Development of a Haptic Backhoe Testbed

A Thesis
Presented to
The Academic Faculty

by

Joseph George Frankel

In Partial Fulfillment
of the Requirements for the Degree of
Master of Science in Mechanical Engineering

School of Mechanical Engineering
Georgia Institute of Technology
May 2004

Development of a Haptic Backhoe Testbed

Approved by:

Dr. Wayne J. Book, Advisor

Dr. Harvey Lipkin

Dr. Roozbeh Kangari
(College of Architecture)

Date Approved: 7 May 2004

To my sons,

Brannon and Josh

ACKNOWLEDGEMENTS

I am grateful to many people for helping make this project a success. A master's project of this scope and complexity is far beyond the ability of a single person in a time frame of a year or so, and I humbly acknowledge the fact that without a huge amount of help and support from my teammates, colleagues, and family, I could not have gotten so far so fast. Credit is due to Matthew Kontz, my partner, friend, and fellow graduate student in Dr. Book's research group, and to J.D. Huggins, Research Engineer, mentor, and indispensable helper from the most difficult technical situations to the most mundane tasks. Matt and J.D. brought a wealth of skills to the team, and I am honored to have been able to work with them.

I would also like to thank Dr. Wayne Book, for his wisdom, experience, guidance, and support. Without Dr. Book this project could not have possibly begun in the first place. If ever I was veering off down a dead-end path, he pointed me back in the right direction by giving me suggestions or insight that only comes with the extensive knowledge and experience. I am grateful and honored to have been able to work for him.

Thanks are also due to the project sponsors. Thanks go to Paul Meyer, Mark Evans, Dave Knight, and Derek Eagles at John Deere for donating the tractor, backhoe, extra parts, and for their technical expertise. Thanks go to Randy Bobbitt at Sauer-Danfoss for helping us pick the right valve, and to Doug Hedrick for donating the valves. Thanks go to Tim Jordan and Mike Bouts at Balluff for donating the position sensors and helping us make the right choice. Thanks go to Bill Steyer and Jim Raspberry at Georgia Hydraulic Cylinders for donating both time and material by making us custom cylinders. Thanks also go to other donors and/or contributors at various levels, including Hydac, Wika, Berendsen Fluid Power, Daman Manifolds, and Tim Merkens at Brennan Industries.

I would also like to thank my two sons, Brannon and Josh, for patiently waiting for me to finish grad school while living far away from their family. I would also like to thank my

parents, Sam and Candy, for their encouragement and belief in me.

Finally, but certainly not least, I would like to thank my fiancee, Tami, for her love, support, faith, and encouragement. Not only did she encourage me to go to grad school, but in times of frustration and confusion she always seemed able to give me the right boost. I also owe her a debt of gratitude for putting up with my obsession with building the haptic backhoe, from times when I was focused and consumed, to times when I went on and on about techncal details. Behind every good man there is a good woman.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A commercial backhoe has been modified for haptic control research at Georgia Tech's Fluid Power and Motion Control Center (FPMC). Electrohydraulic valves and feedback sensors have been retrofitted to the backhoe and interfaced with a haptic joystick through a computerized control system. The resulting system provides force feedback to the hand of the operator as he or she manipulates the bucket with the joystick in Cartesian space. This system has been constructed for use as a platform for ongoing research in the area of haptic controls for the fluid power industry.

The work presented herein is divided into seven chapters. The first chapter introduces the haptic backhoe concept and provides some motivation for the project. The second chapter presents the current state of *haptics–for–hydraulics* research as presented in scientific literature. The third chapter presents kinematic and dynamic modeling of the haptic backhoe components for use both in simulation and control. The fourth chapter presents simulation results from the model derived in the preceeding chapter. The fifth chapter describes the design of the physical system. The sixth chapter presents initial test results of the backhoe moving under closed–loop haptic control. The last chapter describes the current state of the system and suggests several areas for future exploration.

It is hoped that the haptic backhoe will continue to serve as a useful research tool for many years into the future.

CHAPTER 1

INTRODUCTION

Figure 1: Trenching with the John Deere Model 47 Backhoe

The advent of a new generation of haptic input devices has opened up new possibilities for the fluid power industry. It may now be possible for a novice to become proficient at operating heavy earthmoving equipment much more quickly than was previously possible with the use of haptic control interfaces.

Heavy equipment operators have the ability to manipulate backhoes and loaders by actuating manual levers that act directly on the flow control valves. Only after extensive training and experience can operators subconciously solve the inverse kinematic relationships between lever displacements and bucket trajectories as they work, which is a learned skill that novices do not posess. It takes time to acquire a "feel" for the non-intuitive lever motions necessary to load a truck or dig a trench effectively and efficiently, because the

command motions actuate joint space variables and both the desired end-effector trajectory and visual feedback exist primarily in Cartesian space. Even more difficult is the ability to sense the forces experienced by the end-effector when the only feedback available to the operator consists of the observed bucket speed through the soil, the engine's response to a load, or pressure waves propagated back to the user's hand through the valve spool and control lever.

The traditional method to control a hydraulic excavator has been accomplished with the use of manual proportional valves. For example, a typical diesel powered earthmoving vehicle generates hydraulic oil flow with a gear pump that is mechanically driven by the engine, where flow rate remains constant at any given engine speed. At idle, oil flows continuously through a hydraulic circuit at low pressure until a portion of that oil is diverted toward a resistive load (i.e. cylinder) with a valve, causing a rise in pressure to overcome the resistance and maintain constant flow through the pump. The operator controls the proportion of the total flow that is directed to each load by displacing the spools in the valves through a direct mechanical connection between his or her hand and the spool. The spool must be displaced from its zero position to allow pressure to rise and fluid to flow from the pump to the cylinders and cause the cylinder to move. Each lever may control either one or two degrees of freedom of the excavator. Since a movement of one of the levers causes a velocity in one of the cylinders, an operator must do the inverse kinematics in his or her head to produce a desired bucket motion in Cartesian space. Although this may be second nature to an expert, it can be quite difficult for a novice to master quickly.

Therefore, a novice operator with an earthmoving task has heretofore been forced to either hire a professional, which can be expensive especially for a small job, or rent the equipment and learn how to use it him or herself before the task can be completed. However, if the traditional direct-acting valve control levers are replaced with a haptic interface, and appropriate valves and joint position sensors are installed on the excavator, three significant potential improvements may result. First, kinematic transformations can be performed as part of the real-time control loop so that the operator thinks and works solely in Cartesian space. Second, representations of forces experienced by the end-effector can be displayed

Figure 2: The PHANToM Haptic User Interface

to the users hand via the active nature of the haptic interface. Third, controls can be physically separated from the equipment and teleoperated if desired because the mechanical lever commands have been replaced with electric signals.

To prove this concept, a haptically operated backhoe has been developed at the Georgia Institute of Technology for testing and evaluating haptic control algorithms. The master haptic input device is the Personal Haptic Interface Mechanism (PHANToM) 1.0 available from Sensable Technologies, illustrated in figure 2. This device is capable of six degree of freedom position sensing and three degree of freedom force feedback in its 5"x7"x10" workspace, and comes complete with necessary drivers and configurable software.

The slave is a John Deere model 47 backhoe, donated to Georgia Tech by the John Deere company for research purposes, with four degrees of freedom and a 9.5'x18'x13' workspace. Figure 3 is a photo of the backhoe the day it was delivered to Georgia Tech.

Control is implemented with host and target computers running Matlab/Simulink/xPC Target software available from Mathworks. Electrohydraulic valves from Sauer-Danfoss and magnetorestrictive position sensors from Balluff have also been retrofitted to the backhoe.

Using the above mentioned hardware, the initial concept was to create a haptic control system for the backhoe, where the master (joystick) and slave (backhoe) communicate

Figure 3: The John Deere Model 47 Backhoe, as delivered

bilaterally, such that the master generates a position reference for the slave and the slave returns a position error to the master to use for haptic force computation as illustrated in figure 4.

Although this is the simplest approach to setting up the system and proving the concept, it is only the beginning. The goals of the project have been to create a testbed for haptic control algorithms on hydraulic earthmoving equipment. At present, a basic haptic control system has been devised, proving the haptic backhoe concept and setting the stage for future work. Now that the electrically controlled valves, sensors, computers, and basic control software have been developed, future investigators will have a platform at their disposal well suited for testing and developing more sophisticated controls. Potential areas for future research using the haptic backhoe are described in the Literature Review (chapter 2) and Conclusions (chapter 7).

The material presented herein will describe the initial work getting the project off the ground and setting up the haptic backhoe for the first time. Information is divided into two major categories: (1) modeling and analysis, and (2), design, construction and initial testing. In the first category, complete mathematical models of the valves, cylinders, and backhoe dynamics will be derived, simulated, and validated with experimental data. System identification techniques will be used to generate the valve model from data taken in the

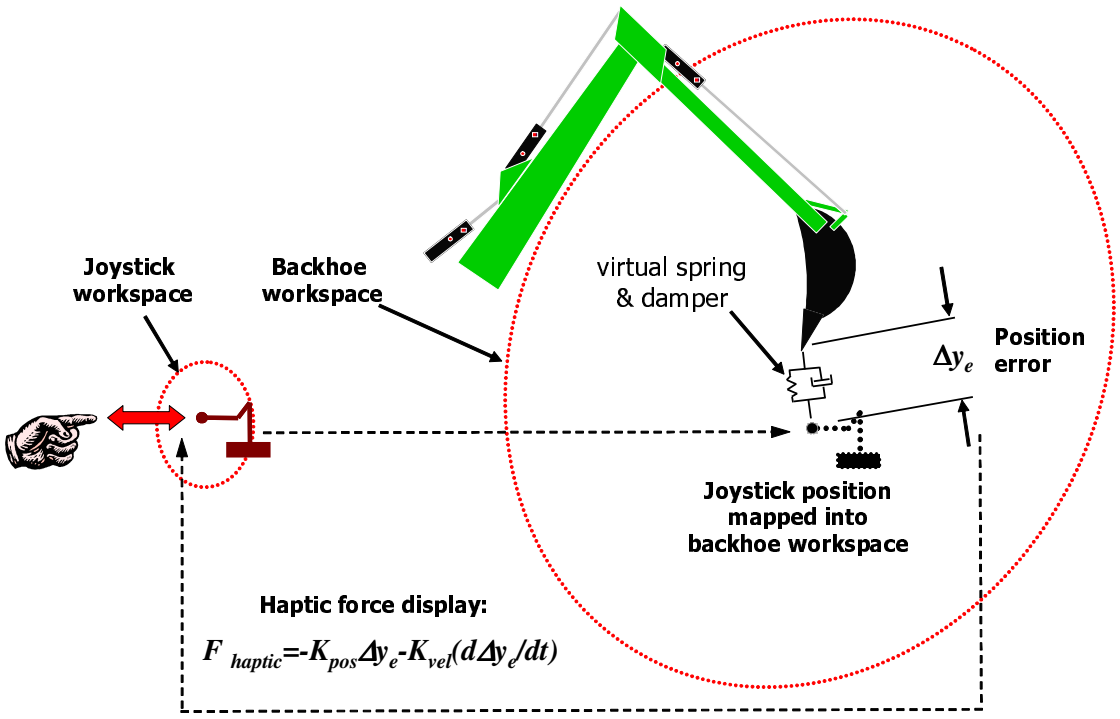


Figure 4: The Haptic Backhoe initial concept

lab. A solid model will be used for estimation of the backhoe's mass and inertia properties, and for animating the results of simulations. Also, kinematic transformations necessary for controller design will be derived.

In the second category, mechanical, electrical, and software design will be presented. The closed-loop haptic system will be shown first to work on a single degree of freedom, and then later extended to include control on all four degrees of freedom. Controller development will be described, and initial haptics testing will be documented. Finally, suggested areas of future research using the haptic backhoe system will be proposed.

Dubbed the Haptically Enhanced Robotic Excavator (HEnRE), figure 5 is a photo of the backhoe at the point of completion of the present work.

A comprehensive website documenting the development of the haptic backhoe, including archives of all component specifications, control software, Pro/E models, and reports is located at

http://www.imdl.gatech.edu/jfrankel/backhoe.htm

Figure 5: Haptic Backhoe with team members Matt Kontz (lower left) and Joe Frankel (upper right). Missing from the photo is the third team member, J.D. Huggins.

CHAPTER 2

LITERATURE REVIEW

In recent years, there has been a great deal of research in the area of haptic feedback and control. Much of this work can be divided into two areas: interfaces for virtual environments, where a user can "feel" the virtual surfaces of a CAD model, such as in [4], and robotic teleoperation, where the user feels a representation of the forces experienced by a mechanism under remote control, such as in [17], [22], and [14]. Other areas receiving somewhat less attention include assisted manufacturing and assembly [29], assisted surgery and surgical training [10], and tactile aids for the visually impaired [9]. The haptic interface is typically a joystick–like mechanism with one to six degrees of freedom, with position sensors and force displaying actuators. More exotic, higher order interfaces have also been created such as instrumented gloves [40] and digitized deformable surfaces [30].

In addition, a large amount of literature is available on the electrohydraulic control of hydraulic earthmoving equipment, for example [11], [27], and [1]. However, much less work has been done using a haptic interface as part of the electrohydraulic control system. The most relevant work the author is aware of is described in [31], [33], and [23]. Other relevant works include [6] and [18]. However, with the advent of a new generation of commercially available haptic interfaces such as the PHANToM, the field of *haptics-for-hydraulics* will undoubtedly expand significantly in the future due to the enhancements that can be made to excavation using haptic control.

This leads to the question: how can haptics be used to improve the control of hydraulic earthmoving equipment? An effort to create a laboratory testbed to develop new haptics–for–hydraulics technologies would require laying groundwork in several areas. One can envision the need for a suitable excavator, haptic interface, electrohydraulic valves, and computer control system. Components should be selected such that the system is flexible, so that various control algorithms can be investigated as ideas unfold with little or no

hardware modifications.

As a parallel effort, mathematical models of the excavator, valves, cylinders, and soil would provide useful insight into the dynamics of the system. Models could also be used for both controller design [42], as well as to predict the performance of various control algorithms in simulation before implementing them on the physical hardware. In addition, models could also be adapted for real-time execution, for purposes such as endpoint force estimation [20], force tracking [11], dynamic representation in the master computer during teleoperation [14], and state observers [27]. Whether used on– or off–line, for real–time control, simulating the response of a proposed control algorithm, or simply to gain insight into the system dynamics, mathematical models would be valuable and useful tools for developing and testing new haptics–for–hydraulics technologies.

The logical starting point for developing a comprehensive system model is a kinematic analysis. Kinematic relationships between joint angles, velocities, and torques and their corrsponding end effector positions, velocities, and forces are well known and documented [5], [35], [16], [28]. Using standardized Denavit–Hartenberg notation and the geometric Jacobian matrix, these relationships can be computed quickly and efficiently in generalized coordinates. Kinematic algorithms developed for this project are presented in section 3.2.

The next logical component of a system model would be a description of the dynamics that relate applied forces to the resulting motions in the excavator's links. Two main approaches can be found in the literature: recursive Newton-Euler dynamic models based on $\Sigma F = ma$ that relate the motion of one link to the next in a serial chain, and nonrecursive LaGrangian dynamic models based on kinetic and potential energy. Both of these modeling approaches are also well known and documented [5], [39]. It is not surprising the two approaches have their advantages and disadvantages. The LaGrangian model seems to be the most prevalent in the literature [11], [28] because it provides the most intuitive insight into the dynamics of the system, albeit at the cost of computational complexity [41]. The Newton-Euler model, on the other hand, can be solved with fewer calculations:

> "...LaGrange's equation gives the designer physical insight needed to understand the behavior of the overall system, but the resulting equations are often computionally complex." [16]

Because the mathematical model of the backhoe is to be used primarily for insight into the system dynamics and is not intended to be run in real time, a LaGrangian dynamic model will be developed, which is described in section 3.3.2.

The next logical component of the system model would be a model of the valves. A common approach to modeling hydraulic control valves begins with an analysis of fluid flow through a sharp-edged orifice, which describes the oil flow past the valve spool [12], [13], [24], [28]. Based on the assumption of constant energy along a streamline—i.e. Bernoulli's equation—the flow Q through a sharp-edged orifice can be modeled as proportional to the square root of the pressure drop across the orifice Δp, where in general $Q = C_d A\sqrt{\Delta p}$. The discharge coefficient C_d is a parameter of the orifice, and A is the cross-sectional flow area. An excellent early work describing the modeling of valves can be found in [25].

However, because of the sophistication of modern electrohydraulic valves, an accurate valve model would also require dynamic elements in addition to the orifice flow equations to fully capture its performance. Much of the literature reviewed reports the use of servo valves for hydraulic control [1], [11], [27]. These typically have 50–100Hz bandwidths [26] and cost upwards of $2500 each. In contrast, sponsors of this project have requested that low–cost valves be used to more closely emulate hardware that might end up on a commercial model if haptic feedback were to be put into production. Section 3.4 describes the Sauer-Danfoss PVG32 valve that was selected for the haptic backhoe, and the work done generating a model for it. This valve has a bandwidth on the order of 6Hz and a single unit cost around $1500, a price that could certainly be reduced for volume production. Previous researchers have also generated a model of the PVG32 in [3], but only a second order linear model is given, where all the dynamics are assumed to take place in the main spool mass. Regardless of the valve used, the performance of the hydraulic system will be determined by the valves, and as such a mathematcial model of the valves will be a critical component of the overall system model.

The next modeling component would need to represent the forces between the slave and its environment. In the literature, soil–tool interaction forces are typically modeled as a mass/spring/damper system. For example, a one–dimensional model of the relationship

between the force exerted by the slave on the environment and the resulting slave position would take the form $X(s)/F(s) = 1/(ms^2 + bs + k)$, with the spring and damper terms representing the soil impedance [31], [42].

A more sophisticated model has been developed and described in [6]. This is the most realistic soil model discovered in the literature to date. A dynamic, real-time digging simulation with gross soil deformations and bucket filling is displayed in a virtual environment, while force feedback is also displayed to the operator via a haptic magnetically levitated joystick.

The final piece of the puzzle necessary to assemble a haptics–for–hydraulics testbed would be the controller. In designing a contoller with haptic force feedback, the two most fundamental goals are to provide both stability and *transparency*. The latter is achieved when the operator cannot distinguish between manipulating the master and manipualting the slave [31]. It can be shown mathematically that some types of haptic controllers can provide perfect transparency, while others cannot. A review of the available literature indicates that the approach to controller design is usually a combination of one or more of position control, rate control, force control, or impedance control, although other techniques have been proposed and shown to perform effectively [42].

Position control is the simplest method to control the slave. With position control, the master position is scaled and mapped directly into the slave's workspace to provide a desired reference. Position errors can be regulated based upon Cartesian, joint, or cylinder space variables. Unfortunately, problems arise as soon as large soil–tool interaction forces are present—i.e. when digging—even though this method does provide satisfactory results during unconstrained motion. Therefore, the controller may need to switch into another mode when the bucket comes in contact with the soil:

> "Simple trajectory control almost never suffices unless the mechanism can completely overpower the resistance during digging. Hence, most methods that control the bucket during earthmoving operations are coupled to force or position feedback." [37]

Rate control, on the other hand, typically starts by defining a datum point in the master's task space, and a velocity command is generated for the slave based on the master's displacement from the datum. This is the most common form of joystick control. Experiments have shown that operator's prefer rate control over position control, reporting that it provides superior accuracy and a lighter work load [23]. However this method is also not without its shortcomings. It can be shown that, although perfect transparency can be achieved with rate control, hand forces must be integrated and environment forces must be differentiated, resulting in a system with a limited range of stability [20].

Force control, the third type of basic control, can be used to produce a desired force on the environment based on the master position by adjusting cylinder pressures. The force applied to the slave environment becomes the result of these pressures and the geometric configuration of the slave. One simple force control scheme has been presented in [1], where a reference is tracked such that a desired force is exerted on the environment by a single hydraulic cylinder. In another work, a sliding mode force controller that tracks a linear second order model of the cylinder rod dynamics has been presented in [27], and extended into a more comprehensive nonlinear excavator model in [11].

Another type of force control involves creating a force on the environment based on the force exerted onto the master. This method is most useful when the slave is in contact with the environment, and provides a realistic experience for the operator when digging compared to position or rate control. However, instability problems arise as the feedback gain is increased, especially when the bucket first comes in contact with the soil [23].

Additionally, force control requires a measurement of the forces experienced at the end-effector, which has been accomplished using either pressure transducers [27] or load pins at the joints [11], [31], [2].

Impedance control is a hybrid scheme that compromises between position control and force control. When the slave's environment has low impedance—i.e. moving in free space—the controller is in position mode and the master impedance is set high. On the other hand, when the slave's environment has high impedance—i.e. digging—the controller is in force mode and the master impedanece is set low. In position mode, the master displays a

high impedance to the operator for good trajectory tracking, and the slave acts as a force source/position sensor. In force mode, the master displays low impedance to the operator to minimize effort while digging, and the slave acts as a position source/force sensor. The controller transitions between control modes based on the ratio of the slave's force on the environment to the slave's velocity. A great deal or work has been done in this area at the University of British Columbia and is presented in [31], [11], [20], and [32]. However, these assume a known, constant slave environment impedance. For a more flexible model, an adaptive control method for mapping unknown environment impedance in real time is presented in [22].

Many other types of control schemes exist for future exploration. For example, one technique has been proposed that seeks to maximize both force and position transparency using H_∞ optimization [42]. Using this approach, a closed–loop transfer function is derived, based on a known plant model and an unknown controller, that relates a vector of user inputs and disturbances to a vector of errors in perfect tranparency. Then, the controller can be designed to minimize the ∞–norm of the transfer function, so that zero output corresponds to perfect transparency. Recall that perfect transparency implies that both scaled force representations and kinematic relationships are maintained perfectly between master and slave. Obviously, this will only be possible over a finite range of frequencies. It is the author's opinion that this approach shows the greatest promise for future research and (regretfully) lies beyond the scope of the present work.

It should be noted that haptics–for–hydraulics research is also underway in private industry, where results are held proprietary and therefore unavailable in literature. For example, John Deere representatives indicate that Caterpillar, Inc. is exploring autonomous excavation using an ummanned trackhoe. Also, Kraft Telerobotics claims to have developed a complete multi–degree of freedom haptic control system suitable for retrofit on a variety of heavy equipment. Originally intended for hazardous material handling, this system was reported in 1992 in [19]. However, verbal communications with Kraft indicate that they are not willing disclose technical details.

Based on the literature review described above, as well as the John Deere company's

desire to explore haptics–for–hydraulics control algorithms, a testbed was constructed at the Georgia Institute of Technology's Intelligent Machine Dynamics Laboratory (IMDL) to develop haptic control technologies for the fluid power industry. A model 47 backhoe, donated by John Deere, has been retrofitted with electrohydraulic valves, position sensors, pressure sensors, a haptic display interface, and control computers. In addition, a comprehensive mathematical model of the backhoe was developed, both of which are described in this work.

Obviously, the scope of a project intended to test haptic–for–hydraulic control algorithms can easily go beyond the bounds of a typical master's thesis. The work described herein primarily relates to system design and integration, where the functional hardware and mathemetical models are itended to be passed on to future researchers. As such, only the most preliminary experimental results are presented.

CHAPTER 3

MODELING

This section describes the kinematic relationships between the bucket position, cylinder positions, and joint angles, and also the dynamic relationships between the cylinder forces, joint torques, and resulting motions of the backhoe's links. These four articulated links are the swing, boom, stick, and bucket.

3.1 Component Frames and Labeling Conventions

The backhoe can be modeled as a 4R (four revolute joint) hybrid serial-parallel mechanism, with the swing joint axis normal to the ground and the remaining three joint axes parallel to the ground.

3.1.1 Notation

Figure 6 illustrates the component frames and joint angles as used throughout this work. Shaded regions represent the four links of the backhoe, dashed lines represent cylinders, and black dots represent pin joints which move with the link to which they are attached. Note that the relevant points in the mechanism have been labeled with numbers O_0-O_4 for the joint axis origins and letters A-H for the remaining axes relevant to the analysis. When describing linear and angular dimensions, the following convention has been ascribed: All linear (scalar) dimensions are given as r_{xy}, where x and y are either the numeric or alphabetic joint labels, with numbers always before letters, and arranged in ascending order. For example, the distance from point B to origin O_1 would be r_{1B}, etc. Angular quantities are given in the format θ_{xyz}, where x, y, and z are the three points describing the angle in the counterclockwise direction. For example, the angle between the lines $\overline{O_2 O_3}$ and $\overline{EO_3}$ is given by θ_{23E}. See Figure 6.

Scalar quantities are shown in italics. For example, the swing angle is given by θ_1.

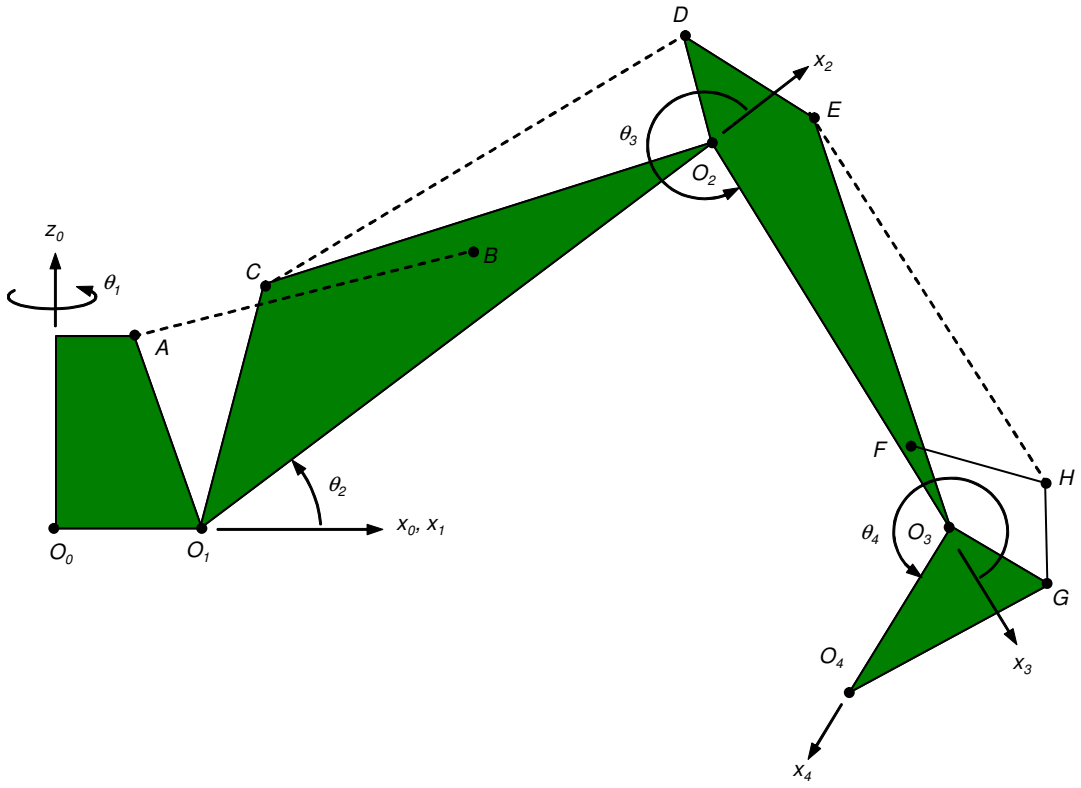


Figure 6: Backhoe Component Frames and Joint Angles

Table 1: Denavit-Hartenberg Parameters

Joint #	Link Lengths a_i	Joint Angles θ_i	Joint Offsets d_i	Twist Angles α_i
1	8.5 in	θ_1	0	90°
2	48 in	θ_2	0	0
3	38.7 in	θ_3	0	0
4	18.5 in	θ_4	0	0

Vector and matrix quantities are shown in boldface format, with one-dimensional column vectors in lower case and two-dimensional matrices in uppercase. For example, $\mathbf{a} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \end{bmatrix}^T$ indicates the four element link length vector and $\mathbf{M}$ is the 4x4 manipulator inertia matrix.

Position vectors are indicated in the format $\mathbf{p}_{xy}^{f}$, where the superscript f denotes the component frame and the subscript xy denotes the starting and ending points. For example, the notation $\mathbf{p}_{C3}^{2}$ denotes the position vector from point C to origin O_3 in component frame 2 (boom coordinates).

Cylinder lengths are denoted by the vector $\mathbf{y}_c$, with elements y_{ci}, $i = 1 : 5$. Although the backhoe has four degrees of freedom, the swing link is associated with the first two cylinders, so y_{c3} controls the boom, y_{c4} the stick, and y_{c5} the bucket. See Figure 6.

3.1.2 Denavit-Hartenburg Parameters

Some of the kinematic transformations make use of homogeneous transformation matrices and the geometric Jacobian. For these computations, the Denavit-Hartenberg (DH) labeling convention is used as in [35]. Link origins O_i are located at the distal ends of each link, the link lengths a_i are measured from origin O_{i-1} to O_i along x_i and joint angles θ_i are measured about z_{i-1}. The joint angles vary with the backhoe configuration and represent the generalized coordinates for the formulation given in section 3.3.2 The DH parameters for the backhoe are given in Table 1.

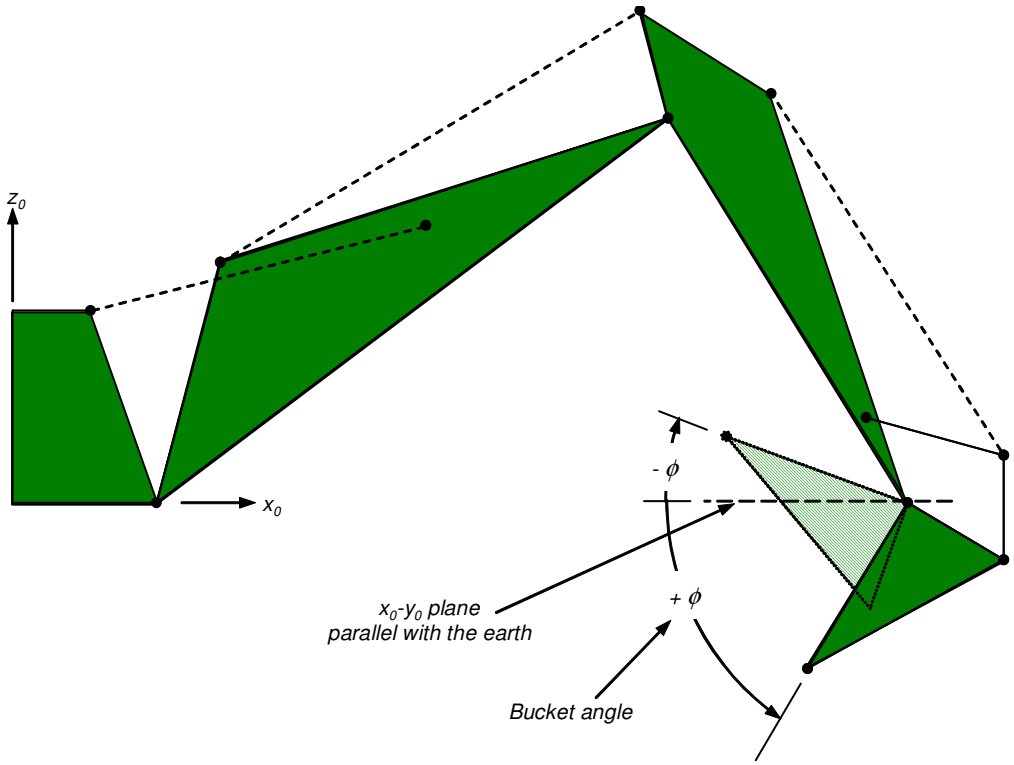


Figure 7: Bucket Angle

3.1.3 Bucket angle

The bucket angle ϕ has been defined as the angle between the $\mathbf{x_4}$ axis (bucket) and the $\mathbf{x_0 - y_0}$ plane, which is assumed to be parallel with the earth. The bucket angle is defined as positive downward as illustrated in Figure 7. This angle will be useful for describing the desired bucket trajectory in base frame coordinates. Finally, subscripts and indices in parenthesis are used interchangeably in some cases. For example, $\theta_1 = \theta(1)$, etc.

3.2 Kinematics

During simulation, the orientation of the backhoe can be expressed as a vector of coordinates in one of three vector spaces: cylinder space, joint space, or Cartesian space. In each case, exactly four elements are required to uniquely determine the backhoe's orientation, or pose. In order to simulate and/or control a desired motion, sets of equations are required to transform coordinates between vector spaces which must be computed at each simulation time step. Table 2 illustrates the coordinates in each vector space.

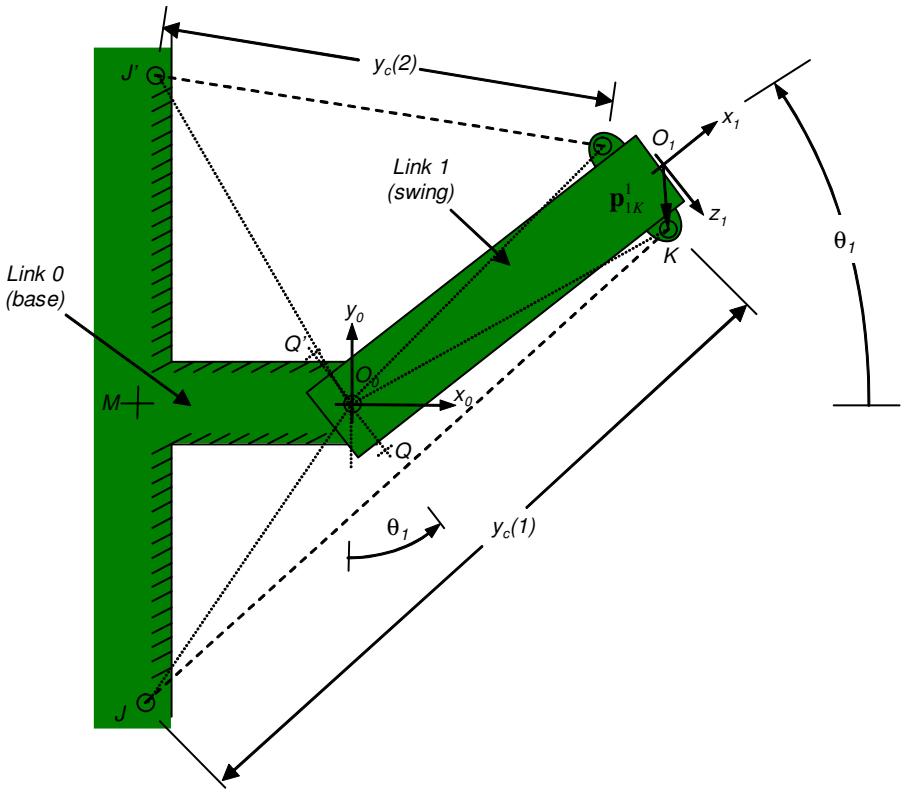


Figure 8: Link 1 Transformation

Table 2: Vector Spaces of Backhoe Coordinates

Associated Link	Cylinder Space	Joint Space	Cartesian Space
Swing	y_{c1}, y_{c2}	θ_1	y
Boom	y_{c3}	θ_2	z
Stick	y_{c4}	θ_3	x
Bucket	y_{c5}	θ_4	ϕ

3.2.1 Cylinder Space to Joint Space Transformation

This section derives the equations necessary to compute the joint angles from the cylinder lengths by analyzing each link independently from base to tip. During simulation, all quantities are either known constants or functions of the input variables y_{ci}.

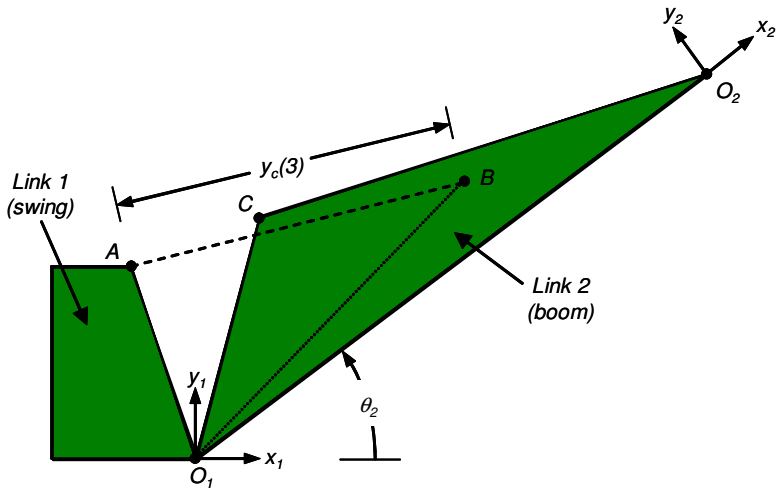


Figure 9: Link 2 Transformation

3.2.1.1 Swing joint angle

Figure 8 is a sketch of the base (tractor) and swing links as viewed from above. The transformation from the known cylinder length y_{c1} to the joint angle θ_1 is computed as follows:

$$\theta_{K0J} = \cos^{-1}\left(\frac{r_{0K}^2 + r_{0J}^2 - y_{c1}^2}{2r_{0K}r_{0J}}\right) \tag{1}$$

$$\theta_1 = \theta_{K0J} - \theta_{K0Q} - \theta_{MJ0} \tag{2}$$

Note that in practice, the angle θ_{K0J} must be checked to see whether it is greater than 180°, which requires knowledge of y_{c2} as well.

3.2.1.2 Boom joint angle

Figure 9 is a sketch of the swing and boom links as viewed from the side. The transformation from the boom cylinder length y_{c3} to the joint angle θ_2 is as follows.

$$\theta_{A1B} = \cos^{-1}\left(\frac{r_{1A}^2 + r_{1B}^2 - y_{c3}^2}{2r_{1A}r_{1B}}\right) \tag{3}$$

$$\theta_2 = \pi - \theta_{01A} - \theta_{A1B} - \theta_{B12} \tag{4}$$

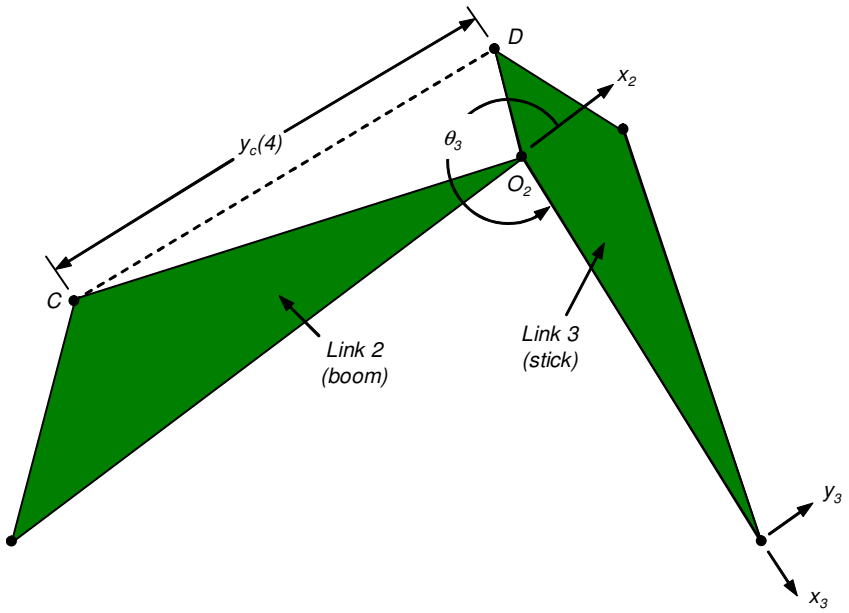


Figure 10: Link 3 Transformation

3.2.1.3 Stick joint angle

Figure 10 is a sketch of the boom and stick links as viewed from the side. Note that the knuckle links between points F, G, and H are not shown around O_3 in figure 10. The transformation from the stick cylinder length y_{c4} to the joint angle θ_3 is as follows.

$$\theta_{C2D} = \cos^{-1}\left(\frac{r_{2C}^2 + r_{2D}^2 - y_{c4}^2}{2r_{2C}r_{2D}}\right) \tag{5}$$

$$\theta_3 = 3\pi - \theta_{12C} - \theta_{C2D} - \theta_{D23} \tag{6}$$

3.2.1.4 Bucket joint angle

Figure 11 is a sketch of the stick and bucket links as viewed from the side. The transformation from the boom cylinder length y_{c5} to the joint angle θ_4 is as follows.

$$\theta_{EFH} = \cos^{-1}\left(\frac{r_{EF}^2 + r_{FH}^2 - y_{c5}^2}{2r_{EF}r_{FH}}\right) \tag{7}$$

$$\theta_{HF3} = \pi - \theta_{DFE} - \theta_{EFH} \tag{8}$$

$$r_{3H} = \sqrt{r_{3F}^2 + r_{FH}^2 - 2r_{3F}r_{FH}\cos\left(\theta_{HF3}\right)} \tag{9}$$

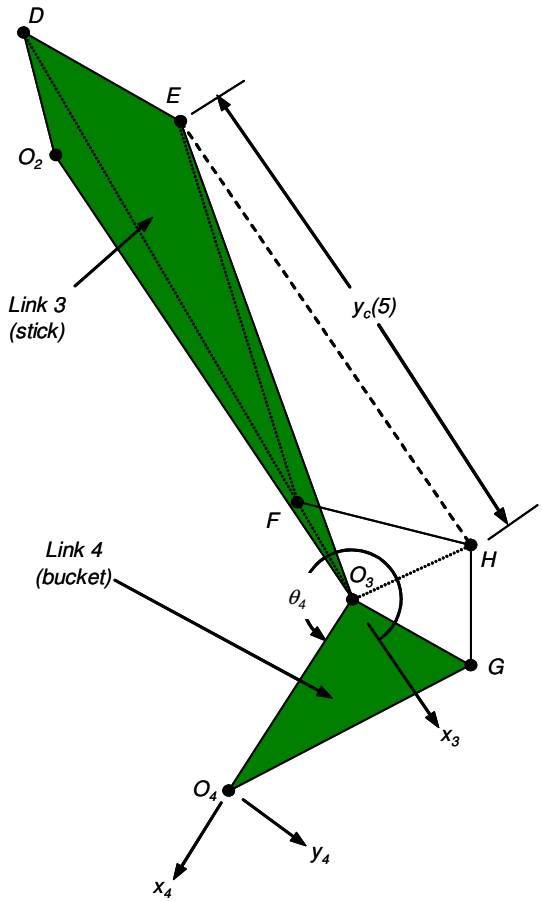


Figure 11: Bucket Parameters

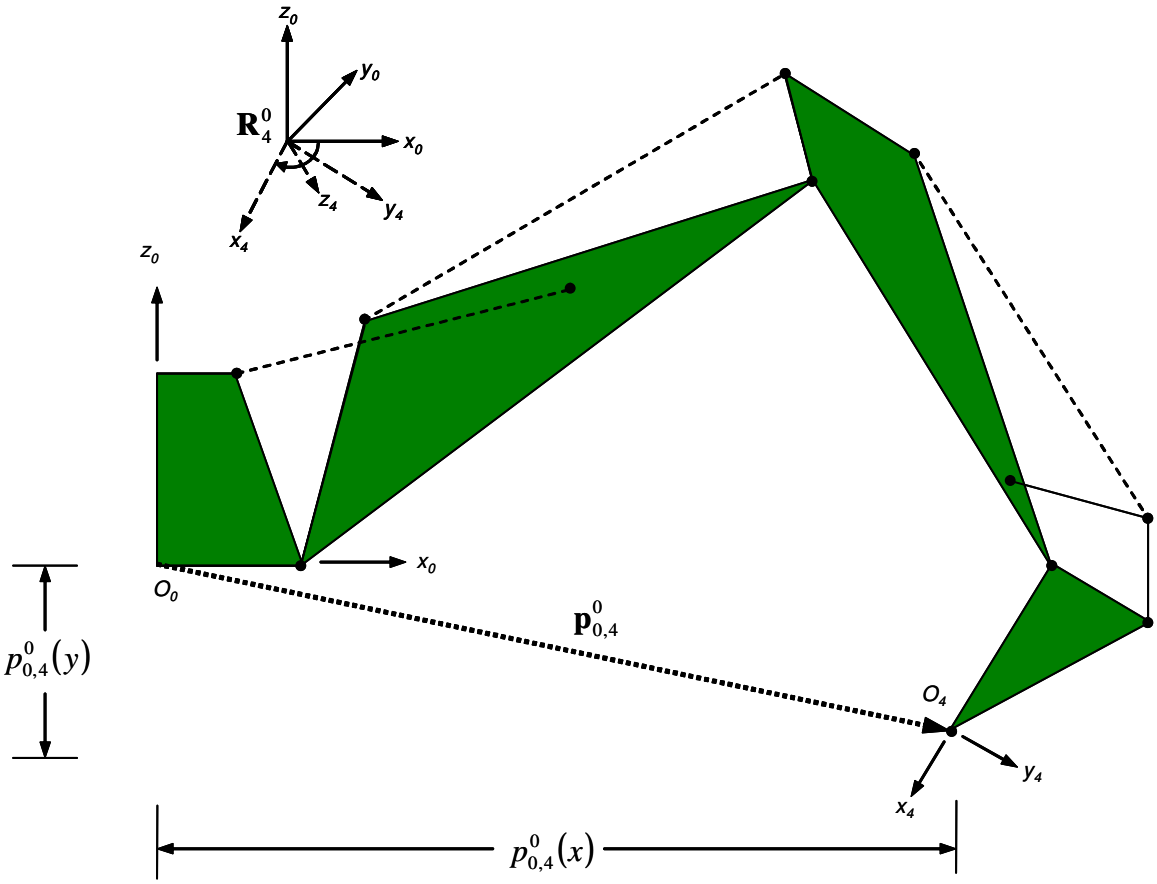


Figure 12: Bucket Position

$$\theta_{F3H} = \cos^{-1}\left(\frac{r_{3F}^2 + r_{3H}^2 - r_{FH}^2}{2r_{3F}r_{3H}}\right) \tag{10}$$

$$\theta_{H3G} = \cos^{-1}\left(\frac{r_{3H}^2 + r_{3G}^2 - r_{GH}^2}{2r_{3H}r_{3G}}\right) \tag{11}$$

$$\theta_4 = 3\pi - \theta_{F3H} - \theta_{H3G} - \theta_{G34} - \theta_{23D} \tag{12}$$

Thus with knowledge of the cylinder lengths yc, the joint angles θ can be calculated from Equations (1)-(12).

