lumped parameter model for the valve includes the inductance, resistance, and torque constant for the motor portion of the valve. The model includes the mass, stiffness, and damping of the valve pipe. Nonlinear elements in the model include hysteresis and saturation of the valve motor. The hysteresis is modeled by calculating motor torque as a function of current. An equation modeling flow forces was implemented with reasonable correlation between the model and experimental results. The effects of coulombic and viscous friction in the piston seals are modeled. The equations for the orifice areas are derived from equations describing the area of the segment of a circle as a function of its fundamental dimensions. These equations allow for variable port diameters and lapping, and incorporate the clearance between the valve pipe tip and the receiver plate. The control structure is simple linear control with three feedback loops: inner force feedback, velocity feedback, and outer position feedback. For the computer simulation, the model equations and nonlinearities are coded into the ACL simulation language. The differential equations are solved with Gear's stiff integration method. The valve model is verified by performing tests to collect data to use in the model. Small adjustments are made to gain better correlation between the model and the test data. The system model is verified by using the same step input and initial conditions for the model and test systems. Large differences are found and eliminated through iterations and debugging. Good correlation is found between the model and test data for current input, valve position, piston position, load position, load velocity, and force in the compliant member. The time delay for piston velocity is due to filtering of the piston position signal.

A method for modeling hydraulic systems is also described in [Kleinsteuber, 1994]. A polynomial network modeling technique is used to model hydraulic actuation systems used in heavy-duty machines. The polynomial networks are derived using a software package called Abductory Induction Mechanism (AIM) that generates C source code to be inserted into the simulation program. This method of modeling does not require the manipulation of state space variables, and requires only easily obtained experimental measurements to completely model the hydraulic system. The model is implemented by bypassing the derivation of the pump pressure and modeling the entire range of flow into and out of the actuator. The final approach used two modular inductive networks to eliminate the derivation of the orifice values. The results of the AIM model are compared to those of an iterative model. The AIM model produces very similar results and reduces the computational load by 86.7 percent.

1.2.3 Variable Length Beam Papers

Because a LRM will probably undergo axial motion during deployment, the subject of variable length beams is explored. The papers surveyed here offer mathematical models of the dynamics of a beam is axial motion or axial and rotational motion. Experimental results are also presented.

A beam of variable length, or with axial motion, is studied in [Yuh, 1991]. The beam has rotational as well as translational motion. One end is clamped and the other end is free. Newton's second law is used to derive a partial differential equation to describe the beam deflection, similar to [Tabarrok, 1974]. An approximate dynamic model for the

lateral motion of the beam is derived for the design of a multivariable control system. A number of experimental tests were run with various axial and rotational motions, and data on tip-deflection was collected. There is good correlation between experimental and computer simulation results. Additional tests were run using the computer simulation showing the effect of beam rotation is significant only in combination with fast axial motion, and not with slow axial motion.

A variable length flexible beam is also considered in [Suzuki, 1994]. An experimental model of a flexible beam with both rotational and translational motion is described. A dynamic model of this apparatus is derived using the Lagrangian method. The tip acceleration is measured for different fixed and varying beam lengths. The beam frequency for various beam lengths is determined experimentally and compared to the theoretical values. The frequency of vibration of the beam is characterized well by the dynamic model.

The dynamics and control of a translating flexible beam with a tip mass is presented in [Tadikonda, 1992]. An experimental set up consisting of a flexible link that slides in and out of a rigid support is described. The eigenfunctions of a fixed-free beam are used as the trial functions to derive the dynamics. The equations of motion are obtained using Lagrange's equations. The equation derived for flexible motion reduces to that in [Tabarrok, 1974] in the absence of an end mass. The beam is extended and retracted with and without an end mass, moving 1.0 meters in 20.0 seconds. The amplitude of the flexible motion increased during extension and decreased during retraction. The

frequencies of the vibrations decrease as the end mass increases. The results demonstrate that the effect of the elastic motion during translational motion is substantial.

1.3 Description of the Thesis Topic

This thesis will describe in more detail the simulation of the flexible beam test bed and a modification of the controller written by D.S. Kwon. . Also, the problem of a variable length flexible manipulator will be addressed.

The hydraulic actuator controller described in [Kwon, 1995] is designed to compensate for actuator dynamics, increase bandwidth, and reduce structural vibrations. The dynamics of the high-capacity hydraulic actuator are compensated by using a load-compensated velocity feedforward loop. This allows good tracking with the actuator. A pressure feedback loop is used to increase the bandwidth. In order to prevent the excitement of the resonant frequency of the system, the command trajectory is shaped with filtering methods [Kwon, 1994]. The dynamics of the actuator were not modelled in Kwon's controller. A model of the hydraulic actuator dynamics was added to the controller to more accurately represent the behavior of the system. The integration iteration used in the hydraulic model was nested in the controller loop of the flexible beam simulation.

An investigation of a variable length flexible manipulator is included in this thesis because during tank waste removal operations the inertia of the system will change because the configuration and loading of the manipulator will change. The manipulator may be shortest in length while it is being positioned over the tank opening and extend

in length as it reaches into the tank. When the manipulator is pulling waste from the tank its payload will vary substantially. The beam length is varied by altering the mass properties of the beam through customized code embedded in the simulation.

These topics will be explored in the next four chapters. The second chapter will describe the simulation created to test the behavior of the flexible beam test bed. The third chapter will present the formulation of hydraulic equations used in the simulation. The results from the simulation tests will be presented in the fourth chapter, and conclusions will be made in the fifth chapter.

Chapter 2

Building the Simulation

In this chapter the flexible beam test bed and its simulation will be described, and the features of the simulation software and its implementation will be explained. The control law used to command the hydraulic actuator will be derived using equations from [Merritt, 1967].

2.1 Simulating the PNL Flexible Beam Test Bed

The flexible beam test bed was built at PNL to study the behavior of and control problems associated with long and flexible manipulators. The test bed is a one degree of freedom (DOF) system consisting of a flexible beam with a hydraulic actuator at the base. The steel flexible beam is 4.17 meters long, 0.30 meters tall, 1.91 centimeters thick, and weighs approximately 187.7 kilograms. At the end of the beam a 70 kilogram Schilling Titan II arm is mounted on a 170 kilogram platform. The platform is supported by an air bearing and ensures planar motion of the beam. The base actuator is a rack and pinion type rotary hydraulic actuator manufactured by Flo-Torque. A Parker ST10-5 servo valve (five gallons/minute at 1000 psi) was used with a Parker BD90 servo valve amplifier.

A simulation of the PNL FBTB was written using TeleGRIP software [Deneb, 1994].

This simulation allowed different controllers to be tested without the actual hardware. The simulation displays a graphical representation of the FBTB, contains the dynamic properties of the beam, and calls the controller for the actuator. The dynamic properties, mass, mass moment of inertia, and centroid are calculated by TeleGRIP using the specified dimensions and density of the part. The simulation can compute and display the position, velocity, acceleration, and torque of any manipulator joint.

To model the flexible behavior of the beam in the simulation, the simulated beam was divided into six rigid sections. The section at the base of the beam is 0.42 meters long, the four sections in the middle of the beam are 0.83 meters long, and the section at the end of the beam is 0.42 meters long. All of the sections are 0.3 meters tall and 1.91 centimeters thick. Between each section a torsional spring and damper were placed, forming five additional "joints" in the beam. A spring constant of 39.5 KN-m/rad and a damping coefficient of 90.4 N-m/rad/s were used between each of the sections. It is evident that the value of this chosen spring constant is reasonable when the equivalent spring constant of a cantilever beam is considered. A force acting at the end of a beam in a direction perpendicular to the length of the beam generates a torque at the base of the beam that can be equated to the torque caused by a torsional spring at the base of the beam with an angular deflection sufficient to cause the end point deflection produced by the applied force. The equation for this equivalent spring constant,

$$k_\theta = \frac{3EI}{L}, \tag{1}$$

is derived in many vibrations textbooks. The equivalent spring constant of a 0.83 meter

long beam of the same material and cross-section as the simulated beam ($E = 2.07 \times 10^{11}$ N/m^2, $I = 1.74 \times 10^{-7}$ m^4) is 131.1 KN-m/rad. For a 4.17 meter long beam, the equivalent spring constant is 26.2 KN-m/rad. The simulated beam is 4.17 meters in length and is made of segments that are 0.83 meters in length. This combination of segments should have a spring constant that is smaller than that of an individual segment but larger than that of a full length beam. As expected, the spring constant used between each segment in the simulation, 39.5 KN-m/rad, does fall between the value of 131.1 KN-m/rad for the segment and 26.2 KN-m/rad for the entire beam. The values for the spring constant and damping coefficient used in the simulation were obtained by adjusting these values until the first and second natural frequencies of the simulated beam matched those of the actual PNL FBTB.

2.2 Implementing Customized Code

In addition to solving the kinematics for and assigning dynamic properties to a model, the TeleGRIP simulation can utilize customized code. The software is especially designed to facilitate user-written controllers. The code, written in C, is added to the dynamic shared object library that is provided by TeleGRIP. Detailed instructions on adding user-written controllers to the library are given in Appendix B. For this simulation, a PD controller with a robust notch filter was included and code to model the hydraulic actuator was written. The TeleGRIP program and the customized code communicate information as shown in Figure 2. In Figure 2, τ is the torque computed by the hydraulic model. J is the inertia, B is the damping factor, and K is the stiffness of the system modeled in

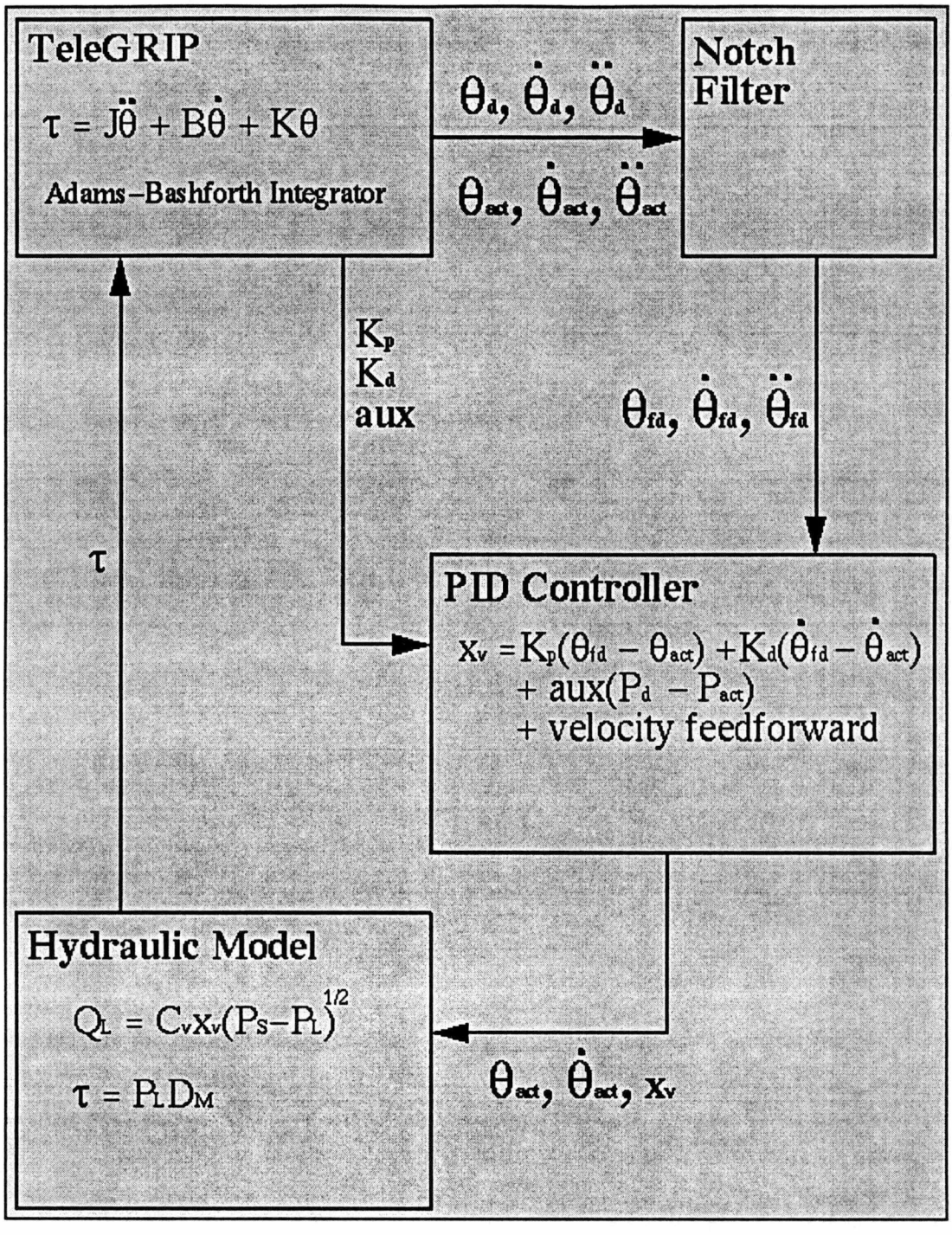


Figure 2: Diagram of the Embedded Code Used in the Simulation

TeleGRIP. θ_{act}, $\dot{\theta}_{act}$, and $\ddot{\theta}_{act}$ are the actual joint position, velocity, and acceleration of the model. θ_d, $\dot{\theta}_d$, and $\ddot{\theta}_d$ are the desired joint position, velocity, and acceleration generated by TeleGRIP's trajectory planner. θ_{fd}, $\dot{\theta}_{fd}$, and $\ddot{\theta}_{fd}$ are the filtered desired joint position, velocity, and acceleration computed by the robust notch filter. K_p, K_d, and aux are controller gains provided by the user, and x_v is the commanded servo valve position. P_d and P_{act} are the desired and actual hydraulic system pressures, and P_S and P_L are the supply and load pressures. Q_L is the load flow in the hydraulic system, D_M is the volumetric displacement of the motor, and C_v is the valve coefficient.