3.2.2 Forward Displacement Analysis

The Forward Displacement Analysis (FDA) computes the position and orientation of the bucket from the joint angle vector θ.

Beginning with the joint angles θ that have been computed as described in section 3.2.1, the position and orientation of each component frame (i.e. each link) are computed

recursively from the base O_0 to the bucket tip O_4 with the successive multiplication of homogeneous transformation matrices [35]. These transformations, computed from the joint angles and the Denavit-Hartenberg parameters listed in Table 1, are given by

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_i^{i-1} & \mathbf{p}_{i-1,i}^{i-1} \\ \mathbf{0}^{(1x3)} & 1 \end{bmatrix} \tag{13}$$

where $\mathbf{R}_i^{i-1}$ is the 3x3 rotation matrix from frame *i-1* to frame i and $\mathbf{p}_{i-1,i}^{i-1}$ is the position vector of origin i with respect to origin *i-1* expressed in the *i-1* component frame.

The position and orientation of the end effector (i.e. bucket tip) is then computed from

$$\mathbf{B} = \mathbf{A}_1^0\mathbf{A}_2^1\mathbf{A}_3^2\mathbf{A}_4^3 = \begin{bmatrix} \mathbf{R}_4^0 & \mathbf{p}_{0,4}^0 \\ \mathbf{0}^{(1x3)} & 1 \end{bmatrix} \tag{14}$$

where $\mathbf{B}$ is termed the *bucket displacement matrix*. Figure 12 illustrates the transformation.

The rotation matrix $\mathbf{R}_4^0$ can be interpreted as a matrix of projections of the base frame unit vectors onto bucket frame unit vectors,

$$\mathbf{R}_4^0 = \begin{bmatrix} \mathbf{x}_0\cdot\mathbf{x}_4 & \mathbf{x}_0\cdot\mathbf{y}_4 & \mathbf{x}_0\cdot\mathbf{z}_4 \\ \mathbf{y}_0\cdot\mathbf{x}_4 & \mathbf{y}_0\cdot\mathbf{y}_4 & \mathbf{y}_0\cdot\mathbf{z}_4 \\ \mathbf{z}_0\cdot\mathbf{x}_4 & \mathbf{x}_0\cdot\mathbf{y}_4 & \mathbf{z}_0\cdot\mathbf{z}_4 \end{bmatrix} \tag{15}$$

and $\mathbf{p}_{0,4}^0 = \begin{bmatrix} p_{0,4}^0(x) & p_{0,4}^0(y) & p_{0,4}^0(z) \end{bmatrix}^T$ is the bucket tip relative to the base frame, expressed in base frame coordinates.

Once the bucket displacement matrix is computed, the bucket angle ϕ is then found from the $\mathbf{x_4}$ vector relative to the $\mathbf{x_0}$-$\mathbf{y_0}$ plane. However, since the $\mathbf{x_1}$ axis is constrained to rotate in the $\mathbf{x_0}$-$\mathbf{y_0}$ plane and $\mathbf{y_1}$ is always parallel to $\mathbf{z_0}$, the bucket angle is computed from

$$\phi = \text{atan2}(y_1\cdot x_4, x_1\cdot x_4) + \pi \tag{16}$$

Note that in Matlab, the output of the *atan2* function is limited to $-\pi < \phi < +\pi$ (see Appendix D), so in the case that $\mathbf{x_1}\cdot\mathbf{x_4} < 0$ and $\mathbf{y_1}\cdot\mathbf{x_4} > 0$, the bucket is curled up tight

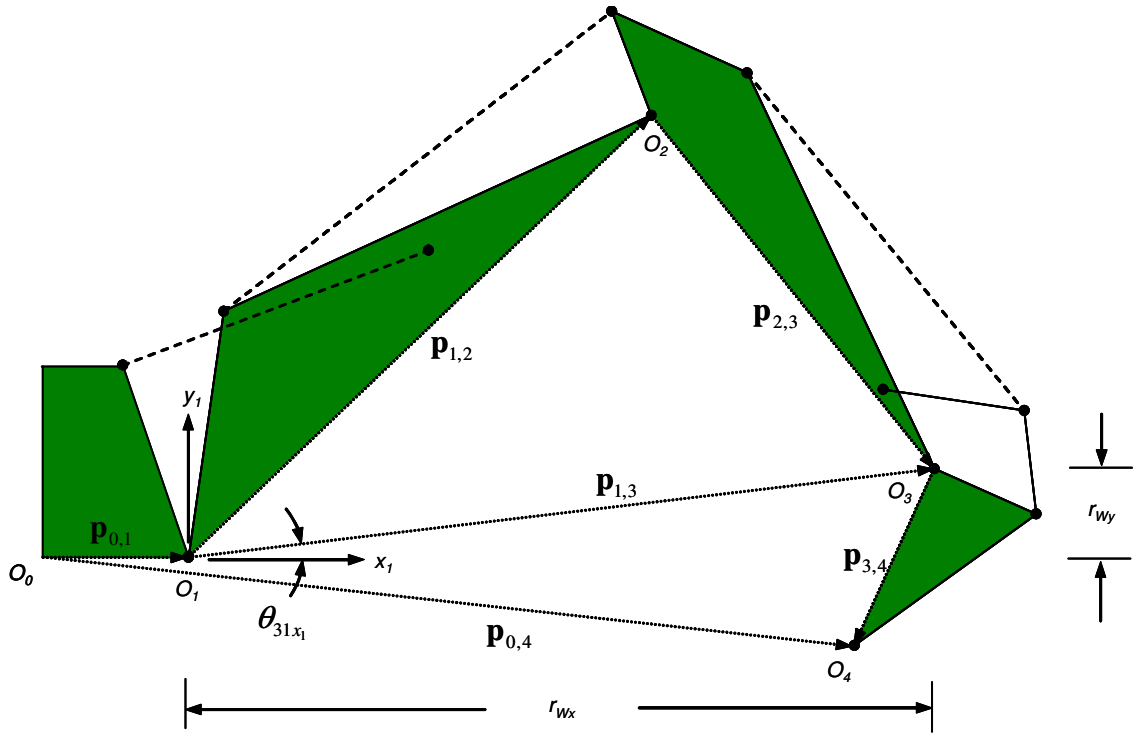


Figure 13: Reverse Displacement Vectors

next to the stick, and equation (17) becomes

$$\phi = \text{atan2}\left(\frac{z_0 \cdot x_4}{x_0 \cdot x_4}\right) - \pi. \tag{17}$$

Thus the position and orientation of the bucket can be computed from knowledge of the joint angles by using equations (13)-(17).

3.2.3 Reverse Displacement Analysis

The reverse displacement analysis (RDA) is used to compute the joint angle vector θ from the bucket tip position $\mathbf{p}^0_{0,4} = \mathbf{p}^\mathbf{0}_\mathbf{b}$, and the bucket angle ϕ. Although this operation is the inverse of that described in section 3.2.2, it is computed quite differently. Figure 13 illustrates the quantities relevant to the RDA.

3.2.3.1 Swing angle by RDA

The swing angle θ_1 is decoupled from the other links and can be computed directly from the x and y components of the bucket position vector $\mathbf{p}_{0,4}^0$:

$$\theta_1 = atan2\left(p_{0,4}^0(y), p_{0,4}^0(x)\right) \tag{18}$$

3.2.3.2 Boom and stick angles by RDA

The remaining three links form a planar arm, and the joint angles are computed as follows. Using the rotation matrices $\mathbf{R}_1^0$ and $\mathbf{R}_4^0$ to express all vectors in frame 1, the wrist position O_3 relative to O_1 is

$$\mathbf{p}_{1,3}^1 = \mathbf{p}_{0,4}^1 - \mathbf{p}_{0,1}^1 - \mathbf{p}_{3,4}^1 = \mathbf{R}_1^0\mathbf{p}_{0,4}^0 - \begin{bmatrix} a_1 \\ 0 \\ 0 \end{bmatrix} - \mathbf{R}_4^0 \begin{bmatrix} a_4 \\ 0 \\ 0 \end{bmatrix} \tag{19}$$

where elementary rotations can be multiplied in the same manner as homogeneous transformations:

$$\mathbf{R}_4^0 = \mathbf{R}_1^0\mathbf{R}_2^1\mathbf{R}_3^2\mathbf{R}_4^3 \tag{20}$$

and the distance and angle formed from O_1 to the wrist O_3 is

$$r_{13} = \sqrt{\left(p_{1,3}^1(x)\right)^2 + \left(p_{1,3}^1(y)\right)^2} = \sqrt{(r_{Wx})^2 + (r_{Wy})^2} \tag{21}$$

$$\theta_{31x_1} = \tan^{-1}\left(\frac{p_{1,3}^1(y)}{p_{1,3}^1(x)}\right) = \tan^{-1}\left(\frac{r_{Wy}}{r_{Wx}}\right) \tag{22}$$

Now that all the sides of the triangle $O_1O_2O_3$ are known, the interior angles can be found using the cosine law,

$$\theta_{321} = \cos^{-1}\left(\frac{a_2^2 + a_3^2 - r_{13}^2}{2a_2a_3}\right) \tag{23}$$

$$\theta_{213} = \cos^{-1}\left(\frac{a_2^2 + r_{13}^2 - a_3^2}{2a_2r_{13}}\right) \tag{24}$$

and then the angles θ_2 and θ_3 can be found from

$$\theta_2 = \theta_{31x_1} + \theta_{213} \tag{25}$$

$$\theta_3 = \pi + \theta_{321} \tag{26}$$

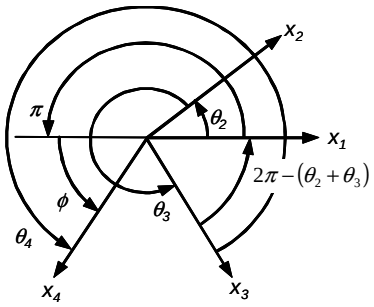


Figure 14: θ_4 Angle analysis

3.2.3.3 Bucket angle by RDA

Figure 14 illustrates the relationship between the bucket angle ϕ and the joint angles θ_2, θ_3 and θ_4. The last angle θ_4 is found from the bucket angle ϕ, where

$$[2\pi - (\theta_2 + \theta_3)] + \pi + \phi = \theta_4 \tag{27}$$

or

$$\theta_4 = \phi - \theta_2 - \theta_3 + 3\pi \tag{28}$$

3.2.4 Joint Space to Cylinder Space Transformation

The joint space to cylinder space transformation is used to compute the cylinder lengths $\mathbf{y}_c$ from the joint angles θ. This is the inverse of the operation described in section 3.2.1. Combined with the reverse displacement algorithm, these routines are useful to compute the required cylinder length time histories for control purposes.

3.2.4.1 Swing cylinder length

The lengths of the swing cylinders y_{c1} and y_{c2} are computed from θ_1 as follows.

$$\theta_{Q0J} = \tan^{-1}\left(\frac{r_{0M}}{r_{JM}}\right) + \theta_1 \tag{29}$$

$$\theta_{K0J} = \theta_{Q0J} + \frac{\pi}{2} - \theta_{10K} \tag{30}$$

$$y_{c1} = \sqrt{(r_{0J})^2 + (r_{0K})^2 - 2r_{0J}r_{0K}\cos(\theta_{K0J})} \tag{31}$$

$$\theta_{J'0Q'} = \tan^{-1}\left(\frac{r_{0M}}{r_{JM}}\right) - \theta_1 \tag{32}$$

$$\theta_{J'0K'} = \theta_{J'0Q'} + \frac{\pi}{2} - \theta_{10K} \tag{33}$$

$$y_{c2} = \sqrt{(r_{0J})^2 + (r_{0K})^2 - 2r_{0J}r_{0K}\cos(\theta_{J'0K'})} \tag{34}$$

Refer to Figure 8 for equations 29-34.

3.2.4.2 Boom cylinder length

The boom cylinder length is found by rearranging equations (3) & (4):

$$\theta_{A1B} = \pi - \theta_{01A} - \theta_2 - \theta_{B12} \tag{35}$$

$$y_{c3} = \sqrt{(r_{1A})^2 + (r_{1B})^2 - 2r_{1A}r_{1B}\cos(\theta_{A1B})} \tag{36}$$

Refer to Figure 9 for equations (35) & (36).

3.2.4.3 Stick cylinder length

The stick cylinder length is found by rearranging equations (5) & (6):

$$\theta_{C2D} = \pi - \theta_{D23} - \theta_{12C} - \theta_3 \tag{37}$$

$$y_{c4} = \sqrt{(r_{2C})^2 + (r_{2D})^2 - 2r_{2C}r_{2D}\cos(\theta_{C2D})} \tag{38}$$

3.2.4.4 Bucket cylinder length

The bucket cylinder length is found from four angle additions and four cosine laws in the geometry between the knuckle links *FH* and *GH*. Refer to Figure 11 for equations (39)-(46).

$$\theta_{x_3 3G} = 2\pi - \theta_{G34} - \theta_4 \tag{39}$$

$$\theta_{G3F} = \pi - \theta_{x_3 3G} + \theta_{23D} \tag{40}$$

$$r_{FG} = \sqrt{(r_{3F})^2 + (r_{3G})^2 - 2r_{3F}r_{3G}\cos(\theta_{G3F})} \tag{41}$$

$$\theta_{3FG} = \cos^{-1}\left(\frac{r_{3F}^2 + r_{FG}^2 - r_{3G}^2}{2r_{3F}r_{FG}}\right) \tag{42}$$

$$\theta_{HFG} = \cos^{-1}\left(\frac{r_{FG}^2 + r_{FH}^2 - r_{GH}^2}{2r_{FG}r_{FH}}\right) \tag{43}$$

The angle θ_{HF3} depends on whether $\theta_4 + \theta_{G34} > 2\pi$:

if

$$\theta_4 + \theta_{G34} > 2\pi$$

then

$$\theta_{HF3} = \theta_{HFG} + \theta_{3FG}$$

otherwise

$$\theta_{HF3} = \theta_{HFG} - \theta_{3FG} \tag{44}$$

then

$$\theta_{EFH} = \pi - \theta_{DFE} - \theta_{HF3} \tag{45}$$

$$y_{c5} = \sqrt{(r_{EF})^2 + (r_{FH})^2 - 2r_{EF}r_{FH}\cos(\theta_{EFH})} \tag{46}$$

3.2.5 Workspace Transformations

During bilateral operation in position control mode, the position of the backhoe must be mapped into the workspace of the PHANToM and vice versa. The error between the PHANToM's position and the backhoe's position in the PHANToM's workspace is used to compute haptic forces—i.e. the virtual spring connection—where the haptic force $\mathbf{f_h}$ is parallel in direction and proportional in magnitude to the position error vector $\mathbf{p_b^p} - \mathbf{p_p^p}$ as illustrated in figure 15(a). Simultaneously in the backhoe's workspace, the PHANToM's position is used as a Cartesian reference for the backhoe's controller to follow, as illustrated in figure 15(b). Note that on the backhoe side, both Cartesian references coming from the PHANToM and measured positions coming from the position sensors must be transformed back and forth between Cartesian space and cylinder space as described in sections 3.2.1-3.2.4.

3.2.5.1 PHANToM Workspace

In the Ghost software, the PHANToM's local origin O_p^p is located at the wrist position when all the links are at 90 degrees. From the operator's perspective, the coordinate axes are $\mathbf{x_p^p}$ to the right, $\mathbf{y_p^p}$ is straight up, and $\mathbf{z_p^p}$ points toward the operator. See figure 15(a). By fixing a point in the workspace of the PHANToM that represents the origin of the backhoe O_0^p, the instantaneous position of the backhoe can then be mapped into the PHANToM's

workspace relative to O_0^p. This mapping requires a rotation $\mathbf{R_0^p}$ from the orientation of the backhoe's base frame O_0^0 to the PHANToM's frame O_p^p, a scaling by the factor K_s to account for the relative size of the two workspaces and for unit conversions, and finally a translation from O_p^p to O_0^p, where

$$\mathbf{p}_b^p = \mathbf{p}_{O_pO_0}^p + K_s \mathbf{R}_0^p \mathbf{p}_b^0 = \mathbf{T}_0^p \mathbf{p}_b^0, \tag{47}$$

$$\mathbf{T}_0^p = \begin{bmatrix} 0 & -K_s & 0 & \mathbf{p}_{O_pO_0}^p(x) \\ 0 & 0 & K_s & \mathbf{p}_{O_pO_0}^p(y) \\ -K_s & 0 & 0 & \mathbf{p}_{O_pO_0}^p(z) \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{48}$$

and $\mathbf{p}_b^0 = \mathbf{p_{0,4}^0} = \begin{bmatrix} \mathbf{p}_b^0(x) & \mathbf{p}_b^0(y) & \mathbf{p}_b^0(z) & 1 \end{bmatrix}^{\mathbf{T}}$. See [35] for more details on this transformation. Also, the value of the workspace scaling factor is $K_s = 2.1167 \left[\frac{mm}{in}\right]$, where mm are measured in the PHANToM workspace and in are measured in the backhoe workspace.

With the position of the backhoe known from equation (47), the haptic force $\mathbf{f_h}$ is computed from

$$\mathbf{f}_h = \mathbf{k}_p \left(\mathbf{p}_b^p - \mathbf{p}_p^p \right), \tag{49}$$

where k_h is the virtual spring rate. Equation (49) can be directly entered into the Ghost software in xyz coordinates. Although it is not necessary to define the scaling factor and spring rate as constant in all three xyz directions, the initial software has been set up this way for simplicity.

3.2.5.2 Backhoe Workspace

With the transformation $\mathbf{T_0^p}$ defined in equation (48) above, the inverse transformation can conveniently be used to map the position of the PHANToM into the backhoe's workspace:

$$\mathbf{p}_p^0 = \left(\mathbf{T}_0^p\right)^{-1} \mathbf{p}_p^p \tag{50}$$

which is illustrated in figure 15(b). This position becomes the tracking reference for the backhoe's controller to follow, after being transformed into cylinder lengths.

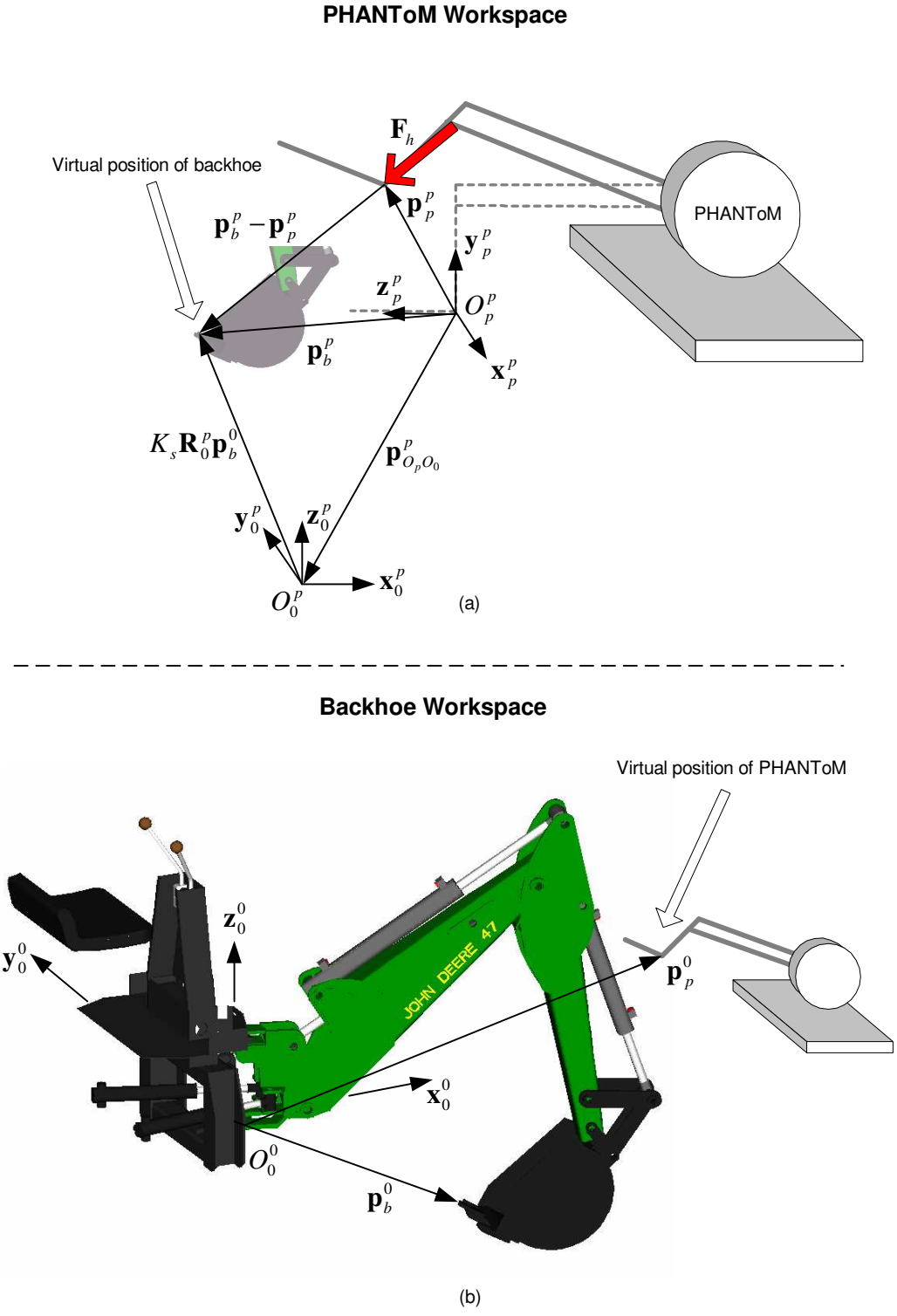


Figure 15: Workspace Transformations

3.2.5.3 Bucket Angle

The PHANToM has six encoders that can produce signals representing both the xyz position of the wrist and the orientation of an attached stylus. However, the actuators can only produce haptic forces in the three xyz directions. The actuators cannot produce torques about the wrist. Therefore, although the angle of the stylus can be used as a reference for the bucket angle ϕ, no torque feedback is available for this degree of freedom of the backhoe, and the bucket angle reference coming from the PHANToM is a unilateral command.

The gimbal angles of the PHANToM stylus are available in the Ghost software from the getGimbalAngles() function. The vector returned contains $\begin{bmatrix} \theta & \phi & \psi \end{bmatrix}_p$, where θ_p is the angle between the projection of the stylus onto the $\mathbf{x_p} - \mathbf{z_p}$ plane and the $\mathbf{z_p}$ axis measured about the $\mathbf{y_p}$ axis, ϕ_p is the angle between the logitudinal axis of the stylus and the $\mathbf{x_p} - \mathbf{z_p}$ plane measured positive in the $+\mathbf{y_p}$ direction, and ψ_p is the rotational angle of the stylus about its longitudinal axis, all measured in radians. Therefore, the instantaneous bucket angle reference is, by coincidence, simply the negative of the ϕ angle from the PHANToM:

$$\phi^b_{b,ref} = -\phi^p_p \tag{51}$$

The C++ code used to retrieve the bucket angle reference from the PHANToM is given in appendix C.2

3.3 Dynamics

3.3.1 Actuator Force to Joint Torque Transformations

When simulating the motion of the backhoe's links in response to forces applied to them by the hydraulic cylinders, a transformation is necessary to convert from cylinder rod forces to joint torques. Formally, this step would involve replacing the three-dimensional rod forces $\mathbf{f}_r$ with a combination of both equivalent couple moments and forces translated to the joint axes. However, since we are only interested in the torque applied to the joints, the internal forces that intersect the joint axes will be neglected. Note that the magnitude of the applied joint torques τ_r depends on both the rod forces $\mathbf{f}_r$ and the cylinder lengths $\mathbf{y}_c$.

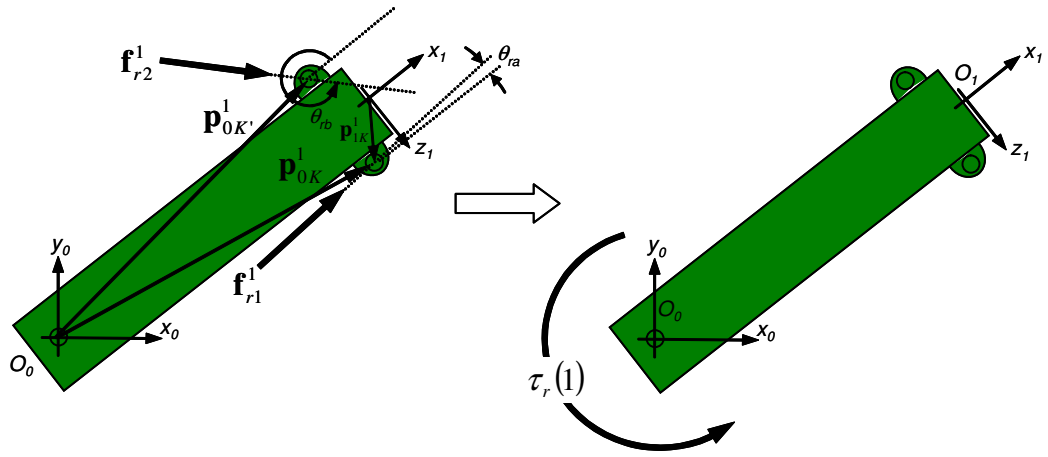


Figure 16: Swing cylinder force to joint 1 torque transformation

3.3.1.1 Torque applied to swing link

The first joint torque τ_{r1} at O_0 about the $\mathbf{z}_0$ axis is computed from the contribution of the two swing cylinder forces $\mathbf{f}_{r1}$ and $\mathbf{f}_{r2}$. Figures 8 and 16 illustrate the transformation.

$$r_{JQ} = \sqrt{(r_{0J})^2 + \left(p^1_{1K}(3)\right)^2 - 2r_{0J}p^1_{1K}(3)\cos(\theta_{Q0J})} \tag{52}$$

$$\theta_{r1} = \cos^{-1}\left(\frac{r^2_{QK} + y^2_{c1} - r^2_{JQ}}{2r_{QK}y_{c1}}\right) \tag{53}$$

$$r_{J'Q'} = \sqrt{(r_{0J'})^2 + \left(p^1_{1K}(3)\right)^2 - 2r_{0J}p^1_{1K}(3)\cos(\theta_{J'0Q'})} \tag{54}$$

$$\theta_{r2} = 2\pi - \cos^{-1}\left(\frac{r^2_{QK} + y^2_{c2} - r^2_{J'Q'}}{2r_{QK}y_{c2}}\right) \tag{55}$$

$$\mathbf{p}^1_{0K} = \begin{bmatrix} r_{QK} & 0 & p^1_{1K}(3) \end{bmatrix}^T \tag{56}$$

$$\mathbf{p}^1_{0K'} = \begin{bmatrix} r_{QK} & 0 & -p^1_{1K}(3) \end{bmatrix}^T \tag{57}$$

$$\mathbf{f}^1_{r1} = |f_r(1)| \begin{bmatrix} \cos(\theta_{r1}) & 0 & \sin(\theta_{r1}) \end{bmatrix}^T \tag{58}$$

$$\mathbf{f}^1_{r2} = |f_r(2)| \begin{bmatrix} \cos(\theta_{r2}) & 0 & \sin(\theta_{r2}) \end{bmatrix}^T \tag{59}$$

$$\tau_{r1} = \left(\mathbf{p}^1_{0K} \times \mathbf{f}^1_{r1} + \mathbf{p}^1_{0K'} \times \mathbf{f}^1_{r2}\right) \cdot \mathbf{z}_0 \tag{60}$$

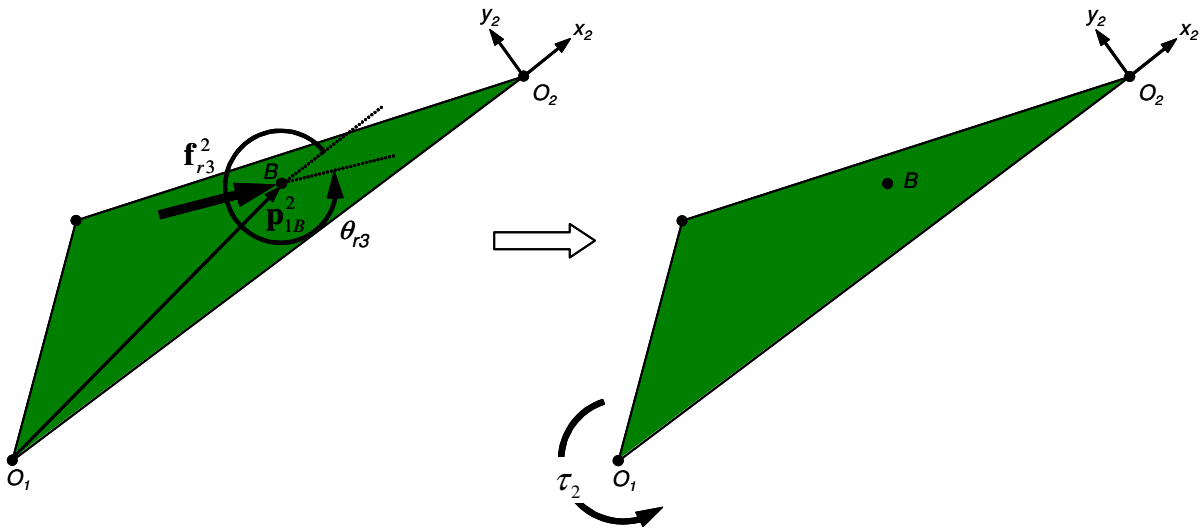


Figure 17: Boom cylinder force to joint 2 torque transformation

Note that the five-element cylinder force vector $\mathbf{f}_r$ does not contain the direction information of the swing cylinder force vectors $\mathbf{f}_{r1}^1$ and $\mathbf{f}_{r2}^1$ used in equations (58) & (59); i.e.

$$\mathbf{f_r} = \begin{bmatrix} f_r(1) & f_r(2) & f_r(3) & f_r(4) & f_r(5) \end{bmatrix}^T \tag{61}$$

which is related to equations (58) & (59) only by the magnitude of the first two elements.

3.3.1.2 Torque applied to boom

Figures 9 and 17 illustrate the transformation of the force applied by the boom cylinder $\mathbf{f}_{r3}$ into the applied torque τ_{r2} at O_1 about $\mathbf{z_1}$.

$$\theta_{1BA} = \cos^{-1}\left(\frac{y_{c3}^2 + r_{1B}^2 - r_{1A}^2}{2y_{c3}r_{1B}}\right) \tag{62}$$

$$\theta_{r3} = 2\pi + (\theta_{B12} - \theta_{1BA}) \tag{63}$$

$$\mathbf{p}_{1B}^2 = r_{1B}\begin{bmatrix} \cos(\theta_{B12}) & \sin(\theta_{B12}) & 0 \end{bmatrix}^T \tag{64}$$

$$\mathbf{f}_{r3}^2 = |f_r(3)|\begin{bmatrix} \cos(\theta_{r3}) & \sin(\theta_{r3}) & 0 \end{bmatrix}^T \tag{65}$$

$$\tau_{r2} = \left(\mathbf{p}_{1B}^2 \times \mathbf{f}_{r3}^2\right) \cdot \mathbf{z}_1 \tag{66}$$

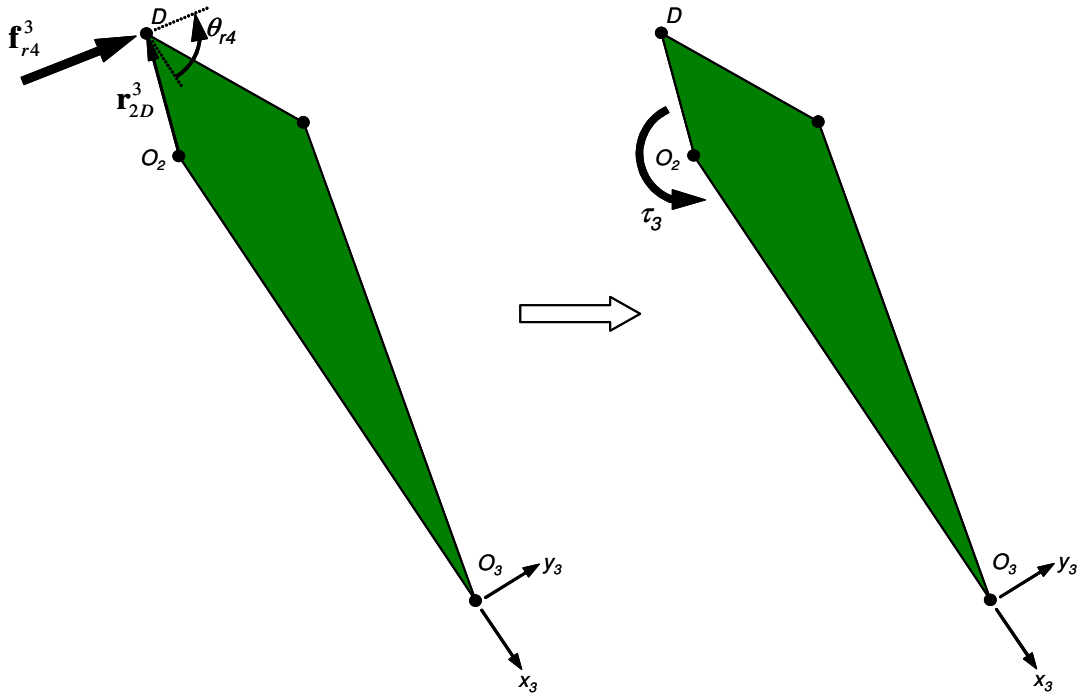


Figure 18: Stick cylinder force to joint 3 torque transformation

3.3.1.3 Torque applied to stick

Figures 10 and 18 illustrate the transformation of the force applied by the stick cylinder $\mathbf{f}_{r4}$ into the applied torque τ_{r3} at O_2 about $\mathbf{z}_2$.

$$\theta_{2DC} = \cos^{-1}\left(\frac{r_{2D}^2 + y_c^2\left(4\right) - r_{2C}^2}{2r_{2D}y_c\left(4\right)}\right) \tag{67}$$

$$\theta_{x_3'2D} = \pi - \theta_{D23} \tag{68}$$

$$\theta_{r4} = \pi - \theta_{2DC} - \theta_{x_3'2D} \tag{69}$$

$$\mathbf{p}^3_{2D} = r_{2D}\begin{bmatrix} \cos\left(\theta_{D23}\right) & \sin\left(\theta_{D23}\right) & 0 \end{bmatrix}^T \tag{70}$$

$$\mathbf{f}^3_{r4} = \left|f_r\left(4\right)\right|\begin{bmatrix} \cos\left(\theta_{r4}\right) & \sin\left(\theta_{r4}\right) & 0 \end{bmatrix}^T \tag{71}$$

$$\tau_r\left(3\right) = \left(\mathbf{p}^3_{2D} \times \mathbf{f}^3_{r4}\right) \cdot z_2 \tag{72}$$

3.3.1.4 Torque applied to bucket

Figure 19 illustrates the points of interest in computing the torque applied to the bucket. Knowledge of the force exerted by the bucket cylinder $\mathbf{f}_{r5}$ and the length of the bucket

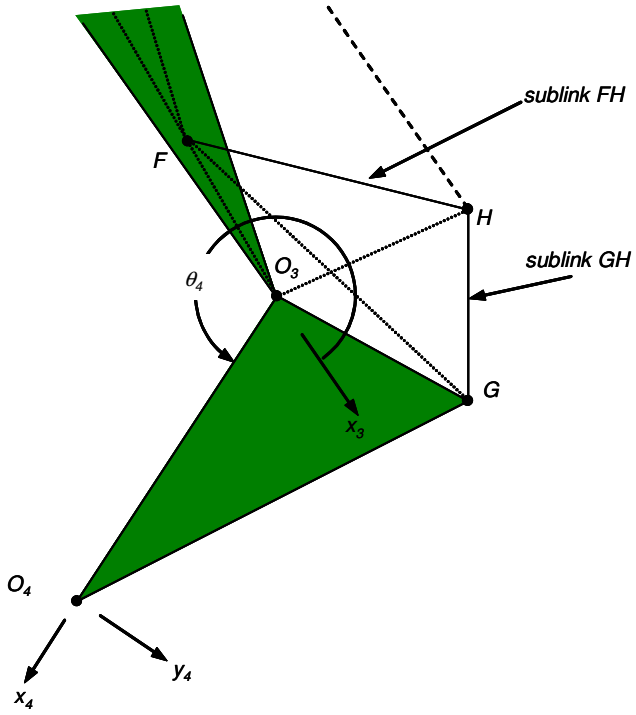


Figure 19: Bucket links

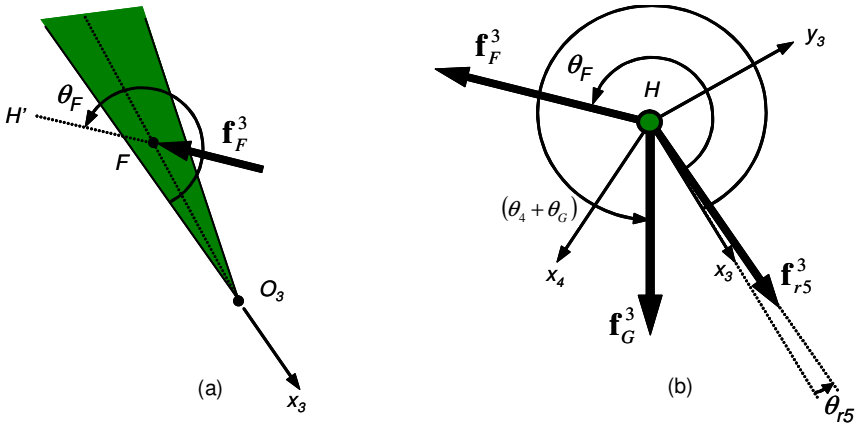


Figure 20: Bucket cylinder joint angle analysis

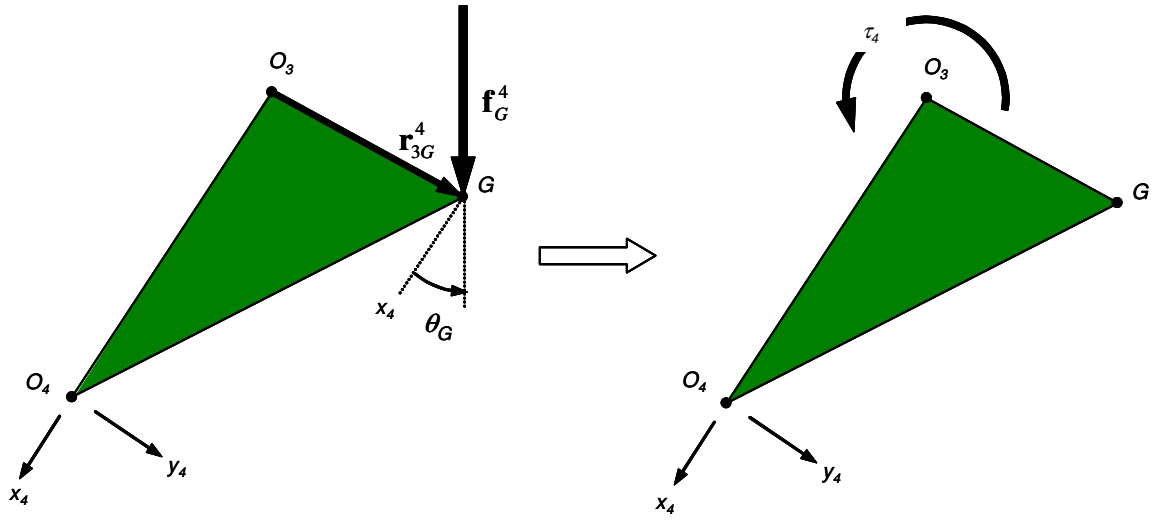


Figure 21: Bucket cylinder force to joint 4 torque transformation

cylinder y_{c5} is required for the transformation.

This transformation is solved by first analyzing the angles of the bucket sublinks FH and GH relative to the stick and bucket, summing forces at the pin connection at H, and finally solving simultaneously for the magnitude of the orthogonal components of the force $\mathbf{f}_G^4$ that link GH exerts on the bucket at point G.

Figure 21 illustrates the transformation of the force $\mathbf{f}_G^4$ exerted by the sublink GH onto the bucket, and the corresponding applied torque τ_{r4} at O_3 about $\mathbf{z}_3$.

Figure 20(a) illustrates the geometry of the sublink FH relative to the stick. Figure 20(b) illustrates the forces applied to the pin at point H.

The first step is to find the angles θ_G, θ_{r5}, and θ_F:

$$\theta_{EFH} = \cos^{-1}\left(\frac{r_{EF}^2 + r_{FH}^2 - y_c^2\,(5)}{2r_{EF}r_{FH}}\right) \tag{73}$$

$$\theta_{HF3} = \pi - \theta_{DFE} - \theta_{EFH} \tag{74}$$

$$r_{3H} = \sqrt{r_{3F}^2 + r_{FH}^2 - 2r_{3F}r_{FH}\cos\left(\theta_{HF3}\right)} \tag{75}$$

$$\theta_{F3H} = \cos^{-1}\left(\frac{r_{3F}^2 + r_{3H}^2 - r_{FH}^2}{2r_{3F}r_{3H}}\right) \tag{76}$$

$$\theta_{H3G} = \cos^{-1}\left(\frac{r_{3H}^2 + r_{3G}^2 - r_{GH}^2}{2r_{3H}r_{3G}}\right) \tag{77}$$

$$\theta_4 = 3\pi - \theta_{F3H} - \theta_{H3G} - \theta_{G34} - \theta_{23D} \tag{78}$$

$$\theta_{x_3 3G} = 2\pi - \theta_4 - \theta_{G34} \tag{79}$$

The angle θ_{G3F} depends on whether $\theta_4 + \theta_{G34} > 2\pi$:

if

$$\theta_4 + \theta_{G34} > 2\pi$$

then

$$\theta_{G3F} = \pi - \theta_{x_3 3G} - \theta_{23D}$$

otherwise

$$\theta_{G3F} = \pi - \theta_{x_3 3G} - \theta_{23D} \tag{80}$$

$$r_{FG} = \sqrt{(r_{3F})^2 + (r_{3G})^2 - 2r_{3F}r_{3G}\cos(\theta_{G3F})} \tag{81}$$

$$\theta_{FG3} = \cos^{-1}\left(\frac{r_{FG}^2 + r_{3G}^2 - r_{3F}^2}{2r_{FG}r_{3G}}\right) \tag{82}$$

$$\theta_{FGH} = \cos^{-1}\left(\frac{r_{FG}^2 + r_{GH}^2 - r_{FH}^2}{2r_{FG}r_{GH}}\right) \tag{83}$$

The angle θ_{3GH} depends on whether $\theta_4 + \theta_{G34} > 2\pi$:

if

$$\theta_4 + \theta_{G34} > 2\pi$$

then

$$\theta_{3GH} = \theta_{FGH} - \theta_{FG3}$$

otherwise

$$\theta_{3GH} = \theta_{FGH} + \theta_{FG3} \tag{84}$$

The angle θ_G is defined as the angle that the two-force link GH makes with $\mathbf{x}_4$:

$$\theta_G = \theta_{G34} - \theta_{3GH} \tag{85}$$

$$\theta_{3FG} = \cos^{-1}\left(\frac{r_{3F}^2 + r_{FG}^2 - r_{3G}^2}{2r_{3F}r_{FG}}\right) \tag{86}$$

$$\theta_{HFG} = \cos^{-1}\left(\frac{r_{FG}^2 + r_{FH}^2 - r_{GH}^2}{2r_{FG}r_{FH}}\right) \tag{87}$$

The angle θ_{HF3} depends on whether $\theta_4 + \theta_{G34} > 2\pi$:

if

$$\theta_4 + \theta_{G34} > 2\pi$$

then

$$\theta_{HF3} = \theta_{HFG} - \theta_{3FG}$$

otherwise

$$\theta_{HF3} = \theta_{HFG} + \theta_{3FG} \tag{88}$$

$$\theta_{3FH'} = \pi - \theta_{HF3} \tag{89}$$

$$\theta_F = (\pi - \theta_{3FH'} - \theta_{23D}) + \pi \tag{90}$$

$$\theta_{FHE} = \cos^{-1}\left(\frac{r_{FH}^2 + y_c^2(5) - r_{EF}^2}{2 r_{FH} y_c(5)}\right) \tag{91}$$

$$\theta_{r5} = -\theta_{23D} + \theta_{HF3} - \theta_{FHE} \tag{92}$$

Now that the angles θ_G, θ_F, and θ_{r5} are known from equations (85), (90), and (92), a force balance on pin H in the $\mathbf{x}_3$ and $\mathbf{y}_3$ directions yields

$$f_{r5}\cos(\theta_{r5}) + f_F\cos(\theta_F) + f_G\cos(\theta_4 + \theta_G) = 0 \tag{93}$$

$$f_{r5}\sin(\theta_{r5}) + f_F\sin(\theta_F) + f_G\sin(\theta_4 + \theta_G) = 0 \tag{94}$$

where the mass of pin H has been neglected. Rearranging equations (93) and (94) solving simultaneously for the magnitudes of the unknown forces $\mathbf{f}_F$ and $\mathbf{f}_G$,

$$\begin{bmatrix} f_F \\ f_G \end{bmatrix} = \begin{bmatrix} \cos(\theta_F) & \cos(\theta_4 + \theta_G) \\ \sin(\theta_F) & \sin(\theta_4 + \theta_G) \end{bmatrix}^{-1} \begin{bmatrix} -f_{r5}\cos(\theta_{r5}) \\ -f_{r5}\sin(\theta_{r5}) \end{bmatrix} \tag{95}$$

See Figure 20(b).

The torque θ_{r4} applied to the bucket at O_3 about $\mathbf{z}_3$ by the force $\mathbf{f}_G$ is then found from

$$\mathbf{p}_{3G}^4 = r_{3G}\begin{bmatrix} \cos(\theta_{G34}) & \sin(\theta_{G34}) & 0 \end{bmatrix}^T \tag{96}$$

$$\mathbf{f}_G^4 = f_G\begin{bmatrix} \cos(\theta_G) & \sin(\theta_G) & 0 \end{bmatrix}^T \tag{97}$$

$$\tau_{r4} = \left(\mathbf{p}_{3G}^4 \times \mathbf{f}_G^4\right) \cdot \mathbf{z}_3 \tag{98}$$

3.3.2 LaGrangian Dynamic Model

In the field of robotics, a common approach to the dynamic modeling of serial manipulators is to use the principles laid out by Joseph-Louis LaGrange in 1788 in his work entitled the Méchanique Analytique [5], [35], [39]. Unlike the Newton-Euler approach, the LaGrangian approach bases its equations of motion on the kinetic and potential energy of the system in terms of a set of generalized coordinates, which are the rotational and translational positions and velocities of the robot's joints. The advantage to the LaGrangian approach is that there is no need to compute reaction forces at the connections between links, which are typically of no interest to the analysis. The LaGrangian function is defined as the difference between the kinetic and potential energy of the mechanical system:

$$L = K - U \tag{99}$$

where K is the scalar sum of the kinetic energy of all the links

$$K = \frac{1}{2}\sum_{i=1}^{n} \mathbf{v}_{ci}^T \mathbf{m}_i \mathbf{v}_{ci} + \omega_i^T \mathbf{I}_i \omega_i \tag{100}$$

and U is the scalar sum of the potential energy of all the links

$$U = -\sum_{i=1}^{n} m_i \mathbf{g}^T \mathbf{p}_{0,c_i} \tag{101}$$

In equation (100), $\mathbf{v}_{ci}$ is the 3x1 translational velocity vector of link i written at the center of mass, and ω_i is the 3x1 rotational velocity vector of link i, both relative to the base frame. The mass matrix $\mathbf{m}_i$ is the diagonal 3x3 matrix where $\mathbf{m}_i = \text{diag}\left(\begin{bmatrix} m_i & m_i & m_i \end{bmatrix}\right)$, and the lower case has been used to reserve the upper case $\mathbf{M}$ for the results of the present formulation. The inertia matrix $\mathbf{I}_i$ is the diagonal 3x3 inertia tensor containing the three principle inertias.

In equation (101), the negative sign accounts for the fact that the gravitational vector $\mathbf{g} = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T$ points in the negative $\mathbf{z}_0$ direction. Also, both the gravitational vector $\mathbf{g}$ and the position vectors of the center of masses relative to the base frame $\mathbf{p}_{0,c_i}$ are expressed in base frame coordinates and the superscripts have been omitted. The index n is the number of degrees of freedom of the system.

Equation (100) can be rewritten by using the theory of instantaneous screw motion [21] by defining the relationship between the velocities of each link $\mathbf{v}_{c_i}$ and ω_i and the joint rates $\dot{\theta}$ using the geometric *Jacobian* matrix:

$$\mathbf{J}_{vi} = \begin{bmatrix} \mathbf{J}_{vi}^1 & \mathbf{J}_{vi}^2 \ldots & \mathbf{J}_{vi}^i & \mathbf{0} & \mathbf{0} \ldots & \mathbf{0} \end{bmatrix} \tag{102}$$

$$\mathbf{J}_{\omega i} = \begin{bmatrix} \mathbf{J}_{\omega i}^1 & \mathbf{J}_{\omega i}^2 \ldots & \mathbf{J}_{\omega i}^i & \mathbf{0} & \mathbf{0} \ldots & \mathbf{0} \end{bmatrix} \tag{103}$$

where the elements of the *3 × n link Jacobian submatrices* $\mathbf{J}_{v_i}$ and $\mathbf{J}_{w_i}$ are defined by

$$\mathbf{J}_{vi}^j = \begin{cases} \mathbf{z}_{j-1} \times \mathbf{p}_{j-1,ci}^{j-1} & \text{for a revolute joint} \\ \mathbf{z}_{j-1} & \text{for a prismatic joint} \end{cases} \tag{104}$$

$$\mathbf{J}_{\omega i}^j = \begin{cases} \mathbf{z}_{j-1} & \text{for a revolute joint} \\ 0^{(3\times 1)} & \text{for a prismatic joint} \end{cases} \tag{105}$$

and $j = 1 : i$. The vector $\mathbf{p}_{j-1,c_i}^{j-1}$ is the position of the current link's center of mass relative to the previous link's origin expressed in the previous link's coordinates. Note that the complete Jacobian matrix

$$\mathbf{J}_i = \begin{bmatrix} \mathbf{J}_{vi} \\ \mathbf{J}_{\omega i} \end{bmatrix} \tag{106}$$

is written separately for each link.

With the Jacobian matrices defined for each link, the linear and angular velocities of each link can be written in the generalized coordinates as

$$\begin{bmatrix} \mathbf{v}_{ci} \\ \omega_i \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{vi} \\ \mathbf{J}_{\omega i} \end{bmatrix} \dot{\theta} \tag{107}$$

Substituting the components of equation 107 into 100 yields

$$K = \frac{1}{2} \dot{\theta}^T \left(\sum_{i=1}^{n} \mathbf{J}_{vi}^T \mathbf{m}_i \mathbf{J}_{vi} + \mathbf{J}_{\omega i}^T \mathbf{I}_i \mathbf{J}_{\omega i} \right) \dot{\theta} \tag{108}$$

The quantity in parenthesis in equation (108) is termed the $n \times n$ *manipulator inertia matrix* $\mathbf{M}$, where

$$\mathbf{M} = \sum_{i=1}^{n} \left(\mathbf{J}_{vi}^T \mathbf{m}_i \mathbf{J}_{vi} + \mathbf{J}_{\omega i}^T \mathbf{I}_i \mathbf{J}_{\omega i} \right) \tag{109}$$

so that

$$K = \frac{1}{2}\dot{\theta}^T \mathbf{M}\dot{\theta} \tag{110}$$

Now we are ready to form the equations of motions in terms of the generalized coordinates. Substituting (101) and (110) into (99) yields

$$L = \frac{1}{2}\dot{\theta}^T \mathbf{M}\dot{\theta} + \sum_{i=1}^{n} m_i \mathbf{g}^T \mathbf{p}_{0,c_i} \tag{111}$$

The equations of motion can now be described by taking the derivative of equation (111) with respect to θ, $\dot{\theta}$, and time t:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i} = \tau_i \tag{112}$$

Substituting equation (99) into (112), this becomes

$$\frac{d}{dt}\left(\frac{\partial K}{\partial \dot{\theta}_i}\right) - \frac{\partial K}{\partial \theta_i} - \frac{\partial U}{\partial \theta_i} = \tau_i \tag{113}$$

where we have used the fact that K is a function of both θ and $\dot{\theta}$, but U is only a function of θ. The joint torque τ_i is the torque of link *i-1* acting on link i along the $\mathbf{z}_{i-1}$ axis. Note that τ_i may be a sum of torques from more than one source, such as from both actuator forces and forces applied to the end effector.

Evaluating the first term in equation 113 yields

$$\frac{d}{dt}\left(\frac{\partial K}{\partial \dot{\theta}_i}\right) = \sum_{j=1}^{n} M_{ij}\ddot{\theta}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} \frac{\partial M_{ij}}{\partial \theta_k}\dot{\theta}_j\dot{\theta}_k \tag{114}$$

where the manipulator inertia matrix has been expanded into a sum of scalar elements before the product rule has been applied.

Evaluating the second term of equation (113) yields

$$\frac{\partial K}{\partial \theta_i} = \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} \frac{\partial M_{jk}}{\partial \theta_i}\dot{\theta}_j\dot{\theta}_k \tag{115}$$

and evaluating the third term of equation (113) yields

$$\frac{\partial U}{\partial \theta_i} = -\sum_{j=1}^{n} m_j \mathbf{g}^T \mathbf{J}_{vj}^i \tag{116}$$

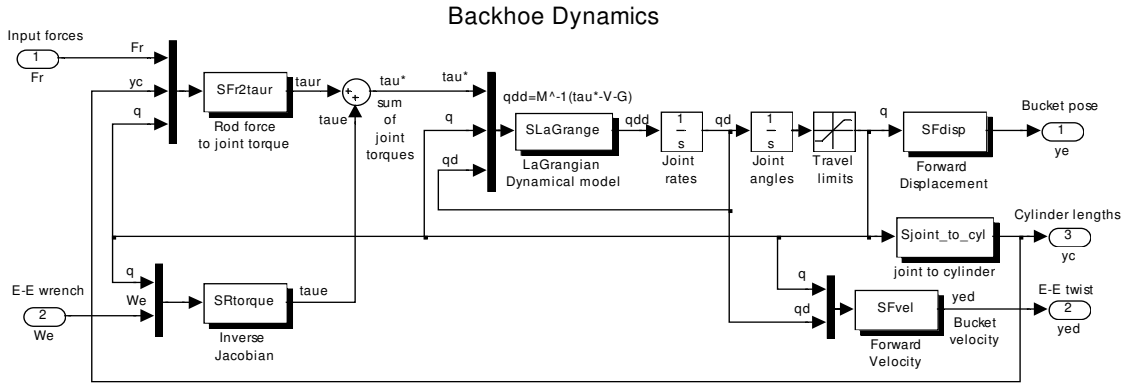


Figure 22: LaGrangian dynamic model block diagram

where the fact that $\frac{\partial \mathbf{p}_{0,cj}}{\partial \theta_i} = \frac{\partial \mathbf{p}_{j-1,cj}}{\partial \theta_i} = \mathbf{J}^i_{vj}$ has been used from equation (104). Substituting equations (114)-(116) into (113) and combining terms yields the final form of the LaGrangian dynamic equation:

$$\sum_{j=1}^{n} M_{ij}\ddot{\theta}_j + \sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial M_{ij}}{\partial \theta_k} - \frac{1}{2}\frac{\partial M_{jk}}{\partial \theta_i}\right)\dot{\theta}_j\dot{\theta}_k - \sum_{j=1}^{n} m_j \mathbf{g}^T \mathbf{J}^i_{vj} = \tau_i \tag{117}$$

which is valid for each degree of freedom, *i=1:n*. Each of the n equations of the form given by (117) can be written more compactly in matrix form:

$$\mathbf{M}(\theta)\,\ddot{\theta} + \mathbf{V}\left(\theta, \dot{\theta}\right) + \mathbf{G}(\theta) = \tau^* \tag{118}$$

where

$$\mathbf{V} = \sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial M_{ij}}{\partial \theta_k} - \frac{1}{2}\frac{\partial M_{jk}}{\partial \theta_i}\right)\dot{\theta}_j\dot{\theta}_k \tag{119}$$

$$\mathbf{G} = -\sum_{j=1}^{n} m_j \mathbf{g}^T \mathbf{J}^i_{vj} \tag{120}$$

and $\mathbf{M}$ is given in equation (109). The vector $\mathbf{V}$ is the *velocity coupling vector*, which contains the velocity-dependent centripetal and coriolis terms, and the vector $\mathbf{G}$ is the *vector of gravitational torques*. The variables $\mathbf{V}$ and $\mathbf{G}$ have been left as capitals to be consistent with the literature. Figure 22 is a block diagram of the relevant portion of the backhoe simulation. Thus the bucket position and orientation can be computed from the sum of the applied joint torques τ^*, along with the necessary initial joint positions θ and

velocities $\dot{\theta}$ at $t = 0$. The dynamic model given by equation (118) was constructed using CAD software for the mass and inertia properties, and simulation software to compute the dynamic quantities. The elements of the variables $\mathbf{M}$, $\mathbf{V}$, and $\mathbf{G}$ given by equations (109), (119), and (120) were manipulated with Matlab's Symbolic Toolbox and both the code and results are listed in Appendix F. The mass and inertia properties calculated from the solid model in Pro/ENGINEER are given in Appendix E.

3.4 Valve Modeling and System Identification

The John Deere 4410 tractor supplies constant hydraulic oil flow (at constant engine speed) to its attachments via a gear pump connected to the output shaft of the diesel engine, where the flow rate is proportional to engine speed and pressure floats to load. This is typically the case for mobile hydraulic equipment. Because the supply flow rate is constant provided the engine speed remains constant, open-center valves are much more appropriate than closed-center valves for this project. A constant flow source connected to a closed-center valve requires the extra complexity of a flow divider and parallel circuit to handle all off-peak flow, whereas an open-center valve does not. However, this greatly reduces the design space for the haptic backhoe because of the limited selection of electrohydraulic open-center valves on the market.

3.4.1 The PVG32 Electrohydraulic Valve

The electrohydraulic valve chosen for the haptic backhoe is the model PVG32 from Sauer-Danfoss. After some initial investigation, this valve was selected by engineers at John Deere partially for its suitability and relatively low cost, but mostly because of the long term business relationship between John Deere and Sauer-Danfoss.

The PVG32 is an open-centered flow control valve suitable for mobile hydraulic applications involving constant supply flow from a gear pump. The modular design allows for stacking up to ten valves into one compact block, and a wide variety of spool sizes and actuation techniques are available from the manufacturer. For this project, a 10 gpm spool, high-performance electrical actuation, and pressure compensation options were selected to give the desired performance. Figure 23 is a photo of the PVG32 with three valve sections.

The PVG32 valve block consists of a pump module on one end, an end cover on the other end, and a number of valve modules stacked side-by-side in between. The following passage from the PVG32 Technical Manual [34] describes the sequence of events that occur when valve diverts flow to a load. Refer to Figure 24 for section views of the individual valve modules.

"When the pump is started and the main spools in the individual basic modules

Figure 23: The PVG32 Electrohydraulic Control Valve

(11) are in the neutral position, oil flows from the pump, through connection P, across the pressure adjustment spool (6) to tank. The oil flow led across the pressure adjustment spool determines the pump pressure (stand-by pressure).

When one or more of the main spools are actuated, the highest load pressure is fed through the shuttle valve circuit (10) to the spring chamber behind the pressure adjustment spool (6), and completely or partially closes the connection to tank.

Pump pressure is applied to the right-hand side of the pressure adjustment spool (6). The pressure relief valve (1) will open should the load pressure exceed the set value, diverting pump flow back to tank.

In a pressure-compensated basic module the compensator (14) maintains a constant pressure drop across the main spool both when the load changes and when a module with a higher load pressure is actuated." (p. 5)

The main spool, labeled 11 in figure 24, receives the input force from either the electronics module (not shown) or the manual lever, which drives the response of the rest of the valve. Once the main spool is displaced, both the pressure adjustment and compensation spools react to the sensed load automatically, and the PVG32 delivers steady-state flow to load proportional to the input.

The next sections describe the work done testing and modeling the PVG32's performance.

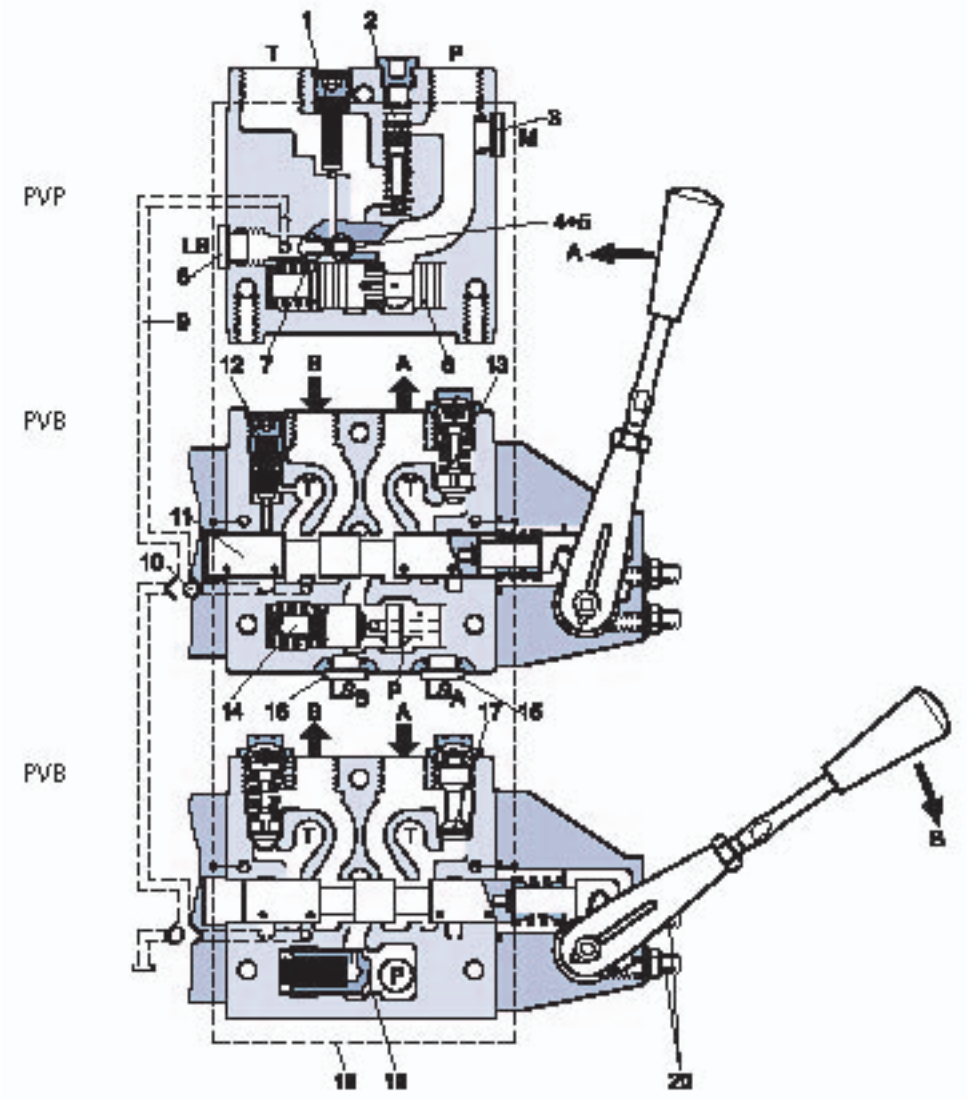


Figure 24: Cross-sections of the PVG32 valve

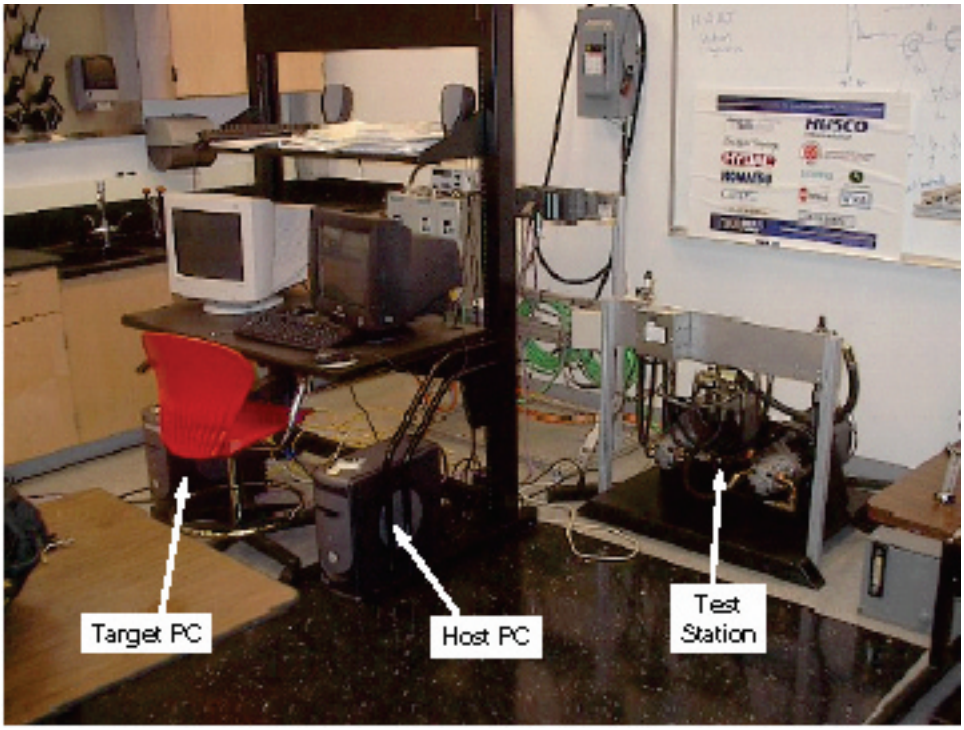


Figure 25: Hardware-in-the-loop (HIL) simulator

3.4.2 Hardware-in-the-Loop Simulator

One of the goals of the haptic backhoe project was to build a mathematical model that could capture the all the relevant dynamics of the system, and ultimately be used to provide insight and understanding, predict performance, and aid as a design tool. The major portion of the work in this area involved modeling the PVG32 valve. To accomplish this task, the Hardware-in-the-Loop (HIL) simulator at Georgia Tech's Fluid Power and Motion Control center (FPMC) was used to collect input/output data. Figure 25 is a photo of the HIL components. See [7] and [8] for more information on this system.

Figure 26 is a photo of the PVG32 mounted on the HIL simulator for testing. A small hydraulic load cylinder was connected to the valve's workports, fitted with a Temposonics magnetostrictive position sensor for position measurement. Valve commands and sensor measurements were generated and measured respectively through a National Instruments 6052-E A/D card. Control software was written in Matlab/Simulink/xPC Target using a two-computer host/target arrangement. The pump speed was set at 1710 rpm, controlled

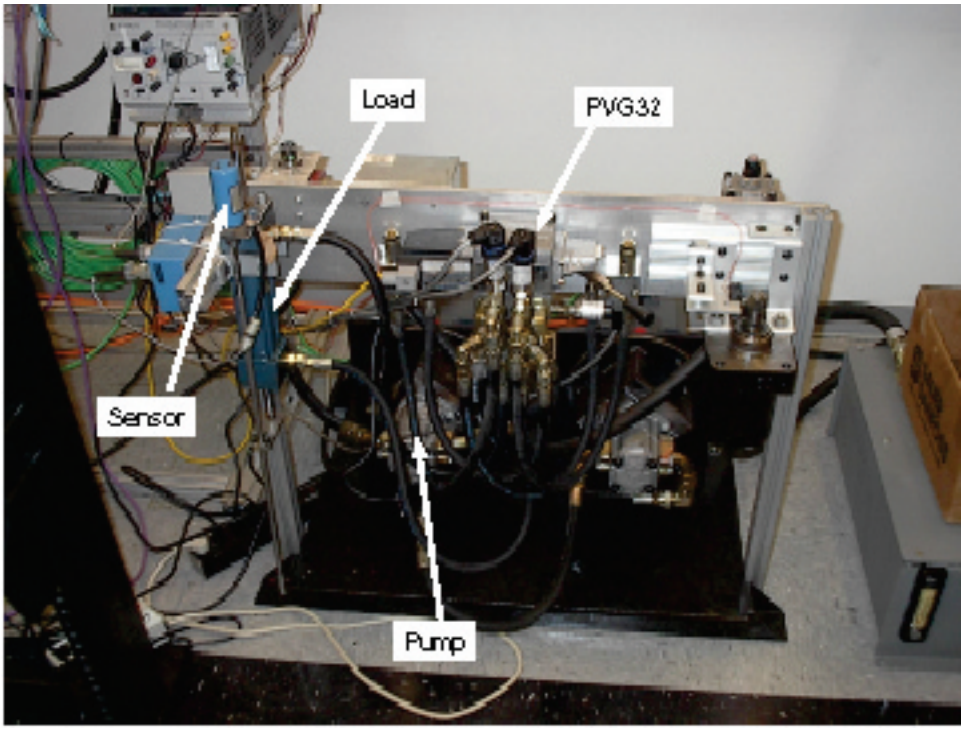


Figure 26: PVG32 mounted on the HIL simulator

separately by a PLC running Simatics software from Siemens. This pump speed corresponded to the nominal flow rating of 8.6 gpm from the tractor's pump, at it's nominal engine speed of 2600 rpm.

3.4.3 HIL Testing

It was desired to generate a transfer function that would relate the input voltage U_s to the output flow rate Q from the valve. Using the HIL simulator equipment, the command voltage and cylinder position signals U_s and y_c were recorded for a series of fixed sine sweeps between 1.0 and 8.8 Hz, in 0.2 Hz increments at an amplitude of 1.5 V. The amplitudes were averaged over a period of ten seconds, and recorded at each frequency.

Since the valve can be considered a velocity source rather than a position source, and the flow rate and cylinder positions are related by

$$Q(t) = A\frac{dy_c(t)}{dt}, \tag{121}$$

the amplitudes of the position measurements were multiplied by the cylinder area and

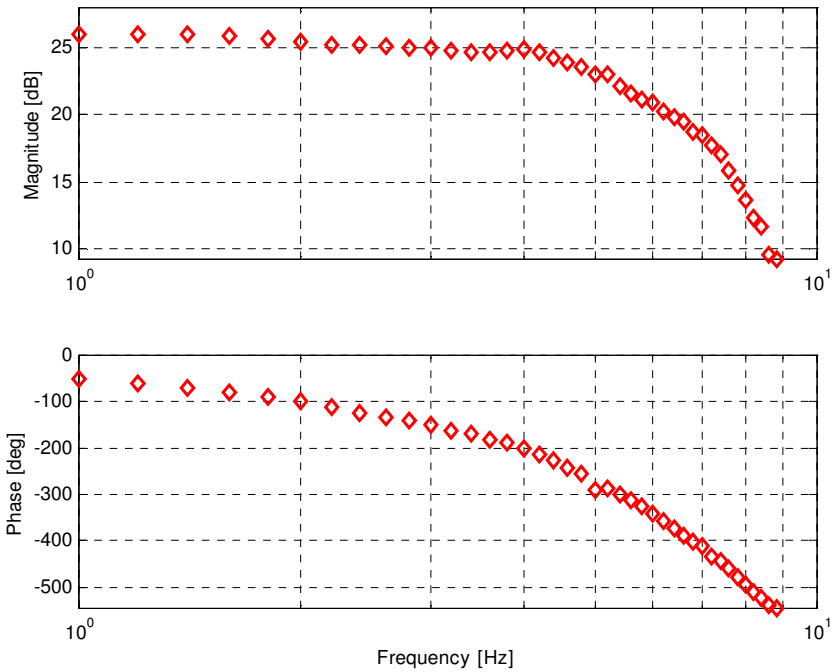


Figure 27: PVG32 Valve & cylinder frequency response

the angular frequency at each frequency, so that

$$\frac{Q(j\omega)}{U_s(j\omega)} = A\left.\frac{sY_c(s)}{U_s(s)}\right|_{s=j\omega} = \left|\frac{\omega Y_c(j\omega)}{U_s(j\omega)}\right| \angle +90^\circ \tag{122}$$

which adds 20dB/decade and 90° phase shift over the full range of frequencies. This method of converting position to velocity is preferable to taking numerical time derivatives, because the sensor noise has been effectively removed beforehand during the averaging step when the wave amplitudes were computed. The reduced sine sweep data, after application of equation (122), is illustrated in figure 27.

Next, a series of step response tests were conducted over a range of input amplitudes between 0.5 V and 1.8 V. In this case, the cylinder position vs. time data was first numerically differentiated using Matlab's forward difference function `diff`, then filtered at at 50 Hz using a second order Butterworth filter block `butter` from Simulink, and finally multiplied by the cylinder area A. Note that all data was sampled at 1 kHz. The reduced step response data is shown in figure 28.

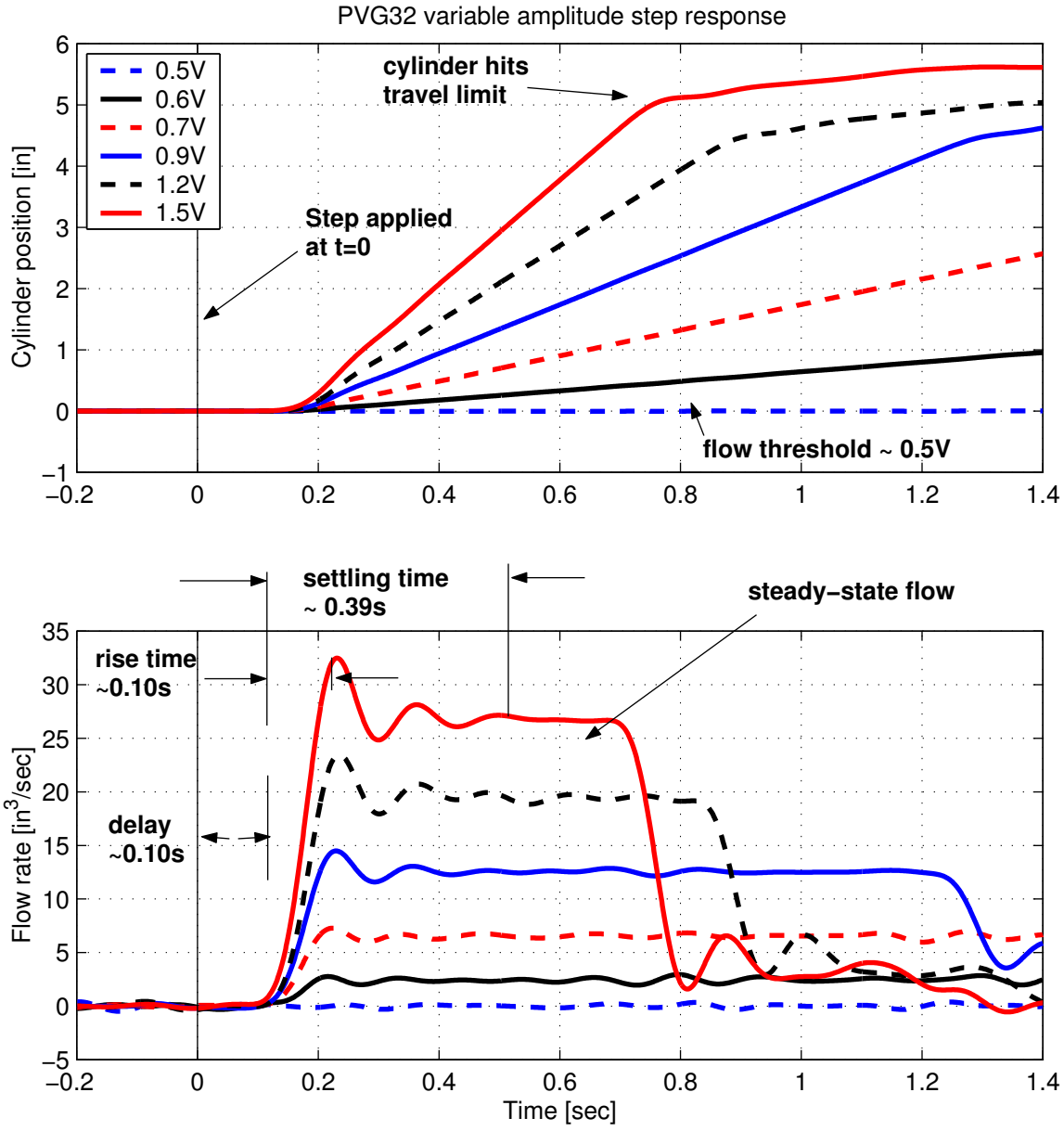


Figure 28: PVG32 Step Response

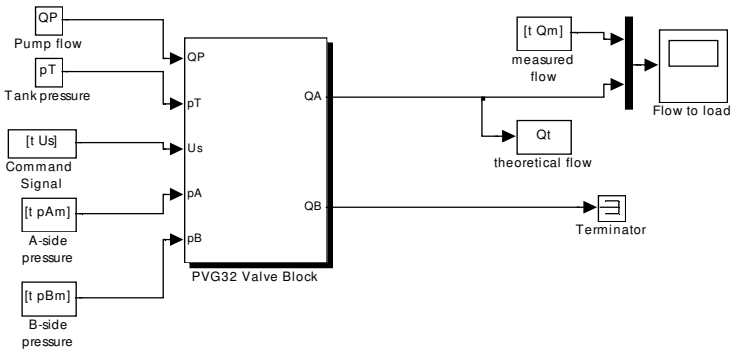


Figure 29: PVG32 Simulink model

It can be seen in figure 28 that the PVG32 exhibits no flow to load for a command voltage U_s less than about 0.5 V, which is due to the deadband in the spool. Also, there is a time delay of about 100 ms between the time when the step command is applied and when the valve responds. This is due to the inherent nature of how the valve operates: once the coil applies a magnetic force to shift the spool, it takes time for the main spool to shift, the load sensing port to open, the shuttle valve to shift, the pressure wave to propagate back to the pressure spool, the pressure spool to shift, the gallery pressure to rise, and then finally for the oil to be driven to load. These characteristics ultimately govern the bandwidth of the valve.

3.4.4 Parameter Optimization

The data shown in figures 27 and 28 were used to generate a mathematical model that would accurately predict the response of the PVG32. Many attempts were made to capture its behavior in simulation. Trial models ranged from simple, purely linear transfer functions to complex nonlinear simulations representing all the masses, spring rates, orifice flows, deadbands, coulomb friction, fluid compliance, etc., attempting to optimize as many as 23 simultaneous unknown parameters. The final form is a slightly modified linear model, based on a third order transfer function augmented with nonlinear blocks to represent spool overlap and the nonlinear steady state gain. The final model was generated in Simulink, and the top-level system is shown in Figure 29

It can be seen from figure 29 that the PVG32 valve model has five inputs and two

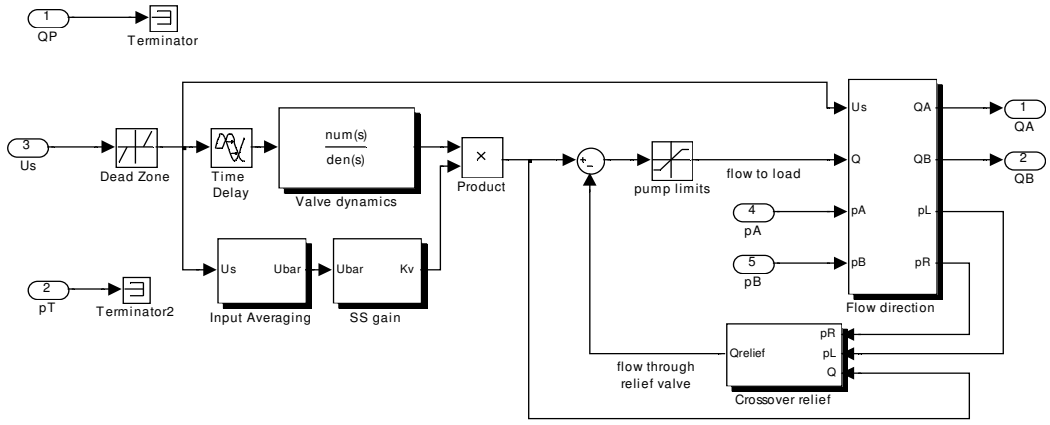


Figure 30: PVG32 Valve Block

outputs. The inputs are the command voltage sent to the coil, flow from the pump, return line pressure, workport A pressure, and workport B pressure. The outputs are the flow rates delivered to port A and port B. Note that this is a two-port admittance model, where the output is the *through* variable, flow rate (i.e. velocity) and the feedback is the *across* variable, pressure (i.e. force).