The TeleGRIP model uses a Hamiltonian dynamics formulation and an Adams-Bashforth integrator to solve for the joint values of the system using a given torque. Through the graphical user interface, the user gives the input command for the new position of the manipulator model (e.g. a step command to a new angular location). A trajectory is planned by TeleGRIP according to the input command, and the new joint values are sent to the notch filter. The notch filter used is the one described in [Bills, 1994], which uses a cascade of two notch filters to remove inputs that would excite the first two resonant frequencies. The notch filter sends its filtered joint values to the PD controller at each time step. The controller also receives the controller gains from TeleGRIP. The controller calculates the desired position of the valve and sends this value to the hydraulic model. The hydraulic model uses the requested valve position and the actual joint position and velocity to calculate the torque generated by the hydraulic actuator. This torque is sent back to TeleGRIP to calculate the new joint values. The

graphical representation of the model is updated according to the new joint values calculated by the TeleGRIP dynamics routine. The specifics of the PD controller are described later in this chapter. The hydraulic model is described in more detail in the third chapter.

2.3 Implementing the Controller

The PD controller used in this simulation is described in [Kwon, 1995]. It is designed to improve the tracking performance of the hydraulic actuator by increasing bandwidth and increase the stability of the controller. The controller consists of a proportional term, a derivative term, a velocity feedforward term, and a pressure feedback term. The velocity feedforward term is included because of the nonlinear relationship between the valve position and the actuator torque. This nonlinear relationship may cause a given valve opening to result in an unexpected torque command. To obtain the velocity feedforward term which is used to improve controller bandwidth, the desired valve opening, x_v, is expressed in terms of the desired flow rate, Q_{Ld},

$$x_{vd} = \frac{Q_{Ld}}{C_V \sqrt{P_S - P_L}}, \tag{2}$$

and the load pressure is calculated by the equation

$$P_L = \frac{\tau}{D_M}. \tag{3}$$

The desired flow rate is expressed in terms of the desired joint velocity:

$$Q_{Ld} = D_M \dot{\theta}_{fd} + C_{tm} P_L + \frac{V_t}{4\beta_e} \dot{P}_L \,, \tag{4}$$

where C_{tm} is the total leakage coefficient for the motor, V_t is the total volume of the motor, and β_e is the bulk modulus of the fluid. For simplicity the $\frac{V_t}{4\beta_e} \dot{P}_L$ term of equation (4) is neglected because the fluid is assumed to be compressed, and the $C_{tm}P_L$ term is neglected because leakage flow is typically small compared to the $D_M\dot{\theta}_{fd}$ term. Equations (3) and (4) are substituted into equation (2) to obtain the velocity feedforward term,

$$x_{vd} = \frac{D_M \, \dot{\theta}_{fd}}{C_v \sqrt{P_S - \frac{\tau}{D_M}}} \,. \tag{5}$$

For the pressure feedback term, which is used to improve controller stability, the desired and actual pressures are calculated as

$$P_d = \frac{J\ddot{\theta}_{fd}}{D_M} \,, \tag{6}$$

and

$$P_{act} = \frac{J\ddot{\theta}_{act}}{D_M} \,. \tag{7}$$

The command that the controller sends to the servo valve is

$$x_v = K_p(\theta_{fd} - \theta_{act}) + K_d(\dot{\theta}_{fd} - \dot{\theta}_{act}) + x_{vd} + aux(P_d - P_{act}) \; . \tag{8}$$

The controller gains are entered by the user into a pop-up window provided by the TeleGRIP user interface. For this simulation, the gains were adjusted until the best performance from the controller was obtained. Two different sets of gains were found, one set for the original beam length of approximately four meters and another set for the case when the beam is retracted to approximately two meters. The notch filter was also adjusted for each of the two cases to prevent the excitement of the first and second resonant frequencies of the particular beam length. For a beam length of four meters, the gains used are K_p = 268 volt/rad, K_d = 3 volt-sec/rad, and aux = 2.6×10^{-8} volt-m^2/N. For a beam length of two meters, the gains used are K_p = 780 volt/rad, K_d = 2.2 volt-sec/rad, and aux = 4.0×10^{-8} volt-m^2/N. The code for the PD controller is provided in Appendix_A.

2.4 Varying the Beam Length

During deployment into a waste storage tank, a long-reach manipulator will be required to change its length to perform tank entry or waste removal operations. Varying length introduces a number of problems not necessarily associated with fixed length manipulators. For example, inertia varies significantly and, more importantly, frequencies and mode shapes vary as length changes. This section will describe how the FBTB

simulation was modified to allow the flexible beam to vary in length. The actual FBTB at PNL has only one DOF and does not vary in length, but studying a simple two DOF system with rotating and prismatic joints is the simplest case. The PNL FBTB was selected as a first case example. Future efforts will focus on similar two DOF systems being developed and constructed at ORNL.

2.4.1 Adding a Prismatic Joint

A prismatic joint was added to the simulation of the flexible beam to allow translational motion. The joint was placed in the center of the beam so that the beam could be retracted to about 2 meters in length or extended to its full length of about 4 meters. Being able to vary the length by a factor of two was determined to be representative of LRMs being planned for waste clean up [Kress, 1995]. To prevent centrifugal force from moving the prismatic joint out while the beam is rotating about its base, the friction value of the prismatic joint was set very large to act as a brake. When the base joint was commanded to rotate the beam, the prismatic joint was commanded to either extend or retract the beam. Because of the motion planner used by TeleGRIP the motions of the rotational and prismatic joints must be completed at the same time. This prevents the rotational joint from moving at its maximum speed and does not allow the velocity of the translational motion to be varied. Due to these limitations this method of varying the beam length was not used. In addition, there was no interest in studying the control of the prismatic joint.

2.4.2 Shifting the Mass of the Beam

Rather than moving a section of the beam with a joint, translation of the beam can be effected by shifting the mass of the beam. This can be done easily using the dynamic properties structure provided by TeleGRIP's shared library. This structure contains the dynamic properties of a part, including density, mass, centroid location, and mass moments of inertia. By modifying the mass, mass moment of inertia, and centroid location of various parts in the simulation, the beam can be "translated."

When the beam is at its full length of about 4 meters, the mass is distributed as shown in Figure 3a, where m = 18.8 kg and M = 37.6 kg. When the beam is retracted to a length of 2 meters, the mass is distributed as shown in Figure 3b. This provides a good model of the link translating into itself as in a telescoping or folding system.

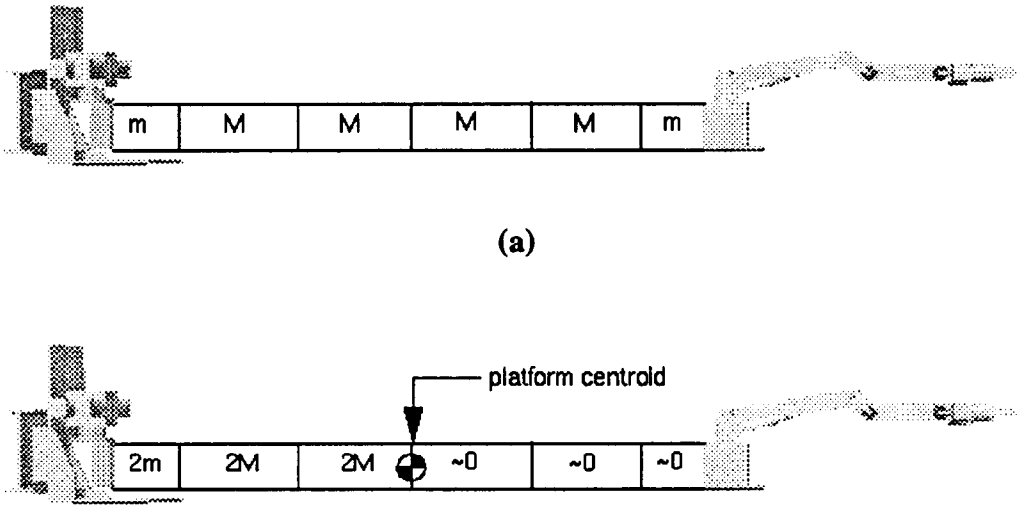


Figure 3: Mass Distribution of the Flexible Beam: (a) For a 4 Meter Long Beam, (b) For a 2 Meter Long Beam.

For the two meter long beam the centroid of the platform and Schilling arm is placed two meters from the base of the beam. In the graphical representation of the beam the platform and Schilling arm appear in the same location, but their mass acts at the location shown in Figure 3b. In Figure 3b the sections at the end of the beam have nearly no mass but not zero mass because TeleGRIP's dynamic analysis cannot accept a part with zero mass. The mass of these end sections is about 0.01 kg.

The beam is translated from four to two meters or from two to four meters by gradually shifting the mass from the configuration in Figure 3a to that in Figure 3b or vice versa. When the beam is translated from two meters to four meters, mass is first removed from the first section at the base of the beam and added to the fourth section of the beam. When the first section has been reduced from 37.6 kg to 18.8 kg then mass is removed from the second section. When the fourth section has been increased from nearly zero to 37.6 kg then mass is added to the fifth section of the beam. As the beam mass is being shifted the position of the platform centroid is also moved. The centroid of the platform is moved along the length of the beam as the beam is "extended." When the beam is translated from four meters to two meters the mass is shifted similarly. Mass is first removed from the sixth section at the end of the beam and added to the third section. When the mass of the sixth section has been reduced from 18.8 kg to nearly zero then mass is taken from the fifth section of the beam, et cetera.

The time taken to translate the beam can be varied. The mass of the beam is shifted based on the cycle time of the simulation. The cycle time is updated every one-tenth of a second, so the mass can be moved in discrete units every one-tenth of a second. To

translate the beam more slowly and continuously, less mass is shifted at one time.

As the length of the beam changes the inertia of the system used for the pressure feedback calculation in equations (6) and (7) also changes. The inertia of the system used in equations (6) and (7) is recalculated as the beam length is changing. The value of the inertia is updated for beam lengths of two, three, and four meters.

Chapter 3

Modeling the Hydraulic Actuator

In this chapter the model of the base hydraulic actuator of the PNL FBTB is described. A model of the actuator is included in the simulation to make the behavior of the system more realistic and to allow hydraulic parameters such as pressure and torque to be used in the controller calculations. Included in the model description are the valve equation, the Runge-Kutta formulation used to calculate the load pressure, and other nonlinearities associated with the actuator.

3.1 Calculating the Load Flow

The hydraulic model receives the commanded valve position from the controller and the current joint values from TeleGRIP for use in its calculations. The load flow for the actuator is calculated by the nonlinear equation

$$Q_{L[i]} = C_v x_v \sqrt{P_S - P_{L[i]}} \,, \tag{9}$$

where C_v is the overall valve coefficient, x_v is the commanded valve position, P_S is the supply pressure and P_L is the load pressure, or by the linearized equation

$$Q_{L[i]} = K_q x_v - K_c P_{L[i]} , \tag{10}$$

where K_q is the valve flow gain and K_c is the valve flow pressure coefficient. Equations (9) and (10) were each tested in the simulation, and the nonlinear equation (9) was used for experiments with the simulation. For equation (9) the overall valve coefficient is calculated from the flow equation by assuming that all other parameters are at their maximum values. For this system, the values would be: $Q_L = 3.1545 \times 10^{-4}$ m^3/sec, x_v = 10 volts, $P_S - P_L = 6.895 \times 10^6$ N/m^2, so C_v would be 262.58 m^2(N)$^{1/2}$/volt-sec. The supply pressure used is 2.07×10^7 N/m^2. In previous studies ([Kwon, 1994], [Kwon, 1995]) the load pressure was a measured parameter. For this simulation the load flow was calculated using a Runge-Kutta integration method.

3.2 Calculating the Load Pressure

The calculation of the load flow begins with the equation, from [Merritt, 1974], of the time rate of change of the load flow,

$$\dot{P}_L = \frac{4 \beta_e}{V_t} (Q_L - D_M \dot{\theta}_M - C_{tm} P_L) , \tag{11}$$

where β_e is the effective bulk modulus, V_t is the total volume in both chambers of the actuator, D_M is the volumetric displacement of the motor, and C_{tm} is the total leakage coefficient. The equation of the angular acceleration of the motor is calculated as

$$\ddot{\theta}_m = \frac{D_M P_L - D_M\left(\frac{\dot{\theta}_m}{|\dot{\theta}_m|}\right) - B_m \dot{\theta}_m - D_m C_d \mu \dot{\theta}_m}{J_m}, \tag{12}$$

where B_m is the damping coefficient of the motor, Cd and μ are damping and friction coefficients, and J_m is the inertia of the motor. Equations (11) and (12) were used in the Runge-Kutta formulation as follows:

$$(kP)_1 = \left[\frac{4\beta_e}{V_t}\left(Q_{L[i]} - D_M \dot{\theta}_m - C_{tm} P_{L[i]}\right)\right]\Delta t \tag{13}$$

$$(k\dot{\theta}_m)_1 = \left[\frac{D_M P_{L[i]} - D_M\left(\frac{\dot{\theta}_m}{|\dot{\theta}_m|}\right) C_f P_S - B_m \dot{\theta}_m - D_M C_d \mu \dot{\theta}_m}{J_m}\right]\Delta t \tag{14}$$

$$(kP)_2 = \left(\frac{4\beta_e}{V_t}\left\{C_v x_v \sqrt{P_S - \left[P_{L[i]} + \frac{1}{2}(kP)_1\right]} - D_M\left[\dot{\theta}_m + \frac{1}{2}(k\dot{\theta}_m)_1\right]\right.\right.$$

$$\left.\left. - C_{tm}\left[P_{L[i]} + \frac{1}{2}(kP)_1\right]\right\}\right)\Delta t \tag{15}$$

$$(k\dot{\theta}_m)_2 = [\frac{D_M[P_{L[i]} + \frac{1}{2}(kP)_1] - D_M\frac{\dot{\theta}_m}{|\dot{\theta}_m|}C_fP_S - B_m[\dot{\theta}_m + \frac{1}{2}(k\dot{\theta}_m)_1]}{J_m}$$

$$\frac{- C_dD_M\mu[\dot{\theta}_m + \frac{1}{2}(k\dot{\theta}_m)_1]}{}] \Delta t \tag{16}$$

$$(kP)_3 = (\frac{4\beta_e}{V_t}\{C_vx_v\sqrt{P_S - [P_{L[i]} + \frac{1}{2}(kP)_2]} - D_M[\dot{\theta}_m + \frac{1}{2}(k\dot{\theta}_m)_2]$$

$$- C_{tm}[P_{L[i]} + \frac{1}{2}(kP)_2]\}) \Delta t \tag{17}$$

$$(k\dot{\theta}_m)_3 = [\frac{D_M[P_{L[i]} + \frac{1}{2}(kP)_2] - D_M\frac{\dot{\theta}_m}{|\dot{\theta}_m|}C_fP_S - B_m[\dot{\theta}_m + \frac{1}{2}(k\dot{\theta}_m)_2]}{J_m}$$

$$\frac{- C_dD_M\mu[\dot{\theta}_m + \frac{1}{2}(k\dot{\theta}_m)_2]}{}] \Delta t \tag{18}$$

$$(kP)_4 = (\frac{4\beta_e}{V_t}\{C_vx_v\sqrt{P_S - [P_{L[i]} + (kP)_3]} - D_M[\dot{\theta}_m + (k\dot{\theta}_m)_3]$$

$$- C_{tm}[P_{L[i]} + (kP)_3]\}) \Delta t \tag{19}$$

$$(k\dot{\theta}_m)_4 = [\frac{D_M[P_{L[i]} + (kP)_3] - D_M\frac{\dot{\theta}_m}{|\dot{\theta}_m|}C_fP_S - B_m[\dot{\theta}_m + (k\dot{\theta}_m)_3]}{J_m}$$

$$\frac{- C_dD_M\mu[\dot{\theta}_m + (k\dot{\theta}_m)_3]}{}] \Delta t \ . \tag{20}$$

$$P_{L[i+1]} = P_{L[i]} + \frac{1}{6}[(kP)_1 + 2(kP)_2 + 2(kP)_3 + (kP)_4] \quad (21)$$

In equations (13) through (20) the time step for the Runge-Kutta method, Δt, is 0.00001 seconds. The program runs through the integration, by the index *i*, one hundred times. This allows the hydraulic model to correspond to the 1000 Hz sampling rate of the TeleGRIP simulation. Using a smaller time step does not significantly increase the accuracy of the simulation but does increase the time required to run the simulation. The time required to run a thirty second simulation varies from about thirty-five minutes on a Silicon Graphics Indy workstation to about twenty minutes on a Silicon Graphics Onyx workstation. The load pressure is used to compute the torque generated by the actuator with the equation

$$T_{g[i]} = P_{L[i]} D_m \, . \quad (22)$$

When the program is finished running the integration the final torque value, $T_{g[99]}$, is returned to the TeleGRIP simulation to be used in computing new joint values.

3.3 Other Nonlinearities

The hydraulic model also includes other nonlinearities that are present in the actuator, saturation and friction. The actuator in the PNL FBTB saturates at 1000 N-m, so the torque returned from the hydraulic model was limited to values between -1000 N-m and 1000 N-m. If the torque calculated by the hydraulic model exceeded 1000 N-m then a

value of 1000 N-m was returned. The actuator in the FBTB also exhibits the effects of coulomb friction at low torque. If the torque calculated by the hydraulic model was less than 8 N-m, then a torque of zero was returned to the TeleGRIP simulation. These nonlinearities along with the nonlinear orifice flow equation, equation (9), and the variation of beam inertia with respect to time constitute the most significant nonlinearities affecting the dynamic behavior. The variation of the hydraulic fluid properties with respect to time and temperature was not considered. These effects were considered beyond the scope of this study.

Chapter 4

Simulation Results

This chapter will describe the experiments run with the simulation of the flexible beam test bed using the controller and hydraulic model discussed previously. The results from experiments on fixed length beams, variable length beams, and various beam translation times are presented.

4.1 Testing the Simulation

To test the controller and hydraulic model, the beam was moved both rotationally and translationally. For each experiment the beam was moved from a rotational position of twenty degrees to a desired position of zero degrees. Experiments were run for four cases: with the beam length fixed at four meters, fixed at two meters, varying from two to four meters, and varying from four to two meters. The translational motion of the beam was completed in seven seconds. For each of these cases two tests were run, one with the controller tuned for a two meter beam and one with the controller tuned for a four meter beam.

The characteristics of these tests that were observed were overshoot, rise time, and settling time. These values are listed in Table 1. The overshoot, given as a percentage, is the amount that the actuator position exceeded the desired position. The rise time is

Table 1. Response Characteristics for Various Beam Motions and Controllers

Description	Overshoot (%)	Rise Time (seconds)	Settling Time (seconds)
Fixed 4m Beam Tuned for 2m	8.8	12.9	> 30
Fixed 4m Beam Tuned for 4m	2.95	4.5	4.5
Fixed 2m Beam Tuned for 2m	0.45	15.4	15.9
Fixed 2m Beam Tuned for 4m	27.6	4.6	25.6
Variable 2m -> 4m Beam Tuned for 2m	0.5	15.4	15.4
Variable 2m -> 4m Beam Tuned for 4m	30.7	4.8	> 30
Variable 4m -> 2m Beam Tuned for 2m	7.7	15.4	> 30
Variable 4m -> 2m Beam Tuned for 4m	2.85	4.5	4.5

the time taken for the actuator to come within five percent of the desired position. The settling time is the time taken for the actuator to stay within plus or minus five percent of the desired position. The overshoot tended to be the smallest for the tests where the controller was tuned for the length at which the beam was fixed, or, for the variable length beams, the overshoot tended to be the smallest for the tests where the controller was tuned for the length at which the beam began its translation. The rise time also corresponded to the controller used. When the controller was tuned for the two meter beam, the rise time was about fifteen seconds. When the controller was tuned for the four meter beam, the rise time was about four and one half seconds. The settling time, like the overshoot, was the smallest when the controller was tuned for the length at which the beam was fixed, or the length where the beam started its translational motion.

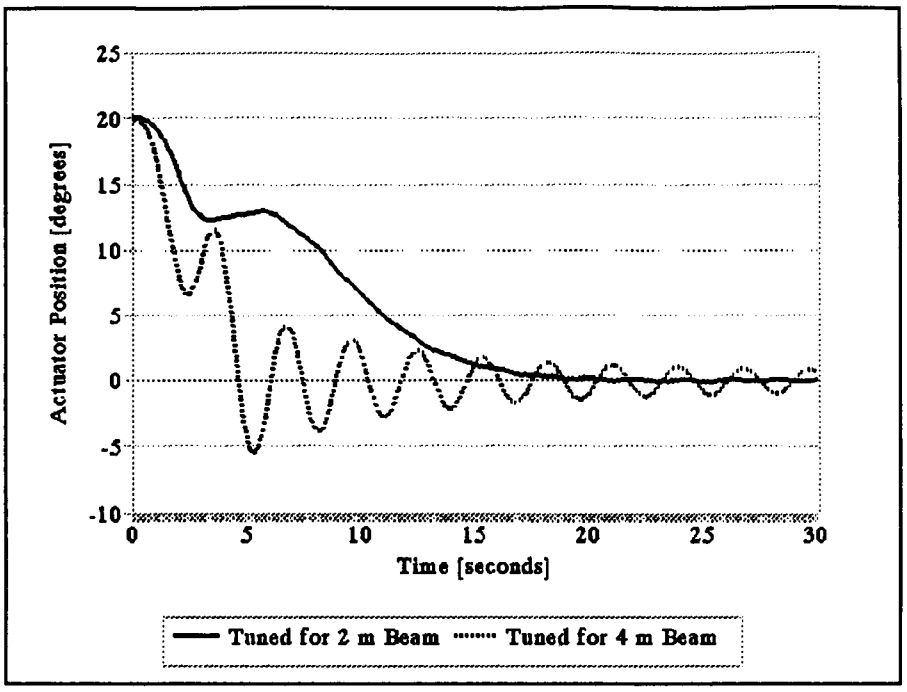
To further test the simulation, the time taken to translate the beam was varied. The beam was rotated twenty degrees as it was translated from two meters to four meters. For each case the controller was tuned for a two meter beam.

The overshoot, rise time, and settling time were also observed for this test, and are shown in Table 2. The values for the two and four meter fixed length beams are included for comparison. In general, the overshoot increases as the translation time decreases. For translation times of one second or less the overshoot, rise time, and settling time remain constant.

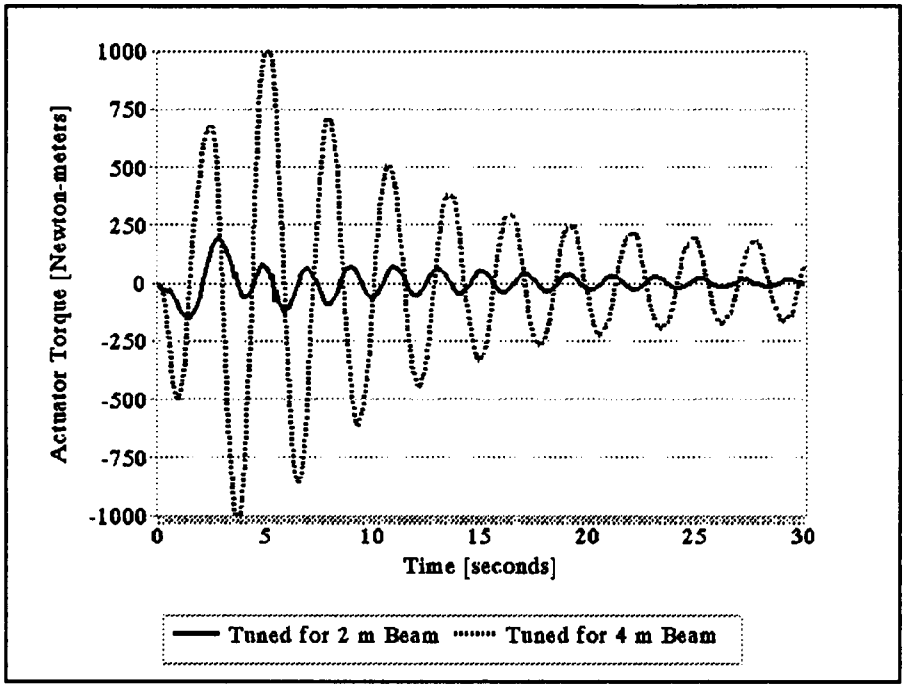
The position and torque responses of the hydraulic actuator for the tests are shown in Figures 4 through 8. Figure 4 shows the position as a function of time of the actuator for a two meter fixed length beam. Two different responses are shown, one using a

Table 2. Response Characteristics for Various Beam Translation Times with Controller Tuned for a 2 Meter Beam

Description	Translation Time (seconds)	Overshoot (%)	Rise Time (seconds)	Settling Time (seconds)
Fixed 2 m Beam	—	0.45	15.4	15.9
Variable 2-4 m Beam	7.0	0.5	15.4	15.4
Variable 2-4 m Beam	5.0	0.75	15.3	15.3
Variable 2-4 m Beam	1.0	2.6	15.1	16.7
Variable 2-4 m Beam	0.5	2.6	15.1	16.7
Variable 2-4 m Beam	0.1	2.6	15.1	16.7
Fixed 4 m Beam	—	8.8	12.9	> 30