Figure 30 shows the contents of the subsystem 'PVG32 valve block' from figure 29. This figure illustrates all the important features of the model. The most critical element is the 'Valve dynamics' block, which contains the transfer function of the linear model, upon which the performance of the model is based and all other features are subsidiary.

The shape of the bode plot in figure 27 indicated that the valve may be a 3rd order system. Therefore, a 3rd order model was assumed with unknown pole locations. To find the poles, the Matlab `fmincon` function was used to call the Simulink model and vary the parameters until the error between the data and the model were optimized. After the poles were identified, the transfer function relating output flow rate Q to input voltage U_s, including the observed time delay, is

$$G_v(s) = \frac{Q(s)}{U_s(s)} = \frac{K\omega_n^2 e^{-T_d s}}{(s+p_1)(s^2+2\varsigma\omega_n s+\omega_n^2)} = \frac{31450Ke^{-0.096s}}{s^3+51s^2+1908s+31450} \tag{123}$$

Equation (123) alone fits the frequency response curve shown in figure 27 well for a steady

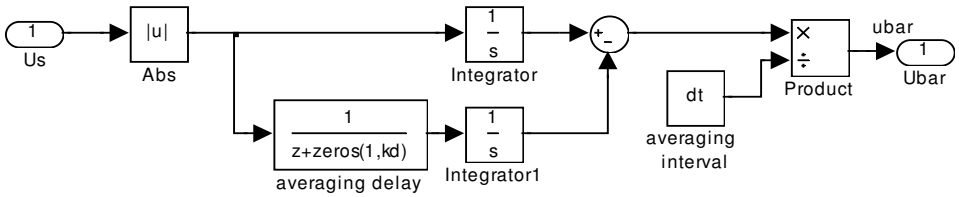


Figure 31: PVG32 Input averaging subsystem

state gain of approximately $K = 34$. However the fit between the step and sinusoidal time traces is good only for a small range of input voltage amplitude, indicating that the steady state gain $K = K(u_s)$ depends on the input. The best model fit was achieved by *averaging* the input magnitude for a period of time and then computing the steady state gain required to produce this output from the experimental data. The average input is computed from

$$\bar{u}_s(t) = \frac{1}{\Delta t} \int_{t-\Delta t}^{t} u_s(t)\, dt \tag{124}$$

and the steady state gain is computed from

if

$$|\bar{u}_s| < 0.55V$$

then

$$K(\bar{u}_s) = \frac{c_2 \bar{u}_s^2 + c_1 \bar{u}_s + c_0}{\bar{u}_s}$$

otherwise

$$K(\bar{u}_s) = 0 \tag{125}$$

The time averaging interval was adjusted to $\Delta t = 0.16$s, and the coefficients c_0, c_1, and c_2 were found using the steady-state portions of figure 28. The Simulink subsystems 'Input Averaging' and 'SS gain' in figure 30 represent equations (124) and (125), and are shown in figures 31 and 32, respectively.

The contents of the 'Flow direction' block in figure 30 checks the sign of the input signal and diverts the computed flow to workport A or B accordingly. The deadband block was also added ahead of the transfer function to model the effect of spool overlap. The pressure

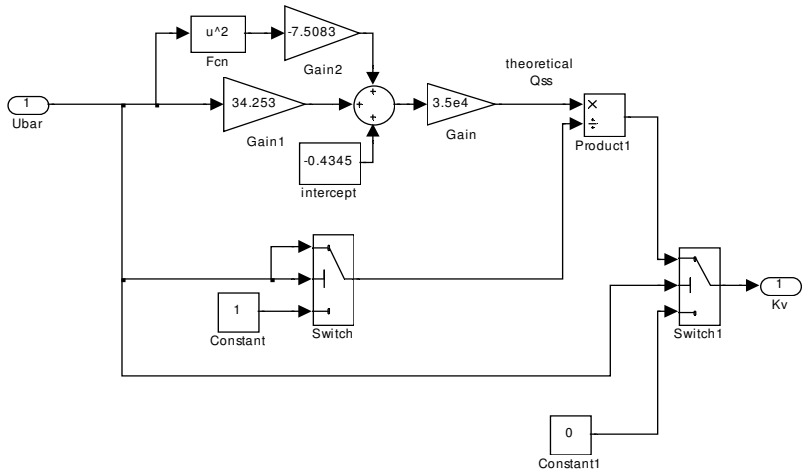


Figure 32: PVG32 Steady-state gain subsystem

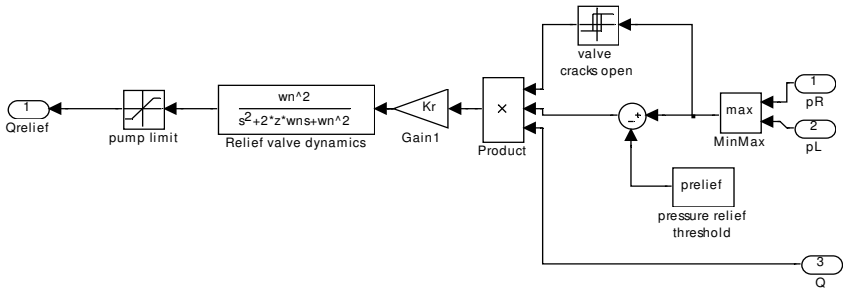


Figure 33: PVG32 relief valve model

relief valve was also modeled to simulate the effects observed when the HIL load cylinder hit the end of its travel, as illustrated in figure 33.

See Appendix G for the Matlab script written to optimize the PVG32 model parameters.

3.4.5 PVG32 model validation

Once the PVG32 model parameters had been identified, the step and sinusoidal data from the HIL testing were sent to the PVG32 model to validate performance. Figures 34 and 35 compare the model prediction to the experimental data for a range of step sizes and driving frequencies, respectively.

Figure 36 illustrates the frequency response of the model over the range of experimental data.

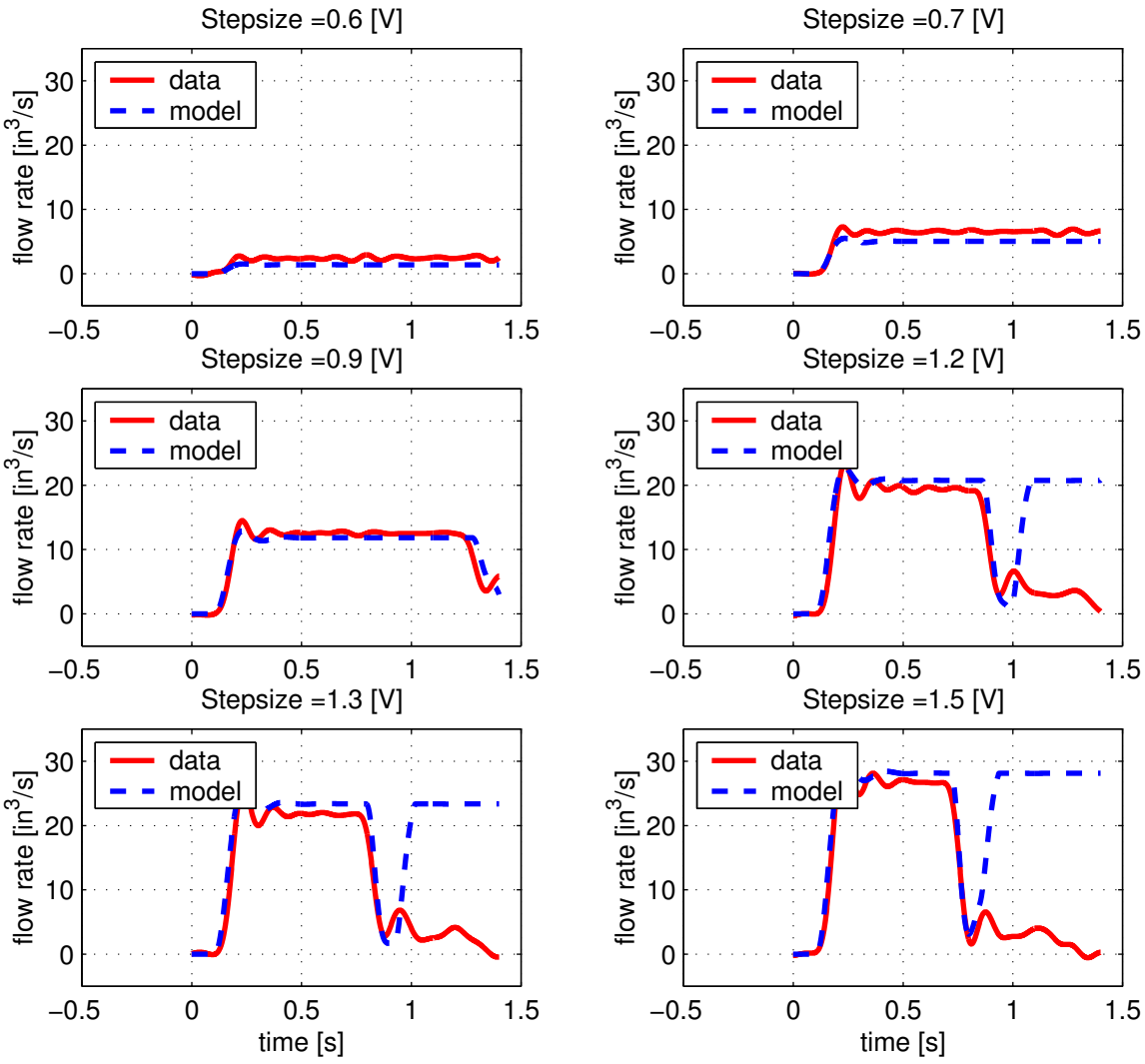


Figure 34: PVG32 model step response validation

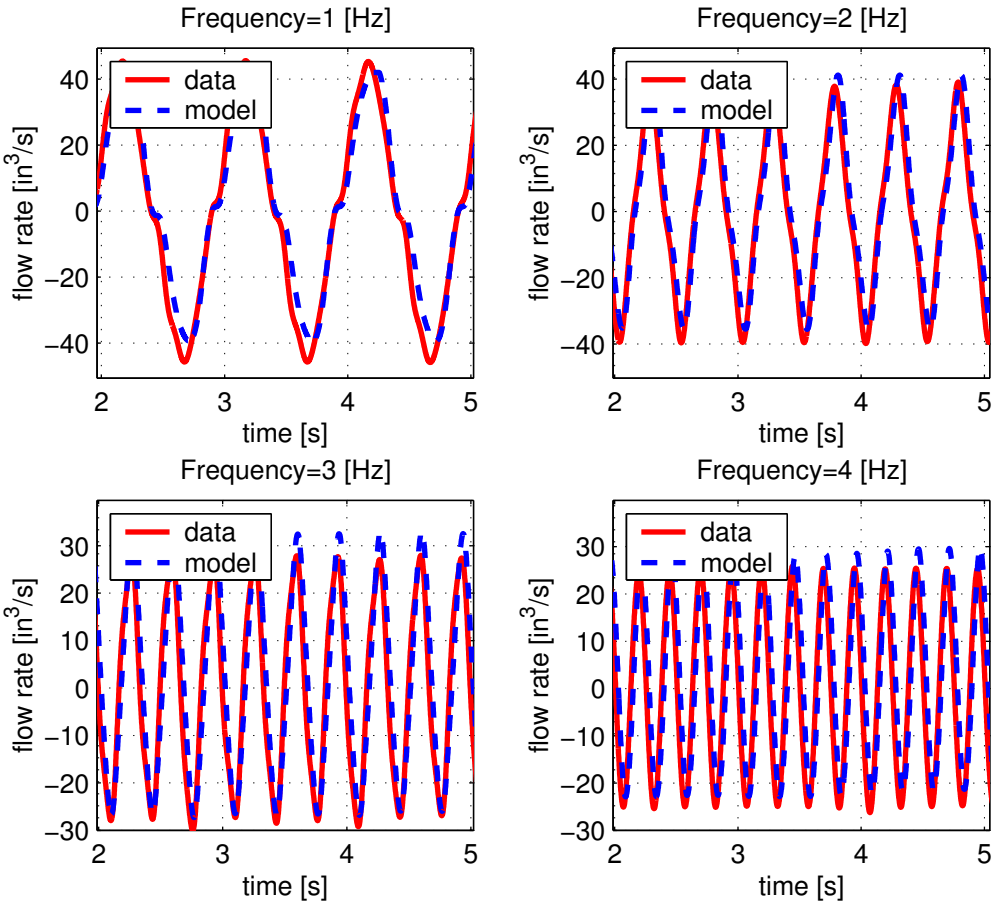


Figure 35: PVG32 model sinusoidal response validation

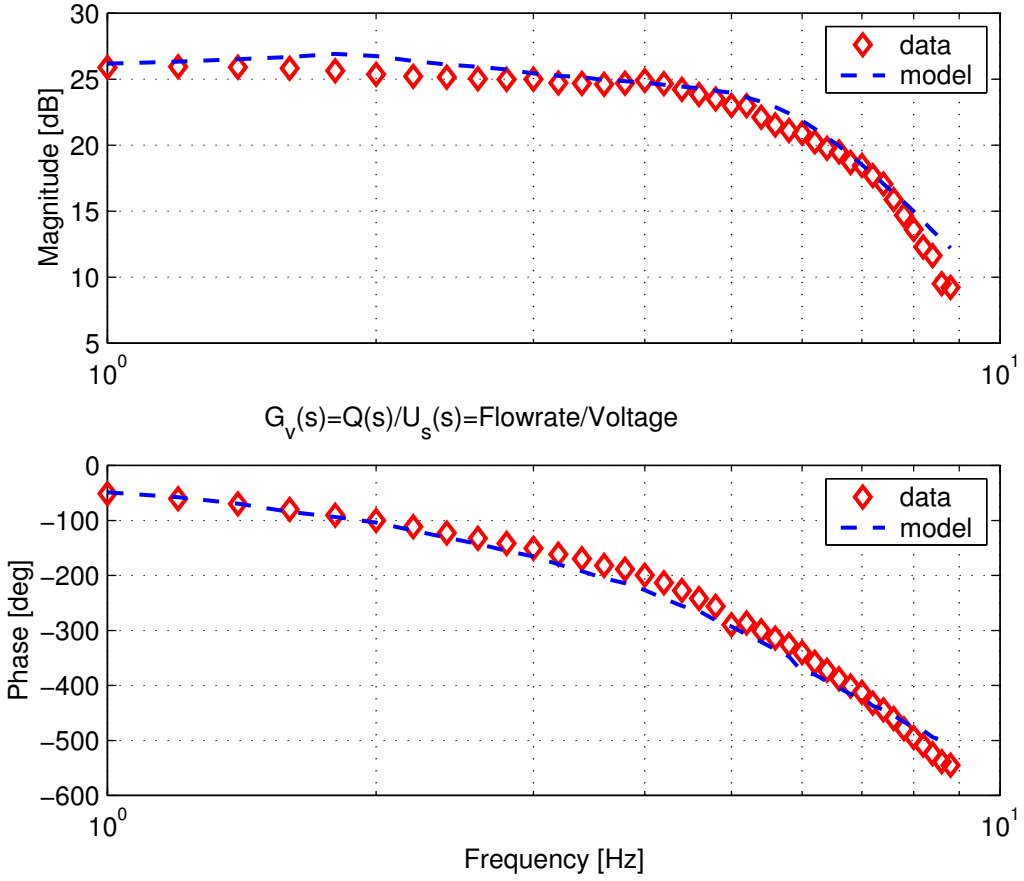


Figure 36: PVG32 model frequency response validation

As can be seen, the model fits the data extremely well for all the data sets available. Note that in the step response plots show in figure 34, oscillations occur in the data during the settling period which are not predicted by the model. This discrepency is most likely due to unmodeled nonlinearities in the valve such as Coulomb friction in the spool. Consider that the valve model dynamics are fundamentally based on the linear characteristics given in equation (123), and that the parameters of equation (123) were optimized by minimizing the errors between the measured output data and model prediction using the same input. In order to optimize the pole locations, damping characteristics in the dynamic case would be overestimated to account for the Coulomb friction in the static case, resulting the oscillations observed in figure 34.

3.5 Hose and Cylinder Compliance

When hydraulic fluid under pressure is forced into the hoses and cylinders downstream from the valve, the fluid will compress and the walls will stretch in a springlike manner dependent upon the bulk modulus of the oil and the elastic modulus of the hose and cylinder walls [25], [13], [28]. The compliance C of a control volume defines the change in volume with respect to the change in pressure:

$$C = \frac{\Delta V}{\Delta p} \tag{126}$$

Considering a control volume surrounding the oil, hose, and either the cap or rod end of the cylinder, the combined compliance can be defined as

$$C = \frac{V_{cyl} + V_{hose}}{\beta} + \frac{dV_{hose}}{dp} + \frac{dV_{cyl}}{dp} \tag{127}$$

where the first term in equation (127) represents the bulk compression of the fluid, and the second and third terms represent the stretch in the hose and cylinder walls, respectively. Pressure is assumed to be instantaneously constant throughout the control volume.

The bulk modulus of the oil relates the change in pressure to a change in volume:

$$\beta = -V_0 \frac{dp}{dV} \tag{128}$$

where β is the bulk modulus of the oil and V_0 is the initial volume. The value of β ranges between 250,000 and 400,000 psi for degassed oil, but can be dramtically reduced by the presence of only a small amount of trapped air [25].

For small circumference changes, the tangential stretch of the hoses and cylinders can be approximated using models of an axially loaded beam and the hoop stress in a thin-walled cylinder:

$$\delta = \frac{\pi p D^2}{2tE} \tag{129}$$

where t is the thickness of the cylinder wall, D is the diameter, and E is the elastic modulus. The volume enclosed is then the initial volume plus the volume increase due to pressure:

$$V(p) = V_0 + \Delta V = \frac{L}{4\pi}\left(\pi D + \frac{\pi p D^2}{2tE}\right)^2 \tag{130}$$

where equation (129) has been used to express the change in circumference. Taking the derivative of equation (130) with respect to pressure yields the compliance of a cylinder:

$$\frac{dV}{dp} = \frac{\pi L D^3}{4tE}\left(1 + \frac{pD}{2tE}\right) \tag{131}$$

The compliance of the hoses and cylinder walls can then be computed using equation (131) and their respective values of L, D, t, and E.

Once the overall compliance of each control volume is known from equation (127), pressure can be computed by integrating the flow across the boundaries, adding the volume swept out by the piston, and dividing by the compliance,

$$p_c(t) = \frac{\int Q_c dt - (y_C(t) - y_{C0})\, A_c}{C_c} + p_{c0} \tag{132}$$

$$p_r(t) = \frac{\int Q_r dt + (y_C(t) - y_{C0})\, A_r}{C_r} + p_{r0} \tag{133}$$

where y_C is the cylinder position, Q is the flow into the volume, A is the piston area, the subscripts c and r denote the cap and rod-side volumes, respectively, and y_{C0}, p_{c0} and p_{r0} are initial conditons. The net force on the rods due to pressure is then

$$F_r(t) = A_c p_c(t) - A_r p_r(t) \tag{134}$$

where friction between the piston seals and cylinder walls has been neglected. Note that the backhoe has eight total compressible volumes, one cap volume and one rod volume, for each of the five cylinders. The two swing cylinders are considered to have only one cap and one rod side volume for the two because of their crossover connection. See figure 46

Figure 37 illustrates the compliance model of the oil, hoses, and cylinders. Equations (132), (133), and (134) were used in the overall backhoe model to relate the flow coming from the valves to the forces exerted on the backhoe links, and are implemented in the block labeled 'Hoses & Cylinders' in figure 41.

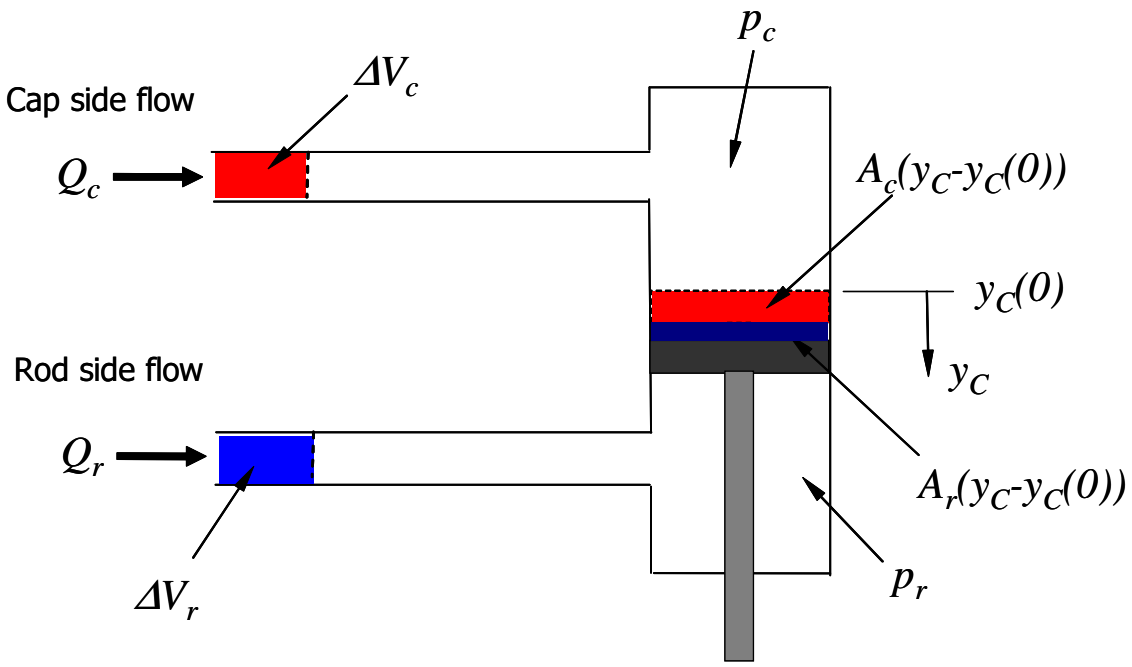


Figure 37: Hose and cylinder compliance model

CHAPTER 4

SIMULATIONS

4.1 Bucket Trajectory Simulation

Perhaps the most common use of a backhoe is to dig a trench. The bucket motions involved in a trenching operation consist of the follwing sequence:

1. reach out with the bucket
2. plunge down into the soil
3. draw the bucket along the bottom of the hole to fill it with dirt
4. lift the dirt out of the hole
5. swing the bucket over to the spoil pile
6. unload the bucket onto the spoil pile
7. swing the bucket back and repeat the process.

This is most easily visualized in Cartesian space as a time-varying vector of the form $\mathbf{y}_e = [x(t)\ y(t)\ z(t)\ \phi(t)]^T$. The Cartesian trajectory can be specified using chosen lengths for reach, plunge, scoop, etc., as long as the trajectory lies in the *cylindrical* space of the backhoe. Once specified, the joint angles and cylinder lengths required to produce this motion can be computed using the transformations laid out in sections 3.2.3 and 3.2.4.

To validate the kinematic transformations, a typical trenching trajectory has been specified in figure 38, where the plot represents the trajectory of the bucket tip in 3D space over a 30-second trenching cycle. The bucket angle ϕ is not shown.

This trajectory has been transformed through the Reverse Displacement routine to compute the required joint angles, and then back through a Forward Displacement routine to locate each link origin O_i in 3D space vs. time. The resulting trajectories of *all* the link origins are plotted in figure 39 in the vertical $\mathbf{x}_0$-$\mathbf{z}_0$ plane. The bold dashed lines represent the trajectory of each joint, and the thin solid lines represent the backhoe configuration

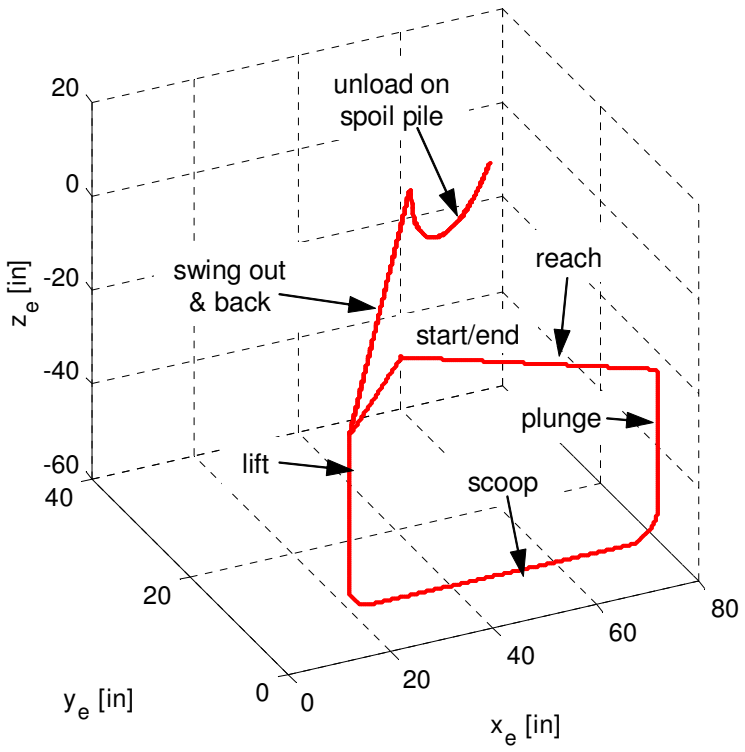


Figure 38: Bucket tip trajectory for a trenching operation

along those trajectories at 0.6 sec intervals. The 'swing out', 'swing back', and 'unload on spoil pile' segments lie in the third dimension and have been omitted for clarity.

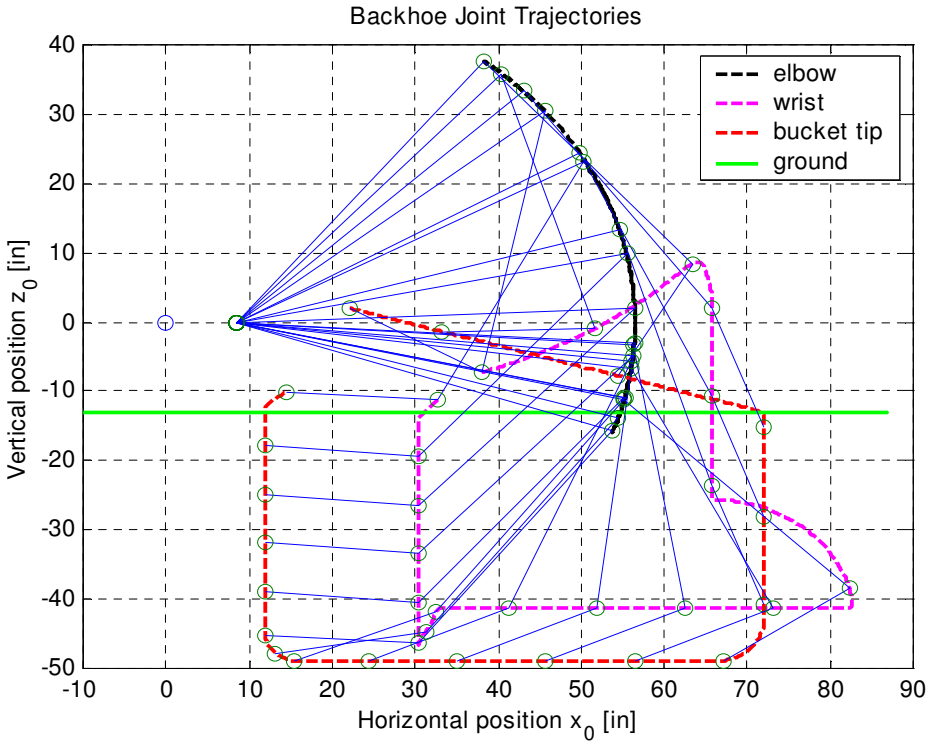


Figure 39: Trenching simulation joint trajectories

Figure 40 illustrates the cylinder space vectors for the boom, stick, and bucket during this simulation. Note that these trajectories could be used as reference commands in the controller if fully autonomous digging was desired.

4.2 One Degree of Freedom Model Validation

Tests were conducted early on in the project to prove the haptic backhoe concept on a single degree of freedom. During these tests, the dipperstick cylinder was equipped with a position sensor, and a single PVG32 valve was used to actuate that cylinder under closed–loop control. Part of these tests consisted of collecting the stick cylinder's response to a sinusoidal position reference. Data was collected for the valve command signals (input) and cylinder lengths (output), which is used in this section to validate the combined system

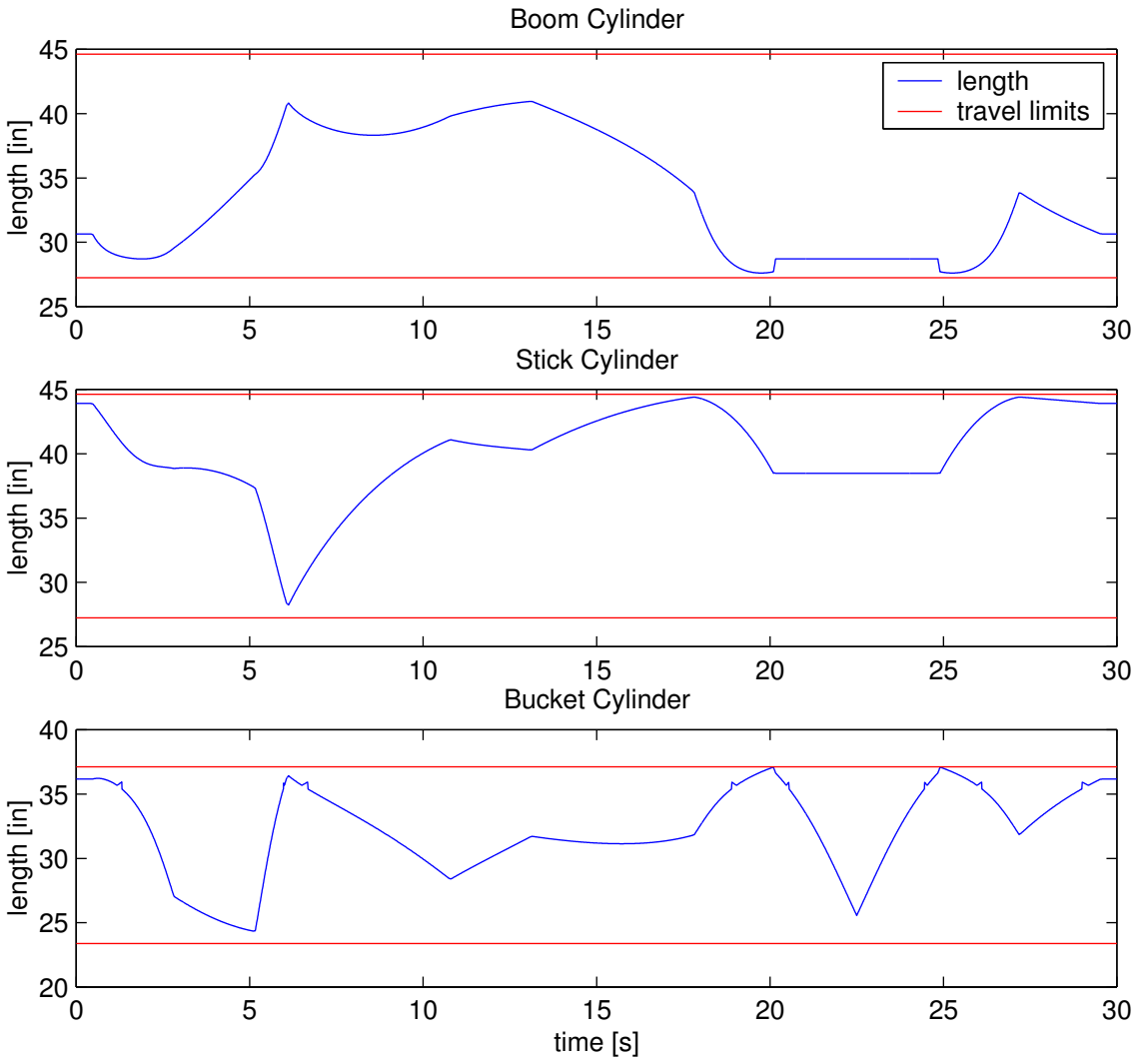


Figure 40: Trenching simulation cylinder trajectories

model presented in sections 3.4 and 3.3. See section 5.1 for more detail on the one–degree–of–freedom tests.

To validate the combined valve and backhoe models, the same input voltage that was sent to the valve during the 1–dof testing was also input to the combined model, and the stick cylinder length that was predicted by the model was compared with the data.

Figure 41 illustrates the top-level block diagram used for modeling the 1–dof system. The contents of the valve block can be found in figures 29 through 33. The dynamic model of the backhoe links is shown in figure 22.

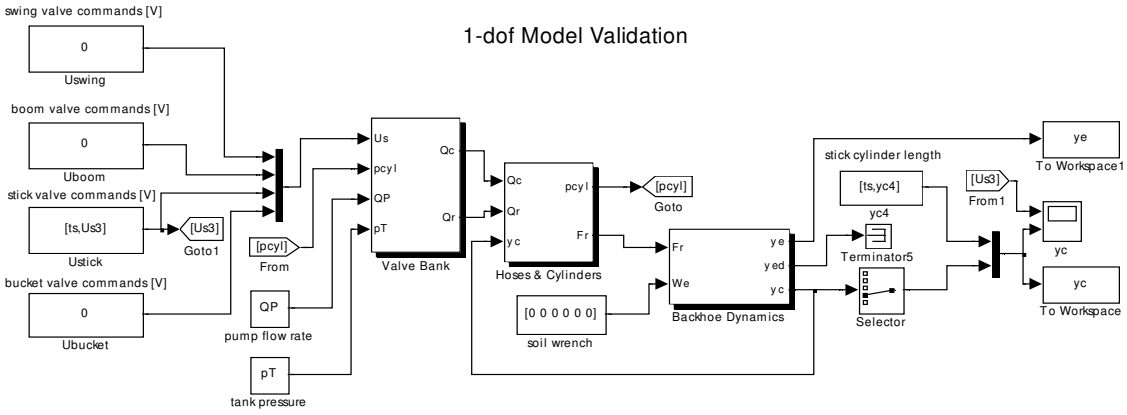


Figure 41: 1 Degree of Freedom System Model

Figure 42 compares several quantities between the data and the model. Figure 42(a) shows the command voltage sent to the valve during testing, which was a 0.4Hz, 1V sine wave. The wave is distorted due to the feedback regulator that was running to eliminate drift during the test. Noise spikes occurred when the position sensor rod, fixed to the boom, struck the magnet fixed to the stick. Figure 42(b) shows the flow rate from the valve into the cap end of the cylinder as predicted by the model. The effects of the spool deadband are clearly visible in this plot. Figure 42(c) compares the cylinder length predicted by the model with the data collected during the 1-dof testing.

As can be seen, the model predicts the cylinder position very well. Note that there is a small amount of drift in the model due to a high sensitivity to error in the zero reference of the input voltage. Also, the model prediction curve has been shifted vertically to account

for nonzero initial conditions at the beginning of the data set.

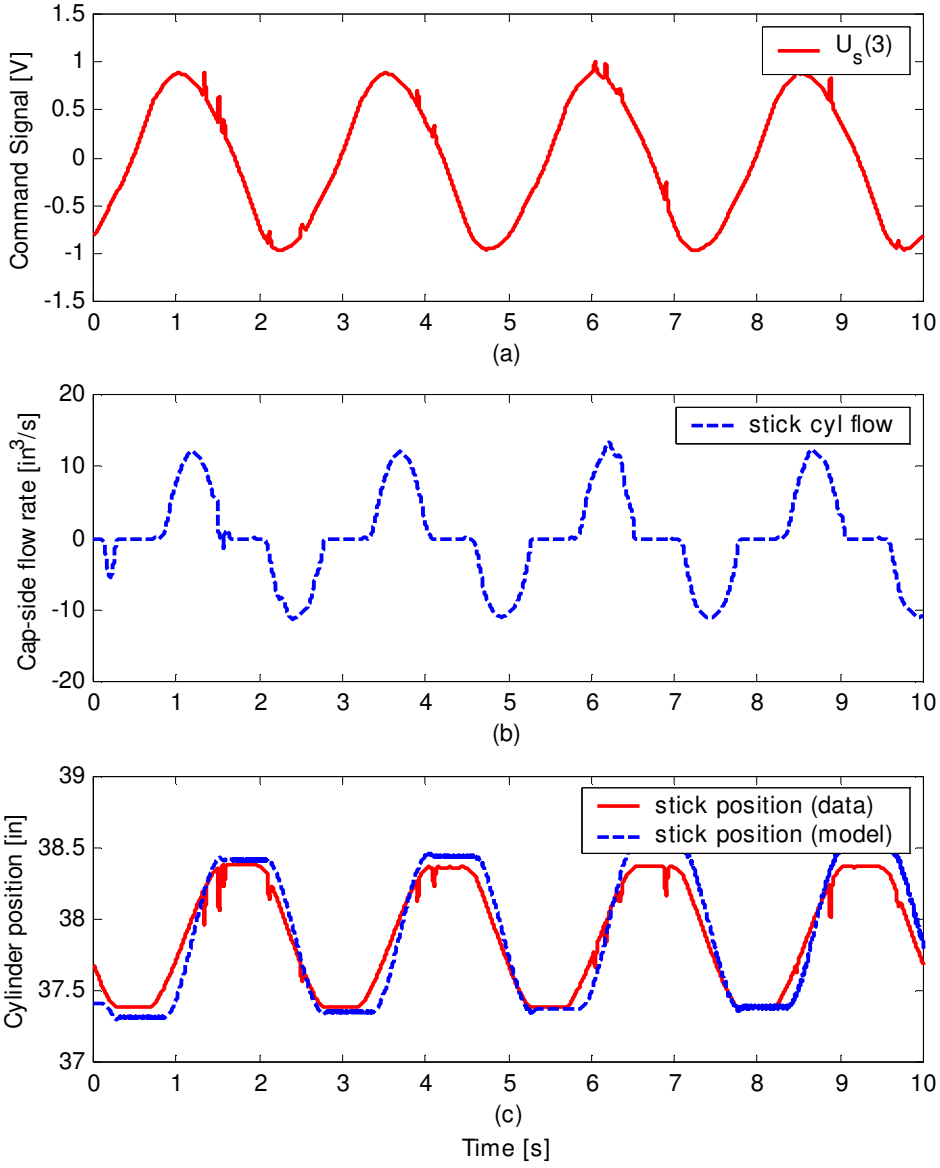


Figure 42: One degree of freedom system model validation

4.3 Four Degree of Freedom Model Validation

The next step in developing the system model consisted of extending the simulation to four degrees of freedom and comparing the response with experimental data. Before this could be done, however, the real system had to be built by retrofitting position sensors and PVG32 valves to all four degrees of freedom so that data could be collected and used for

model validation. Retrofitting the sensors and valves represented the majority of the work building the haptic backhoe, and was the primary focus for several months. The design and construction of the four degree of freedom haptic backhoe is described below in section 5.2.

Once the real system was complete, the simulation was extended to model the motion of all the backhoe's links. Input/ouput data was collected similar to the tests described in section 4.2 above as all four functions were operated simultaneously. This data was then used to validate the system model as described below.

4.3.1 Sinusoidal tracking

The first simulation consisted of using sinusoidal cylinder length references and closed-loop proportional control. Valve command voltages produced by the controller and position measurements from the sensors were logged from the backhoe, and then the same time traces of the valve commands were used in simulation as open-loop input to the plant model. Figure 43 illustrates both the data collected from the real system and the comparison with the model output using the same valve commands. It was found that the model output was highly sensitive to small errors in the spool deadband and the cylinder friction. These two parameters were adjusted slightly to best fit the model to the data.

4.3.2 Trajectory tracking

Next, a digging trajectory similar to the one illustrated in figure 39 was input to both the backhoe and the model, and again, the simulation output was compared with the collected data. First, the desired Cartesian trajectory shown in figure 39 was transformed to desired cylinder lengths as described in sections 3.2.3 and 3.2.4, and then used as a cylinder length reference for both the backhoe and the model. Defining the input as vector of the closed–loop command voltages produced by the controller, and the output as the vector of cylinder lengths, input/output from both the backhoe and the model are shown in figure 44.