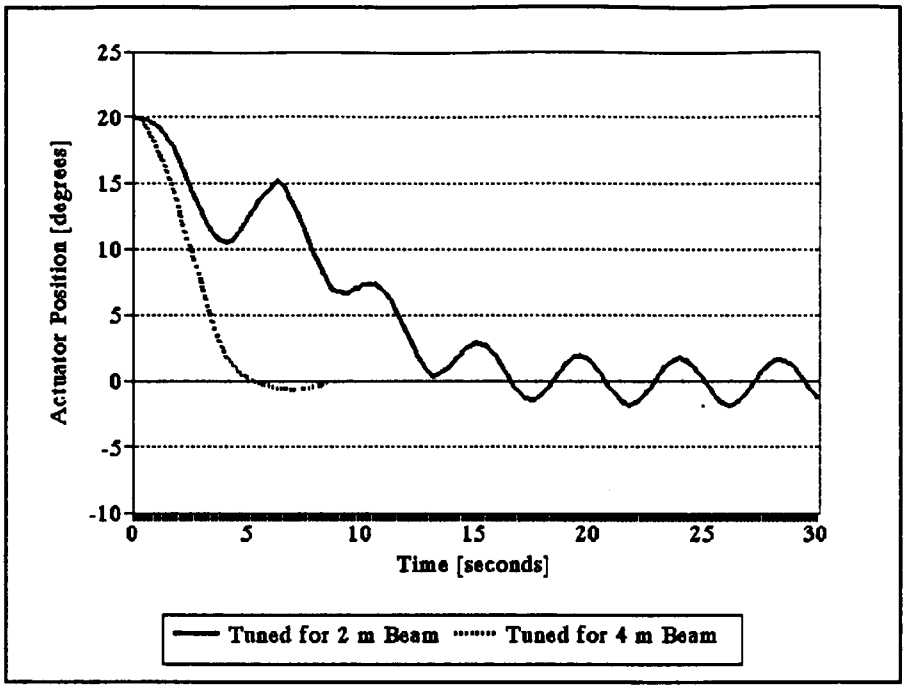


(a)

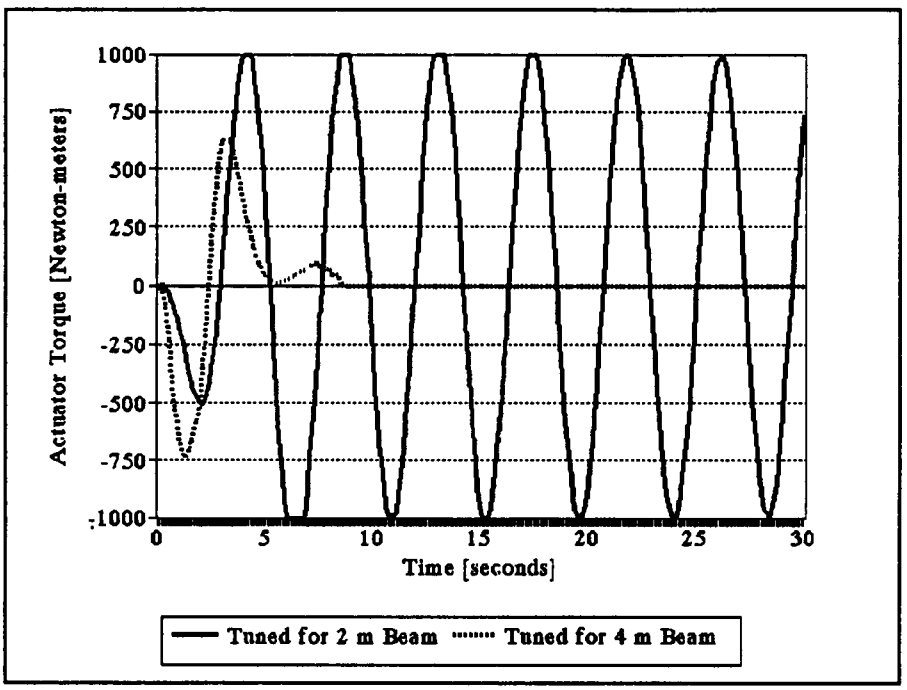


(b)

Figure 4: Hydraulic Actuator Response for a Two Meter Fixed Length Flexible Beam: (a) Position, (b) Torque

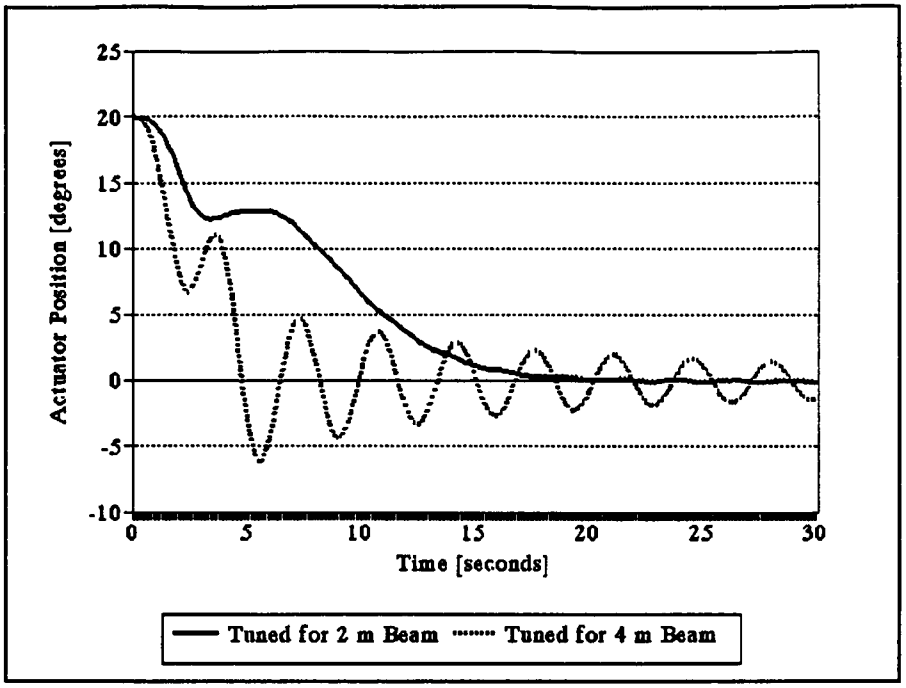


(a)

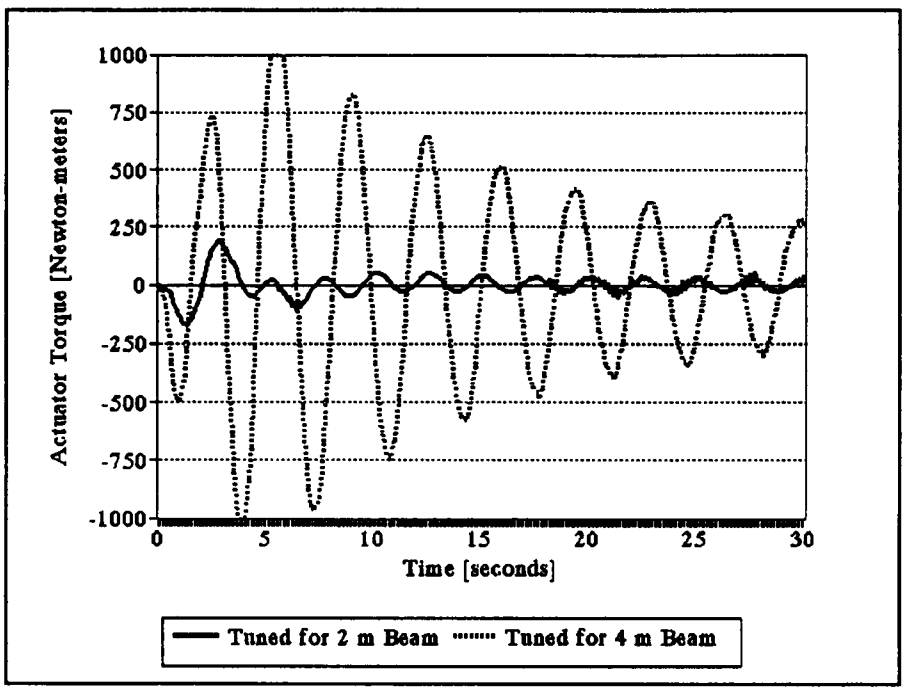


(b)

Figure 5: Hydraulic Actuator Response for a Four Meter Fixed Length Flexible Beam: (a) Position, (b) Torque

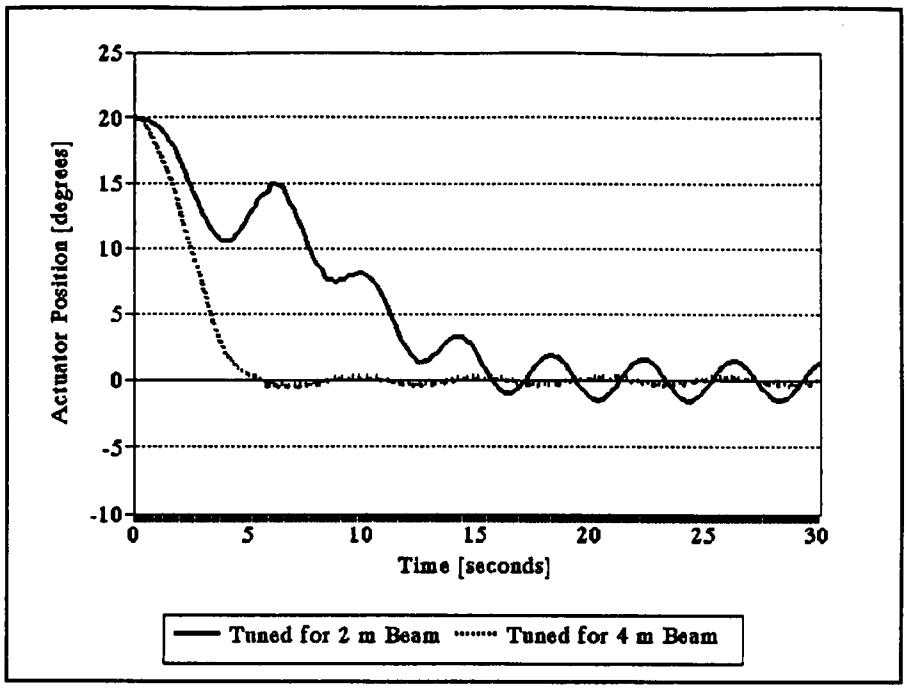


(a)

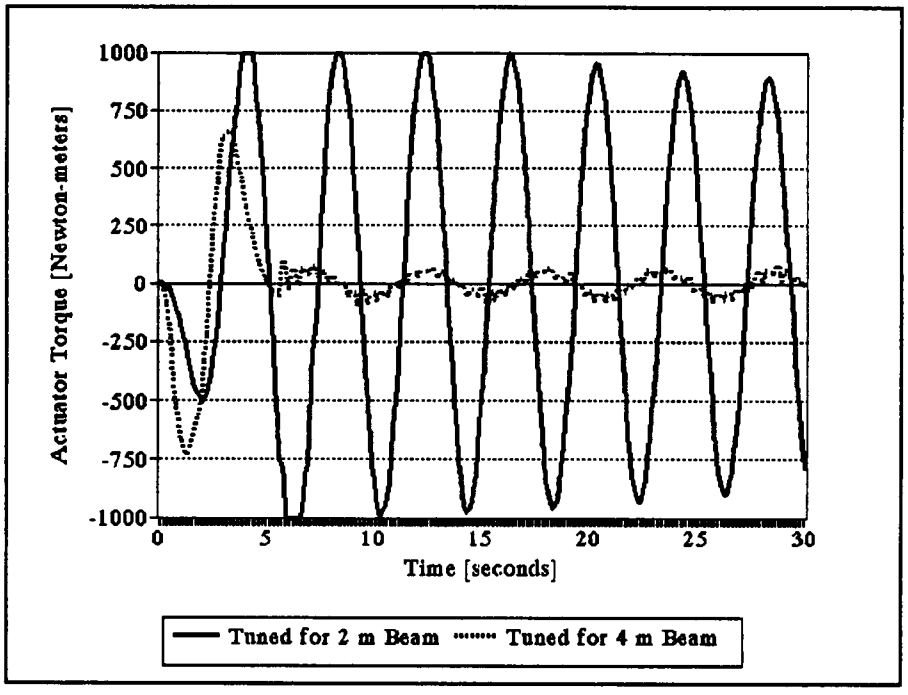


(b)

Figure 6: Hydraulic Actuator Response for a Two to Four Meter Variable Length Flexible Beam: (a) Position, (b) Torque

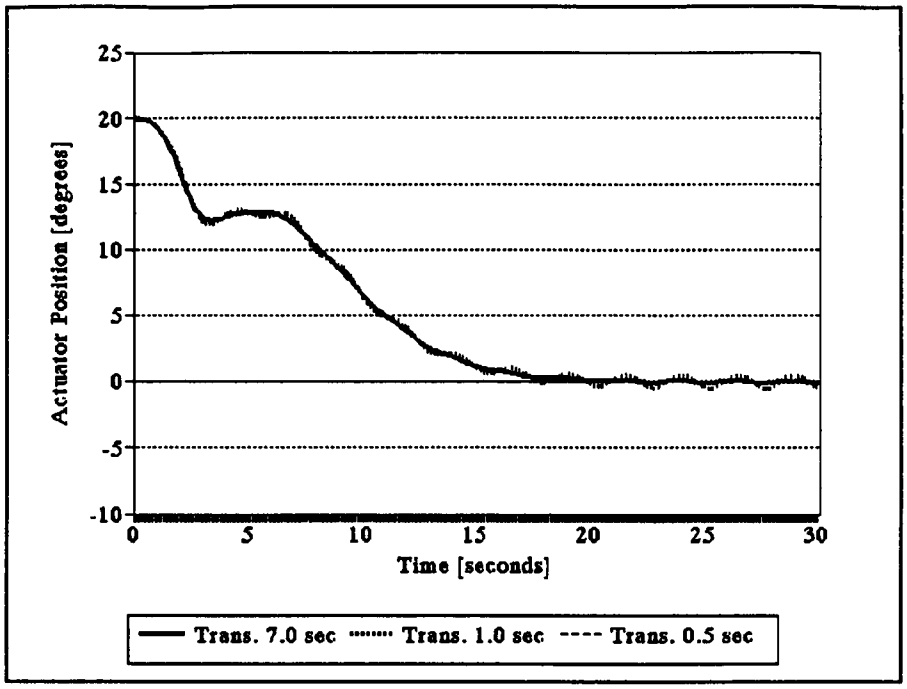


(a)

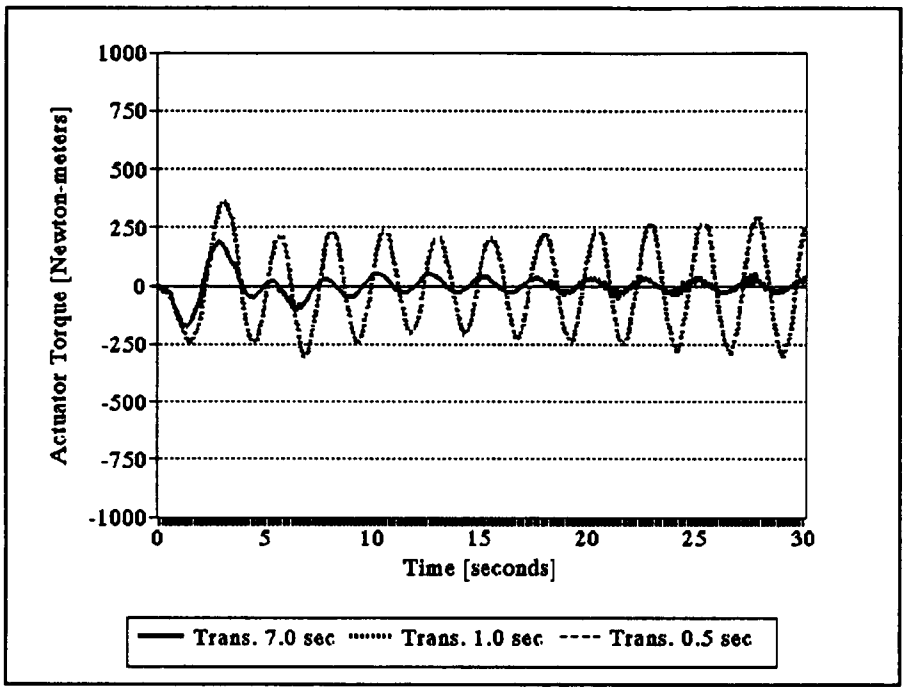


(b)

Figure 7: Hydraulic Actuator Response for a Four to Two Meter Variable Length Flexible Beam: (a) Position, (b) Torque



(a)



(b)

Figure 8: Hydraulic Actuator Response for a Variable Length Flexible Beam with Different Translation Times: (a) Position, (b) Torque

controller tuned for a two meter beam and one using a controller tuned for a four meter beam. As expected considering the results given previously, the response using a controller tuned for a two meter beam is the better response. The response from the controller tuned for a four meter beam shows more overshoot, a longer settling time, and, as Figure 4b shows, requires more torque. Figure 5 shows the response for a four meter fixed length beam, and similarly to Figure 4, it shows that the controller tuned for the four meter beam gives the better response. In Figure 6 the results for a beam varying in length from two to four meters are shown. Again, two responses are shown, one for a controller tuned for a two meter beam and one for a controller tuned for a four meter beam. The position and torque responses show that the controller tuned for a two meter beam gives the better result, with less oscillation and a smaller torque requirement. In Figure 7 the results for a beam varying in length from four to two meters are shown. As expected the controller tuned for four meters provides the best position and torque responses. In Figure 8 the beam varies in length from two to four meters using one controller tuned for a two meter beam, and three different translation times are shown. As expected the slowest translation time, seven seconds, results in the least oscillation and the least torque requirement. The results for the one second and one half second translation times are very similar in position and torque response. The torque for the faster translation times increases until it reaches a maximum value of plus or minus 500 N-m after about 60 seconds.

4.2 Verifying the Model

The results obtained by these tests agree reasonably well with the simulations and experiments reported in [Kwon, 1994] and [Kwon, 1995]. The simulation and experimental results in these papers were for a fixed length beam only, since the PNL FBTB does not have a prismatic joint, but they do confirm the operation of the filter and controller. Experimental tests performed by Kwon, et al. on the PNL FBTB showed that the first resonant frequency of the system is approximately 0.3 Hz. Figures 5 and 6 show that the four meter long simulated beam vibrates at about 0.3 Hz. This value can be confirmed with the following equation from [Laura, 1974],

$$f_1 = \frac{1}{2\pi}\left(\frac{y_1}{L}\right)^2 \sqrt{\frac{EI}{\rho A}}, \tag{23}$$

where L = 4.17 m, E = 2.07×10^{11} N/m^2, I = 1.742×10^{-7} m^4, ρ = 7830 kg/m^3, A = 5.73×10^{-3} m^2, and y_1 is the root of the transcendental equation derived in [Laura, 1974]. For an end mass to beam mass ratio of 240.5 kg to 187.7 kg, or 1.28, $y_1 \cong 1.16$. Using equation (23) the first natural frequency of the four meter beam is 0.35 Hz.

Chapter 5

Conclusions

In this thesis the problem of a flexible robot link was explored through a simulation of the PNL FBTB. In the simulation the hydraulic actuator of the FBTB was modeled and the beam was varied in length. The study of hydraulically actuated flexible robot links of varying configuration is important to LRM technology. The application of LRMs to tank waste retrieval will require steady control during operation.

The simulation of the FBTB was run on Deneb Robotics' TeleGRIP software. This software provided dynamic properties and analysis for a graphical representation of the hardware. The capability of the software to run user-written code for customized control algorithms was used to include a PD controller with velocity feedforward and pressure feedback terms and a robust notch filter in the simulation. The length of the flexible beam was varied by shifting the mass of the beam.

The base hydraulic actuator of the FBTB was modeled using equations for load flow, time rate of change of load pressure, and angular velocity. A Runge-Kutta integration method was used to calculate the load pressure. The torque generated by the actuator was calculated from the load pressure and the torque value was returned to TeleGRIP to generate new joint values.

The behavior of the flexible beam was observed for several different tests. The beam

was held at fixed lengths of two and four meters as it was rotated by the base actuator through an angle of twenty degrees. The beam was also translated in length from two to four meters and from four to two meters as it was rotated twenty degrees. For each of these tests the controller was tuned for both a two meter long beam and a four meter long beam. Tuning the controller involved adjusting the gains and setting the notches of the notch filter. These tests showed that the best performance from the beam is obtained when the controller is tuned for the length at which it begins its move. For instance, for a beam fixed at two meters or a beam that will translate from two to four meters the controller should be tuned for a two meter long beam for the best performance. Likewise, for a beam fixed at four meters or a beam that will translate from four to two meters the controller should be tuned for a four meter long beam. Tuning the controller properly will reduce the overshoot, rise time, and settling time of the actuator position and reduce the torque required from the actuator. The speed at which the beam was translated was varied to find the effect on the performance. When the speed of translation was increased the overshoot and settling time of the actuator also increased, but the performance was not significantly degraded. The increase in translation speed did cause a significant increase in the torque required from the actuator. The continued constant amplitude and frequency oscillations seen during the faster translation times is limit cycling caused by structural resonances. Translation speed must be adjusted to prevent limit cycling.

For further work on the LRM problem, an experimental test stand is being constructed at ORNL. This test stand has a Parker HTR 3586 N-m base rotary hydraulic actuator, a Parker EH-series hydraulically actuated translational joint with a 3.49 cm diameter rod,

LVDT position feedback sensors, and MOOG 760 series servovalves. The test stand is approximately 2.2 meters in length with a 0.5 meter variable length flexible section. Experiments performed on this test stand will be important for confirming the results of the simulation experiments. Other work can be done on the simulation by making modifications to the existing controller or by adding a new controller. Different controllers can be implemented that use fuzzy logic, end point tracking, or other sensory feedback. Another area that can be the subject of further research is changing the configuration of the Schilling arm used as an end-effector on the flexible beam.

Bibliography

Bibliography

[Bills, 1994] Bills, K.C., Kwon, D.S., Kress, R.L., Baker, C.P., "Model of Rotary-Actuated Flexible Beam with Notch Filter Vibration Suppression Controller and Torque Feedforward Load Compensation Controller," Proceedings of the 1994 Deneb Users Group Meeting. St. Clair, Michigan, October 1994.

[Deneb, 1994] Deneb Robotics, Inc., TeleGRIP Users' Manual: TeleGRIP 2.3. 1994.

[Jansen, 1991a] Jansen, J.F., Burks, B.L., Babcock, S.M., Kress, R.L., Hamel, W.R., "Long-Reach Manipulation for Waste Storage Tank Remediation," Modeling and Control of Compliant and Rigid Motion Systems. ASME, 1991.

[Jansen, 1991b] Jansen, J.F., March-Leuba, S., Kwon, D.S., Babcock, S.M., Burks, B.L., Kress, R.L., Hamel, W.R., Long-Reach Manipulation for Waste Storage Tank Remediation. FY 1991 Report, ORNL/TM-11999.

[Kleinsteuber, 1994] Kleinsteuber, S., Sepehri, N., "Abductory Induction Modeling of Hydraulic Actuation Systems," Robotics and Manufacturing: Recent Trends in Research, Education, and Applications. Vol. 5, ASME Press, 1994.

[Kress, 1995] Kress, R., Jansen, J., Bills, K., Gizelar, A., Basher, A., Modeling and Control for Flexible-Link Robotic Systems. FY 1995 Report, ORNL/TM to be published.

[Kwon, 1993] Kwon, D.S., March-Leuba, S., Babcock, S.M., Hamel, W.R., Key Design Requirements for Long-Reach Manipulators. ORNL/TM-12251. September 1993.

[Kwon, 1994] Kwon, D.S., Hwang, D.H., Babcock, S.M., Burks, B.L., "Input Shaping Filter Methods for the Control of Structurally Flexible, Long-Reach Manipulators," Proceedings of the 1994 IEEE Conference on Robotics and Automation. San Diego, California, May 1994.

[Kwon, 1995] Kwon, D.S., Hwang, D.H., Babcock, S.M., Kress, R.L., Lew, J.Y., Evans, M.S., "Design and Experimental Evaluation of Flexible Manipulator Control Algorithms," Proceedings of the ANS Sixth Topical Meeting on Robotics and Remote Systems. Monterey, California, February 1995.

[Laura, 1974] Laura, P.A.A., Pombo, J.L., Susemihl, E.A., "A Note on the Vibrations of a Clamped-Free beam with a Mass at the Free End," Journal of Sound and Vibration. Vol. 37, No. 2, 1974.

[McLain, 1989] McLain, T.W., Iversen, E.K., Davis, C.C., Jacobsen, S.C., "Development, Simulation, and Validation of a Highly Nonlinear Hydraulic Servosystem Model," Proceedings of the 1989 American Control Conference. Pittsburgh, Pennsylvania, June 21-23, 1989.

[Merritt, 1967] Merritt, Herbert E., Hydraulic Control Systems. John Wiley and Sons, New York. 1967.

[Salcudean, 1994] Salcudean, S.E., Drexel, P.A., Ben-Dov, D., Taylor, A.J., Lawrence, P.D., "A Six Degree-of-Freedom, Hydraulic, One Person Motion Simulator," Proceedings of the 1994 IEEE Conference on Robotics and Automation. San Diego, California, May 1994.

[Sepehri, 1994] Sepehri, N., Lawrence, P.D., Sassani, F., Frenette, R., "Resolved-Mode Teleoperated Control of Heavy-Duty Hydraulic Machines," ASME Journal of Dynamic Systems, Measurement and Control. Vol. 116, June 1994.

[Suzuki, 1994] Suzuki, E., Choi, B.S., Yuh, J., "Experimental Study on a Flexible Beam," Robotics and Manufacturing: Recent Trends in Research, Education, and Applications. Vol. 5, ASME Press, 1994.

[Tabarrok, 1974] Tabarrok, B., Leech, C.M., Kim, Y.I., "On the Dynamics of an Axially Moving Beam," Journal of The Franklin Institute. Vol. 297, No. 3, March 1974.