As can be seen in figure 44, the model does a reasonable job at predicting the response of the backhoe to an arbitrary reference trajectory. However there is plenty of room for improvement. Considering the complexity of both the real system and the (32^{nd} order) model, and the limited amount of measurement data available, some parameters could

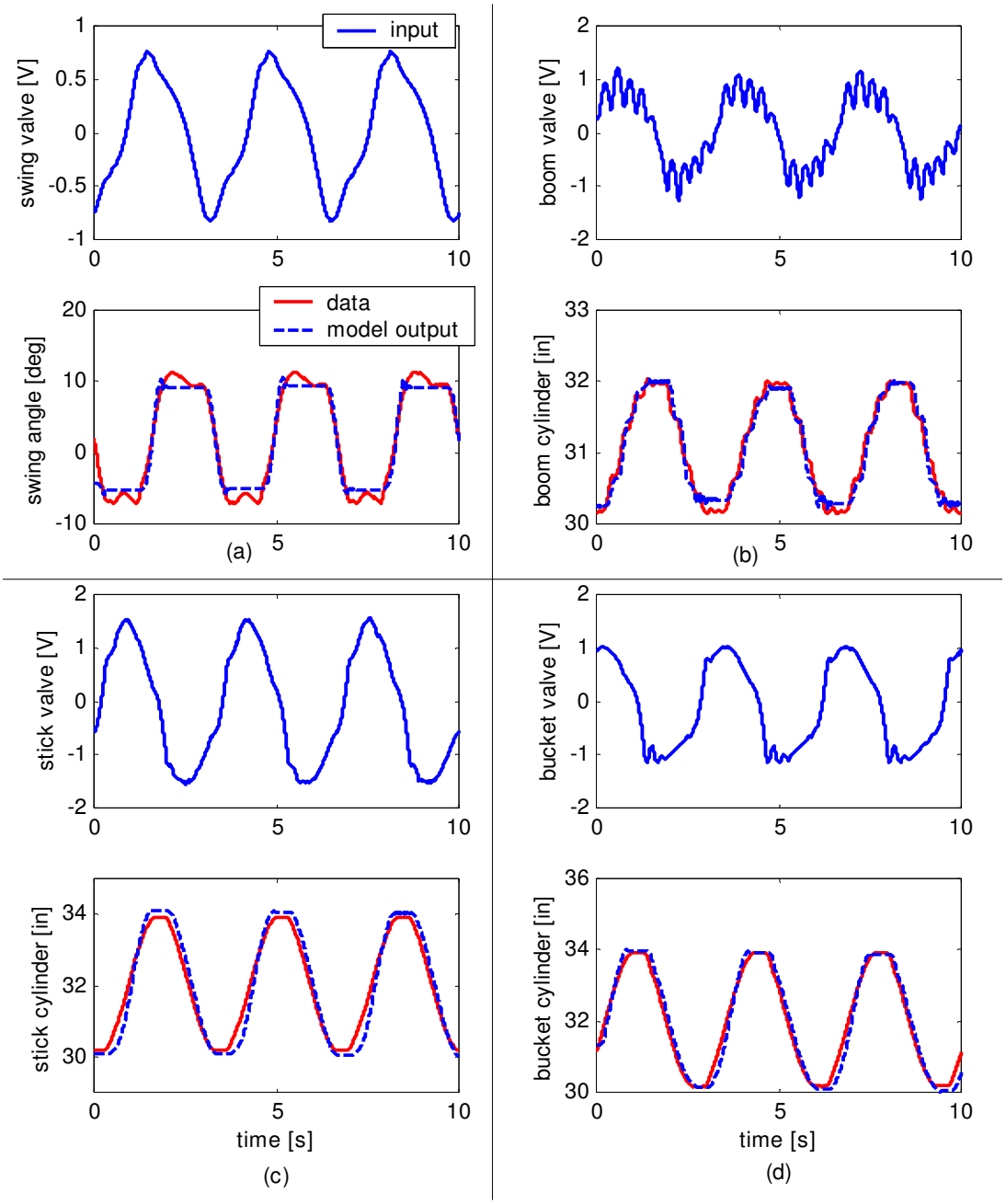


Figure 43: Response to a sinusoidal cylinder reference: data vs. model

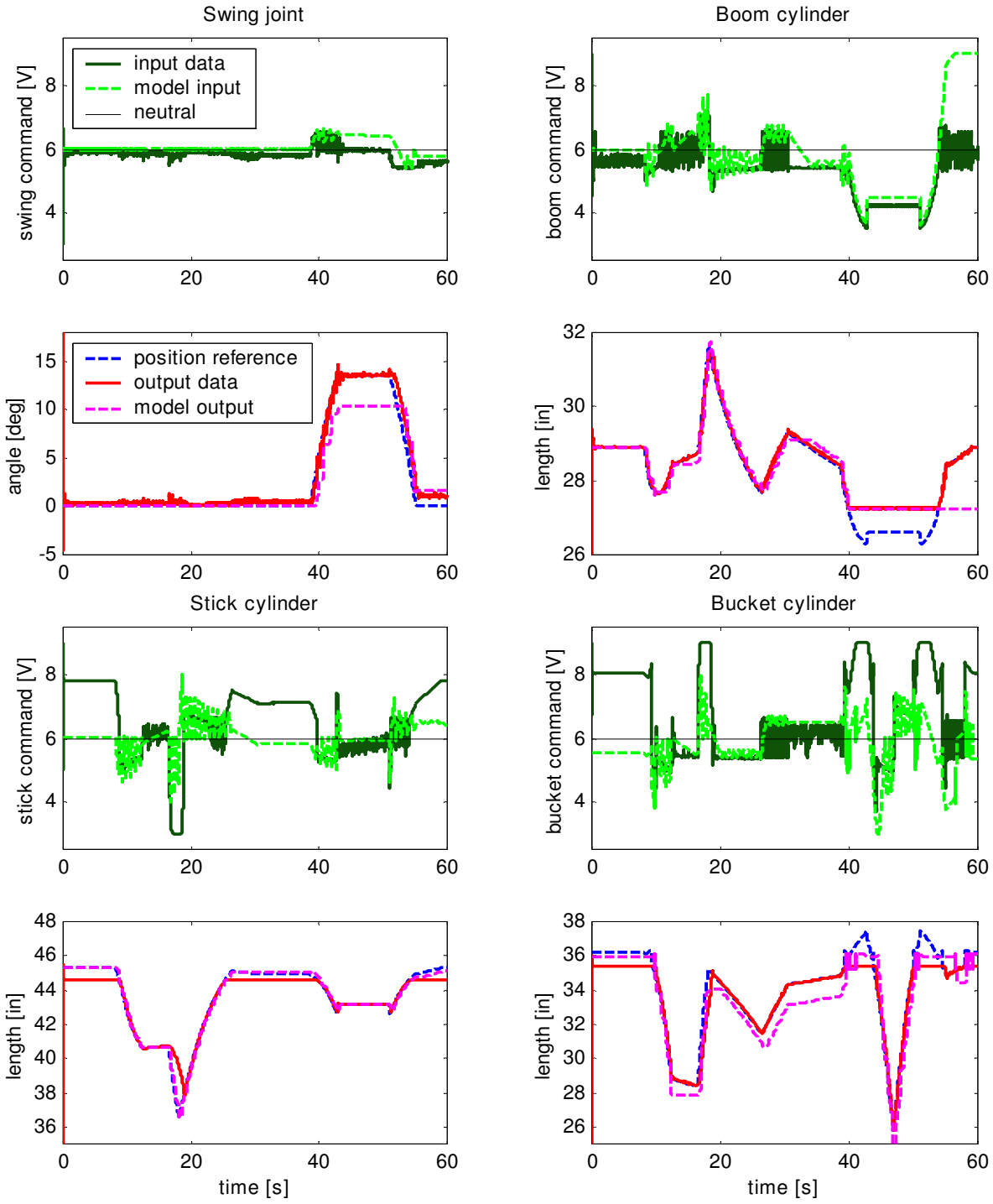


Figure 44: Trajectory tracking: data vs. model

only be estimated by adjusting the model and re–running the simulation. These uncertain parameters include friction between the seals in the inner surfaces of the cylinder walls, the pressure drop across hoses and fittings due to viscous resistance in the oil, the flow coefficient of oil returning to tank through the valve, and the dynamics of the relief valves. At the time the data shown in figure 44 was collected, pressure sensors were not yet installed in the backhoe, making both Coulomb and viscous friction difficult to estimate. Consequently, viscous friction has been modeled as negative torque feedback coming from the joint rates, and Coulomb friction has been neglected completely. Other parameters have either been estimated or neglected. It is the author's opinion that future refinements of the model should include measurements of these parameters, all of which could be identified much more accurately from pressure measurements designed for this purpose.

As a means to visualize the simulation output, the Pro/ENGINEER Mechanism Design extension was used to define joints and joint drivers, and the simulation results were animated and saved as video files. See figure 53 for an illustration of the model used for animation. Video files of the animated solid model in .mpg format can be downloaded from the backhoe website or from the CD-ROM.

CHAPTER 5

DESIGN

5.1 Proof-of-Concept System Design

The haptic backhoe was initially set up with control of only one degree of freedom to illustrate the proof of concept. To accomplish this, the dipperstick was plumbed with a PVG32 valve, retrofit with a Temposonics position sensor, and controlled with the PHANToM. Closed-loop haptic control of the backhoe was achieved for the first time on Oct. 29, 2003. Figure 45 illustrates the hardware used for this setup.

Figure 45: One-degree-of-freedom Haptic Backhoe

5.1.1 Hydraulic Modifications

The original Gresen V20 valves installed in the backhoe have a 'power beyond' port with a removable plug at the end of the valve bank, from which high pressure flow can be accessed and used for additional hydraulic loads. The pump connection on the PVG32 valve was connected to the power beyond port and the tank connection on the PVG32 was spliced into the Gresen valve's return line with a tee. Figure 46 shows the hydraulic circuit for the 1-dof system, which has been taken from [15] and modified slightly to illustrate the addition of the PVG32 valve.

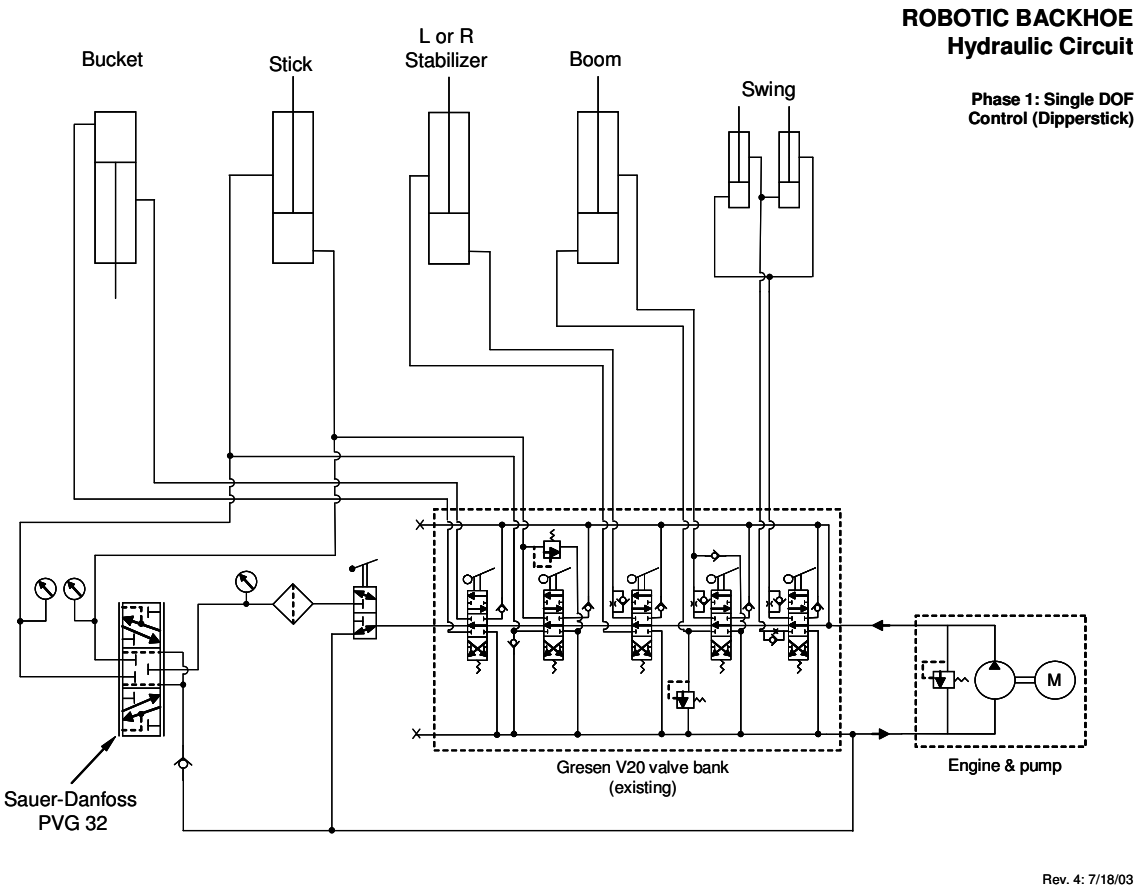


Figure 46: Hydraulic circuit with PVG32 valve addition

Although this arrangement worked well enough to prove the haptic backhoe concept, it was not optimal. It was observed during testing that the Gresen valves produced much higher flow rates (i.e the backhoe moved around much faster) than the PVG32. Because of

this observation, and because of the low performance and stalling observed when practicing digging earlier, it is suspected that the main relief valve in the Gresen bank was preventing adequate pressure buildup in the PVG32 for good performance. When the PVG32 valves were replumbed later for 4-dof control, they were connected in parallel with the Gresen valves rather than in series, bypassing the Gresen's pressure relief.

5.1.2 PHANToM Control Software

The standard configuration of a Simulink/xPC Target control system consists of using two computers, a "Host" PC for developing the control program in the Simulink environment, and a "Target" PC for executing the real–time code on a fast, low–level operating system. The real–time code is generated from the Simulink model by Real–Time Workshop on the Host PC and loaded into memory on the Target PC when the program is compiled. By separating the programming and execution of control code on two different computers, the Target PC's processor can be dedicated to real–time control and avoid the interruptions inherent in more sophisticated operating systems such as MS Windows.

For the 1–dof system, two computers were used in the Host/Target configuration described above, and additionally, the PHANToM "Ghost" software was installed on the Host PC and executed using MS Visual C++. Therefore, the Host PC served to both create and compile the model running the backhoe on the Target PC, as well as to control the PHANToM in real–time with the Ghost software.

The Ghost software was programmed such that the full range of vertical (y-axis) motion of the PHANToM corresponded to the full range of motion of the dipperstick cylinder. Position feedback was performed with a Temposonics magnetostrictive linear transducer attached to the outside of the cylinder. The Simulink model running on the Target PC was programmed to read the voltage from both the position sensor and the reference coming from the Host PC, compute an error signal, and generate a control signal to send to the valve. The Host PC, also running the Ghost software, read the cylinder position, compared it with the PHANToM stylus position, and computed the haptic force to display to the operator's hand. The two programs exchanged information via UDP Send and Receive

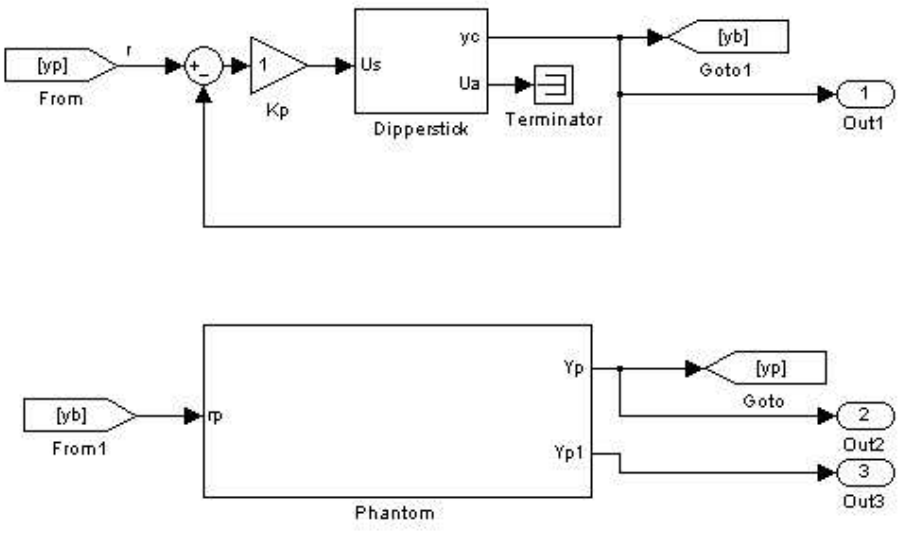


Figure 47: One degree of freedom haptic control software

blocks available in Simulink.

The Simulink portion of the program was written with the help of Matthew Kontz, who was also solely responsible for Ghost software programming. More information on the proprietary Ghost software can be found in [36] and [17]. The C code is given in appendix C.

Figure 47 illustrates the top level Simulink program for the 1-dof haptic control system. Figure 48 illustrates the contents of the Dippertsick block, which includes the xPC drivers for the National Instruments PCI-6052E A/D card. Figure 49 shows the contents of the Phantom block, including the UDP Send and UDP Receive blocks and the linear transducer conversions for both the measurement and reference signals.

Once the system shown in figure 45 was assembled, two types of tests were conducted. For the first test, the dipperstick cylinder was given a computer-generated sinusoidal reference command, and controlled with simple proportional feedback (Gain $K = 1[V/in]$) to eliminate drift. The dipperstick tracked the reference command reasonably well for frequencies below about 0.5 Hz, but at any higher frequency the combination of the spool deadband and the backlash in the wrist joints served to only excite vibration in the backlash of the pin joints, plus it made quite a racket. The data collected at 0.4 Hz is given in figure 42

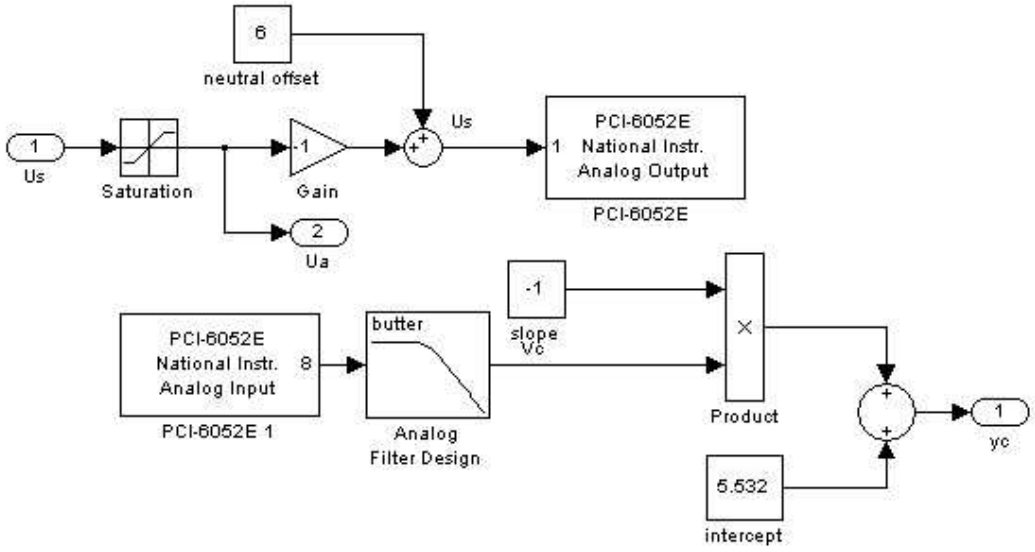


Figure 48: Dipperstick block

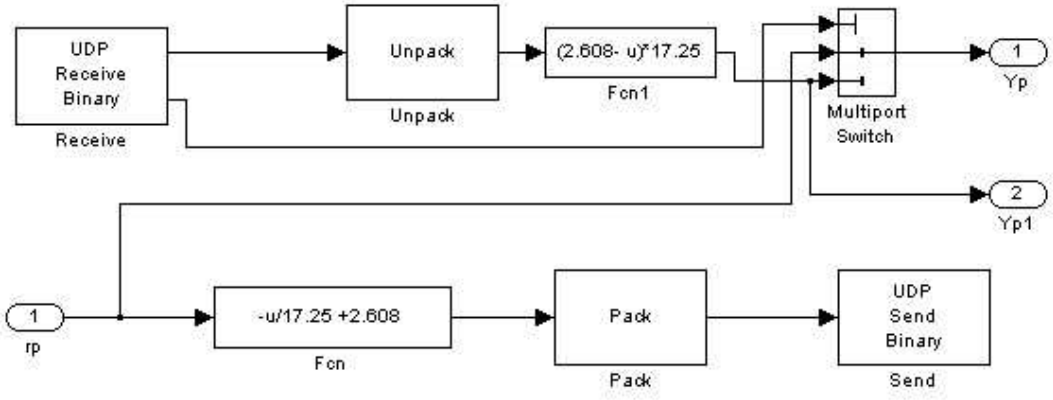


Figure 49: Phantom block

along with the model prediction as described in sections 3.3, 3.4, and 4.2. The backhoe was later rebuilt with new pins and bearing surfaces in an effort to minimze backlash.

For the second test, the PHANToM block shown in figures 47 and 49 was added to the program, and haptic control was performed by moving the PHANToM up and down while the dipperstick tracked the PHANToM's position reference in the backhoe's workspace. The backhoe performed as expected, with a spring-like sensation clearly evident linking the motion of the dipperstick with the motion of the PHANToM. This test represented the first successful closed-loop haptic control of the backhoe in unconstrained space.

It should be noted that when the PHANToM was given a light tap and then left unconstrained, marginal stability was observed as the PHANToM and backhoe vibrated in unison at approximately 5 Hz and 1" amplitude. This observed behavior is cause to suspect a need to add some damping to the controller to maintain stability [22].

It should also be noted that data collected from the 1–dof system was used to validate the system model by comparing it with simulation results, as described in section 4.2.

5.2 Final System Design

5.2.1 Hardware Modifications to the Original Backhoe

Significant improvements were made upgrading the system shown in figure 45 to include control of all four joints. The design and construction portion of the project can be described as an intense, well organized team effort between Research Engineer J.D. Huggins, PhD student Matthew Kontz, and the author, the results of which are described in the present section. Nearly every design decision was made as a team, which made possible the excellent results. Appreciation is given to J.D. for providing a tremendous amount of guidance and expertise as well as performing a significant portion of the hydraulic and metal fabrication work. Matt is credited with the construction of the power supply/junction box and associated cables and connectors, and with much of the software development. These upgrades are described in the following sections.

5.2.1.1 Valves

Three more valve modules were added to the PVG32 assembly to provide a total of four independent electrohydraulic functions. This was a convenient option inherent in the design of the PVG32. The valves were discounted deeply (nearly donated) by Sauer-Danfoss. Assembly was donated courtesy of Berendsen Fluid Power.

5.2.1.2 Cylinders

Custom hydraulic cylinders based on the original cylinder dimensions were fabricated with gun-drilled rods for housing internal position sensors. The cylinders were donated courtesy of Georgia Hydraulic Cylinders, Inc.

5.2.1.3 Sensors

Linear position sensors were installed into the boom, stick, and bucket cylinders. These were model BTL-E micropulse style, courtesy of Balluff. These sensors operate using magnetrostrictive measurement principles. A potentiometer was mounted on the swing joint axis. Pressure transducers for each cap and rod volume and a pressure gage for main gallery pressure were installed courtesy of WIKA.

5.2.1.4 Backhoe

The backhoe itself was completely dismantled and rebuilt with all new pins, bearing surfaces, new fasteners, and paint. All parts used in the rebuild were OEM, courtesy of John Deere.

5.2.1.5 Supports

The valves and pressure transducers were secured to the backhoe with a steel frame, along with a tray to support the PHANToM. Electronics modules for the BTL-E micropulse sensors were mounted inside electrical outlet boxes fixed to the cylinders.

5.2.1.6 Electrical

Digital valve commands generated by the TargetPC are converted to analog with a Measurement Computing DDA-06 PCI D/A card. Analog sensor signals received by the TargetPC are read and digitized with a Keithley-Metrabyte DAS-1602 PCI A/D card.

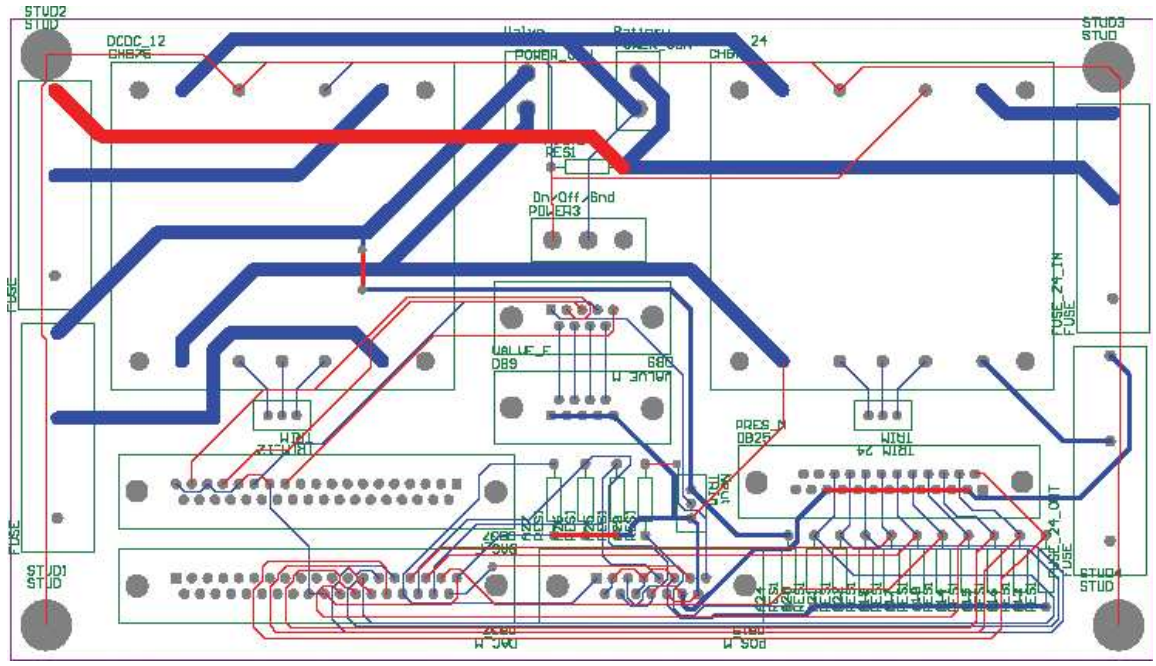


Figure 50: Power supply / signal junction board layout

A power supply/junction box was designed and built to regulate 12 and 24V power coming from the on-board electrical system, and also to interface measurement and control signals between the TargetPC, sensors, and valves. Figure 50 illustrates the board layout design created in Protel. Figure 51 illustrates the completed power supply/junction box.

Figure 51: Power supply / signal junction box, assembled

5.2.1.7 Hydraulics

All hydraulic lines not involving relative motion between the ends were assembled using hard stainless steel tubing and JIC fittings. The PVG32 valves were plumbed in parallel with the original Gresen V20 valves, with a manual 3-way selector installed to toggle between manual and haptic control modes. The 3-way valve was donated courtesy of Hydac. Figure 52 is a photo of the integrated PVG32 and Gresen V20 hydraulic valve systems.

Figure 52: Haptic Backhoe hydraulic system

5.2.2 Mechanical Design

Figure 53 illustrates the Pro/ENGINEER model used for mechanical design, showing the assembly of the major components. This model was also used for mass and inertia estimation as described in Appendix E, and animating simulation results from Matlab as described in section 4.3. Figure 54 is a photo of the backhoe during assembly, near completion.

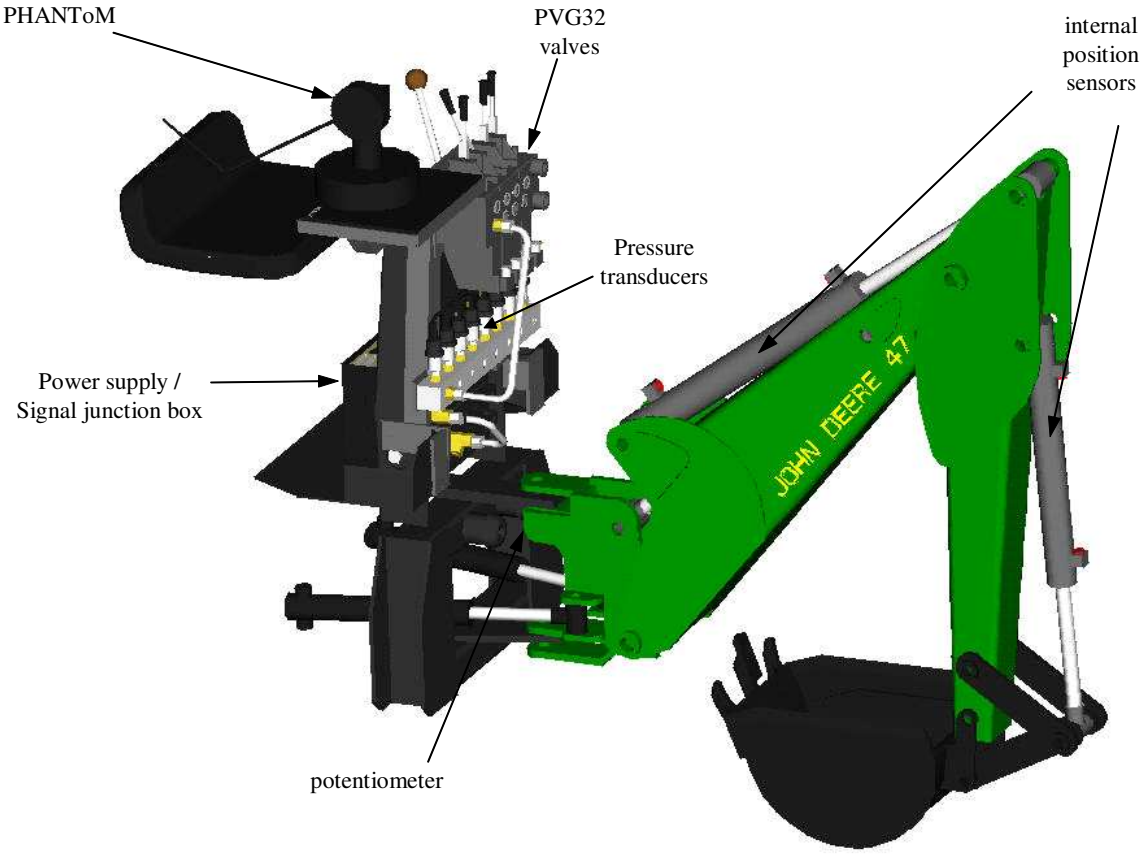


Figure 53: Haptic Backhoe design model

Figure 54: Haptic Backhoe, mid-assembly

5.2.3 System Integration

Figure 55 illustrates the interconnections of the main components of the haptic backhoe system. The PHANToM in the upper left passes encoder signals to and receives motor commands from the HostPC via the PHANToM driver box (not shown in the diagram). The Host and Target PC's exchange reference commands via an internet connection using UDP protocol. The TargetPC generates valve control signals and receives analog sensor signals via two 37 D-sub ribbon cables. The power supply / signal junction box regulates 12 and 24V power to the valves and sensors, and also routes sensor signals between the ribbon cables and the individual cables connecting the valves and sensors.

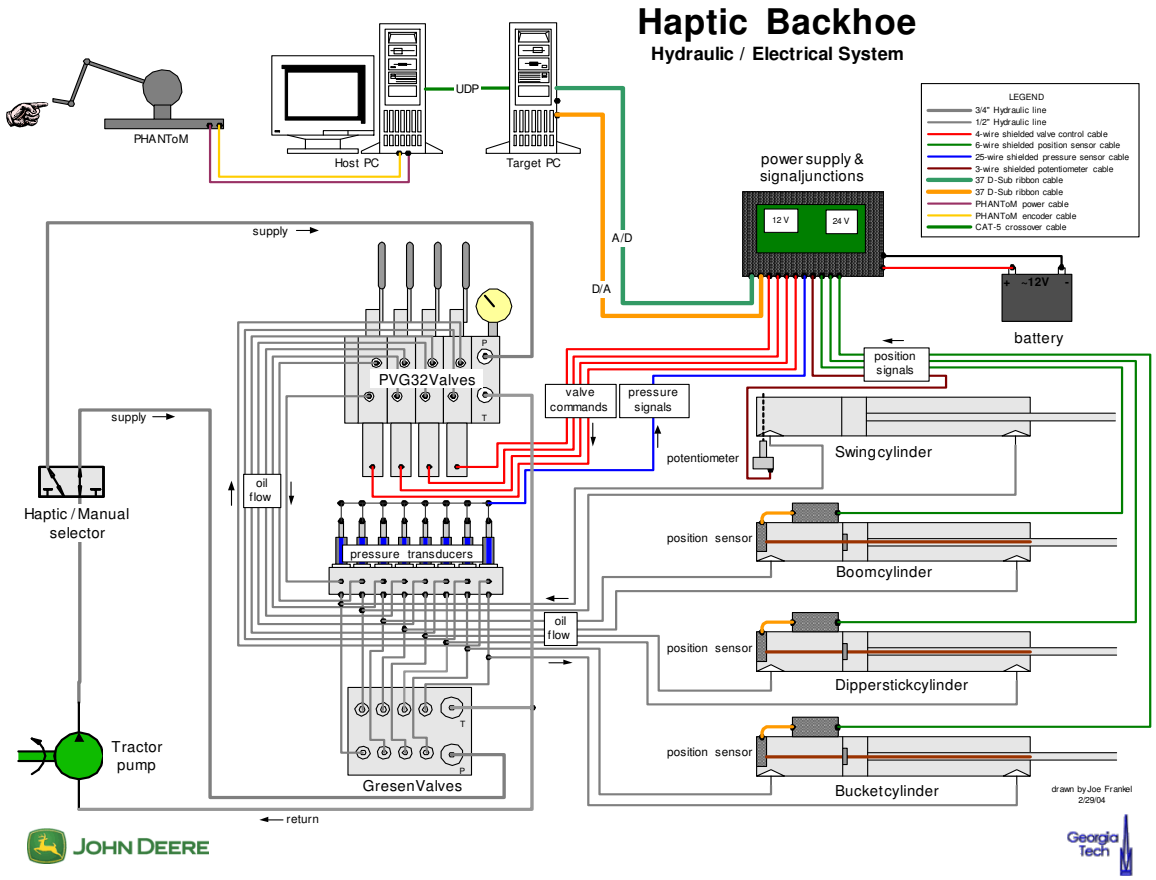


Figure 55: Haptic Backhoe system interconnectivity

Swing angle is measured with a 5kΩ rotary potentiometer aligned with the swing axis. The 4-20mA output from the BTL-E sensors on the boom, stick, and bucket are sent through

precision 500Ω resistors to measure cylinder length. The tractor supplies hydraulic power to either the PVG32 or the original Gresen V20 valves. Pressure is measured for each cap and rod at the manifold connecting the two valve assemblies.

5.2.4 Control Software

The PHANToM and backhoe communicate bilaterally through a Simulink model running on the slave (Target) computer and the GHOST software running on the master (Host) computer, as described above in section 5.1, with additional blocks to transform between Cartesian, joint, and cylinder vector spaces. The PHANToM generates the reference for the backhoe and the backhoe generates the reference for the PHANToM.

Figure 56 illustrates the Simulink model created to control the backhoe through the Target PC. The Backhoe block transmits commands from the controller to the valve via the DDA-06 analog output card, and receives sensor voltages via the DAS-1602 analog input card. The Backhoe block also converts sensor voltages to cylinder lengths from the calibration data listed in Appendix B, and outputs the cylinder lengths in real time.

The Task-Space-to-Joint-Space (T2J), Joint-Space-to-Cylinder-Space (J2C), Cylinder-Space-to-Joint-Space (C2J), and Joint-Space-to-Task Space (J2T) blocks contain the equations given in sections 3.2.1 through 3.2.4, which tranform PHANToM and backhoe position coordinates between vector spaces as necessary. The Cartesian coordinates of the PHANToM and backhoe are mapped into virtual positions in each other's workspaces inside the PHANToM block. Cylinder length errors are regulated inside the Controller block by sending appropriate voltage commands to the valves on the backhoe.

All user position/force interaction passes through the PHANToM block to and from the GHOST software running on the master (Host) computer via UDP sockets. The Host PC handles conversions from encoder positions on the PHANToM to Cartesian space and also generates actuator commands for displaying haptic forces to the hand of the operator.

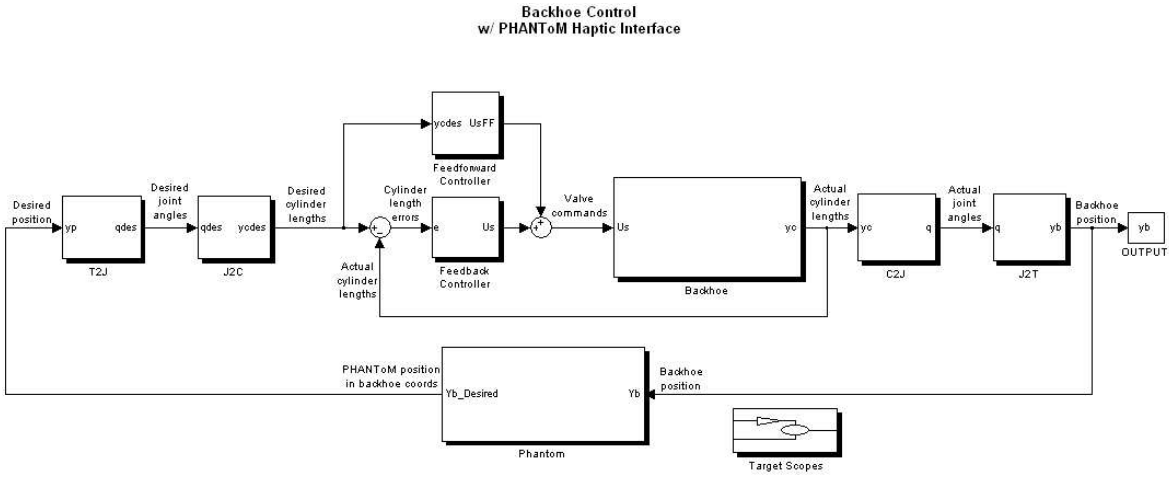


Figure 56: Closed-Loop Haptic Control Software

CHAPTER 6

TESTING AND RESULTS

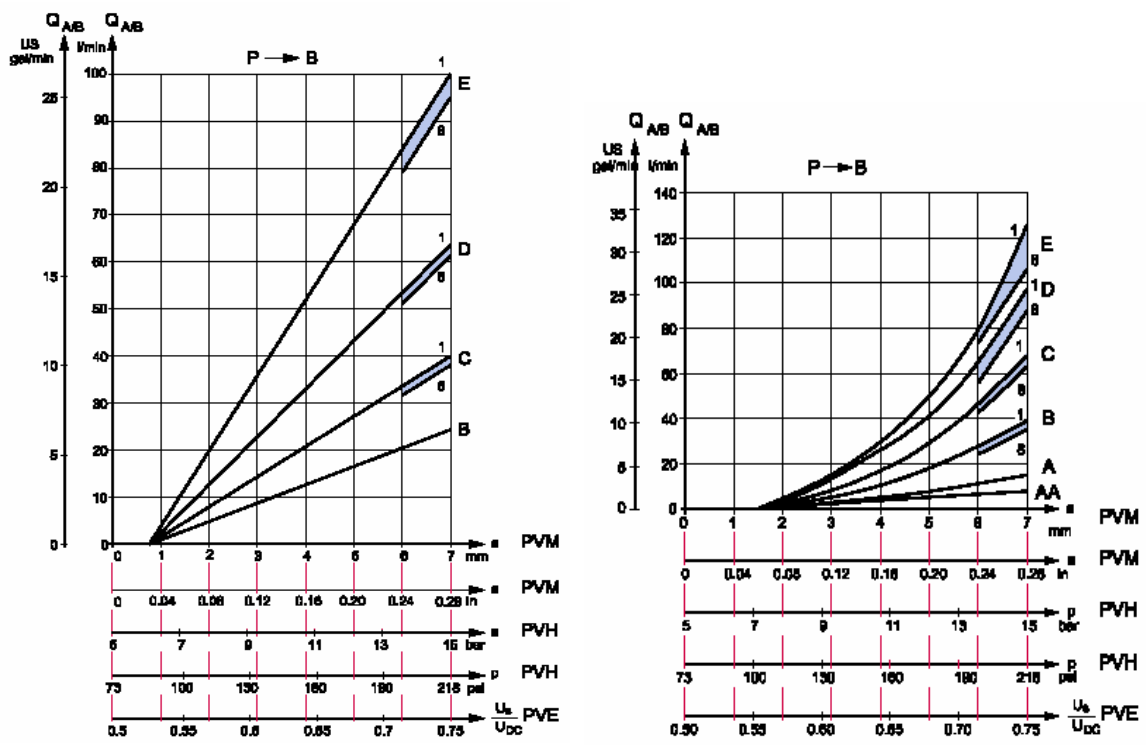
After all the hardware construction, wiring, and sensor calibration was complete, the first task at hand was to design a simple position controller that could move the backhoe around in free space by tracking a computer generated reference. This would serve to debug the basic system and establish a baseline upon which more sophisticated controllers could be designed.

6.1 Controller design and tuning

To accomplish simple position tracking, a proportional feedback controller based on cylinder length error was designed and tested. The first test consisted of sending the controller a sinusoidal cylinder length trajectory as a reference. Figure 43 in section 4.3 illustrates the valve command (input) and cylinder position (output) data collected from the first tests. Although the backhoe was able to track a sine wave on all four cylinders, it was immediately obvious that there were vibration problems in the swing and boom joints. These oscillations can be seen in the swing and boom joint data in figure 43.

After consulting with Randy Bobbitt, engineer at Sauer–Danfoss, some potential sources of vibration were identified. First, it was suspected that the pressure compensators built into the PVG32 was conflicting with the backhoe's position controller. Second, the pressure compensators require a minimum load pressure to work against, and it was suspected that they were choking off the flow under negative loading conditions. This occurs when the load pressure drops too low and tries to suck oil from the valve, such as in the cap end of the boom cylinder when the backhoe is being lowered, because gravity is pulling on the backhoe and trying to lower it faster than the valve is delivering oil to fill the cylinder. Third, it was suspected that the DDA-06 D/A board might not sourcing and/or sinking the signal current to the valves correctly.

The purpose of the pressure compensators is to adjust the pressure drop across the (main) flow control spool such that a linear relationship is maintained between voltage command and flow rate. The pressure compensation spool is illustrated in figure 24, in the center block, lower spool. The two rated flow curves with and without pressure compensation are illustrated in figure 57, copied from the PVG32 manual [34].



(a) PVG32 oil flow with pressure compensation

(b) PVG32 oil flow without pressure compensation

Figure 57: PVG32 Flowrate vs. voltage command, with and without pressure compensation

The pressure compensator essentially functions as an inner hydraulic pressure control loop within the outer position control loop. Because it is designed to adjust the pressure across the main spool, depending on both the input voltage and the sensed load pressure, and because the backhoe's controller is designed to regulate cylinder position, it was suspected that interaction between the two control loops was causing the vibrations.

Another issue to consider was the fact that the valve spools have a deadband, so that when the spools are within the deadband, no opening exists for oil to flow through. This

deadband is approximately 0.55V "wide" on each side of a 6V neutral signal, as measured during the initial HIL simulator tests described in section 3.4. Therefore, a strictly linear control law based on cylinder length error will produce no motion if the command signal is within the deadband, and so the errors can never be reduced below a minimum amount for any given proportional gain without some other type of control action. During testing, a deadband jump–through block was added to correct the deadband problem by adding or subtracting from the linear controller's command signal depending on the sign of the error.

In an effort to identify the source of the vibrations, several steps were taken based upon the discussion with Randy Bobbitt. First, the control signal was checked. The output signals going from the D/A card to the swing and boom valves were fed back into the computer via the A/D card to check if the voltage at the valve control terminals was the same as the commanded voltage computed by the controller. Figure 58 shows the input/output data on the swing and boom joints, including the signals that were fed back through the A/D card. Although a small bias of about 50mV existed between the signal sent vs. received in the swing joint, it was determined from this data that the D/A card was producing the command voltages correctly. The problematic oscillations are clearly visible in this plot also.