[Tadikonda, 1992] Tadikonda, S.S.K., Baruh, H., "Dynamics and Control of a Translating Beam With a Prismatic Joint," ASME Journal of Dynamic Systems, Measurement and Control. Vol. 114, September, 1992.

[Yuh, 1991] Yuh, J., Young, T., "Dynamic Modeling of an Axially Moving Beam in Rotation: Simulation and Experiment," ASME Journal of Dynamic Systems, Measurement and Control. Vol. 113, March 1991.

Appendices

Appendix A. Source Code

```
/* File Name: dyn_ctl_ex_mod6.c
        Created by: IGRIP
        Modified by D. S. Kwon 7/26/94
        Modified date: 7/28/94, 8/2/94,  8/18/94, 8/24/94
        Modified by A. M. Gizelar 10/94
        Comments for modification:
        Ver 1.0: run the filter function only for specified joints selectively  7/28/94
        ver 2.0: Initialization of filter, sampling rate has been improved 8/2/94
        ver 3.0: initialization bug in unix machine fixed   8/10/94
        ver 4.0: printf statement added  8/15/94
        Ver 5.0: DBPRINT debugger printf statement added 8/18/94
        Ver 6.0: Bug fixed:  static * fil_plan_xx, and printf statement disabled
      Ver 7.0: Modification for hydraulic model added
        Ver 8.0: Modification for variable beam length

*/
/*************************************************************************
 *
 *                        dyn_example_control
 *
 *  Inputs:     name          type          description
 *              ----          ----          -----------
 *              data_pointer  **char              pointer to the ctl data
                              struct
 *                                          for this device
 *              eval_time     float         the elapsed time (is seconds)
 *                                          since the start of the move
 *
 *  Outputs:
 *              data_pointer  **char              pointer to the ctl data
                                    struct
 *                                          for this device
 *
 *  Description:
 *      This is an example PID controller routine for IGRIP dynamics.
 *      The code computes the control torques at each joint by applying
 *      the control law to the error between planned and actual joint
 *      trajectory states.
 *
 *      NOTE: This example assumes the robot links to form a tree
 *      structure with all joints parallel to each other.  This results
```

```
*      in a simplistic dynamic coupling with no inertia variation
*      during a move. The mass array is assumed to contain the link
*      mass value for a translational joint or else the moment of inertia
*      value about the joint for the rotational case.
*      There are no gravitational, frictional, or stiffness effects.
*
*      The default IGRIP units are as follows:
*
*      Quantity        Units           Abbreviation
*
*      Mass            kilogram        kg
*      Length                  millimeter      mm
*      Time            second                  s
*      Angle                   radian          rad
*
*      Force           Newton                  N
*      Torque                  Newton-meter            N-m
*      Inertia         kg-meter^2      kg-m^2
*
*      Joint Torques may be force units or torque units, depending on
*      the joint type.
*
*      Viscous Friction is expressed in terms of units of Force/Torque
*      per unit joint velocity, depending on the joint type.
*
*      Joint Stiffness is expressed in terms of units of Force/Torque
*      per unit joint displacement, depending on the joint type.
*
*  Modification History:
*
*      cre     user     dd mm yy     decoupled example
*
*  copyright   1992   Deneb Robotics Incorporated
*
***************---------- MICA ----------***************
    Part has been added to use the digital_filters
*---------- MICA ----------*
*
*************************************************************************/

#define         MAX_DOF   10
#define         mmTOm 0.001

/* #define DBPRINT  */
```

```
typedef struct
        {
            float  err_xx [MAX_DOF];        /* control errors */
            float  err_ii [MAX_DOF];
            float  err_vv [MAX_DOF];

            float  prev_err_xx [MAX_DOF]; /* prev control errors */
            float  prev_err_ii [MAX_DOF];
        }
         dev_ctl_data_struct;

static dev_ctl_data_struct *dev_ctl_data;

float cycle_time;


/***************---------- MICA ----------***************/
/*  Input shaping filter "Robust Notch Filter" is used
    which is in "filter_IGRIP_C6.c"                    */

#include "filter_for_2m_beam.c"

/* #include "shlibdefs.h" */
/* #include "shlibdefs_mod.h" */
/* Determine the joints which will use the filters  */
#define USE_FILTER 1
#define USE_NO_FILTER 0
int filter_use_joint [MAX_DOF] ={1,0,0,0,0, 0,0,0,0,0};
/*int filter_use_joint [MAX_DOF] ={1,1,1,1,1, 1,1,1,1,1};*/
/*int filter_use_joint [MAX_DOF] ={0,0,0,0,0, 0,0,0,0,0};*/



/*----------  MICA ----------*/

/*****************************************************************
 *
 *    dyn_example_control_hydr
 *
 *****************************************************************/

int
dyn_example_control_hydr( data_pointer, eval_time )
```

```
char **data_pointer;
float eval_time;
{

int   dof, joint;
int   *joint_type;

float *initial_thetas,
      *final_thetas;

float del_time;
float convg_position, convg_speed;

float *plan_xx; /* planned thetas */
float *plan_vv;
float *plan_aa;

float *curr_xx; /* actual thetas */
float *curr_vv;
float *curr_aa;

float *tau;     /* control forces */

/***********  hydr model  ******************/
float *valve;      /*  valve position  */

               /* control gains */
float *Kpp;    /* proportional  */
float *Kii;    /* integral      */
float *Kdd;    /* derivative    */
               /* Additional gains (not used for the IGRIP default PID controller)  */
float *aux1;   /* feedforward gain= Jtotal   */
float *aux2;   /* 1st filter frequency----  pressure feedback gain      */
float *aux3;   /* 1st filter pole/zero ratio alpha    */

ig_node *node_16_4;   /** for variable inertias **/
ig_node *node_16_4_1;
ig_node *node_32_8;
ig_node *node_32_8_1;
ig_node *node_32_8_2;
ig_node *node_32_8_3;
/*float cycle_time;*/
ig_node *node_pallet;   /** for variable mass **/
ig_node *node_azimuth;
```

```
ig_node *node_upperarm;
ig_node *node_forearm;
float mass_old164;
float mass_old1641;
float mass_old328;
float mass_old3281;
float mass_old3282;
float mass_old3283;
float Jbeam, Mbeam, l_beam;
float *Jhyd;
float fpv,fpa,fpp,ca,cx,cv;
float Kp2, Kp4, Kd2, Kd4, A2_2, A2_4;

/* ---------- MICA filter part ----------*/
static int init_flag;

static float *fil_plan_xx; /* filtered planned thetas */
static float *fil_plan_vv;
static float *fil_plan_aa;
/* need to allocate the memory to use as an array of fil_plan_xx[joint]
*  Array of fil_plan_xx is the filtered joint trajectory          */

extern void digital_filters(int init_flag, int dof, int joint, float del_time,
                            float *plan_xx, float *fil_plan_xx,
                            float *plan_vv, float *fil_plan_vv,
                            float *plan_aa, float *fil_plan_aa);

/***********  hydr model  ******************/

float hydraulic_model( float theta_m, float theta_m_dot, float theta_m_2dot,
                       float xv);

char *part_16_4;    /*** for variable inertias ***/
char *part_16_4_1;
char *part_32_8;
char *part_32_8_1;
char *part_32_8_2;
char *part_32_8_3;
char *part_pallet;
char *part_azimuth;
char *part_upperarm;
char *part_forearm;

/***********  hydr model  ******************/
```

```
float DM;            /* volumetric disp. in metric units */
float PS;           /* supply pressure in metric units */
float CV;
float Jmb;
 DM = 0.0008919444 /** 1000 * 1000 * 1000*/;
 PS = 20684271 /*/ 1000 / 1000*/;
CV = .0000000006935999;
 Jmb = .0007117;


Kp2 = 780.0;
Kp4 = 268.0;
Kd2 = 2.2;
Kd4 = 3.0;
A2_2 = 0.04/1000000.0;
A2_4 = 0.026/1000000.0;



part_16_4 = "flex_beam:beam_16.4";     /*** for variable inertias ***/
part_16_4_1 = "flex_beam:beam_16.4_1";
part_32_8 = "flex_beam:beam_32.8";
part_32_8_1 = "flex_beam:beam_32.8_1";
part_32_8_2 = "flex_beam:beam_32.8_2";
part_32_8_3 = "flex_beam:beam_32.8_3";
part_pallet = "flex_beam:pallet";
part_azimuth = "flex_beam:azimuth";
part_upperarm = "flex_beam:upperarm";
part_forearm = "flex_beam:forearm";


/* ----------  MICA filter part ----------*/

/******************* begin execution ********************/
        #ifdef DBPRINT
           printf("NEW file dyn_ctl_ex_mod6.c\n ");
        #endif
    /*
     * set up controller data pointer
     */
    if( *data_pointer == NULL )
    {
        *data_pointer = (char *) malloc( sizeof( dev_ctl_data_struct ) );
         printf("data pointer was null = %x \n", *data_pointer);
```

```
    }

    dev_ctl_data = (dev_ctl_data_struct *) *data_pointer;
        #ifdef DBPRINT
         printf("device control pointer value= %x \n", dev_ctl_data);
        #endif



    /*
     * set up device data pointers
     */
    dof = /*mot_inq_dof()*/1;

    joint_type = mot_get_joint_types();

    initial_thetas = mot_get_initial_thetas();
    final_thetas   = mot_get_final_thetas();

    curr_xx = mot_get_current_thetas();
    curr_vv = mot_get_current_joint_speeds();
    curr_aa = mot_get_current_joint_accels();

    plan_xx = mot_get_plan_thetas();
    plan_vv = mot_get_plan_joint_speeds();
    plan_aa = mot_get_plan_joint_accels();

    /*
     * set up dynamics data pointers
     */
    Kpp = dyn_get_proportional_gains();
    Kii = dyn_get_integral_gains();
    Kdd = dyn_get_derivative_gains();


    aux1= dyn_get_aux1_gains();
    aux2= dyn_get_aux2_gains();
    aux3= dyn_get_aux3_gains();

    tau = dyn_get_joint_torques();


/***********  hydr model  ******************/
    valve = (float*)malloc(dof*sizeof(float));
```

```
    /*
     * calc time interval:     Moved from the PID control part
     */
    del_time = mot_inq_heart_rate();

        #ifdef DBPRINT
        printf(" The DOF of the manipulator is == %d  \n", dof );
        printf(" The sampling freq of the simulationr is == %f  \n", del_time );
        #endif
/*****************************************************************
*
*       initialize stuff
*
*****************************************************************/

     /*** for variable inertias ***/
node_16_4 = dyn_get_named_node_handle( part_16_4 );
node_16_4_1 = dyn_get_named_node_handle( part_16_4_1 );
node_32_8 = dyn_get_named_node_handle( part_32_8 );
node_32_8_1 = dyn_get_named_node_handle( part_32_8_1 );
node_32_8_2 = dyn_get_named_node_handle( part_32_8_2 );
node_32_8_3 = dyn_get_named_node_handle( part_32_8_3 );
node_pallet = dyn_get_named_node_handle( part_pallet );
node_azimuth = dyn_get_named_node_handle( part_azimuth );
node_upperarm = dyn_get_named_node_handle( part_upperarm );
node_forearm = dyn_get_named_node_handle( part_forearm );


    if( dyn_inq_state_reset() )
    {
        #ifdef DBPRINT
        printf(" dyn_inq_state_reset has been called \n");
        #endif
        #ifdef DBPRINT
        printf("  \n");
        #endif

        /*
         * initialize state
         */
         /* ---------------  from MICA filter part ---------------*/
    fil_plan_xx=(float*)malloc(dof*sizeof(float));
```

```
    fil_plan_vv=(float*)malloc(dof*sizeof(float));
    fil_plan_aa=(float*)malloc(dof*sizeof(float));

        init_flag=0;
        /* ---------- MICA ----------*/

              for( joint = 0; joint < dof; joint++ )
              {
           dev_ctl_data->prev_err_xx[ joint ] = 0.0;
           dev_ctl_data->prev_err_ii[ joint ] = 0.0;
           }
           /* -------  from MICA ----------------------  */
        #ifdef DBPRINT
        printf("digital_filters will be called for initialization  \n");
        #endif
           digital_filters(init_flag, dof, joint, del_time,
                                        plan_xx, fil_plan_xx,
                                        plan_vv, fil_plan_vv,
                                        plan_aa, fil_plan_aa);
/* _input, _output arguments are addresses */
                init_flag=1;
                printf("Initialization of filters finished  ----------\n");
                /*----------- MICA -------------------  */
        }

/*****************************************************************
*
*       PID controller
*
*****************************************************************/
    /*
     * calc time interval :  Moved to the top of the initalization part
     */

    /*
     * calc control error terms
     */

    /* ---------- from MICA ----------*/
    if (init_flag !=1) {
        printf (" The filter has not been initialized ===== need to stop \n");
    }
```

```
    else {
    #ifdef DBPRINT
        printf(" Do loop of filtering  for each joint will start \n");
        #endif

    }


    for( joint = 0; joint < dof; joint++ )
    {
        if (filter_use_joint[joint]==USE_FILTER) {

 /* _input, _output arguments are addresses */
        dev_ctl_data->err_xx[joint] = fil_plan_xx[joint] - curr_xx[joint];
        dev_ctl_data->err_vv[joint] = fil_plan_vv[joint] - curr_vv[joint];
fpa = fil_plan_aa[joint];
ca = curr_aa[joint];

                        #ifdef DBPRINT
                        printf(" Trajectory filtering was successful for joint = %d -----\n",
joint );
                        #endif


                        }
                else if (filter_use_joint[joint]==USE_NO_FILTER) {
                /*  ----------MICA ----------*/

                dev_ctl_data->err_xx[joint] = plan_xx[joint] - curr_xx[joint];
                dev_ctl_data->err_vv[joint] = plan_vv[joint] - curr_vv[joint];
                        #ifdef DBPRINT
                        printf(" Trajectory filtering was not used for joint = %d -----\n",
joint );
                        #endif

                        }
                else {
                        printf (" Use of filters for each joint were not defined");
                        }
    }

    /*
     * calc integral error term
     */
```

```
    for( joint = 0; joint < dof; joint++ )
    {
        dev_ctl_data->err_ii[joint] = dev_ctl_data->prev_err_ii[joint]
                        + dev_ctl_data->prev_err_xx[joint]*del_time;
    }

    /*
     * set up device control torques using PID law
     */


    for( joint = 0; joint < dof; joint++ )
    {
        if( joint_type[joint] == TRANSLATIONAL )
        {
            tau[joint] = Kpp[joint] * dev_ctl_data->err_xx[joint]  * mmTOm
                    + Kii[joint] * dev_ctl_data->err_ii[joint] * mmTOm
                    + Kdd[joint] * dev_ctl_data->err_vv[joint] * mmTOm;
        }
        else
        {

/***********  hydr model  ********************/


            valve[joint] = Kpp[joint] * dev_ctl_data->err_xx[joint]
                    + Kii[joint] * dev_ctl_data->err_ii[joint]
                    + Kdd[joint] * dev_ctl_data->err_vv[joint]

                    /*  + aux1[joint] * fil_plan_vv[joint]   */
/* aux1 = 1 */          + aux1[joint] * ( DM*fil_plan_vv[joint] /
                                (CV*sqrt( PS-(tau[joint]/DM) )) )
                    /*    - aux2[joint] * (tau[joint] /DM);*/
                      + aux2[joint] * ( (Jbeam*fil_plan_aa[joint]/DM)
                            - (Jbeam*curr_aa[joint] /DM) )/1000000.0;

fpv = ( DM*fil_plan_vv[joint] /(CV*sqrt( PS-(tau[joint]/DM) )) );
cx = ( (Jbeam*fil_plan_aa[joint]/DM)- (Jbeam*curr_aa[joint] /DM) );

        tau[joint] = hydraulic_model(curr_xx[joint], curr_vv[joint],
curr_aa[joint], valve[joint]);
```

```
        }
    }

cycle_time = mot_inq_cycle_time();

/*** for variable mass 2m -> 4m ***/

   if( (cycle_time >= 0.0) && (cycle_time <= 7.0/5.0) )
   {
   mass_old164 = node_16_4->dyn_props->mass;
   node_16_4->dyn_props->mass = ( 37.4917 - 93.7295/7.0*cycle_time );
   node_16_4->dyn_props->inertia[0] = node_16_4->dyn_props->inertia[0] /