Next, the suspected negative loading problem was checked. For most of the feedback gains that were used, it was observed that the boom only oscillated when lowering and not raising, and the swing joint only oscillated when moving right to left and not left to right. The hypothesis was that gravity was causing an uncontrollable pressure drop in the boom cylinder when lowering, and momentum was causing a similar problem in the swing cylinders. The lack of oscillation in the swing when moving left to right could only be explained by either asymmetric resistance between the two cylinders or asymmetric flow characteristics in the valve.

To check the effect of backpressure on the boom, a small throttle valve was temporarily installed in the line connected to the cap end of the boom cylinder. The throttle valve was slowly closed as the boom moved up and down under closed–loop control with a sinusoidal reference. Unfortunately, as the gallery pressure increased to over 1000psi (as indicated on

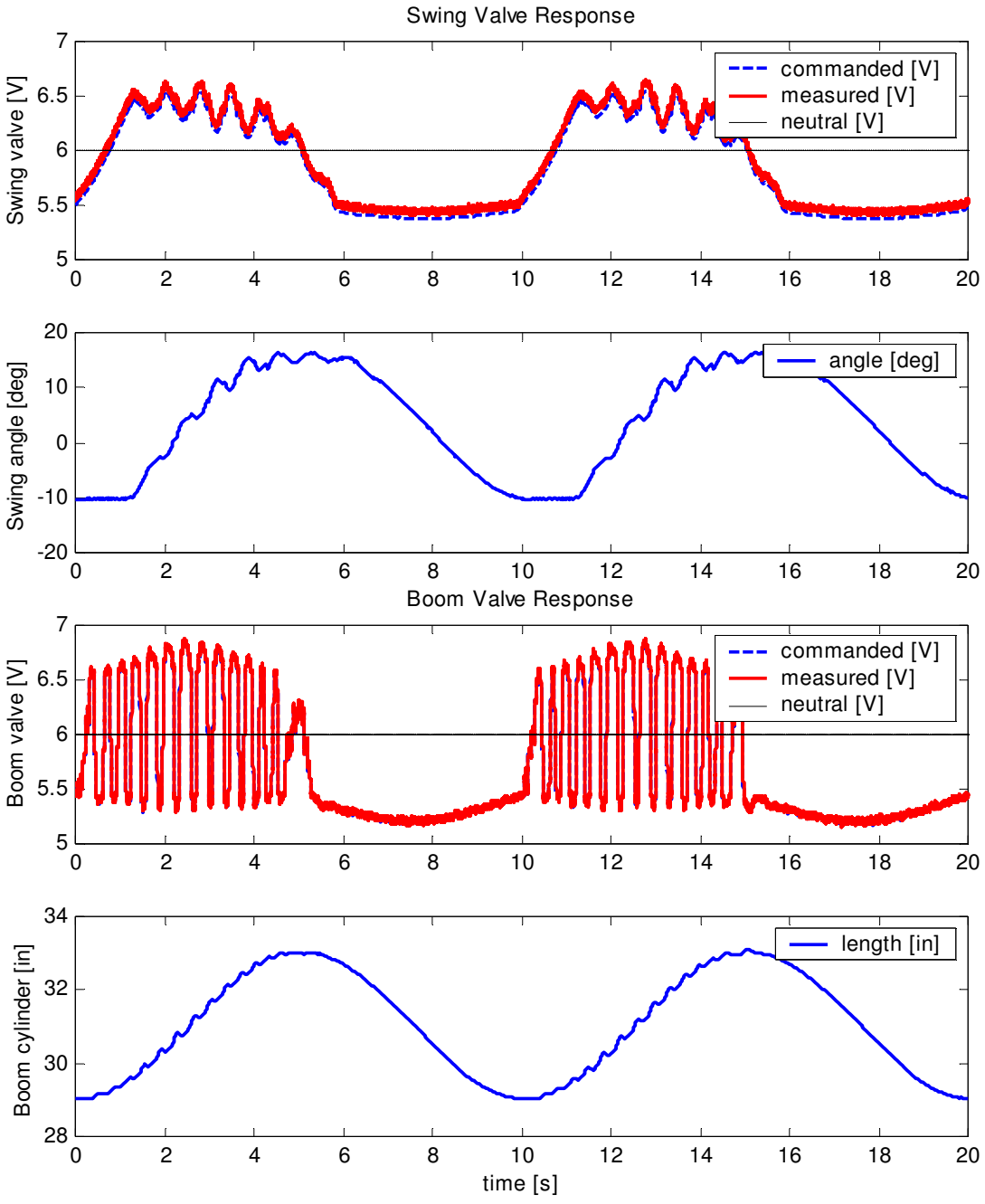


Figure 58: Swing and boom oscillations

the pressure gage) the boom continued to oscillate as it lowered. This observation did not support the claim that the oscillations could be due to negative loading.

The next step taken to identify the source of vibration was to compare open– and closed–loop control. A 0.7Hz, 0.8V square wave was sent to each of the valves as an open–loop command. The backhoe did not oscillate at all in this case, but rather moved smoothly in both directions for all four functions. Data collected from this test is given in figure 59.

Since open–loop commands did not induce vibration, it appeared that the cause was somehow related to closed–loop control. In order to minimize the proportion of the command signal generated by the feedback controller, a feedforward controller was developed based upon inverting the empirical relationship between command voltage and steady state flowrate (i.e. cylinder velocity). This relationship was estimated by taking the numerical derivative of the position data shown in figure 59. Lines were fit through the estimated cylinder velocity resulting from a 0.8V command in each direction, and the results were used in a linear lookup table in the feedforward controller. These values are shown on the estimated velocity plots in figure 59. Thus for a desired cylinder *velocity*, the feedforward controller was programmed to produce a corresponding voltage, in a linear fashion based on the velocity data. Note that using this technique to generate a feedforward signal requires a derivative of the reference position, which can be notoriously noisy and may require aggressive filtering.

The feedforward controller was then implemented, with a ± 0.55V neutral jump–through for the spool deadband, and the feedback gains were adjusted for the best performance under closed–loop control with a computer generated reference. It was found that a tradeoff existed between tracking error and vibration: high feedback gains relative to the feedforward gains induced vibration but reduced tracking error, whereas low feedback gains relative to the feedforward gains reduced vibrations at the cost of increased tracking error.

Vibrations could be almost completely removed in the boom cylinder using this approach, as illustrated by the data collected from the boom cylinder in figure 60. Cylinder tracking errors remained less than 0.5in during this test. Feedback gains, which were scheduled differently for raising and lowering, are also show on the plot. Note that a *positive*

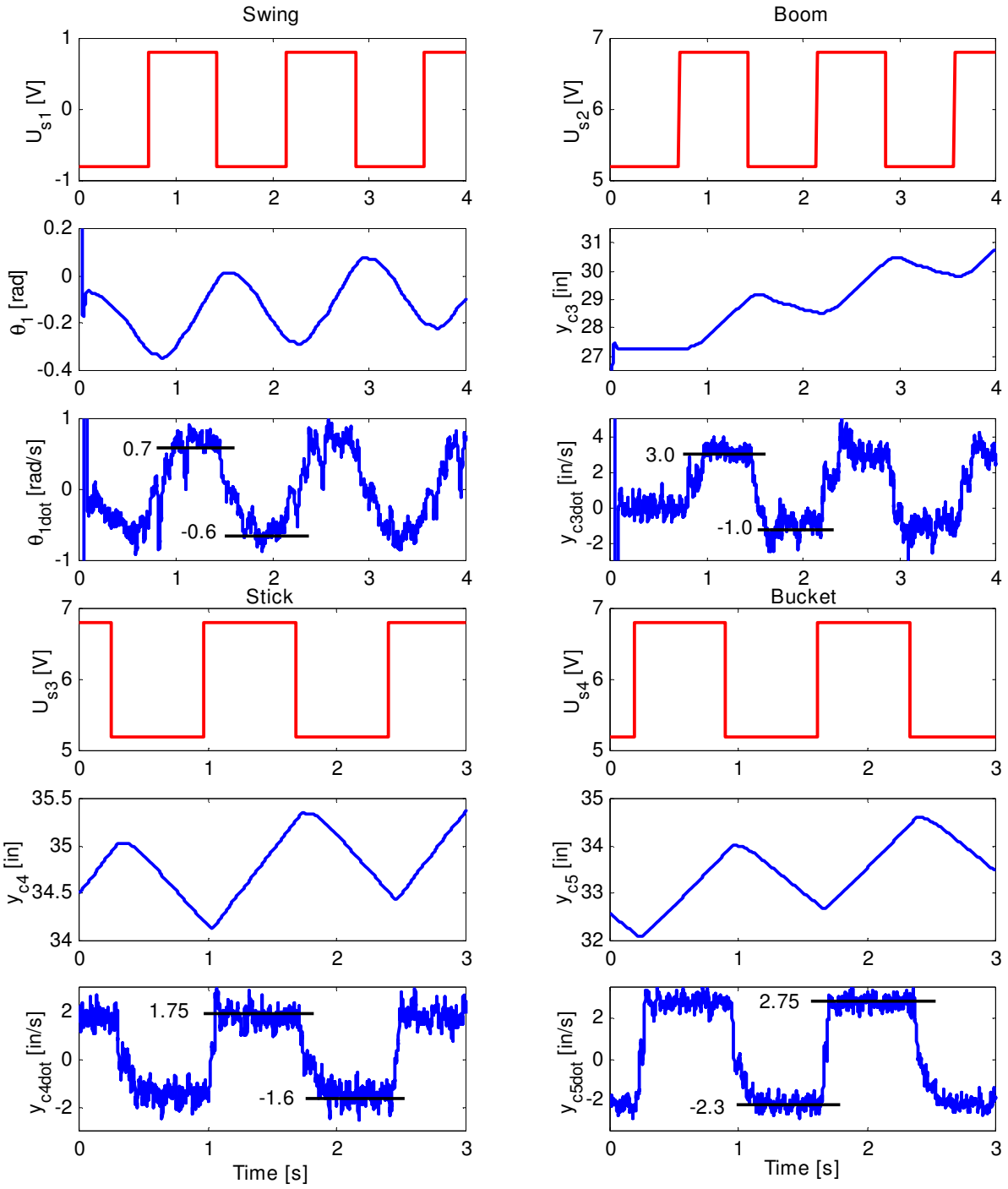


Figure 59: Open–loop response to square wave voltage

valve command corresponds to an increase in cylinder length and *lowering* the boom. The effect of the jump–through command is also evident by the vertical jumps across 6V (neutral) when the sign of the command changes, which was put in place to compensate for the deadband in the spool.

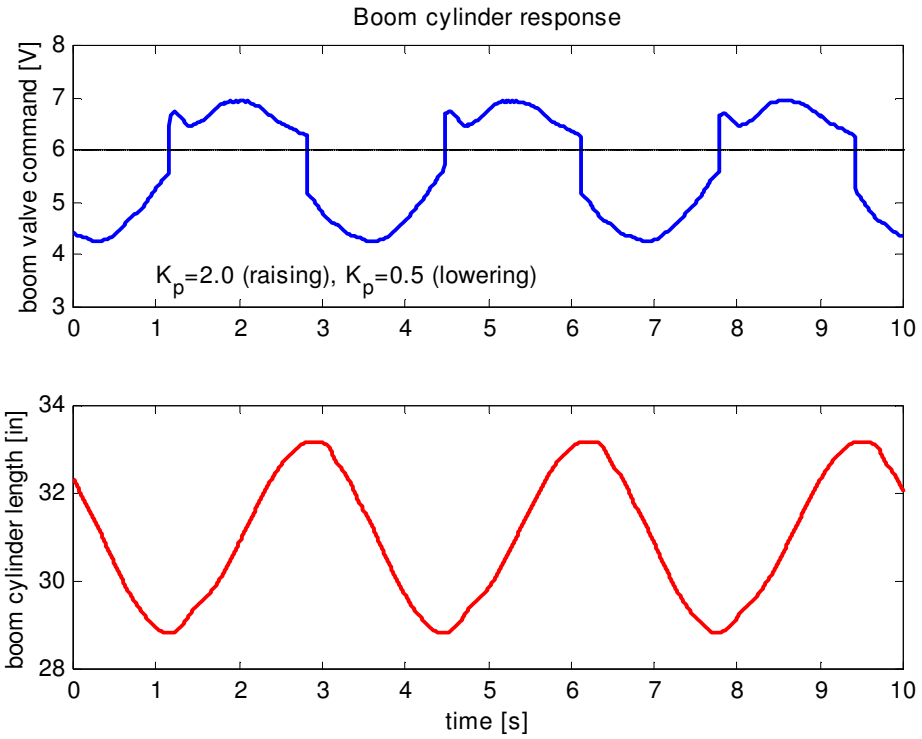


Figure 60: Boom cylinder response with feedforward control

Efforts to remove vibrations in the swing cylinders with feedforward control were not as effective, however. After extensive tuning in both the feedforward and feedback controllers, no combination of gains was found that could significantly reduce the vibrations in the swing cylinders. Although more tuning efforts are certainly warranted, initial tests indicate that feedforward control may not be able to remove the vibrations in the swing joint.

In general, some vibration also existed in the stick and bucket cylinders, but to a much lesser degree. Feedback control with and without feedforward assistance could be used to achieve satisfactory tracking of a computer–generated reference without excessive vibration.

6.2 Haptic position control

After tuning the position controller gains with computer generated commands, the next step was to control the backhoe with the PHANToM. The Simulink software was modified to include vector transformations between all four coordinate systems (PHANToM Cartesian, backhoe Cartesian, joint, and cylinder) so that Cartesian position of the bucket tip, mapped into the PHANToM's workspace, was passed to and from the Ghost software. Forces were displayed to the users's hand proportional to the position error between the PHANToM and the tip of the bucket in the PHANToM's workspace, as illustrated in figure 15.

Data was collected as the PHANToM was moved in a digging–like motion in free space, which is shown in figure 61 for each of the four valves and cylinders. The dashed blue lines in the position plots are the reference commands sent from the PHANToM to the backhoe after being transformed into backhoe cylinder coordinates, and the solid red lines are the resulting output cylinder lengths. The corresponding voltage command plots are the signals produced by the controller.

As figure 61 illustrates, the backhoe is fairly well behaved with reasonably good tracking on the stick and bucket cylinders. However, the swing and boom cylinders do not track nearly as well, and exhibit excessive vibration when operated with the PHANToM. Although the work presented herein does not include a satisfactory solution to the vibration problems, at the time of this writing, eliminating vibration is the primary focus of current efforts.

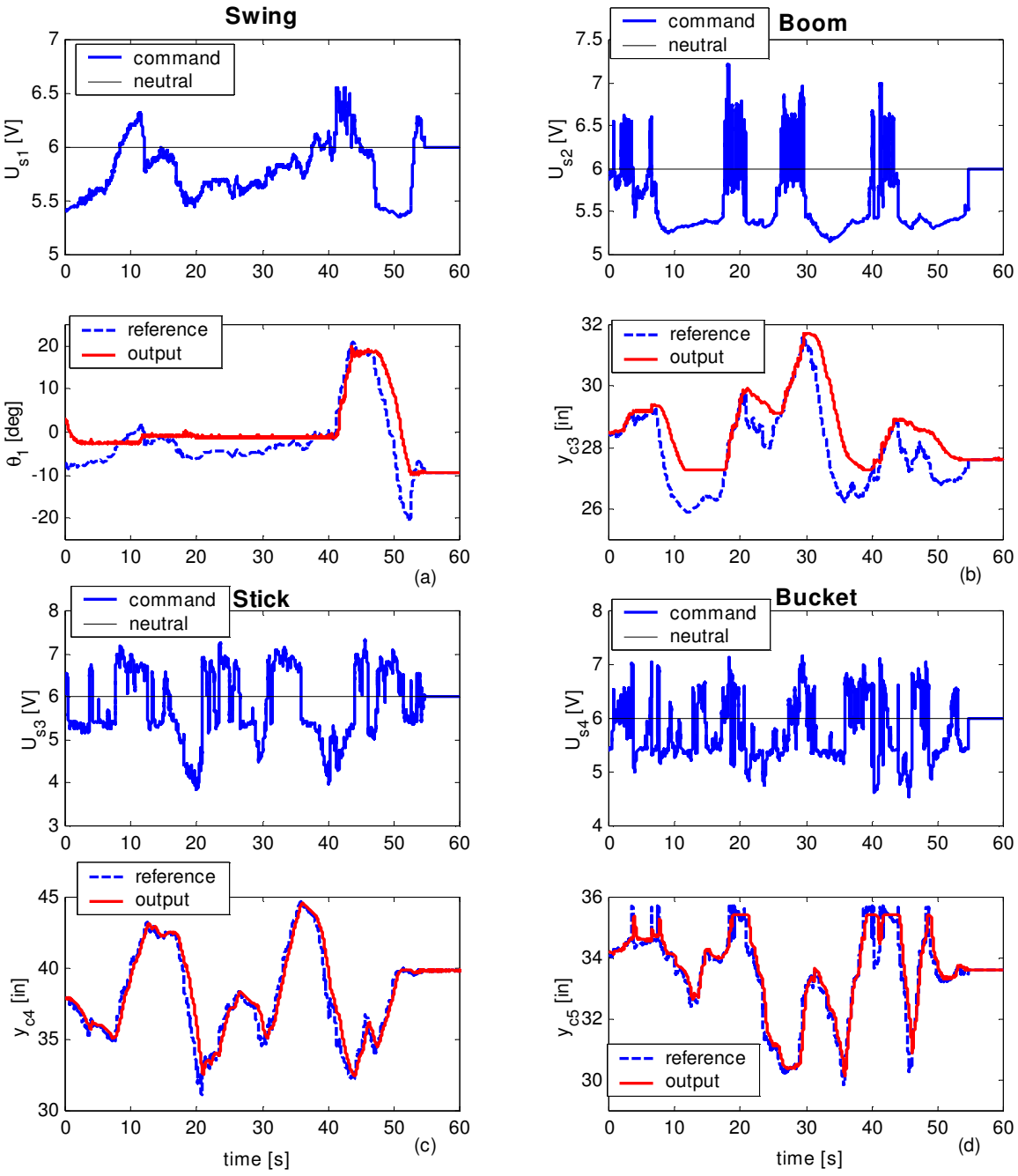


Figure 61: Haptic position control

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

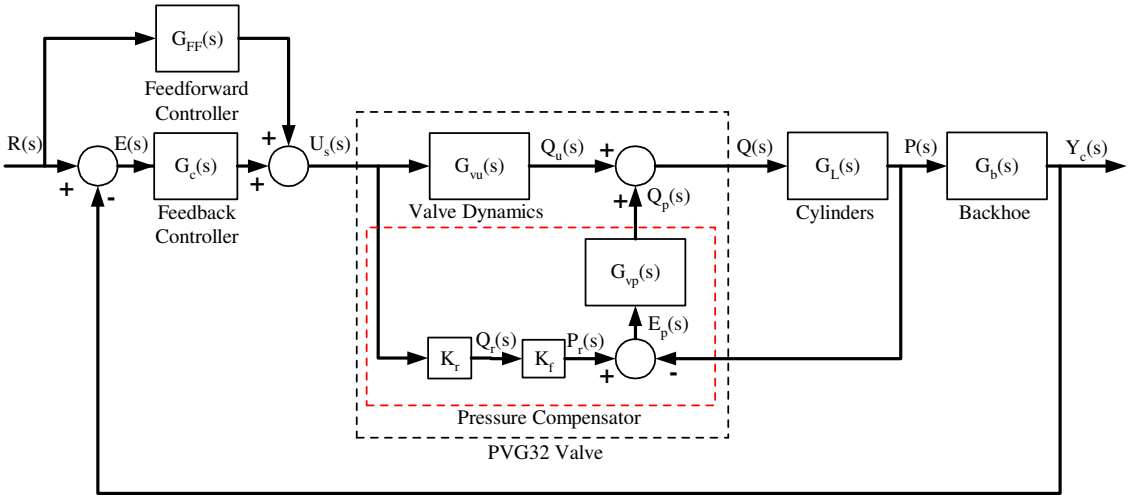
At this point in time, more work remains to be done before the backhoe can be considered a completed test platform for haptics–for–hydraulics control algorithms. Once complete, many options will exist for future research using this system. The following sections describe the suggested plan of action for future work on the backhoe.

7.1 Vibrations and Pressure Compensation

The undesireable vibrations described in section 6.1 need to be eliminated. Evidence gathered thus far points to a conflict between the pressure compensators in the PVG32 valves and the backhoe's position controller. A test needs to be contrived to verify that this is indeed the case, and if so, the current PVG32 valves should be replaced with versions *without* pressure compensation. Randy Bobbitt at Sauer Danfoss reports that the valves' external form factors are identical with and without pressure compensators, so the existing mounting plate and tube fittings should match up without any modifications. Also, if the valves are replaced, smaller flow control spools than size "C" should be considered, especially for the swing valve.

In order to better understand the effect of pressure compensation in the PVG32 valves, a hypothetical linear system model representing the backhoe is given in figure 62.

This model assumes that the flow leaving the valve $Q(s)$ is a linear combination of the flow from the uncompensated valve dynamics $Q_u(s)$ and the "adjustment" flow due to the pressure compensator $Q_p(s)$. The pressure compensation flow is assumed to be a function of the *pressure error*, which is the difference between the sensed load pressure $P(s)$ and the pressure required at any given command signal to maintain the linear relationship between flow and command, so that $Q(s) = K_r U_s(s)$. The relationship K_f between the reference flow $Q_r(s)$ and reference pressure $P_r(s)$ would most likely have to be determined



R(s) = reference cylinder position
E(s) = position error signal
$U_s(s)$ = valve command
$Q_u(s)$ = uncompensated oil flow
$Q_r(s)$ = reference flow @ $U_s(s)$
$P_r(s)$ = reference pressure @ $Q_r(s)$
$E_p(s)$ = pressure error signal
$Q_p(s)$ = compensation flow
Q(s) = total flow to load
P(s) = cylinder pressure
$Y_c(s)$ = cylinder position

Figure 62: Hypothetical linear backhoe model with closed–loop control

experimentally by the valve manufacturer. Assuming this is a reasonable model, it can be shown with block reduction that the closed–loop transfer function of the overall system is

$$\frac{Y_c(s)}{R(s)} = \frac{(G_{FF} + G_c)\left[G_{vu} + K_r K_f G_{vp}\right] G_b G_L}{1 + G_{vp} G_L + \left[G_{vu} + K_r K_f G_{vp}\right] G_b G_L G_c}. \tag{135}$$

where the terms $G_{vp}(s)$ and $[G_{vu}(s) + K_r K_f G_{vp}(s)]$ represent the valve dynamics with pressure compensation, $G_b(s)$ is a linear model of the backhoe, $G_L(s)$ is a model of the cylinders and hoses, and $G_c(s)$ and $G_{FF}(s)$ are the feedback and feedforward controllers, respectively. Each transfer function G in equation (135) is a function of the derivative operator, $G = G(s)$.

The error function for a given reference input, with all terms included, is

$$\frac{E(s)}{R(s)} = 1 - \frac{Y_c(s)}{R(s)} = \frac{1 + G_{vp} G_L - \left[G_{vu} + K_r K_f G_{vp}\right] G_b G_L G_{FF}}{1 + G_{vp} G_L + \left[G_{vu} + K_r K_f G_{vp}\right] G_b G_L G_c}, \tag{136}$$

which should be made equal to zero for ideal performance. Equation (136) indicates that the errors may be minimized or eliminated with the proper choice of $G_{FF}(s)$, if all the other terms in (136) are known, by setting the numerator equal to zero and solving for $G_{FF}(s)$. In practice this may be difficult due to the need for an accurate plant model.

However if a feedforward controller is to be designed based upon a plant model, the valve should be returned to the HIL simulator, connected to a rotary motor, flow meter, and pressure sensors, and tested by varying the load pressure to determine the relationship $Q(s)/P(s)$ for constant values of U_s. It should also be verified that $G_{vu}(s)$ is accurately represented by equation (123) when combined with $G_{vp}(s)$ as shown in figure 62.

If a feedforward controller is not used, $G_{FF}(s) = 0$ and equation (136) becomes

$$\frac{E(s)}{R(s)} = \frac{1 + G_{vp} G_L}{1 + G_{vp} G_L + \left[G_{vu} + K_r K_f G_{vp}\right] G_b G_L G_c}, \tag{137}$$

which can only be made equal to zero by setting $G_c(s) = \infty$. Obviously this is not possible

due to the limitations on the command voltage U_s and the bandwidth of the system. Also, any lightly damped, low frequency poles in $G_{vp}(s)$ could be the source of vibrations and marginal stability observed in the real system.

It is instructive to investigate the form of the system dynamics without pressure compensation where $G_{vp}(s) = 0$. If the pressure compensator is removed, but a feedforward controller is still used, the error function becomes

$$\frac{E(s)}{R(s)} = \frac{1 - G_{vu}G_bG_LG_{FF}}{1 + G_{vu}G_bG_LG_c} \tag{138}$$

Once again, the error function can be minimized by setting $G_{FF} = 1/(G_{vu}G_bG_L)$. This should not be a problem since the valve, cylinder, and backhoe are all minimum phase systems.

Finally, if neither pressure compensation nor feedforward control are used, $G_{vp}(s) = G_{FF}(s) = 0$, and

$$\frac{E(s)}{R(s)} = \frac{1}{1 + G_{vu}G_bG_LG_c}, \tag{139}$$

in which case the closed–loop transfer function becomes

$$\frac{Y_c(s)}{R(s)} = \frac{G_{vu}G_bG_LG_c}{1 + G_{vu}G_bG_LG_c}. \tag{140}$$

Clearly, if the hypothetical model above is a reasonable representation of the backhoe, the pressure compensator in the valve introduces additional complexity and uncertainty into the system, making controller design and achieving transparency much more difficult than it would be without pressure compensation.

7.2 End-Effector Position

Early on in the development of the haptic backhoe, a conscious decision was made to map the origin of the PHANToM O_p^p, which is located at the base of the stylus in the

PHANToM workspace at initialization, to the *tip* of the bucket in the backhoe workspace O_4^b. This workspace transformation is described in section 3.2.5. The result is that an operator feels as if the bucket extends *behind* the base of the stylus. Any change in the stylus angle ϕ_p will produce a motion not only in the bucket cylinder, but also in the boom and stick cylinders because the tip of the bucket is trying to rotate about a fixed point in the backhoe's workspace.

However, it may make more sense to map the base of the stylus to the wrist of the backhoe at O_3^b. Kinematically, this situation may be more intuitive for a user because it would feel as if the bucket extends *in front* of the base of the stylus. In this case, changing the stylus angle would only produce a motion in the bucket cylinder as the bucket tries to rotate about a fixed at the wrist, which is a much less demanding reference command. Changing the end-effector point would require reworking the 'T2J' and 'J2T' blocks shown in figure 56.

It may also make more sense to reorient the PHANToM so that two main horizontal links point *away* from the operator and the stylus is curled back toward the operator, so that the angles of the PHANToM's links approximately correspond to the angles of the backhoe's links. Although this may inhibit some of the user's range of motion with the PHANToM, it may serve to further improve the intuitive feel of controlling the backhoe with the PHANToM.

7.3 Friction Estimation

Many haptic control schemes will need to estimate the forces between the bucket and the environment, and force estimation will be a requirement for haptic transparency. A key element in any force estimation will include an estimation of the friction in the cylinders, because the magnitude of the force applied to the environment will be less than the amount predicted by simply multiplying the differential cylinder pressures by their respective areas and then mapping the resulting forces to the tip of the bucket. The difference will result from friction.

During the rebuild, it was noticed that very little Coulomb friction existed in the custom

cylinders that were retrofitted to the backhoe. Forces on the order of 10 lbs. or less were sufficient to slide the rods inside the cylinders when the oil ports were open to the atmosphere. However, it is expected that during low–speed, high–pressure operation, back pressure due to stiction in the seals will be present, and that during high–speed operation, viscous forces in the oil will be present. The significance of friction in the cylinders is identified in [32], although no distinction is made between Coulomb and viscous types. A stiction force on the order of 6,000N is reported in [20], but this is on much larger forestry machines.

Presently, pressure sensors have been installed but not yet tested. These sensors will provide continuous feedback of the pressures inside the cap and rod of each cylinder, plus the gallery pressure in the valve body—a total of nine signals. Once the pressure sensors are functioning correctly, they can be used to estimate friction parameters in the backhoe using the techniques described in [38].

7.4 Real-Time Force Estimation

Once the pressure sensors are functioning and the friction parameters have been identified, it will be possible to estimate the forces exerted on the environment by the bucket.

First, the rod forces applied to the backhoe's links can be computed from the position and pressure sensor signals:

$$\mathbf{f}_r = \mathbf{A}_c\mathbf{p}_c - \mathbf{A}_r\mathbf{p}_r - \mathbf{b}_r(\dot{\mathbf{y}}_c)\dot{\mathbf{y}}_c - \mathbf{m}_r\ddot{\mathbf{y}}_c \tag{141}$$

where the friction coefficients in the cylinders $\mathbf{b}_r(\dot{\mathbf{y}}_c)$ have been computed from cylinder pressures and knowledge of the friction parameters, and the rod velocities $\dot{\mathbf{y}}_c$ have been computed from numerical differentiation after digital filtering. During any digging operation, the backhoe will be moving very slowly, and the rod accelerations $\ddot{\mathbf{y}}_c$ will be small and should be neglected.

Second, the joint angles can be computed using the cylinder–to–joint transformation described in section 3.2.1:

$$\theta = C2J(\mathbf{y_c}). \tag{142}$$

Note that the joint angles are already computed in the existing position controller and are therefore available for use in other real–time functions.

Third, the torque about the joints due to cylinder pressures can be computed from the actuator force to joint torque algorithm described in section 3.3.1:

$$\tau_{\mathrm{r}} = fr\_to\_taur\left(\mathbf{f}_r, \theta\right) \tag{143}$$

Both the actuator forces on the links $\mathbf{f_r}$ and the wrench from the soil-bucket interaction $\mathbf{w_e} = \mathbf{J^{-T}}\tau_{\mathbf{e}}$ at the bucket tip contribute to the net torque about the joints. By expressing equation (118) in terms of the two components of torque, the backhoe dynamic equation becomes

$$\mathbf{M}\left(\theta\right)\ddot{\theta} + \mathbf{V}\left(\theta, \dot{\theta}\right) + \mathbf{G}\left(\theta\right) = \tau_r + \mathbf{J}^T\mathbf{w}_e. \tag{144}$$

Note that the geometric Jacobian matrix is also a function of the joint angles, $\mathbf{J} = \mathbf{J}(\theta)$.

Solving (144) for the end-effector wrench, the bucket–environment forces are

$$\mathbf{w}_e = \mathbf{J^{-T}}[\mathbf{M}\left(\theta\right)\ddot{\theta} + \mathbf{V}\left(\theta, \dot{\theta}\right) + \mathbf{G}\left(\theta\right) - \tau_{\mathbf{r}}]. \tag{145}$$

If the velocities and accelerations are small, inertial and velocity forces are small, and the wrench exerted on the bucket by the soil is simply due to gravity and static cylinder pressure. In that case equation (145) becomes

$$\mathbf{w}_e = \mathbf{J}^{-T}[\mathbf{G}\left(\theta\right) - \tau_r]. \tag{146}$$

Thus, a real–time force estimator can be implemented using the functions *C2J* and *fr_to_taur*, equations (141) and (145), the geometric Jacobian $\mathbf{J}(\theta)$, and the gravitational vector $G(\theta)$, which is listed in Appendix F.

In the case that velocities are not small, all of the terms in equations (141) and (144) should be included, at the expense of significantly increasing computational overhead in the controller. This would be a good topic for further investigation.

7.5 Control Algorithms

The sections below identify the most common haptic control algorithms found in the literature. Once the force estimation routine has been developed, any of the following algorithms

will be possible to implement on the backhoe, and may be used as starting points for future research.

7.5.1 Position/Position Control

Position/position control is the simplest haptic control law, where the PHANToM and backhoe are connected with a virtual spring. The reference position for the backhoe is the position of the PHANToM mapped into the backhoe's workspace, and the force displayed by the PHANToM is proportional to the position error in the PHANToM's workspace:

$$\mathbf{p}^0_{b,ref} = \mathbf{T}^0_p \mathbf{p}^p_p \tag{147}$$

$$\mathbf{f}^p_h = k_p \left(\mathbf{p}^p_b - \mathbf{p}^p_p \right) \tag{148}$$

where $\mathbf{T^0_p}$ is the workspace mapping transform given by the inverse of equation (48)and $\mathbf{k_h}$ is the virtual spring rate. This is the algorithm employed for the initial setup described in section 6.2. Note that position/position control does not require bucket–enviroment force estimation.

7.5.2 Position/Rate Control

Position/rate control maps the position of the PHANToM to a velocity command for the backhoe, and a haptic force is reflected that is proportional to the difference between the velocity of the backhoe and the position of the PHANToM:

$$\mathbf{p}^0_{b,ref} = k_{vb} \mathbf{R}^0_p \int \mathbf{p}^p_p dt \tag{149}$$

$$\mathbf{f}^p_h = k_{vh} \left(\frac{d}{dt} \mathbf{p}^p_b - k_{vp} \mathbf{p}^p_p \right) \tag{150}$$

where k_{vb} scales the position of the PHANToM to a velocity reference for the backhoe, and k_{vh} and k_{vp} account for force scaling and unit conversions, respectively. Position/rate control is the most common type of joystick control, and is typically used in situations where the workspace dimensions of the master and slave differ by a large amount. Note that position/rate control also does not require bucket–enviroment force estimation.

7.5.3 Position/Force Control

Position/force control generates a reference force for the backhoe to exert on the environment based on the displacement of the PHANToM from an initial reference position $\mathbf{p}^{\mathbf{p}}_{\mathbf{e_0}}$. The haptic force displayed to the user is a scaled representation of the bucket–environment force, less the initial force due to gravity that is present before the bucket comes in contact with the environment $\mathbf{f}^{\mathbf{0}}_{\mathbf{e_0}}$.

$$\mathbf{f}^0_{e,ref} = k_{fb}\left(\mathbf{p}^p_p - \mathbf{p}^p_{e_0}\right) \tag{151}$$

$$\mathbf{f}^p_h = k_{fh}\mathbf{R}^p_0\left(\mathbf{f}^0_e - \mathbf{f}^0_{e_0}\right) \tag{152}$$

where k_{fb} scales the position of the PHANToM into a desired force on the environment, and k_{fh} reduces the measured force on the environment before displaying it to the hand of the operator.

The presence of the terms $\mathbf{p}^{\mathbf{p}}_{\mathbf{e_0}}$ and $\mathbf{f}^{\mathbf{0}}_{\mathbf{e_0}}$ in equations (151) and (152) imply that a reference position and force must be indentified, stored, and reset during program execution. This is because a haptic controller running in position/force mode must either switch or transition from another mode when the bucket comes into contact with the environment. Otherwise, the high gain term k_{fb} commanding the bucket to exert a force on a zero impedance environment will result in instability.

For operating the backhoe in force control mode, equation (146) could be used to determine the cylinder pressure required to produce a given force on the environment, where the reference torques at the joints are

$$\tau_{r,ref} = \mathbf{G}(\theta) - \mathbf{J}^T\mathbf{w}_{e,ref} \tag{153}$$

and the end–effector wrench $\mathbf{w}_{e,ref} = \begin{bmatrix} \mathbf{f}^0_{e,ref} \\ 0^{(3\times1)} \end{bmatrix}$ has no torque terms because the bucket cannot be commanded to exert torques onto the environment. The pressures required are found from inverting the function represented in equation (143) and solving for pressures in equation (141). In that case, assuming the joint and cylinder velocities and accelerations are very small, the relationship between the joint angles, the desired torque at the joints,

and the cylinder pressures can be approximated using

$$\mathbf{A}_c\mathbf{p}_c - \mathbf{A}_r\mathbf{p}_r = taur\_to\_fr\left(\theta, \tau_{r,ref}\right) \tag{154}$$

where the function *taur_to_fr* is the inverse of the algorithm described in section 3.3.1. Thus by representing equations (153) and (154) in real–time code, reference cylinder pressures could be generated for the backhoe controller to regulate a desired force onto the environment.

7.5.4 Impedance Control

Impedance control seeks to create a dynamic relationship between position tracking error and the *force error*, where the force error is defined by referencing the measured end–efector force from a reference force. The reference force is computed by defining the *target impedance* $\mathbf{Z_t(s)}$, so that

$$\left(\mathbf{f}^0_{e,ref} - \mathbf{f}^0_e\right) = \mathbf{Z}_t\left(s\right)\left(\mathbf{p}^0_p - \mathbf{p}^0_b\right), \tag{155}$$

where the target impedance contains a set of constant matrices

$$\mathbf{Z}_t\left(s\right) = \mathbf{M}_t s^2 + \mathbf{B}_t s + \mathbf{K}_t, \tag{156}$$

and the variable s is the derivative operator. The matrices $\mathbf{M_t}$, $\mathbf{B}_t$, and $\mathbf{K}_t$ are design variables that determine the transient dynamics, and asmyptotically drive the position error to zero by adjusting the force applied to the environment.

Combined with haptic force display based on position error, the haptic control laws for impedance control are

$$\mathbf{f}^0_{e,ref} = \mathbf{Z}_t\left(s\right)\left(\mathbf{p}^0_p - \mathbf{p}^0_b\right) + \mathbf{f}_e \tag{157}$$

$$\mathbf{f}^p_h = k_p\left(\mathbf{p}^p_b - \mathbf{p}^p_p\right) \tag{158}$$

where the end–effector force $\mathbf{f_e}$ is exerted by the bucket on the environment, which can be measured using the techniques described in section 7.5.3 above. Thus by adjusting cylinder pressures based on position errors, forces exerted on the environment can be adjusted automatically. This technique has been presneted in [11] where it is shown to work well.

7.5.5 H_∞–Optimal Control

The basic idea of H_∞–optimal control is to minimize a transfer function that relates *transparency* errors to the input signals. The standard H_∞ problem is illustrated in figure 63.

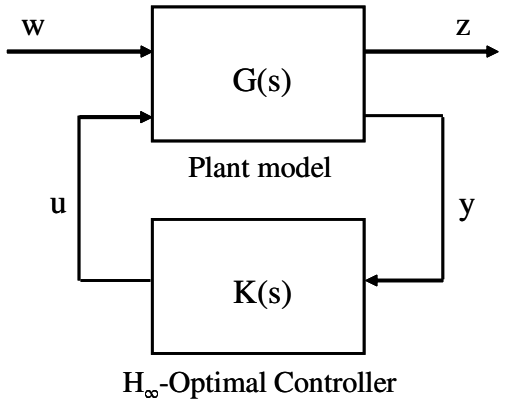


Figure 63: Standard H_∞ problem

In figure 63, $\mathbf{G}(\mathbf{s})$ represents a linear model of the plant, $\mathbf{K}(\mathbf{s})$ is the controller, $\mathbf{y}$ is the vector of measured ouputs, $\mathbf{u}$ is the vector controller commands, $\mathbf{w}$ is the vector of all exogenous inputs, including reference commands, noise, and disturbances, and the vector $\mathbf{z}$ contains the position and force tracking errors. Applied to the backhoe, $\mathbf{G}(\mathbf{s})$ would contain linear models of the valves, cylinders, and backhoe, $\mathbf{y}$ would contain the position and pressure sensor signals, $\mathbf{u}$ would contain the valve command voltages, and $\mathbf{w}$ would contain the reference signals coming from the PHANToM. The vector $\mathbf{z}$ would contain the position and force tracking errors, where

$$\mathbf{z}= \begin{bmatrix} W_1 \left(\mathbf{p}_p^0 - \mathbf{p}_b^0 \right) \\ W_2 \left(\mathbf{f}_{e,ref}^0 - \mathbf{f}_e^0 \right) \end{bmatrix}. \tag{159}$$

The terms W_1 and W_2 are weighting factors that determine the relative emphasis of the two types of errors. The objective is then to find the controller $\mathbf{K}(\mathbf{s})$ that minimizes the tracking errors $\mathbf{z}$.

If the plant model $\mathbf{G}(\mathbf{s})$ is partitioned appropriately, the block diagram shown in figure

63 can be represented as

$$\begin{bmatrix} \mathbf{z} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{G}_{11}(s) & \mathbf{G}_{12}(s) \\ \mathbf{G}_{21}(s) & \mathbf{G}_{22}(s) \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{u} \end{bmatrix}. \tag{160}$$

Rearranging equation (160), it can be shown that the relationship between $\mathbf{z}$ and $\mathbf{w}$ is

$$\frac{\mathbf{z}(s)}{\mathbf{w}(s)} = \mathbf{G}_{11} + \mathbf{G}_{12}\mathbf{K}(\mathbf{I} - \mathbf{G}_{22}\mathbf{K})^{-1}\mathbf{G}_{21}. \tag{161}$$

If a controller $\mathbf{K}(\mathbf{s})$ can be found such that equation (160) is zero over all operating frequencies, perfect transparency will be achieved. This may or may not be possible depending on the form of $\mathbf{G}(\mathbf{s})$, and may have multiple solutions. This technique has been presented in [42] and shown to work well on a single–dof, highly linear system.

Since the backhoe is not a linear system, modeling errors will always exist and perfect transparency will not be possible. However, if system identification is first performed on the backhoe over all possible joint angles, and then a linear model is generated in real–time based on that information, it is conceivable that the model could be used as an input the to optimal controller. As mentioned previously in section 2, it is the author's opinion that H_∞–optimal control shows the greatest promise for providing the operator a sense of transparency when operating the haptic backhoe. Unfortunately, this approach would require extensive testing and modeling, and is beyond the scope of the present work.

7.6 Obstacle Detection and Avoidance

When the time comes that the backhoe can be controlled such that reasonable transparency is achieved between the joystick and the backhoe, the most valuable testing for industry sponsors will be in the area of obstacle detection and avoidance. The ability to detect underground obstacles— such as water, gas, and power lines—and avoid damage to them will be a significant technological advancement for the construction industry, and will undoubtedly generate substantial interest and support from John Deere. Communications with John Deere representatives thus far have indicated this to be their primary interest in the project. Therefore, it is recommended that in the future, obstacle detection and avoidance be held as the primary goal and purpose of the haptic backhoe project.