                                 mass_old164 * node_16_4->dyn_props->mass;
   node_16_4->dyn_props->inertia[1] = node_16_4->dyn_props->inertia[1] /
                                 mass_old164 * node_16_4->dyn_props->mass;
   node_16_4->dyn_props->inertia[2] = node_16_4->dyn_props->inertia[2] /
                                 mass_old164 * node_16_4->dyn_props->mass;
   node_pallet->dyn_props->centroid.xx = -2000.0 + 2175.0/35.0 * cycle_time;
   node_azimuth->dyn_props->centroid.xx = -2000.0 + 2014.850/35.0 * cycle_time;
   node_upperarm->dyn_props->centroid.xx = -2000.0 + 2478.60/35.0 * cycle_time;
   node_forearm->dyn_props->centroid.xx = -2000.0 + 2222.80/35.0 * cycle_time;
   l_beam = 2.0;
   }
   if( (cycle_time >= 28.0/5.0) && (cycle_time <= 35.0/5.0) )
   {
   mass_old1641 = node_16_4_1->dyn_props->mass;
   node_16_4_1->dyn_props->mass = ( 0.01 + 93.7295/35.0*cycle_time );
   node_16_4_1->dyn_props->inertia[0] = node_16_4_1->dyn_props->inertia[0] /
                              mass_old1641 * node_16_4_1->dyn_props->mass;
   node_16_4_1->dyn_props->inertia[1] = node_16_4_1->dyn_props->inertia[1] /
                              mass_old1641 * node_16_4_1->dyn_props->mass;
   node_16_4_1->dyn_props->inertia[2] = node_16_4_1->dyn_props->inertia[2] /
                              mass_old1641 * node_16_4_1->dyn_props->mass;
   }
   if( (cycle_time >= 7.0/5.0) && (cycle_time <= 21.0/5.0) )
   {
   mass_old328 = node_32_8->dyn_props->mass;
   node_32_8->dyn_props->mass = ( 74.9835 - ( (187.459/21.0)*cycle_time ) );
   node_32_8->dyn_props->inertia[0] = node_32_8->dyn_props->inertia[0] /
                                 mass_old328 * node_32_8->dyn_props->mass;
   node_32_8->dyn_props->inertia[1] = node_32_8->dyn_props->inertia[1] /
                                 mass_old328 * node_32_8->dyn_props->mass;
   node_32_8->dyn_props->inertia[2] = node_32_8->dyn_props->inertia[2] /
                                 mass_old328 * node_32_8->dyn_props->mass;
```

```
node_pallet->dyn_props->centroid.xx = -2000.0 + 2175.0/21.0 * cycle_time;
node_azimuth->dyn_props->centroid.xx = -2000.0 + 2014.850/21.0 * cycle_time;
node_upperarm->dyn_props->centroid.xx = -2000.0 + 2478.60/21.0 * cycle_time;
node_forearm->dyn_props->centroid.xx = -2000.0 + 2222.80/21.0 * cycle_time;
l_beam = 3.0;
}
if( (cycle_time >= 21.0/5.0) && (cycle_time <= 35.0/5.0) )
{
mass_old3281 = node_32_8_1->dyn_props->mass;
node_32_8_1->dyn_props->mass = ( 74.9835 - ( (187.459/35.0)*cycle_time ) );
node_32_8_1->dyn_props->inertia[0] = node_32_8_1->dyn_props->inertia[0] /
                              mass_old3281 * node_32_8_1->dyn_props->mass;
node_32_8_1->dyn_props->inertia[1] = node_32_8_1->dyn_props->inertia[1] /
                              mass_old3281 * node_32_8_1->dyn_props->mass;
node_32_8_1->dyn_props->inertia[2] = node_32_8_1->dyn_props->inertia[2] /
                              mass_old3281 * node_32_8_1->dyn_props->mass;
node_pallet->dyn_props->centroid.xx = -2000.0 + 2175.0/7.0 * cycle_time;
node_azimuth->dyn_props->centroid.xx = -2000.0 + 2014.850/7.0 * cycle_time;
node_upperarm->dyn_props->centroid.xx = -2000.0 + 2478.60/7.0 * cycle_time;
node_forearm->dyn_props->centroid.xx = -2000.0 + 2222.80/7.0 * cycle_time;
l_beam = 4.0;
}
if( (cycle_time >= 0.0) && (cycle_time <= 14.0/5.0) )
{
mass_old3282 = node_32_8_2->dyn_props->mass;
node_32_8_2->dyn_props->mass = ( 0.01 + ( (187.459/14.0)*cycle_time ) );
node_32_8_2->dyn_props->inertia[0] = ( (node_32_8_2->dyn_props->inertia[0])
                        / mass_old3282  * (node_32_8_2->dyn_props->mass) );
node_32_8_2->dyn_props->inertia[1] = ( (node_32_8_2->dyn_props->inertia[1])
                        / mass_old3282  * (node_32_8_2->dyn_props->mass) );
node_32_8_2->dyn_props->inertia[2] = ( (node_32_8_2->dyn_props->inertia[2])
                        / mass_old3282  * (node_32_8_2->dyn_props->mass) );

}
if( (cycle_time >= 14.0/5.0) && (cycle_time <= 28.0/5.0) )
{
mass_old3283 = node_32_8_3->dyn_props->mass;
node_32_8_3->dyn_props->mass = ( 0.01 + 187.459/28.0*cycle_time );
node_32_8_3->dyn_props->inertia[0] = node_32_8_3->dyn_props->inertia[0] /
                              mass_old3283 * node_32_8_3->dyn_props->mass;
node_32_8_3->dyn_props->inertia[1] = node_32_8_3->dyn_props->inertia[1] /
                              mass_old3283 * node_32_8_3->dyn_props->mass;
node_32_8_3->dyn_props->inertia[2] = node_32_8_3->dyn_props->inertia[2] /
                              mass_old3283 * node_32_8_3->dyn_props->mass;
```

```
  }
  if (cycle_time >= 7.0)
  {
  l_beam = 4.0;
  }

/******* ^^^ variable mass 2m -> 4m ^^^ *******/

/*
  Kpp[0] = Kp2 + ( (Kp4 - Kp2) * ( ( l_beam - 2.0 ) / ( 2.0 ) ) );
  Kdd[0] = Kd2 + ( (Kd4 - Kd2) * ( ( l_beam - 2.0 ) / ( 2.0 ) ) );
  aux2[0] = A2_2 + ( (A2_4 - A2_2) * ( ( l_beam - 2.0 ) / ( 2.0 ) ) );
*/
/******* ^^^ variable mass and variable gains  ^^^ *******/

/*** for variable mass 4m -> 2m ***/
/*
  if( (cycle_time >= 0.0) && (cycle_time <= 7.0/5.0) )
  {
  mass_old164 = node_16_4->dyn_props->mass;
  node_16_4->dyn_props->mass = ( 18.7459 + 93.7295/7.0*cycle_time );
  node_16_4->dyn_props->inertia[0] = node_16_4->dyn_props->inertia[0] /
                                     mass_old164 * node_16_4->dyn_props->mass;
  node_16_4->dyn_props->inertia[1] = node_16_4->dyn_props->inertia[1] /
                                     mass_old164 * node_16_4->dyn_props->mass;
  node_16_4->dyn_props->inertia[2] = node_16_4->dyn_props->inertia[2] /
                                     mass_old164 * node_16_4->dyn_props->mass;
  node_pallet->dyn_props->centroid.xx = 174.699 - 2175.0/35.0 * cycle_time;
  node_azimuth->dyn_props->centroid.xx = 14.8465 - 2014.850/35.0 * cycle_time;
  node_upperarm->dyn_props->centroid.xx = 478.614 - 2478.60/35.0 * cycle_time;
  node_forearm->dyn_props->centroid.xx = 222.837 - 2222.80/35.0 * cycle_time;
  l_beam = 4.0;
  }
  if( (cycle_time >= 28.0/5.0) && (cycle_time <= 35.0/5.0) )
  {
  mass_old1641 = node_16_4_1->dyn_props->mass;
  node_16_4_1->dyn_props->mass = ( 18.7459 - 93.7295/35.0*cycle_time );
  node_16_4_1->dyn_props->inertia[0] = node_16_4_1->dyn_props->inertia[0] /
                                       mass_old1641 * node_16_4_1->dyn_props->mass;
  node_16_4_1->dyn_props->inertia[1] = node_16_4_1->dyn_props->inertia[1] /
                                       mass_old1641 * node_16_4_1->dyn_props->mass;
  node_16_4_1->dyn_props->inertia[2] = node_16_4_1->dyn_props->inertia[2] /
                                       mass_old1641 * node_16_4_1->dyn_props->mass;
  }
```

```
if( (cycle_time >= 7.0/5.0) && (cycle_time <= 21.0/5.0) )
{
mass_old328 = node_32_8->dyn_props->mass;
node_32_8->dyn_props->mass = ( 37.4918 + ( (187.459/21.0)*cycle_time ) );
node_32_8->dyn_props->inertia[0] = node_32_8->dyn_props->inertia[0] /
                                   mass_old328 * node_32_8->dyn_props->mass;
node_32_8->dyn_props->inertia[1] = node_32_8->dyn_props->inertia[1] /
                                   mass_old328 * node_32_8->dyn_props->mass;
node_32_8->dyn_props->inertia[2] = node_32_8->dyn_props->inertia[2] /
                                   mass_old328 * node_32_8->dyn_props->mass;
node_pallet->dyn_props->centroid.xx = 174.699 - 2175.0/21.0 * cycle_time;
node_azimuth->dyn_props->centroid.xx = 14.8465 - 2014.850/21.0 * cycle_time;
node_upperarm->dyn_props->centroid.xx = 478.614 - 2478.60/21.0 * cycle_time;
node_forearm->dyn_props->centroid.xx = 222.837 - 2222.80/21.0 * cycle_time;
l_beam = 3.0;
}
if( (cycle_time >= 21.0/5.0) && (cycle_time <= 35.0/5.0) )
{
mass_old3281 = node_32_8_1->dyn_props->mass;
node_32_8_1->dyn_props->mass = ( 37.4918 + ( (187.459/35.0)*cycle_time ) );
node_32_8_1->dyn_props->inertia[0] = node_32_8_1->dyn_props->inertia[0] /
                                     mass_old3281 * node_32_8_1->dyn_props->mass;
node_32_8_1->dyn_props->inertia[1] = node_32_8_1->dyn_props->inertia[1] /
                                     mass_old3281 * node_32_8_1->dyn_props->mass;
node_32_8_1->dyn_props->inertia[2] = node_32_8_1->dyn_props->inertia[2] /
                                     mass_old3281 * node_32_8_1->dyn_props->mass;
node_pallet->dyn_props->centroid.xx = 174.699 - 2175.0/7.0 * cycle_time;
node_azimuth->dyn_props->centroid.xx = 14.8465 - 2014.850/7.0 * cycle_time;
node_upperarm->dyn_props->centroid.xx = 478.614 - 2478.60/7.0 * cycle_time;
node_forearm->dyn_props->centroid.xx = 222.837 - 2222.80/7.0 * cycle_time;
l_beam = 2.0;
}
if( (cycle_time >= 0.0) && (cycle_time <= 14.0/5.0) )
{
mass_old3282 = node_32_8_2->dyn_props->mass;
node_32_8_2->dyn_props->mass = ( 37.4918 - ( (187.459/14.0)*cycle_time ) );
node_32_8_2->dyn_props->inertia[0] = ( (node_32_8_2->dyn_props->inertia[0])
                     / mass_old3282  * (node_32_8_2->dyn_props->mass) );
node_32_8_2->dyn_props->inertia[1] = ( (node_32_8_2->dyn_props->inertia[1])
                     / mass_old3282  * (node_32_8_2->dyn_props->mass) );
node_32_8_2->dyn_props->inertia[2] = ( (node_32_8_2->dyn_props->inertia[2])
                     / mass_old3282  * (node_32_8_2->dyn_props->mass) );

}
```

```
   if( (cycle_time >= 14.0/5.0) && (cycle_time <= 28.0/5.0) )
   {
   mass_old3283 = node_32_8_3->dyn_props->mass;
   node_32_8_3->dyn_props->mass = ( 37.4918 - 187.459/28.0*cycle_time );
   node_32_8_3->dyn_props->inertia[0] = node_32_8_3->dyn_props->inertia[0] /
                              mass_old3283 * node_32_8_3->dyn_props->mass;
   node_32_8_3->dyn_props->inertia[1] = node_32_8_3->dyn_props->inertia[1] /
                              mass_old3283 * node_32_8_3->dyn_props->mass;
   node_32_8_3->dyn_props->inertia[2] = node_32_8_3->dyn_props->inertia[2] /
                              mass_old3283 * node_32_8_3->dyn_props->mass;
   }
   if (cycle_time >= 7.0)
   {
   l_beam = 2.0;
   }
*/
/******* ^^^ variable mass 4m -> 2m ^^^ *******/


/*   l_beam = 4.0;
*/
Mbeam = node_16_4->dyn_props->mass + node_16_4_1->dyn_props->mass
       + node_32_8->dyn_props->mass + node_32_8_1->dyn_props->mass
       + node_32_8_2->dyn_props->mass + node_32_8_3->dyn_props->mass
       + node_pallet->dyn_props->mass + node_azimuth->dyn_props->mass
       + node_upperarm->dyn_props->mass + node_forearm->dyn_props->mass;

Jbeam = Mbeam * l_beam * l_beam / 3.0 ;



       #ifdef DBPRINT
       printf(" control torques using PID law was calculated  \n" );
       #endif
   /*
    * save prev state
    */
   for( joint = 0; joint < dof; joint++ )
   {
       dev_ctl_data->prev_err_xx[joint] = dev_ctl_data->err_xx[joint];
       dev_ctl_data->prev_err_ii[joint] = dev_ctl_data->err_ii[joint];
   }
       #ifdef DBPRINT
       printf(" save prev state ------ !!!OK  (in dyn_ctl_ex)  \n" );
```

```
        #endif

    return( STATUS_OKAY );
}


/*****************************************************************
 *
 *    dyn_example_free_ctl_data_hydr
 *
 *****************************************************************/

dyn_example_free_ctl_data_hydr( data_pointer )

char *data_pointer;

{

/************************ begin execution ************************/

    dev_ctl_data = (dev_ctl_data_struct *)data_pointer;
    return;
}
/*****************************************************************/

/************   hydr model   ******************/

float hydraulic_model(float theta_m, float theta_m_dot, float theta_m_2dot,
                      float xv)
{


double QL[50001], PL[50001]; /* load flow, load pressure */
double Tg[50001], Ps;      /* torque generated, supply pressure */
float Dm; /* volumetric motor displacement */
/*float *theta_m, *xv; /* angular motor position, valve displacement */
/*float *theta_m_dot, *theta_m_2dot;    /* angular motor velocity, acceleration */
float Kq, Kc;      /* valve flow gain, valve flow coeff.  */
float Cv;      /* overall valve coeff. */
float Jm, Bm;      /* motor inertia, viscous damping coeff */
float theta_L;     /* angular motor displacement due to load */
float PL_dot[50001], beta_e, Ctm, Vt;
float delta_t, Cf, Cd, mu;
float kP1, kP2;
```

```
float kP3, kP4;
float kthtdot1, kthtdot2, kthtdot3, kthtdot4;
double aa, c;
float new_torque;
int itime;
float time, final_time, f[4], PLc, PLp, fp;


int i;
int j;


    Kq = 1.925;
    Kc = 0.0048 /*0.0*/;
    Ctm = /*0.0608*/ 0.0;          /* this is really Cv    */
    Cv  = 0.0608;
    Cf  = 0.1;
    mu = 0.1;
    Cd = .05;
    Dm = 54.43;             /*  [in^3]/rad          */
    Jm = 2.432;             /*  lbf/(in/s^2)        */
    Bm = 5000.0 /*0.0*/;            /*  I made this up      */
    Ps = 3000.0;            /* psi, this is made up */
    beta_e = 100000;        /*  psi                 */
    Vt = 191.0;             /*   in^3               */
    delta_t = 0.00001;      /*  sec                  */
    PL[0] = 0.0;            /*  psi                 */
    c = 4.0*beta_e/Vt;

    f[0] = 0.0;
    PLc = 0.0;
    final_time = 0.01;



    for(i=0; i<=99; i++)
    {


       if( PL[i] <= Ps )
       {
       QL[i] = Cv*(xv)*(sqrt(Ps-PL[i]));
       }
       if( PL[i] >= Ps )
```

```
        {
        QL[i] = 0.0;
        }

/*        QL[i] = Kq*(*xv) - Kc*PL[i];
*/


        if( theta_m_dot == 0 ) aa = 1.0;
          else aa = (theta_m_dot)/(fabs(theta_m_dot));

/* linear formulation */

/*        kP1 = ( c*( QL[i] - Dm*(*theta_m_dot) - Ctm*PL[i]) )*delta_t;
        kthtdot1 = ( ( Dm*(PL[i] - aa*Cf*Ps) - (Bm + Cd*Dm*mu)*(*theta_m_dot)
)/(*Jm) )*delta_t;


        kP2 = ( c*( Kq*(*xv) - ( Kc*(PL[i] + 0.5*kP1) )
         - Dm*(*theta_m_dot + 0.5*kthtdot1) - Ctm*(PL[i] + 0.5*kP1) ))*delta_t;
        kthtdot2 = ( ( Dm*(PL[i] - aa*Cf*Ps + 0.5*kP1)
              - (Bm + Cd*Dm*mu)*(*theta_m_dot + 0.5*kthtdot1) )/(*Jm) )*delta_t;


        kP3 = ( c*( Kq*(*xv) - ( Kc*(PL[i] + 0.5*kP2) )
         - Dm*(*theta_m_dot + 0.5*kthtdot2) - Ctm*(PL[i] + 0.5*kP2) ))*delta_t;
        kthtdot3 = ( ( Dm*(PL[i] - aa*Cf*Ps + 0.5*kP2)
              - (Bm + Cd*Dm*mu)*(*theta_m_dot + 0.5*kthtdot2) )/(*Jm) )*delta_t;


        kP4 = ( c*( Kq*(*xv) - ( Kc*(PL[i] + kP3) )
         - Dm*(*theta_m_dot + kthtdot3) - Ctm*(PL[i] + kP3) ))*delta_t;
        kthtdot4 = ( ( Dm*(PL[i] - aa*Cf*Ps + kP3)
                - (Bm + Cd*Dm*mu)*(*theta_m_dot + kthtdot3) )/(*Jm) )*delta_t;
*/

/* nonlinear formulation */

        kP1 = ( c*( QL[i] - Dm*(theta_m_dot) - Ctm*PL[i]) )*delta_t;
        kthtdot1 = ( ( Dm*(PL[i] - aa*Cf*Ps) - (Bm + Cd*Dm*mu)*(theta_m_dot) )/Jm
)*delta_t;


        kP2 = ( c*( Cv*(xv)*(sqrt( Ps - (PL[i] + 0.5*kP1)))
```

```
        - Dm*((theta_m_dot) + 0.5*kthtdot1) - Ctm*(PL[i] + 0.5*kP1) ))*delta_t;
       kthtdot2 = ( ( Dm*(PL[i] - aa*Cf*Ps + 0.5*kP1)
              - (Bm + Cd*Dm*mu)*((theta_m_dot) + 0.5*kthtdot1) )/Jm )*delta_t;


       kP3 = ( c*( Cv*(xv)*(sqrt( Ps - (PL[i] + 0.5*kP2)))
        - Dm*((theta_m_dot) + 0.5*kthtdot2) - Ctm*(PL[i] + 0.5*kP2) ))*delta_t;
       kthtdot3 = ( ( Dm*(PL[i] - aa*Cf*Ps + 0.5*kP2)
              - (Bm + Cd*Dm*mu)*((theta_m_dot) + 0.5*kthtdot2) )/Jm )*delta_t;


       kP4 = ( c*( Cv*(xv)*(sqrt( Ps - (PL[i] + kP3)))
        - Dm*((theta_m_dot) + kthtdot3) - Ctm*(PL[i] + kP3) ))*delta_t;
       kthtdot4 = ( ( Dm*(PL[i] - aa*Cf*Ps + kP3)
                - (Bm + Cd*Dm*mu)*((theta_m_dot) + kthtdot3) )/Jm )*delta_t;



       PL[i+1] = PL[i] + (1.0/6.0)*(kP1 + 2.0*kP2 + 2.0*kP3 + kP4);


       Tg[i] = PL[i] * Dm;



/*        fprintf(data_file, "\t%8.3f\t%8.3f\t%7.3f\n", QL[i], PL[i], Tg[i]);
*/
    }


/*    fclose(data_file);
*/

    new_torque = Tg[99] * 0.1129848;  /* convert in-lb to N-m for tgrip */


   if ((new_torque <= 8.0) && (new_torque >= -8.0))
   {
    return( 0.0 );
   }
   if ((new_torque <= 1000.0) && (new_torque >= -1000.0))
   {
    return( new_torque );
   }
```

```
    if (new_torque >= 1000.0)
    {
     return( 1000.0 );
    }
    if (new_torque <= -1000.0)
    {
     return( -1000.0 );
    }


}
```

Appendix B. Implementing Customized Code

TeleGRIP has a shared library feature which facilitates the implementation of customized code for kinematics or dynamics routines. The user includes his code in the shared library by placing it into the shared library directory, /usr/deneb/tgrip.4d/shlib.src/User/*.c, and adding the main function of the code to the proper header file, /usr/deneb/tgrip.4d/shlib.src/Make/*.h). By using the makefile provided by TeleGRIP the user can recompile the shared libraries to include the customized code. When the shared libraries are recompiled a new shared object file is created, /usr/deneb/tgrip.4d/shlib.src/libdnbusr.so. This file is used by TeleGRIP when the program is executed.

To implement code for a customized controller, the user first inserts his code into the shared library by modifying the controller program template provided by TeleGRIP, /usr/deneb/tgrip.4d/shlib.src/User/ dyn_ctl_ex.c. Then the user modifies the file /usr/deneb/tgrip.4d/ shlib.src/Make/dyn_table.h. An example of how to modify this header file would be:

```
#ifdef file_dyn_ctl_1
#define my_dyn_control_example    dyn_control_1
#include "/usr/deneb/tgrip.4d/shlib.src/User/my_controller.c"
#endif
```

where my_controller.c is the modified template program and my_dyn_control_example is the name of the primary function in this program. After the header file has been

modified the shared library is recompiled using /usr/deneb/tgrip.4d/shlib.src/Make/Makefile. Then the TeleGRIP program can be started and access the customized controller. Because the dyn_control_1 entry of the header file was modified, the user can run the customized code through TeleGRIP by selecting User_1 as the controller through the device dynamic pipeline.

Vita

Angela Michelle Gizelar was born in October, 1968. She graduated from Brentwood High School in Brentwood, Tennessee in June, 1986. She attended the Georgia Institute of Technology where she was a cooperative student at the Georgia Tech Research Institute. She received a degree in Mechanical Engineering from Georgia Tech in December, 1991. In August, 1992 she entered the University of Tennessee, Knoxville where she worked as a research assistant, and in December, 1995 received a Master of Science degree in Mechanical Engineering.