7.7 Performance Gains from Haptic Feedback Control

The second most important goal of the project should be to evaluate the performance gains that can be achieved with haptic control. This includes a quantified measure of the reduction in operator training time and a measure of improvements to digging accuracy. Tests can be envisioned in which a novice operator is asked to dig three holes, one using the original manual valves, one with joystick control only, and one with haptic feedback. The resulting time to execute the task and the variation from the specified hole dimensions would be used as measures of performance.

7.8 Joystick Improvements

A significant drawback to the present system is the limited range of forces that the PHANToM is capable of displaying to the operator. Since the PHANToM is a delicate, high–resolution device designed to be used in the laboratory as an tactile interface with virtual models, it was not designed for the higher forces typically involved in operating heavy–duty hydraulic machinery. Research of the currently available haptic interfaces indicates that no such suitable device has yet been developed and made available on the open market. Therefore, it may be worthwhile to develop a haptic joystick in parallel with the backhoe that is more rugged than the PHANToM, and better suited for operating a backhoe. A tradeoff in force and position resolution for the sake of greater strength and durability than the PHANToM would certainly be acceptable.

7.9 Effects of Valve Bandwidth

One final area of interest to John Deere is the effect of using low–cost, low–performance valves in a haptic/hydraulic control system. If the haptic backhoe concept is ever to be made a viable financial option for mass production, the lowest cost valves will need to be used that can still produce acceptable performance. Although a high–performance, high–cost prototype may best exemplify the benefits of haptic controls, achieving the same results with low–cost components will be far more valuable in industry.

To explore the effect of valve bandwidth on overall performance, either the real system

or the dynamic model presented in chapter 3 could be used to simulate another valve with different dynamics. Of course in the real system, the PVG32 has a maximum response time that cannot be exceeded. It could be made slower, however. On the other hand, if work continues with identifying the valve's response to pressure changes and the model is improved, simulations could be run to investigate the effect of changes to the valve dynamics on the overall response. The outcome of such tests and/or simulations may serve to establish minimum valve specifications for haptics–for–hydraulics control systems.

APPENDIX A

RESOURCE DIRECTORY

This section is a listing technical information and contacts for the components of the haptic backhoe. Every effort has been made to collect and store all pertinent information on the IMDL server at

http://www.imdl.gatech.edu/jfrankel/spec_sheets.htm

However, if this page is unavailable or some information is missing, the following information may be helpful.

TRACTOR & BACKHOE
John Deere
Mark Evans
706-854-3521
EvansMark@JohnDeere.com

Documents: *Tractor Manual, Attachment Manual, Parts Manual*
The tractor and attachment manuals are available in printed form in the IMDL library. All three documents are also available in portable document format (pdf) on CD-ROM.

PVG32 VALVES
Sauer-Danfoss

Doug Hedrick	Randy Bobbitt
864-269-1569	864-644-3049
DHedrick@sauer-danfoss.com	RBobbitt@sauer-danfoss.com

Documents: *PVG32 Valves: Technical Information (pdf)*
This documents contains comprehensive information on the PVG32 valves. It can be downloaded from the Sauer-Danfoss website at *http://www.sauer-danfoss.com*.

POSITION SENSORS
Balluff

Tim Jordan	Jim Birkmeyer	Mike Bouts
800-543-8390	352-409-1635	800-543-8390 Ext. 3261
Tim.Jordan@balluff.com	jim.birkmeyer@balluff.com	mbouts@balluff.com

Documents: *BTL-E Embedded Rod Style Specifications (pdf)*
This documents contains specifications on the BTL-E micropulse position sensors installed in the cylinders. It can be downloaded from the Balluff website at *http://www.balluff.com*.

CYLINDERS
Georgia Hydraulic Cylinder

Jim Raspberry	Bill Styer
770-949-3299	770-949-3299
	bill@gahyd.com

Georgia Hydraulic Cylinders custom–fabricated the cylinders with integrated BTL-E position sensors. Website: *http://www.gahyd.com/*.

PHANToM
Sensable Technologies

Documents: *GHOST SDK API Reference, Version 4.0 (pdf)*
This documents contains information on the Ghost C libraries. It can be downloaded from the Sensable website at *http://www.sensable.com/*.

MANIFOLD
Daman Manifolds

Daman is a supplier of aluminum and steel hydraulic manifolds. Daman donated the manifold mounted to the tractor used for mounting the pressure transducers. Website: *http://www.damanifolds.com/*.

PRESSURE SENSORS
WIKA

Documents: *OEM Pressure Transmitters – Type C-10 (pdf)*
Based in Germany, Wika manufactured and donated the C-10 pressure transmitters. Website: *http://www.ewika.com/WikaSite/GlobalSites.aspx*.

MANUAL VALVE/FILTERS
HYDAC

Documents: *Ball Valve Design, Features, & Options (pdf)*
Based in Germany, HYDAC supplied the 3-way manual select valve and flow meter, as well as supplied the IMDL with hydraulic filters and hydraulic fitting and tubing clamps. Website: *http://www.hydacusa.com/*.

HYDRAULIC FITTINGS
Brennan Industries
Tim Merckens
770-981-8451

Brennan Industries is a supplier of hydraulic fittings, including the SAE O-ring style, JIC style, and NPT style used on the backhoe. Brennan has an excellent website for specifying fittings when designing hydraulic systems at *http://www.brennaninc.com*.

A/D & D/A CARDS
Measurement Computing, Keithley Metrabyte

Documents: *CIO-DDA06 Analog Output Board, CIO-DAS1602/12 ANALOG & DIGITAL I/O BOARD for ISA BUS (from Measurement Computing), Keithley DAS-1600/1400 Series User's Guide (from Keithley Metrabyte) (pdf)*

The DAS-1602 A/D card originally manufactured by Keithley Metrabyte has been discontinued, but an identcal clone is available from Mesaurement Computing, who also supplies the DDA-06 D/A card. Websites: *http://www.measurementcomputing.com/*, *http://www.metrabyte.com/main.jsp*.

ELECTRONIC PARTS

Allied Electronics

Nearly all the electrical connectors and wiring was purchased from Allied Electronics. Website: *http://www.alliedelec.com/*.

MISCELLANEOUS PARTS

McMaster-Carr

A large amount of miscellaneous parts and pieces came from McMaster-Carr. Generally speaking, anything not obviously from another source cam from McMaster. Website: *http://www.mcmaster.com/*.

VALVE DEALER/SERVICE

Berendsen Fluid Power

Jim Trayler	Steve Shock
770-419-3430	770-218-7533

Documents: *Series V20 Directional Control Valves*
Berendsen Fluid Power is a distributor and servicer of Sauer-Danfoss valves. Berendsen assembled the PVG32 four-valve assembly at their testing facility in Marietta, GA before delivery to Georgia Tech. Website: *http://www.bfpna.com*.

PUMPS/VALVES/FITTINGS

Orton Industries (Parker-Hannifin)

Don Robasse
770-986-9999
drobasse@attbi.com

Documents: *Series V20 Directional Control Valves*
Parker is a large distributor of hydraulic pumps, valves, and fittings. Although no parts were purchased from Parker for the backhoe, they are currently the distributor of the original valves installed in the backhoe, the Gresen V20. Website: *http://www.parker.com*.

APPENDIX B

SENSOR CALIBRATION DATA

The figures below contain the position sensor calibration data taken to predict the backhoe position from measured sensor voltages. For the swing angle, voltage has been correlated with the angle θ_1 in radians, measured counterclockwise from $\mathbf{x_0}$ about $\mathbf{z_0}$. (See figure 8). The measurements $\mathbf{y_b}$ in figure 64 are in the $\mathbf{y_0}$ direction, and the angle has been computed from the Pythagorean theorem using 98in as the hypotenuse. For the other three sensors—boom, stick, and bucket—voltage has been correlated with cylinder length between the centers of the pin joints at each end of the respective cylinder. The coefficients of the 2^{nd} order curve fit equations are used in Simulink control software inside the *Backhoe/Position sensor conversions* block.

Swing Sensor Calibration

Tip position [in]		Voltage [V]	Angle [rad]	Angle [deg]
x_b	y_b	V_{c1}	θ_1	θ_1
98	0	5.67	0.00	0.00
na	96	2.41	1.37	78.40
	82	3.29	0.99	56.80
	65	3.91	0.73	41.55
	48.5	4.41	0.52	29.66
	25	5.07	0.26	14.78
	-28	6.39	-0.29	-16.60
	-54.5	7.14	-0.59	-33.79
	-77	7.88	-0.90	-51.79
	-91	8.56	-1.19	-68.21
	-97	9.04	-1.43	-81.81

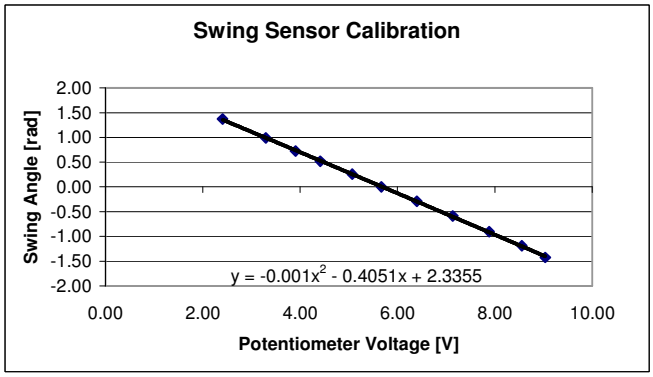


Figure 64: Swing potentiomter calibration data

Boom Sensor Calibration

Cylinder Length [in] y_{c3}	Voltage [V] V_{c1}
38.5	7.17
34	5.06
31	3.73
28.25	2.44

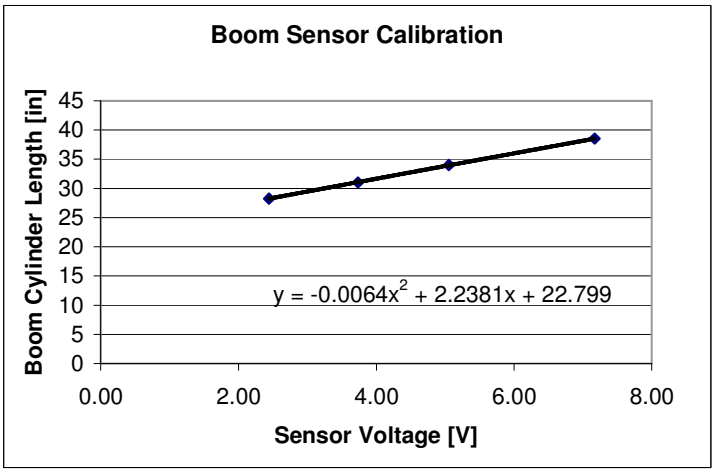


Figure 65: Boom sensor calibration data

Stick Sensor Calibration

Cylinder Length [in] y_{c3}	Voltage [V] V_{c1}
27.25	1.99
38.75	7.25
44.625	10.00

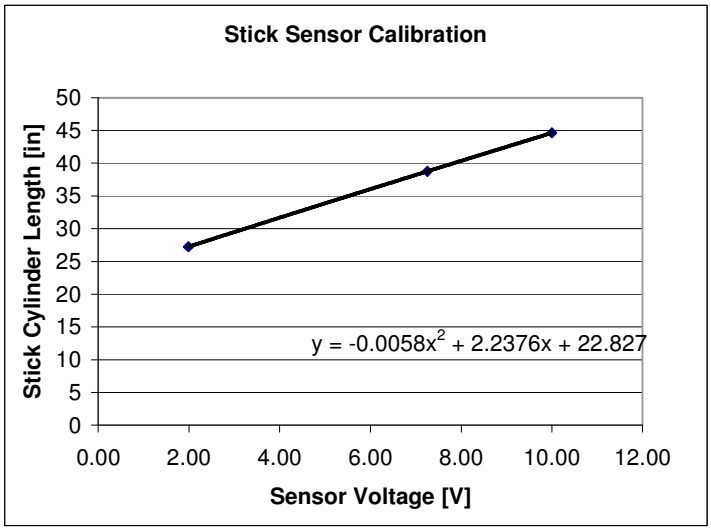


Figure 66: Stick sensor calibration data

Bucket Sensor Calibration

Cylinder Length [in] y_{c3}	Voltage [V] V_{c1}
23.375	2.00
31.75	7.54
23.75	2.22
35.375	9.97

Note: the sensor hits deadband before the cylinder extends all the way

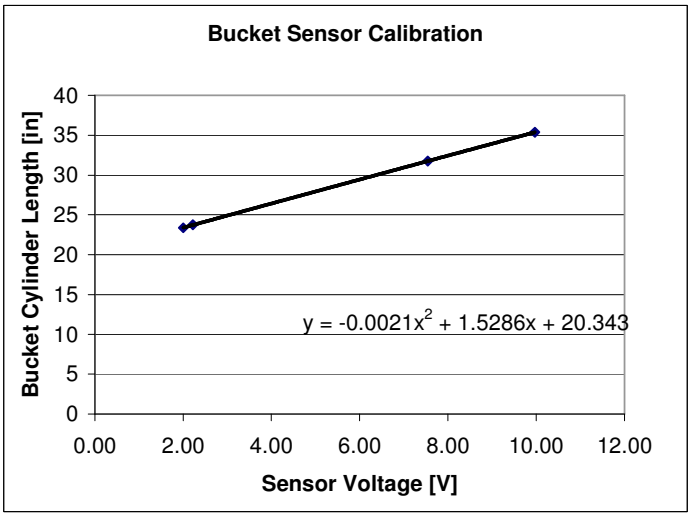


Figure 67: Bucket sensor calibration data

APPENDIX C

GHOST SOFTWARE

The following code was adapted by Matt Kontz to run the Phantom and interface with the Simulink model running the backhoe during the initial tests described in section 6. This code was adapted from the software written to control the IMDL's Hydraulically Actuated Lift (HAL) described in [17].

The main C++ file is called 'BhPhan.cpp', which calls the four header files "BhPhan.h", "DataStorage.h", "DataStruct.h", and "Sock.h" listed below.

C.1 Main source file: BhPhan.cpp

```
//*****************************************************************************
// Filename : BhPhan.cpp
// ---------------------------------------------------------------------------
// This file is and edited version of Main.cpp which is part of an
// extending effects example file provided with the Ghost software.
//
// Author: Matt Kontz <mkontz@mail.com>
// Lab: IMDL ME GaTech
// Created: March 18, 2004
// Edited: na
//
//*****************************************************************************
//      Based on the file : main.cpp
// ---------------------------------------------------------------------------
// This program demonstrates how to extend the gstEffect class
// by creating a viscosity effect.
//
// SensAble Technologies, Inc. Copyright  1999
// All rights reserved.
//*****************************************************************************

#include <gstScene.h>
#include <gstPHANToM.h>
#include <iostream.h>// for cout, cerr
#include <iomanip.h>// for setw, setprecision
#include <fstream.h>// for writing to files
#include <windows.h>// WIN32 Threads
```

```
#include "BhPhan.h" // local header file
#include "DataStruct.h" // Phantom, and Hal data structures
#include "DataStorage.h" // local storage
#include "Sock.h" // udpSocket clas

HANDLE  hThreadB2P, hThreadP2B, hLockHal;

void B2P_Thread(void *);
void P2B_Thread(void *);

int main()
{
/*************************************************************************
Inintialize Socket Classes
*************************************************************************/
// Feynman IP = "128.61.140.140"
// Phantom IP = "128.61.140.190"
// Euler IP = "128.61.140.235" (i.e. HAL)
// Phantom IP = "130.207.153.226" (loading dock)
// xpc-target IP = "130.207.153.241" (loading dock)
// crossover cable = "192.168.0.111"


char *forIP = "192.168.0.111"; // foreign IP
unsigned short forPort = 26401; // foreign Port
unsigned short localPort = 26401; // loal port
udpSocket *sockP2B = new udpSocket(forIP, forPort, localPort);

forPort = 23201; // change foreign Port
localPort = 23201; // change local Port
udpSocket *sockB2P = new udpSocket(forIP, forPort, localPort);

/*************************************************************************
Inintialize mutex for conflict with Hal data
*************************************************************************/
hLockHal = ::CreateMutex(NULL,FALSE,NULL);
/*************************************************************************
Inintialize DataStorage Class
*************************************************************************/
DataStorage *data = new DataStorage();
/*************************************************************************
Get initialization data from Backhoe
*************************************************************************/
cout << "Waiting for initial position from Backhoe" << endl;
PhanStruct Get_Pos = {0,0,0,0,0,3}; // 3 ques for Bh pos
PhanStruct Get_Org = {0,0,0,0,0,2}; // 2 ques for Bh origin
BhStruct Reply = {0,0,0,0,0,0}; // empty structure
```

```
// Get origin (zero frame) of backhoe
Reply.flag;
sockP2B->send((char *) &Get_Org, sizeof(Get_Org));
while (Reply.flag != 2)
sockB2P->recv((char *) &Reply, sizeof(Reply));
data->setOrigin(Reply);

cout << "Origin Received!" << endl;
cout << "x = " << Reply.x << ", y = " << Reply.y << ", z = "
<< Reply.z << endl;

// Get Initial Pos. of Backhoe
sockP2B->send((char *) &Get_Pos, sizeof(Get_Pos));
while (Reply.flag != 3)
sockB2P->recv((char *) &Reply, sizeof(Reply));
data->setBhData(Reply);

cout << "All riiighty THEN!" << endl;
cout << "x = " << Reply.x << ", y = " << Reply.y << ", z = "
<< Reply.z << endl;
/*************************************************************************
Threads Stuff
*************************************************************************/
DWORD ThreadID0; // Thread ID: B2P
DWORD ThreadID1; // Thread ID: P2B

// Create thread argrument structure
struct ThrArgs {
udpSocket *sock;
DataStorage *data;
};
/*************************************************************************
B2P Thread
*************************************************************************/
ThrArgs B2P_Args; // Declare thread argument
B2P_Args.data = data; // Load argument to pass to B2P Thread
B2P_Args.sock = sockB2P;

// Create B2P Thread
hThreadB2P = ::CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) B2P_Thread,
  (LPVOID) &B2P_Args,0,(LPDWORD) &ThreadID0);
::SetThreadPriority(B2P_Thread,15); // 0 = Normal, 31 = Maximum Priority
/*************************************************************************
P2B Thread
*************************************************************************/
ThrArgs P2B_Args; // Declare thread argument
P2B_Args.data = data; // Load argument to pass to B2P Thread
P2B_Args.sock = sockP2B;
```

```
// Create P2B Thread
hThreadP2B = ::CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) P2B_Thread,
  (LPVOID) &P2B_Args,0,(LPDWORD) &ThreadID1);
::SetThreadPriority(P2B_Thread,15); // 0 = Normal, 31 = Maximum Priority
/*****************************************************************************
Inintialize Phantom
*****************************************************************************/

// Create a GHOST scene object.
gstScene *scene = new gstScene;

// create the root separator and set it as the root of the scene graph
    gstSeparator *root = new gstSeparator();
    scene->setRoot(root);

    // prompt the user to place the PHANToM in the reset position
    cout << "Place the PHANToM in its reset position and press <ENTER>."
     << endl;
cin.get();

    // create a PHANToM instance and check to make sure it is valid
    gstPHANToM *myPhantom = new gstPHANToM("Default PHANToM");
    if (!myPhantom || !myPhantom->getValidConstruction()) {
        cerr << "Unable to initialize PHANTOM device." << endl;
        exit(-1);
    }

    // add the PHANToM object to the scene
    root->addChild(myPhantom);

/*****************************************************************************
Inintialize Effect
*****************************************************************************/

// Create the effect and add to PHANToM
    BhEffect *BiLat = new BhEffect(data, hLockHal, hThreadP2B);
    myPhantom->setEffect(BiLat);
    myPhantom->startEffect();

    // start the haptic simulation
    scene->startServoLoop();

/*****************************************************************************
Stop Servo Loop
*****************************************************************************/
    cout << "Press <ENTER> to quit" << endl;
    cin.get();
```

```
scene->stopServoLoop();
/*************************************************************************
Clean up
*************************************************************************/
// Destroy Thread
::TerminateThread(hThreadB2P,ThreadID0);
::TerminateThread(hThreadP2B,ThreadID1);

// flag = 0 tells Hal to stop control and turn of pump
PhanStruct stop = {0,0,0,0,0,0};
stop.flag = 0; // flag=0 -> stops backhoe
sockP2B->send((char *) &stop, sizeof(stop));
while (Reply.flag != 0)
{
sockB2P->recv((char *) &Reply, sizeof(Reply));
sockP2B->send((char *) &stop, sizeof(stop));
}

// Close down sockets
sockP2B->close();
sockB2P->close();

return 0;
}

/*************************************************************************
B2P_Thread
*************************************************************************/

void B2P_Thread(void *ar)
{
struct ThrArgs {
udpSocket *sock;
DataStorage *data;
};
int stop = 0, status, count=0;

ThrArgs Args;
Args = * (ThrArgs *) ar;
DataStorage *data = Args.data; // pointer to data
udpSocket *sockB2P = Args.sock; // pointer to sock
BhStruct msg;

while(stop == 0)
{
sockB2P->recv((char *) &msg, sizeof(msg));
```

```
/* if (msg.time % 1000 == 0)
{
cout << ", x = " << msg.x ;
cout << ", y = " << msg.y ;
cout << ", z = " << msg.z ;
cout << ", phi = " << msg.bucket;
cout << ", flag = " << msg.flag << endl;
} */

status = ::WaitForSingleObject(hLockHal,1); // should never wait
if(status == WAIT_OBJECT_0) // more than 1 ms
{
data->setBhData(msg);
::ReleaseMutex(hLockHal);
}
else
{
//cout << "Couldn't obtain mutex int B2P_Thread." << endl;
//exit(1);
}
count++;
}
}

/*************************************************************************
B2P_Thread
*************************************************************************/

void P2B_Thread(void *ar)
{
struct ThrArgs
{
udpSocket *sock;
DataStorage *data;
};

ThrArgs Args;
Args = * (ThrArgs *) ar;
DataStorage *data = Args.data; // pointer to data
udpSocket *sockP2B = Args.sock; // pointer to sock
PhanStruct msg;

while (1>0)
{
::SuspendThread(hThreadP2B);
msg = data->getPhanData(1);
sockP2B->send((char *) &msg, sizeof(msg));
}
```

```
}
```

C.2 Header file: .h

```
//**************************************************************************
// Filename : PhEffect.h
// ------------------------------------------------------------------------
//
// This file is and edited version of DualEffect.h.
//
// Author: Matt Kontz <mkontz@mail.com>
// Lab: IMDL ME GaTech
// Created: October 19, 2003
// Edited: na
//
// Note: z-towards user, y-up, x to the right
//
//**************************************************************************
//      Based on the file : ViscEffect.h and ViscEffect.cpp
// ------------------------------------------------------------------------
//
// SensAble Technologies, Inc. Coprhyight  1999
// All rights reserved.
//**************************************************************************

#ifndef BH_EFFECT_INCLUDE // if not defined '......'
#define BH_EFFECT_INCLUDE // defines '......' so only happens once.

#include <gstPHANToM.h>
#include <gstEffect.h>
#include <math.h>
#include <windows.h>
#include "DataStorage.h"
#include "DataStruct.h"

class  BhEffect :
public gstEffect
{
private:
DataStorage *data; // pointer to data structure
HANDLE hLockHal; // handle for mutex
HANDLE hThreadP2H; // hanlde for P2H thread
BhStruct Bh; // Backhoe data structure
PhanStruct Phan; // Phantom data structure
int status, Time; // return int for ::WaitForSingleObject
```

```
unsigned int Flag;
BhStruct origin; // x,y,z coordinates of Bh origin in Ph workspace
    double k_x; // spring constant in x
double k_y; // spring constant in y
double k_z; // spring constant in z, and virtual constraints
double xp,yp,zp; // Phantom Coordinates, Phan reference frame
double xb,yb,zb; // backhoe's Coordinates, Phan reference frame
double rpx,rpy,rpz; // Phantom's reference position
double bucket;
double ymin,ymax; // workspace limits y-axis
double Fx,Fy,Fz; // Componets of Force vector to display
double pi;
gstVector p,v,a,gimbal; // pos, vel, accel vectors, gimble angle
public:
//*****************************************************************************
// Constructors
//*****************************************************************************
BhEffect(DataStorage *d, HANDLE hLock, HANDLE hP2H)
//  passes the memory location of data
{
data = d;
hLockHal = hLock;
hThreadP2H=hP2H;
k_x = 0.2;
k_y = 0.2;
k_z = 0.2;
rpx = 0;
rpy = 0;
rpz = 0;
Flag = 3;
origin = data->getOrigin();
pi = 3.14159265;

status = ::WaitForSingleObject(hLockHal,10);
if(status == WAIT_OBJECT_0)
{
Bh = data->getBhData(1);
::ReleaseMutex(hLockHal); // Release MUTually EXclusion lock
}
else
{
//cout << "Couldn't obtain mutex int OpenEffect." << endl;
//exit(1); // stop program
}

}
//*****************************************************************************
// Calculate force for effect at each servo tick.
```

```
//*************************************************************************
    virtual gstVector calcEffectForce(void *phantom)
{
if (!active)
{
return gstVector(0,0,0); // No force if inactive
}
//*************************************************************************
// Retrieve Phantom Data
//*************************************************************************
gstPHANToM *PHANToM = (gstPHANToM *)phantom; // gets current dynamics
p = PHANToM->getPosition(); // stores position data
//s = PHANToM->getSylus();
gimbal = PHANToM->getGimbalAngles();
// Gimbal = theta, phi, twist
// theta is the angle between projection of the pin on to the x-z
// plane and the z-axis, positive anlge in the y direction
// phi angle between stylus and x-z plane, postive up or in y.
// twist is the stylus twist about the stylus' longitudal axis

/* if ( Time % 1000 == 0 )
cout << "Gimbal Angle = " << 180/pi*gimbal << endl; */

//v = PHANToM->getVelocity(); // stores velocity data
//a = PHANToM->getAccel(); // stores acceleration data
//*************************************************************************
// Get the most current data from Backhoe
//*************************************************************************
status = ::WaitForSingleObject(hLockHal,0);
if(status == WAIT_OBJECT_0)
{
Bh = data->getBhData(1);
::ReleaseMutex(hLockHal); // Release MUTually EXclusion lock
}
else
{
//cout << "Couldn't obtain mutex int OpenEffect." << endl;
//exit(1); // stop program
}
//*************************************************************************
// Convert data
//*************************************************************************
Time = data->getTime();
data->incTime();
xp = p[0]; // take Phantom pos. data out of vector form
yp = p[1];
zp = p[2];
xb = Bh.x;
```

```
yb = Bh.y;
zb = Bh.z;
bucket = -gimbal[1]; // or negative phi (phi defined above with gimbal)

//*************************************************************************
// Forces from impedance model - MODE[0]
//*************************************************************************
if (Flag == 3 )
{
rpx = rpx + 0.01*(xb-rpx);
rpy = rpy + 0.01*(yb-rpy);
rpz = rpz + 0.01*(zb-rpz);
if (abs(xb-rpx) < 1)
{
if (abs(yb-rpy) < 1)
{
if (abs(zb-rpz) < 1)
{
Flag = 1;
}
}
}
}
else if (Flag == 1 )
{
rpx = xb;
rpy = yb;
rpz = zb;
}
else
{
rpx = xp;
rpy = yp;
rpz = zp;
}

Fx = k_x*(rpx - xp);
Fy = k_y*(rpy - yp);
Fz = k_z*(rpz - zp);
//*************************************************************************
// Save Phantom data.
// Save both Phantom and HAL data for data file.
// Send Phantom data to HAL
// return force vector
//*************************************************************************
//Phan.flag = 1; // flag = 1 turns on pump
Phan.x = xp;
Phan.y = yp;
```

```
Phan.z = zp;
Phan.bucket = bucket;
Phan.time = Time;
Phan.flag = Flag;

data->setPhanData(Phan);
::ResumeThread(hThreadP2H); // trigger P2H thread to start
return gstVector(Fx, Fy, Fz); // returns force vector
}
//*****************************************************************************
// ? Viscous.cpp/h had this function ?
//*****************************************************************************
virtual gstVector calcEffectForce(void *phantom, gstVector & torques)
{
// Not using torque
torques = gstVector(0,0,0);
return calcEffectForce(phantom);
}
};

#endif // PH_EFFECT_INCLUDE (ascociated with "#ifndef")
```

C.3 Header file: DataStorage.h

```
//*****************************************************************************
DataStorage()
{
Time = 0;
memset(&PhData, 0, sizeof(PhanStruct));
memset(&BhData, 0, sizeof(BhStruct));
memset(&BhOrigin, 0, sizeof(BhStruct));
}
//*****************************************************************************
// Backhoe Orignin
//*****************************************************************************
void setOrigin(BhStruct Bh) { BhOrigin = Bh; }
BhStruct getOrigin() { return BhOrigin; }
//*****************************************************************************
// Time functions
//*****************************************************************************
void incTime() { Time++; }
int getTime() { return Time; }
//*****************************************************************************
// Function to retrieve data
//*****************************************************************************
PhanStruct getPhanData(int n) { return PhData[RECENT-n]; }
```

```
BhStruct getBhData(int n) { return BhData[RECENT-n]; }
//************************************************************************
// Functions to store Backhoe data
//************************************************************************
void setBhData(BhStruct Bh)
{
for(int i = 0; i < RECENT-1; i++)  // Update recent array of Hal data
{
BhData[i] = BhData[i+1];
}
BhData[RECENT-1] = Bh;
}
//************************************************************************
// Functions to store Phan data
//************************************************************************
void setPhanData(PhanStruct Ph)
{
for(int i = 0; i < RECENT-1; i++)  // Update recent array of Hal data
{
PhData[i] = PhData[i+1];
}
PhData[RECENT-1] = Ph;
}
};

#endif // DATA_STORAGE_INCLUDE (ascociated with "#ifndef")
```

C.4 Header file: DataStruct.h

```
//************************************************************************
// Filename : DataStruct.h
// ----------------------------------------------------------------------
//
// This file declare two different data structures.  One is made to store
// data from the Phantom and the second is to store data from Backhoe.
//
// Author: Matt Kontz <mkontz@mail.com>
// Lab: IMDL ME GaTech
// Created: October 19, 2003
// Edited: na
//
//************************************************************************

#ifndef DATA_STRUCTURE_INCLUDE // if not defined '......'
#define DATA_STRUCTURE_INCLUDE // defines '......' so only happens once.
```

```
struct PhanStruct
{
double x;
double y;
double z;
double bucket;
unsigned int time;
unsigned int flag;
};

// Stores all relevant data from Backhoe for each sampling peroid
struct BhStruct
{
double x;
double y;
double z;
double bucket;
unsigned int time;
unsigned int flag;
};

#endif
```

C.5 Header file: Sock.h

```
// Filename : Sock.h
// ------------------------------------------------------------------------
//
// This file is creates the object udpSocket.  This class has four
// associated functions: a constructor, send, recv and close.  Being a
// class object these classes are stand alone and can be used by function
//  using pointers.
//
// If you are using Visual C++ you must include the wsock32.lib library
// under "settings" -> "Link" -> "Input".
//
// Author: Matt Kontz <mkontz@mail.com>
// Lab: IMDL ME GaTech
// Created: July 10, 2002
// Edited: Oct 22, 2002
//
//************************************************************************
//
//      This socket library must be compiled in a C++ environment.  It has
//      been on both a Linux and Windows OS.  A similar library which only
//      uses only C function is Csocket.h.
```

```
//
//      For demos on how to use these libraries see:
//      PhMain.cpp (part of PhanHal.dsp) . Sock.h
//      com.cpp (Linux) .       . . . Sock.h
//      com.c (Linux) .       .       . . Csocket.h
//
//**************************************************************************

#ifndef __SOCK_INCLUDED__
#define __SOCK_INCLUDED__

#include <iostream.h>// For cout and cerr
#include <string.h>// for memset()
#include <stdlib.h>// for atoi() and exit()
#include <stdio.h>// for printf() and fprintf()
#include <errno.h>

#ifdef WIN32
#include <winsock.h>// for socket(), connect(), send(), and recv()
typedef int socklen_t;
#else
#include <sys/types.h>// for socket(), connect(), send(), and recv()
#include <sys/socket.h>// for socket(), connect(), send(), and recv()
#include <netdb.h>// for gethostbyname()
#include <arpa/inet.h>// for sockaddr_in and inet_addr()
#include <unistd.h>// for close()
#endif

class udpSocket
{
private:
int sock; // Socket
unsigned short localPort; // Local port
unsigned short forPort; // Foreign port
struct sockaddr_in localAddr; // Local address
struct sockaddr_in forAddr; // Foreign address
struct hostent *host; // pointer to server information
char *forIP; // Foreign IP address
unsigned int addrLen;
public:
udpSocket(char *fip, unsigned short fp,unsigned short lp)
{
forIP = fip;
forPort = fp;
localPort = lp;
sock = -1; // Less than 0 mean not connected

#ifdef WIN32
```

```
WORD wVersionRequested;
WSADATA wsaData;

wVersionRequested = MAKEWORD(2, 0); // Request Winsock v2.0
if (WSAStartup(wVersionRequested, &wsaData) != 0) // Load Winsock DLL
{
cerr << "WSAStartup() failed" << endl;
exit(1);
}
#endif

// Create a datagram/UDP socket
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
cerr << strerror(errno) << "socket() failed!" << endl;
exit(1);
}

// Construct local address structure
memset(&localAddr, 0, sizeof(localAddr)); // Zero out structure
localAddr.sin_family = AF_INET; // Internet address family
localAddr.sin_addr.s_addr = htonl(INADDR_ANY); // Any incoming interface
localAddr.sin_port = htons(localPort); // Local port

// Bind to the local address
if (bind(sock, (struct sockaddr *) &localAddr, sizeof(localAddr)) < 0)
{
cerr << strerror(errno) << "bind() failed" << endl;
exit(1);
}

// find foreign address
memset((char *) &forAddr, 0, sizeof(forAddr));
int addr = inet_addr(forIP);
forAddr.sin_addr.s_addr = addr;
if(addr != -1)
{
forAddr.sin_family = AF_INET;
}
else
{
host = gethostbyname(forIP);
if (host)
{
forAddr.sin_family = host->h_addrtype;
forAddr.sin_addr.s_addr = *((unsigned long *)host->h_addr_list[0]);
}
else
```

```
{
cerr << strerror(errno) << "Cannot get host information for server."
<< endl;
exit(1);
}
}

forAddr.sin_port = htons(forPort);

addrLen = sizeof(forAddr);

}

void send(char *msg, const int msgLen)
{

// Send the string to the server
if (sendto(sock, msg, msgLen, 0, (struct sockaddr *) &forAddr, addrLen)
!= msgLen)
{
cerr << strerror(errno) << "sendto() sent an incorrent number of bytes"
<< endl;
exit(1);
}
}

void recv(char *buffer, const int msgLen)
{
struct sockaddr_in fromAddr; // Source address of echo
int recvLen; // Length of received response */


// Recv a response
recvLen = recvfrom(sock, buffer, msgLen, 0, (struct sockaddr *)
&fromAddr, (socklen_t *) &addrLen);
if (recvLen != msgLen)
{
cerr << strerror(errno) << "recvfrom() failed: incorrent number of
bytes" << endl;
//exit(1);
}

// Check sender of message
if (fromAddr.sin_addr.s_addr != forAddr.sin_addr.s_addr)
{
cerr << strerror(errno) << "recvfrom() failed: unknown host" << endl;
exit(1);
}
```

```
}

void close()
{
// If the socket is open, close it.
if (sock > -1)
{
#ifdef WIN32
::closesocket(sock);
#else
::close(sock);
#endif
sock = -1;
}

#ifdef WIN32
if (WSACleanup() != 0)
{
cerr << "WSACleanup() failed" << endl;
exit(1);
}
#endif
}
};

#endif
```

APPENDIX D

KINEMATIC AND DYNAMIC TRANSFORMS

D.0.1 Cylinder Space to Joint Space Transformation

```
function q=cyl_to_joint(yc,R)

% This function computes the joint angles of the backhoe based on
% cylinder positions measurements and link dimensions

% Extract necessary dimensions from R matrix
%
%      1   2   3   4   5   6   7    8     9       10      11   12   13   14
%1 R=[R01 R0N RAN RJM R1A R0M R0K  R0J  p11K(1) p11K(2) p11K(3) T01A T10K TMJ0;
%2    R1C R12 R2C R1B R2B 0   0    0    0       0       0       TB12 T12C 0;
%3    R2D RDE R23 R2E R3D REF RDF  R3F  0       0       0       TD23 TDFE T23D;
%4    R34 R3G R4G RFH RGH 0   0    0    0       0       0       TG34 0    0;];

R01=R(1,1);
RJM=R(1,4);
R1A=R(1,5);
R0M=R(1,6);
R0K=R(1,7);
R1B=R(2,4);
T01A=R(1,12);
TB12=R(2,12);
R2C=R(2,3);
R2D=R(3,1);
T12C=R(2,13);
TD23=R(3,12);
REF=R(3,6);
RFH=R(4,4);
TDFE=R(3,13);
R3F=R(3,5)-R(3,7);
R3G=R(4,2);
RGH=R(4,5);
TG34=R(4,12);
R23=R(3,3);
R3D=R(3,5);
p11Kx=R(1,9);
p11Ky=R(1,10);
p11Kz=R(1,11);
```

```
% Expand cylinder lengths from Rc vector
RJK=yc(1);
RAB=yc(3);
RCD=yc(4);
REH=yc(5);

% 1. Compute q(1) from Swing Cylinder measurement yr(1)
R0J=sqrt(RJM^2+R0M^2);
TK0Q=atan((R01+p11Kx)/p11Kz);
TMJ0=atan(R0M/RJM);
TK0J=acos((R0K^2+R0J^2-yc(1)^2)/(2*R0K*R0J));
q(1)=TK0J-TK0Q-TMJ0;

% 2. Compute q(2) from Boom Cylinder measurement yr(2)
TA1B=acos((R1A^2+R1B^2-RAB^2)/(2*R1A*R1B));
q(2)=pi-T01A-TA1B-TB12;

% 3. Compute q(3) from Stick Cylinder measurement yr(3)
TC2D=acos((R2C^2+R2D^2-RCD^2)/(2*R2C*R2D));
q(3)=3*pi-T12C-TC2D-TD23;

% 4. Compute q(4) from Bucket Cylinder measurement yr(4)
TEFH=acos((REF^2+RFH^2-REH^2)/(2*REF*RFH));
THF3=pi-TDFE-TEFH;
R3H=sqrt(R3F^2+RFH^2-2*R3F*RFH*cos(THF3));
TF3H=acos((R3F^2+R3H^2-RFH^2)/(2*R3F*R3H));
TH3G=acos((R3H^2+R3G^2-RGH^2)/(2*R3H*R3G));
T23D=acos((R23^2+R3D^2-R2D^2)/(2*R23*R3D));
q(4)=3*pi-TF3H-TH3G-TG34-T23D;
```

D.0.2 Forward Displacement Analysis

```
function ye=Fdisp(q)

% This function computes end-effector displacement from the joint angles

alpha=[pi/2 0 0 0];
d=[0 0 0 0];

% Compute Homogeneous Transformation Matrices
for i=1:4
   st=sin(q(i));
   ct=cos(q(i));
   sa=sin(alpha(i));
```

```
   ca=cos(alpha(i));
   A{i}=[ct -st*ca st*sa a(i)*ct;
      st ct*ca -ct*sa a(i)*st;
      0 sa ca d(i);
      0 0 0 1];
end

% Compute overall transformation matrix
Y04=A{1}*A{2}*A{3}*A{4};

% Compute bucket angle phi
R12=Rz(q(2));
R23=Rz(q(3));
R34=Rz(q(4));
R14=R12*R23*R34;
X=R14(1,1);
Y=R14(2,1);
theta=atan2(Y,X);
if X < 0 & Y > 0  % atan2 in 2nd quadrant
   phi=theta-pi;
else % atan2 in 1st, 3rd, or 4th quadrant
   phi=theta+pi;
end

% Assemble output
ye=[Y04(1,4) Y04(2,4) Y04(3,4) phi]';

-----------------------------------------------------------------
function Rab=Rz(q);

% This function computes the elementary rotation matrix for an
% arbitrary angle about the y-axis

% Note that input units must be in RADIANS

Rab=[cos(q) -sin(q) 0;
   sin(q) cos(q) 0;
   0 0 1];
```

D.0.3 Reverse Displacement Analysis

```
function q=Rdisp(u,a)

% This function computes the reverse displacement analysis of the backhoe
xe=u(1);
```

```
ye=u(2);
ze=u(3);
phi=u(4);

p1o0o1=[a(1);0;0];
p4o3o4=[a(4);0;0];

% Solve for q(1) directly from endpoint position
q(1)=atan2(ye,xe);

% Rotation matrices
R01=Rx(pi/2)*Ry(q(1));
R04=Rx(pi/2)*Rz(pi)*Ry(-q(1))*Rz(phi);

% Solve position vectors for triangle 123
p0o0o4=[xe;ye;ze];
p0o3o4=R04*p4o3o4;
p1o3o4=R01'*p0o3o4;
p0o0o1=R01*p1o0o1;
p1o1o4=R01'*p0o0o4-p1o0o1;
p1o3o4=R01'*R04*p4o3o4;
p1o1o3=p1o1o4-p1o3o4;

% Use inverse trig and cosine law to get q(3)
PWx=p1o1o3(1);
PWy=p1o1o3(2);
T31x1=atan2(PWy,PWx);
R13=sqrt(PWx^2+PWy^2);
T321=acos((a(2)^2+a(3)^2-R13^2)/(2*a(2)*a(3)));
q(3)=pi+T321;

% Use cosine law to get q(2)
T213=acos((a(2)^2+R13^2-a(3)^2)/(2*a(2)*R13));
q(2)=T31x1+T213;

% Solve for q(4)
q(4)=phi-q(2)-q(3)+3*pi;
```

D.0.4 Joint Space to Cylinder Space Transformation

```
function yr=joint_to_cyl(R,q)

% This function computes the cylinder lengths for a given set of joint angles

% Extract relevant dimensions from the 4x14 R matrix
```

```
%
%      1   2   3   4   5   6   7    8     9       10      11   12   13   14
%1 R=[R01 R0N RAN RJM R1A R0M R0K  R0J  p11K(1) p11K(2) p11K(3) T01A T10K TMJ0;
%2    R1C R12 R2C R1B R2B 0   0    0    0       0       0       TB12 T12C 0;
%3    R2D RDE R23 R2E R3D REF RDF  R3F  0       0       0       TD23 TDFE T23D;
%4    R34 R3G R4G RFH RGH 0   0    0    0       0       0       TG34 0    0;];

%1.
R0M=R(1,6);
RJM=R(1,4);
R0J=R(1,8);
R0K=R(1,7);
T10K=R(1,13);

%2.
T01A=R(1,12);
TB12=R(2,12);
R1A=R(1,5);
R1B=R(2,4);

%3.
TD23=R(3,12);
T12C=R(2,13);
R2C=R(2,3);
R2D=R(3,1);

%4.
TG34=R(4,12);
T23D=R(3,14);
R3F=R(3,5)-R(3,7);
R3G=R(4,2);
RGH=R(4,5);
RFH=R(4,4);
TDFE=R(3,13);
REF=R(3,6);

% == PART 1: Compute Swing Cylinder Lengths yc(1) and yc(2) from q(1) ==

TQ0J=atan(R0M/RJM)+q(1);
TK0J=TQ0J+pi/2-T10K;
yr(1)=sqrt(R0J^2+R0K^2-2*R0J*R0K*cos(TK0J));
TJp0Qp=atan(R0M/RJM)-q(1);
TJp0Kp=TJp0Qp+pi/2-T10K;
yr(2)=sqrt(R0J^2+R0K^2-2*R0J*R0K*cos(TJp0Kp));

% ===== PART 2: Compute Boom Cylinder Length yr(3) from q(2) ====

TA1B=pi-T01A-TB12-q(2);
```

```
yr(3)=sqrt(R1A^2+R1B^2-2*R1A*R1B*cos(TA1B));

% == PART 3: Compute Stick Cylinder Length yr(4) from q(3) ==

T321=q(3)-pi;
TC2D=2*pi-TD23-T321-T12C;
yr(4)=sqrt(R2C^2+R2D^2-2*R2C*R2D*cos(TC2D));

% == PART 4: Compute Bucket Cylinder Length yr(5) from q(4) ==

Tx33G=2*pi-q(4)-TG34;
TG3F=pi-Tx33G+T23D;
RFG=sqrt(R3F^2+R3G^2-2*R3F*R3G*cos(TG3F));
T3FG=acos((R3F^2+RFG^2-R3G^2)/(2*R3F*RFG));
THFG=acos((RFG^2+RFH^2-RGH^2)/(2*RFG*RFH));
if q(4)+TG34>2*pi
   THF3=THFG+T3FG;
else
   THF3=THFG-T3FG;
end
TEFH=pi-TDFE-THF3;
yr(5)=sqrt(REF^2+RFH^2-2*REF*RFH*cos(TEFH));
```

APPENDIX E

MASS & INERTIA PROPERTIES

The data below was calculated from the solid model in Pro/ENGINEER. Mass values were obtained later during the backhoe rebuild with a bathroom scale and were found to vary by less than 5% from the solid model. The values below have been used throughout the LaGrangian dynamic model.

```
BASE ASSEMBLY WITH CYLINDERS
----------------------------

VOLUME =  3.2541377e+02  INCH^3
SURFACE AREA =  1.5227593e+03  INCH^2
AVERAGE DENSITY =  2.2985860e-01 POUND / INCH^3
MASS =  7.4799154e+01 POUND

CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
X   Y   Z    -7.9450335e+00  3.2953767e+00  6.6521587e-02  INCH

INERTIA at CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
(POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  2.3229143e+03 -5.2539074e+02  5.7118446e+01
Iyx Iyy Iyz -5.2539074e+02  5.2772512e+03  1.0302013e+01
Izx Izy Izz  5.7118446e+01  1.0302013e+01  5.3568896e+03

BOOM ASSEMBLY WITH CYLINDER
---------------------------

VOLUME =  4.9585884e+02  INCH^3
SURFACE AREA =  3.4438223e+03  INCH^2
AVERAGE DENSITY =  2.1966004e-01 POUND / INCH^3
MASS =  1.0892037e+02 POUND

CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
X   Y   Z    -2.6439663e+01  3.1270699e+00  0.0000000e+00  INCH

INERTIA at CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
(POUND * INCH^2)
```

```
INERTIA TENSOR:
Ixx Ixy Ixz  2.0186501e+03  7.6399946e+02  0.0000000e+00
Iyx Iyy Iyz  7.6399946e+02  1.5264085e+04  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  1.6083935e+04

STICK ASSEMBLY WITH CYLINDER
----------------------------

VOLUME =  4.5121342e+02  INCH^3
SURFACE AREA =  3.1935502e+03  INCH^2
AVERAGE DENSITY =  2.1115129e-01 POUND / INCH^3
MASS =  9.5274297e+01 POUND

CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
X   Y   Z    -2.3363210e+01  2.8479407e+00  0.0000000e+00  INCH

INERTIA at CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
(POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.3518286e+03 -5.4510614e+02  0.0000000e+00
Iyx Iyy Iyz -5.4510614e+02  2.2033641e+04  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  2.2529499e+04

BUCKET ASSEMBLY WITH CYLINDER
-----------------------------

VOLUME =  2.2684190e+02  INCH^3
SURFACE AREA =  1.7929021e+03  INCH^2
AVERAGE DENSITY =  2.8400000e-01 POUND / INCH^3
MASS =  6.4423099e+01 POUND

CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
X   Y   Z     7.7946254e+00  3.3482795e+00 -7.3503549e-04  INCH

INERTIA at CENTER OF GRAVITY with respect to ASM_DEF_CSYS coordinate frame:
(POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  2.0413482e+03 -4.5279092e+02  5.9602768e-01
Iyx Iyy Iyz -4.5279092e+02  6.2054163e+03 -9.1988628e-02
Izx Izy Izz  5.9602768e-01 -9.1988628e-02  5.9272355e+03
```

APPENDIX F

LAGRANGIAN DYNAMIC MODEL

The LaGrangian dynamic model [39] is given by:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau \tag{162}$$

The code that follows is the result of symbolic manipulation of equations 109, 119, and 120, generated using Matlab's symbolic math toolbox. Note that the joint angles are given in terms of the variable q rather than θ.

F.0.5 Matlab code to generate the terms of the LaGrangian dynamic model

```
% This function performs symbolic manipulation of the dynamic
% variables in the formulation of the LaGrangian dynamic model

% Output is in the form of text files that may be pasted into subsequent
% functions for numerical computations

clear all; close all; clc;

% Declare static joint variables
syms m1 m2 m3 m4
syms I111 I112 I113 I121 I122 I123 I131 I132 I133
syms I211 I212 I213 I221 I222 I223 I231 I232 I233
syms I311 I312 I313 I321 I322 I323 I331 I332 I333
syms I411 I412 I413 I421 I422 I423 I431 I432 I433
syms p1x p1y p1z p2x p2y p2z p3x p3y p3z p4x p4y p4z
syms a1 a2 a3 a4
syms g

% Declare dynamic joint variables
syms q1 q2 q3 q4
syms qd1 qd2 qd3 qd4

% Assemble symbolic inertia matrices
% These will be used to represent the inertia matrices at the local COM's
% and expressed in the local frames
I11=[I111 I112 I113;I121 I122 I123;I131 I132 I133];
```

```
I22=[I211 I212 I213;I221 I222 I223;I231 I232 I233];
I33=[I311 I312 I313;I321 I322 I323;I331 I332 I333];
I44=[I411 I412 I413;I421 I422 I423;I431 I432 I433];

% Position vectors from joint i-1 to COM i and between joints in local frames
p1o0c1=[p1x;p1y;p1z];
p1o0o1=[a1;0;0];

p2o1c2=[p2x;p2y;p2z];
p2o1o2=[a2;0;0];

p3o2c3=[p3x;p3y;p3z];
p3o2o3=[a3;0;0];

p4o3c4=[p4x;p4y;p4z];
p4o3o4=[a4;0;0];

% Rotation matrices between component frames
Rx90=[1 0 0;0 0 -1;0 1 0];
Ry1=[cos(q1) 0 sin(q1);0 1 0;-sin(q1) 0 cos(q1)];
R01=Rx90*Ry1;

R12=[cos(q2) -sin(q2) 0;sin(q2) cos(q2) 0;0 0 1];
R02=R01*R12;

R23=[cos(q3) -sin(q3) 0;sin(q3) cos(q3) 0;0 0 1];
R03=R02*R23;

R34=[cos(q4) -sin(q4) 0;sin(q4) cos(q4) 0;0 0 1];
R04=R03*R34;

% Position vectors from origins to COMs in base frame
% necessary for Jacobian
p0o0c1=R01*p1o0c1;
p0o1c2=R02*p2o1c2;
p0o2c3=R03*p3o2c3;
p0o3c4=R04*p4o3c4;

% z-axes
z00=[0;0;1];

z11=[0;0;1];
z01=R01*z11;

z22=z11;
z02=R02*z22;

z33=z11;
```

```
z03=R03*z33;

% Jacobian matrices
Jv1=[cross(z00,p0o0c1) zeros(3,3)];
Jv2=[cross(z00,p0o0c1) cross(z01,p0o1c2) zeros(3,2)];
Jv3=[cross(z00,p0o0c1) cross(z01,p0o1c2) cross(z02,p0o2c3) zeros(3,1)];
Jv4=[cross(z00,p0o0c1) cross(z01,p0o1c2) cross(z02,p0o2c3) cross(z03,p0o3c4)];

Jw1=[z00 zeros(3,3)];
Jw2=[z00 z01 zeros(3,2)];
Jw3=[z00 z01 z02 zeros(3,1)];
Jw4=[z00 z01 z02 z03];

% Rotate the inertia matrices to the base frame
I01=simple(R01*I11*R01.');
I02=simple(R02*I22*R02.');
I03=simple(R03*I33*R03.');
I04=simple(R04*I44*R04.');

% ======================== Inertia matrix ========================

% Symbolically compute inertia matrix
M=(Jv1.'*m1*Jv1+Jw1.'*I01*Jw1)+(Jv2.'*m2*Jv2+Jw2.'*I02*Jw2)+...
   (Jv3.'*m3*Jv3+Jw3.'*I03*Jw3)+(Jv4.'*m4*Jv4+Jw4.'*I04*Jw4);

% Write matrix terms to data files, element by element
for i=1:4
   for j=1:4
      Mij=simple(M(i,j));
      Mij=char(Mij);
      fid=fopen(strcat(cd,['\M',num2str(i),num2str(j),'.txt']),'w');
      fwrite(fid,Mij);
      fclose(fid);
   end
end

% =================== Velocity coupling vector ===================

% Symbolically compute vector terms
q=[q1;q2;q3;q4];
qd=[qd1;qd2;qd3;qd4];
for i=1:4
   clear Vi
   Vi=class('sym');
   for j=i:4
      for k=1:4
         fid=fopen(strcat(cd,'\M',num2str(i),num2str(j),'.txt'));
         Mij=fscanf(fid,'%c');
```

```
            fclose(fid);

            fid=fopen(strcat(cd,'\M',num2str(j),num2str(k),'.txt'));
            Mjk=fscanf(fid,'%c');
            fclose(fid);

            F1=diff(Mij,q(k));
            F2=diff(Mjk,q(i));
            Vi=Vi+(F1-0.5*F2)*qd(j)*qd(k);
         end
      end

      V=char(simple(Vi));
      V=strrep(V,'char','');


      % Write vector terms to data files, element by element
      switch i
      case 1
         V1=V;
         fid=fopen(strcat(cd,'\V1.txt'),'w');
         fwrite(fid,V1);
         fclose(fid);
      case 2
         V2=V;
         fid=fopen(strcat(cd,'\V2.txt'),'w');
         fwrite(fid,V2);
         fclose(fid);
      case 3
         V3=V;
         fid=fopen(strcat(cd,'\V3.txt'),'w');
         fwrite(fid,V3);
         fclose(fid);
      case 4
         V4=V;
         fid=fopen(strcat(cd,'\V4.txt'),'w');
         fwrite(fid,V4);
         fclose(fid);
      end
   end

   % ======================== Gravitational torque vector ======================

   % Symbolically compute vector terms
   g=sym([0;0;-g]);
   m=[m1 m2 m3 m4];

   for i=1:4
```

```
   clear Gi
   Gi=class('sym');
   for j=1:4
      switch j
      case 1
         Jivj=Jv1(:,i);
      case 2
         Jivj=Jv2(:,i);
      case 3
         Jivj=Jv3(:,i);
      case 4
         Jivj=Jv4(:,i);
      end
      Gi=Gi-m(j)*g.'*Jivj;
   end
   G=char(simple(Gi));
   G=strrep(G,'char','');

   % Write vector terms to data files, element by element
   switch i
   case 1
      G1=G;
      fid=fopen(strcat(cd,'\G1.txt'),'w');
      fwrite(fid,G1);
      fclose(fid);
   case 2
      G2=G;
      fid=fopen(strcat(cd,'\G2.txt'),'w');
      fwrite(fid,G2);
      fclose(fid);
   case 3
      G3=G;
      fid=fopen(strcat(cd,'\G3.txt'),'w');
      fwrite(fid,G3);
      fclose(fid);
   case 4
      G4=G;
      fid=fopen(strcat(cd,'\G4.txt'),'w');
      fwrite(fid,G4);
      fclose(fid);
   end
end

% Reopen the files replace all the qi & qdi terms w/ q(i) & qd(i)
for i=1:4

   % Inertia matrix terms
   for j=1:4
```

```
   fid=fopen(strcat(cd,'\M',num2str(i),num2str(j),'.txt'),'r');
   Mij=fscanf(fid,'%c');
   fclose(fid);
   Mij=strrep(Mij,'q1','q(1)');
   Mij=strrep(Mij,'q2','q(2)');
   Mij=strrep(Mij,'q3','q(3)');
   Mij=strrep(Mij,'q4','q(4)');
   Mij=strrep(Mij,'qd1','qd(1)');
   Mij=strrep(Mij,'qd2','qd(2)');
   Mij=strrep(Mij,'qd3','qd(3)');
   Mij=strrep(Mij,'qd4','qd(4)');
   fid=fopen(strcat(cd,'\M',num2str(i),num2str(j),'.txt'),'w');
   fwrite(fid,Mij);
   fclose(fid);
end

% Velocity coupling vector terms
fid=fopen(strcat(cd,'\V',num2str(i),'.txt'),'r');
Vi=fscanf(fid,'%c');
fclose(fid);
Vi=strrep(Vi,'q1','q(1)');
Vi=strrep(Vi,'q2','q(2)');
Vi=strrep(Vi,'q3','q(3)');
Vi=strrep(Vi,'q4','q(4)');
Vi=strrep(Vi,'qd1','qd(1)');
Vi=strrep(Vi,'qd2','qd(2)');
Vi=strrep(Vi,'qd3','qd(3)');
Vi=strrep(Vi,'qd4','qd(4)');
fid=fopen(strcat(cd,'\V',num2str(i),'.txt'),'w');
fwrite(fid,Vi);
fclose(fid);

% Gravity vector terms
fid=fopen(strcat(cd,'\G',num2str(i),'.txt'),'r');
Gi=fscanf(fid,'%c');
fclose(fid);
Gi=strrep(Gi,'q1','q(1)');
Gi=strrep(Gi,'q2','q(2)');
Gi=strrep(Gi,'q3','q(3)');
Gi=strrep(Gi,'q4','q(4)');
Gi=strrep(Gi,'qd1','qd(1)');
Gi=strrep(Gi,'qd2','qd(2)');
Gi=strrep(Gi,'qd3','qd(3)');
Gi=strrep(Gi,'qd4','qd(4)');
fid=fopen(strcat(cd,'\G',num2str(i),'.txt'),'w');
fwrite(fid,Gi);
fclose(fid);
```

```
end

disp('Symbolic matrix and vector elements save successfully to folder')
```

F.0.6 Output

The elements of the matrix **M** and the vectors **V** and **G** are given below.

```
M(1,1) =
m3*p1z^2+m3*p1x^2+m1*p1x^2+m1*p1z^2+m2*p1z^2+m2*p1x^2+I122-1/2*I311
*cos(2*q(3)+2*q(2))+1/2*I211+1/2*I222+1/2*I321*sin(2*q(3)+2*q(2))
+1/2*I422+1/2*I411+1/2*I221*sin(2*q(2))-1/2*I211*cos(2*q(2))+1/2*I322
*cos(2*q(3)+2*q(2))+1/2*I312*sin(2*q(3)+2*q(2))+1/2*I311+1/2*I322+m4
*p1z^2+1/2*I212*sin(2*q(2))+m4*p1x^2-1/2*I411*cos(2*q(4)+2*q(3)+2
*q(2))+1/2*I421*sin(2*q(4)+2*q(3)+2*q(2))+1/2*I412*sin(2*q(4)+2*q(3)
+2*q(2))+1/2*I422*cos(2*q(4)+2*q(3)+2*q(2))+1/2*I222*cos(2*q(2))

M(1,2) =
-m2*p1z*sin(q(2))*p2x+cos(q(2))*I223+cos(q(2)+q(3))*I323+I413
*sin(q(2)+q(3)+q(4))+sin(q(2)+q(3))*I313-m2*p1z*cos(q(2))*p2y
-m3*p1z*sin(q(2))*p2x-m3*p1z*cos(q(2))*p2y+I423*cos(q(2)+q(3)
+q(4))+sin(q(2))*I213-m4*p1z*sin(q(2))*p2x-m4*p1z*cos(q(2))*p2y

M(1,3) =
-m4*p1z*cos(q(2)+q(3))*p3y-m4*p1z*sin(q(2)+q(3))*p3x+I413
*sin(q(2)+q(3)+q(4))+I423*cos(q(2)+q(3)+q(4))-m3*p1z*sin(q(2)+q(3))
*p3x-m3*p1z*cos(q(2)+q(3))*p3y+sin(q(2)+q(3))*I313
+cos(q(2)+q(3))*I323

M(1,4) =
I423*cos(q(2)+q(3)+q(4))+I413*sin(q(2)+q(3)+q(4))-m4*p1z*cos(q(2)
+q(3)+q(4))*p4y-m4*p1z*sin(q(2)+q(3)+q(4))*p4x

M(2,1) =
sin(q(2))*I231+I432*cos(q(2)+q(3)+q(4))+I431*sin(q(2)+q(3)+q(4))
-m2*p1z*sin(q(2))*p2x-m2*p1z*cos(q(2))*p2y+cos(q(2))*I232-m3*p1z
*cos(q(2))*p2y-m3*p1z*sin(q(2))*p2x-m4*p1z*sin(q(2))*p2x-m4*p1z
*cos(q(2))*p2y+I332*cos(q(2)+q(3))+I331*sin(q(2)+q(3))

M(2,2) =
m3*p2y^2+m3*p2x^2+m2*p2x^2+m2*p2y^2+I233+I433+m4*p2y^2+m4*p2x^2+I333

M(2,3) =
m3*p2y*p3y*cos(q(3))+m3*p2x*p3x*cos(q(3))+m3*p2y*p3x*sin(q(3))
-m3*p2x*p3y*sin(q(3))+I433+I333+m4*p2y*p3x*sin(q(3))+m4*p2y*p3y
*cos(q(3))+m4*p2x*p3x*cos(q(3))-m4*p2x*p3y*sin(q(3))
```

```
M(2,4) =
I433+m4*p2y*p4x*sin(q(3)+q(4))-m4*p2x*p4y*sin(q(3)+q(4))+m4*p2y
*p4y*cos(q(3)+q(4))+m4*p2x*p4x*cos(q(3)+q(4))

M(3,1) =
I331*sin(q(2)+q(3))+I332*cos(q(2)+q(3))-m3*p1z*cos(q(2)+q(3))*p3y
-m3*p1z*sin(q(2)+q(3))*p3x-m4*p1z*sin(q(2)+q(3))*p3x-m4*p1z*cos(q(2)
+q(3))*p3y+I432*cos(q(2)+q(3)+q(4))+I431*sin(q(2)+q(3)+q(4))

M(3,2) =
I433+m3*p2y*p3y*cos(q(3))+I333+m4*p2x*p3x*cos(q(3))-m4*p2x*p3y
*sin(q(3))+m4*p2y*p3y*cos(q(3))+m4*p2y*p3x*sin(q(3))+m3*p2y*p3x
*sin(q(3))+m3*p2x*p3x*cos(q(3))-m3*p2x*p3y*sin(q(3))

M(3,3) =
m3*p3y^2+m3*p3x^2+m4*p3y^2+m4*p3x^2+I433+I333

M(3,4) =
I433+m4*p3y*p4x*sin(q(4))+m4*p3y*p4y*cos(q(4))+m4*p3x*p4x*cos(q(4))
-m4*p3x*p4y*sin(q(4))

M(4,1) =
I432*cos(q(2)+q(3)+q(4))-m4*p1z*sin(q(2)+q(3)+q(4))*p4x-m4*p1z
*cos(q(2)+q(3)+q(4))*p4y+I431*sin(q(2)+q(3)+q(4))

M(4,2) =
-m4*p2x*p4y*sin(q(3)+q(4))+m4*p2x*p4x*cos(q(3)+q(4))+I433+m4*p2y*p4x
*sin(q(3)+q(4))+m4*p2y*p4y*cos(q(3)+q(4))

M(4,3) =
I433+m4*p3y*p4y*cos(q(4))+m4*p3y*p4x*sin(q(4))-m4*p3x*p4y*sin(q(4))
+m4*p3x*p4x*cos(q(4))

M(4,4) = I433+m4*p4y^2+m4*p4x^2

V(1) =
(I312*cos(2*q(3)+2*q(2))-I322*sin(2*q(3)+2*q(2))+I211*sin(2*q(2))
+I221*cos(2*q(2))+I212*cos(2*q(2))+I321*cos(2*q(3)+2*q(2))+I311
*sin(2*q(3)+2*q(2))-I222*sin(2*q(2))+I411*sin(2*q(4)+2*q(3)+2*q(2))
+I421*cos(2*q(4)+2*q(3)+2*q(2))+I412*cos(2*q(4)+2*q(3)+2*q(2))-I422
*sin(2*q(4)+2*q(3)+2*q(2)))*qd(1)*qd(2)+(I312*cos(2*q(3)+2*q(2))-I322
*sin(2*q(3)+2*q(2))+I321*cos(2*q(3)+2*q(2))+I311*sin(2*q(3)+2*q(2))
+I411*sin(2*q(4)+2*q(3)+2*q(2))+I421*cos(2*q(4)+2*q(3)+2*q(2))+I412
*cos(2*q(4)+2*q(3)+2*q(2))-I422*sin(2*q(4)+2*q(3)+2*q(2)))*qd(1)
*qd(3)+(I411*sin(2*q(4)+2*q(3)+2*q(2))+I421*cos(2*q(4)+2*q(3)+2*q(2))
+I412*cos(2*q(4)+2*q(3)+2*q(2))-I422*sin(2*q(4)+2*q(3)+2*q(2)))*qd(1)
*qd(4)+(-m2*p1z*cos(q(2))*p2x-sin(q(2))*I223-sin(q(2)+q(3))*I323+I413
```

```
*cos(q(2)+q(3)+q(4))+cos(q(2)+q(3))*I313+m2*p1z*sin(q(2))*p2y-m3*p1z
*cos(q(2))*p2x+m3*p1z*sin(q(2))*p2y-I423*sin(q(2)+q(3)+q(4))
+cos(q(2))*I213-m4*p1z*cos(q(2))*p2x+m4*p1z*sin(q(2))*p2y)*qd(2)^2
+(-sin(q(2)+q(3))*I323+I413*cos(q(2)+q(3)+q(4))+cos(q(2)+q(3))*I313
-I423*sin(q(2)+q(3)+q(4)))*qd(2)*qd(3)+(I413*cos(q(2)+q(3)+q(4))
-I423*sin(q(2)+q(3)+q(4)))*qd(2)*qd(4)+(m4*p1z*sin(q(2)+q(3))*p3y
-m4*p1z*cos(q(2)+q(3))*p3x+I413*cos(q(2)+q(3)+q(4))-I423*sin(q(2)
+q(3)+q(4))-m3*p1z*cos(q(2)+q(3))*p3x+m3*p1z*sin(q(2)+q(3))*p3y
+cos(q(2)+q(3))*I313-sin(q(2)+q(3))*I323)*qd(3)*qd(2)+(m4*p1z
*sin(q(2)+q(3))*p3y-m4*p1z*cos(q(2)+q(3))*p3x+I413*cos(q(2)+q(3)
+q(4))-I423*sin(q(2)+q(3)+q(4))-m3*p1z*cos(q(2)+q(3))*p3x+m3*p1z
*sin(q(2)+q(3))*p3y+cos(q(2)+q(3*I313-sin(q(2)+q(3))*I323)*qd(3)^2
+(I413*cos(q(2)+q(3)+q(4))-I423*sin(q(2)+q(3)+q(4)))*qd(3)*qd(4)
+(-I423*sin(q(2)+q(3)+q(4))+I413*cos(q(2)+q(3)+q(4))+m4*p1z
*sin(q(2)+q(3)+q(4))*p4y-m4*p1z*cos(q(2)+q(3)+q(4))*p4x)*qd(4)*qd(2)
+(-I423*sin(q(2)+q(3)+q(4))+I413*cos(q(2)+q(3)+q(4))+m4*p1z*sin(q(2)
+q(3)+q(4))*p4y-m4*p1z*cos(q(2)+q(3)+q(4))*p4x)*qd(4)*qd(3)
+(-I423*sin(q(2)+q(3)+q(4))+I413*cos(q(2)+q(3)+q(4))+m4*p1z
*sin(q(2)+q(3)+q(4))*p4y-m4*p1z*cos(q(2)+q(3)+q(4))*p4x)*qd(4)^2

V(2) =
(-1/2*cos(q(2))*I231+1/2*I432*sin(q(2)+q(3)+q(4))-1/2*I431
*cos(q(2)+q(3)+q(4))+1/2*m2*p1z*cos(q(2))*p2x-1/2*m2*p1z*sin(q(2))
*p2y+1/2*sin(q(2))*I232-1/2*m3*p1z*sin(q(2))*p2y+1/2*m3*p1z*cos(q(2))
*p2x+1/2*m4*p1z*cos(q(2))*p2x-1/2*m4*p1z*sin(q(2))*p2y+1/2*I332
*sin(q(2)+q(3))-1/2*I331*cos(q(2)+q(3)))*qd(2)*qd(1)+(-1/2*I331
*cos(q(2)+q(3))+1/2*I332*sin(q(2)+q(3))-1/2*m3*p1z*sin(q(2)+q(3))*p3y
+1/2*m3*p1z*cos(q(2)+q(3))*p3x+1/2*m4*p1z*cos(q(2)+q(3))*p3x-1/2*m4
*p1z*sin(q(2)+q(3))*p3y+1/2*I432*sin(q(2)+q(3)+q(4))-1/2*I431
*cos(q(2)+q(3)+q(4)))*qd(3)*qd(1)+(-m3*p2y*p3y*sin(q(3))-m3*p2x*p3x
*sin(q(3))+m3*p2y*p3x*cos(q(3))-m3*p2x*p3y*cos(q(3))+m4*p2y*p3x
*cos(q(3))-m4*p2y*p3y*sin(q(3))-m4*p2x*p3x*sin(q(3))-m4*p2x*p3y
*cos(q(3)))*qd(3)^2+(1/2*I432*sin(q(2)+q(3)+q(4))+1/2*m4*p1z
*cos(q(2)+q(3)+q(4))*p4x-1/2*m4*p1z*sin(q(2)+q(3)+q(4))*p4y-1/2*I431
*cos(q(2)+q(3)+q(4)))*qd(4)*qd(1)+(m4*p2y*p4x*cos(q(3)+q(4))-m4*p2x
*p4y*cos(q(3)+q(4))-m4*p2y*p4y*sin(q(3)+q(4))-m4*p2x*p4x
*sin(q(3)+q(4)))*qd(4)*qd(3)+(m4*p2y*p4x*cos(q(3)+q(4))-m4*p2x*p4y
*cos(q(3)+q(4))-m4*p2y*p4y*sin(q(3)+q(4))-m4*p2x*p4x*sin(q(3)+q(4)))
*qd(4)^2

V(3) =
(-1/2*I331*cos(q(2)+q(3))+1/2*I332*sin(q(2)+q(3))-1/2*m3*p1z
*sin(q(2)+q(3))*p3y+1/2*m3*p1z*cos(q(2)+q(3))*p3x+1/2*m4*p1z*cos(q(2)
+q(3))*p3x-1/2*m4*p1z*sin(q(2)+q(3))*p3y+1/2*I432*sin(q(2)+q(3)+q(4))
-1/2*I431*cos(q(2)+q(3)+q(4)))*qd(3)*qd(1)+(1/2*m3*p2y*p3y*sin(q(3))
+1/2*m3*p2x*p3x*sin(q(3))-1/2*m3*p2y*p3x*cos(q(3))+1/2*m3*p2x*p3y
*cos(q(3))-1/2*m4*p2y*p3x*cos(q(3))+1/2*m4*p2y*p3y*sin(q(3))+1/2*m4
*p2x*p3x*sin(q(3))+1/2*m4*p2x*p3y*cos(q(3)))*qd(3)*qd(2)+(1/2*I432
```

```
*sin(q(2)+q(3)+q(4))+1/2*m4*p1z*cos(q(2)+q(3)+q(4))*p4x-1/2*m4*p1z
*sin(q(2)+q(3)+q(4))*p4y-1/2*I431*cos(q(2)+q(3)+q(4)))*qd(4)*qd(1)
+(-1/2*m4*p2y*p4x*cos(q(3)+q(4))+1/2*m4*p2x*p4y*cos(q(3)+q(4))+1/2*m4
*p2y*p4y*sin(q(3)+q(4))+1/2*m4*p2x*p4x*sin(q(3)+q(4)))*qd(4)*qd(2)
+(m4*p3y*p4x*cos(q(4))-m4*p3y*p4y*sin(q(4))-m4*p3x*p4x*sin(q(4))-m4
*p3x*p4y*cos(q(4)))*qd(4)^2

V(4) =
(1/2*I432*sin(q(2)+q(3)+q(4))+1/2*m4*p1z*cos(q(2)+q(3)+q(4))*p4x
-1/2*m4*p1z*sin(q(2)+q(3)+q(4))*p4y-1/2*I431*cos(q(2)+q(3)+q(4)))
*qd(4)*qd(1)+(-1/2*m4*p2y*p4x*cos(q(3)+q(4))+1/2*m4*p2x*p4y
*cos(q(3)+q(4))+1/2*m4*p2y*p4y*sin(q(3)+q(4))+1/2*m4*p2x*p4x
*sin(q(3)+q(4)))*qd(4)*qd(2)+(-1/2*m4*p3y*p4x*cos(q(4))+1/2*m4*p3y
*p4y*sin(q(4))+1/2*m4*p3x*p4x*sin(q(4))+1/2*m4*p3x*p4y*cos(q(4)))
*qd(4)*qd(3)

G(1) = 0

G(2) =
m2*g*cos(q(2))*p2x-m2*g*sin(q(2))*p2y+m3*g*cos(q(2))*p2x-m3*g
*sin(q(2))*p2y+m4*g*cos(q(2))*p2x-m4*g*sin(q(2))*p2y

G(3) =
m3*g*p3x*cos(q(2)+q(3))-m3*g*p3y*sin(q(2)+q(3))+m4*g*p3x
*cos(q(2)+q(3))-m4*g*p3y*sin(q(2)+q(3))

G(4) = m4*g*p4x*cos(q(2)+q(3)+q(4))-m4*g*p4y*sin(q(2)+q(3)+q(4))
```

APPENDIX G

PVG32 SYSTEM IDENTIFICATION CODE

This section shows the Matlab script used to optimize the PVG32 simulink model parameters. The `fmincon` function was used to call the simulink model shown in figure 29, and run a simulation using a vector of unknown model parameters. The optimization converged to a solution when the error between the experimental data and the model ouput was less than the specified tolerance.

Parameters to be optimized: $p_1, p_2, p_3, \zeta, \omega_n, d_{band}$, and K_r

```
% ******************************************
%                                          *
%    PVG32 System Identification Routine   *
%                                          *
%******************************************

% This file uses the optimization routine 'fmincon' to
% find the optimum system parameters of the PVG32 valve
% by comparing the model prediction with the step response
% data and minimizing the error between the two.

clear all ; close all ; clc ;

% ========== MODEL PARAMETERS =============

% ---------- "Known" quantities ------------
QP=42;
pT=0;
prelief=900; % note: I think this is in the HIL pump, not the PVG32

% Delay interval for input signal averaging
T=0.001;
dt=0.16;
kd=dt/T;
td=0.096;

% ---------- Guess quantities ------------
p1=-13+33*i;
```

```
p1r=real(p1);
p1c=imag(p1);
p2=conj(p1);
p3=-25;
dband=0.55;
Kr=0.0068;
z=0.68
wn=2*pi/0.18

% Set up vectors of guess quantities and their bounds
X0=[p1r  p1c p3  dband  Kr];
UB=[-12  33 -22  0.55  0.02];
LB=[-13  30 -25  0.45  0];

% ============= NUMERICAL OPTIMIZATION =============

% Optimization parameters
options.MaxIter=150;
options.MaxFunEvals=150;
options.Display='iter';
options.TolFun=1e-8;
options.TolX=1e-8;
options.DiffMaxChange=1e-6;

% Open valve Simulink model
open('PVG32')

% Estimate value of model parameters using numerical optimization
X=fmincon('model_error',X0,[],[],[],[],LB,UB,[],options);

% Assign optimization output to workspace variables
p1r=X(1);
p1c=X(2);
p1=p1r+i*p1c
p2=conj(p1);
p3=X(3)
dband=X(4)
Kr=X(5)

% Transfer function WITHOUT the SS gain in the numerator
num=1;
den=real(conv(conv([1 -p1],[1 -p2]),[1 -p3]));
Gv=tf(num,den)

% ====================== RESULTS ======================
% Check the step response of the optimized model
figure
step=[3 4 6 8 9 11];
```

```
for i=1:length(step)
   data=getdata('s',step(i));
   sim('PVG32')
   if length(Qm)-length(Qt)==1
      Qt=[0;Qt];
   end
   subplot(ceil(length(step)/2),2,i)
   plot(t,Qm,'r','linewidth',2)
   hold on; grid on
   plot(t,Qt,'b--','linewidth',2)
   ylim([-5 35])
   legend('data','model',2)
   xlabel('time [s]')
   ylabel('flow rate [in^3/s]')
   title(strcat('Stepsize = ', num2str(data),' [V]'))
end

% Check the sinusoidal response of the optimized model
figure
step=[1.0 2.0 3.0 4.0];
for i=1:length(step)
   data=getdata('f',step(i));
   sim('PVG32')
   if length(Qm)-length(Qt)==1
      Qt=[0;Qt];
   end
   subplot(ceil(length(step)/2),2,i)
   plot(t,Qm,'r','linewidth',2)
   hold on; grid on
   plot(t,Qt,'b--','linewidth',2)
   legend('data','model',2)
   xlabel('time [s]')
   xlim([2 5])
   ylabel('flow rate [in^3/s]')
   title(strcat('Frequency= ', num2str(data),' [Hz]'))
end

% Check the frequency response of the optimized model
f(1)=1.0;
i=1;

while f(i)<=8.8
   data=getdata('f',f(i));
   sim('PVG32')
   if length(Qm)-length(Qt)==1
      Qt=[0;Qt];
   end
   Magt(i)=amplitude(t,Us,Qt,f(i));
```

```
   Pht(i)=phase(t,Us,Qt,f(i));

   i=i+1;
   f(i)=f(i-1)+0.2;
end

load bodedata2
fm=fe;
Magm=Me;
Phm=Phe;

figure
subplot(2,1,1)
semilogx(fm,Magm,'rd','Linewidth',1.5)
hold on
grid on
semilogx(f(1:end-1),Magt,'b--','Linewidth',1.5)
xlim([1 10])
title('PVG32 Valve Frequency Response')
ylabel('Magnitude [dB]')
legend('data','model')
text(1.5,-1,'G_v(s)=Q(s)/U_s(s)=Flowrate/Voltage')

subplot(2,1,2)
semilogx(fm,Phm,'rd','Linewidth',1.5)
xlim([1 10])
hold on
grid on
semilogx(f(1:end-1),Pht,'b--','Linewidth',1.5)
ylabel('Phase [deg]')
xlabel('Frequency [Hz]')
legend('data','model')
```

This function computes the sum of the square of the errors between the data and the model output:

```
function F=model_error(X)

% This function calls the simulink model 'PVG32' and compares its output
% to the measured output, and computes an error between the two

p1r=X(1);
p1c=X(2);
p1=p1r+i*p1c;
p2=conj(p1);
p3=X(3);
dband=X(4);
Kr=X(5);
```

```
% Create a transfer function from the current set of poles
num=1;
den=real(conv(conv([1 -p1],[1 -p2]),[1 -p3]));

% Assign input to workspace
assignin('base','p1',X(1));
assignin('base','p2',p2);
assignin('base','p3',X(3));
assignin('base','dband',X(4));
assignin('base','Kr',X(5));
assignin('base','num',num);
assignin('base','den',den);

F=0;

% =========== Step data Optimization ================

% Possible data sets
%ampl=[0.5 0.55 0.6 0.7 0.8 0.9 1.1 1.2 1.3 1.4 1.5];
%Set= [ 1    2   3   4   5   6   7   8   9  10  11 ];
Set=[8 9 10];

type='s';
for i=1:length(Set)

data=getdata(type,Set(i));

   % Simulate PVG32 with the current parameters
   sim('PVG32')

   % Compute squared error between model and data
   n=min([length(Qt) length(Qm)]);
   e=Qt(1:n)-Qm(1:n);
   F=F+e'*e;

end

% =========== Frequency data Optimization ================

% Frequencies available: 1.0-8.8[Hz] in 0.2[Hz] increments

Set=[1.0 3.4 6.2];

type='f';
for i=1:length(Set)

data=getdata(type,Set(i));
```

```
   % Simulate PVG32 with the current parameters
   sim('PVG32')

   % Compute squared error between model and data
   n=min([length(Qt) length(Qm)]);
   e=Qt(1:n)-Qm(1:n);
   F=F+e'*e;

end
```

This function computes the mean amplitude of a sinusoidal response vector for generating a Bode plot:

```
function Mag=amplitude(t,Us,Qt,f)

% This function computes the mean amplitudes of the input and
% output time traces

fs=1000;
N=length(t);
nwaves=floor(N*f/fs);
nset=floor(fs/f);

Umax=zeros(nwaves,1);
Umin=zeros(nwaves,1);
Qmax=zeros(nwaves,1);
Qmin=zeros(nwaves,1);

% Scroll through each input & output wave and save its max's and min's
for i=1:nwaves
   Uset=Us((i-1)*nset+1:i*nset);
   Qset=Qt((i-1)*nset+1:i*nset);
   Umax(i)=max(Uset);
   Umin(i)=min(Uset);
   Qmax(i)=max(Qset);
   Qmin(i)=min(Qset);
end
Umag=mean(Umax)-mean(Umin);
Qmag=mean(Qmax)-mean(Qmin);

Mag=20*log10(Qmag/Umag);
```

This function computes the phase shift of a sinusoidal response vector for generating a Bode plot:

```
function Ph=phase(t,Us,Qt,f)

% This function computes the mean amplitudes of the input and
```

```
% output time traces

fs=1000;
N=length(t);
nwaves=floor(N*f/fs);
nset=floor(fs/f);

phase=zeros(nwaves,1);

for i=1:nwaves
    t1(i)=t((i-1)*nset+1);
    t4(i)=t(i*nset);
    Uset=Us((i-1)*nset+1:i*nset);
    Umean=mean(Uset);
    Qset=Qt((i-1)*nset+1:i*nset);
    Qmean=mean(Qset);

    % Pick out the time when Us crosses zero this wave
    j=0;
    t2(i)=t4(i);
    for j=1:length(Uset)-1
        if Uset(j)>=Umean & Uset(j+1)<Umean
            t2(i)=t((i-1)*nset+j);
        end
    end

    % Pick out the time when Qt crosses zero this wave
    j=0;
    t3(i)=t4(i);
    for j=1:length(Qset)-1
        if Qset(j)>=Qmean & Qset(j+1)<Qmean
            t3(i)=t((i-1)*nset+j);
        end
    end

    % Compute the phase shift this wave
    if t3(i) < t2(i)
       tp(i)=(t4(i)-t2(i))+(t3(i)-t1(i));
    else
       tp(i)=t3(i)-t2(i);
    end
    phase(i)=2*pi*f*tp(i);
    if f>=6
       phase(i)=phase(i)+2*pi;
    end
end

Ph=-mean(phase)*180/pi;
```

REFERENCES

[1] Alleyne, A. and Liu, R., "A simplified approach to force control for electrhydraulic systems," *Control Engineering Practice*, vol. 8, pp. 1347–1356, 2000.

[2] Bernold, L. E., Lloyd, J., and Vouk, M., "Equipment operator training in the age of internet2," in *NIST, 19th International Symposium on Automation and Robotics in Construction*, (Gaithersburg, MD), pp. 505–510, September 2002.

[3] Budny, E., Chlosta, M., and Gutkowski, W., "Load-independent control of a hydraulic excavator," *Automation in Construction 12*, pp. 245–254, August 2002.

[4] Buttolo, P., Stewart, P., and Marsan, A., "A haptic hybrid controller for virtual prototyping of vehicle mechanisms," in *Proceedings 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 249–254, March 2002. Ford Motor Co.

[5] Craig, J. J., *Introduction to Robotics*. Reading, MA: Addison-Wesley Pub. Co., 1989.

[6] DiMaio, S. P. and Salcudean, S. E., "A virtual environment for the simulation and programming of excavation trajectories," *Presence*, vol. 10, pp. 465–476, Oct. 2001.

[7] Frankel, J., *Hardware-in-the-Loop (HIL) Simulator Operator's Manual*. Georgia Tech Fluid Power and Motion Control Center, September 2003. Technical report.

[8] Frankel, J., "Modeling, simulation, and design of a haptic backhoe–part 2," October 2003. Powerpoint presentation.

[9] Fritz, J. and Barner, K., "Design of a haptic data visualization system for people with visual impairments," *IEEE Transactions on Rehabilitation Engineering*, vol. 7, pp. 372–384, Sept. 1999.

[10] Gosline, A. H., Salcudean, S. E., and Yan, J., "Haptic simulation of linear elastic media with fluid pockets," in *12th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, (Chicago, IL), March 2004.

[11] Ha, Q. P., Nguyen, Q. H., Rye, D. C., and Durrant-Whyte, H. F., "Impedance control of a hydraulically actuated robotic excavator," *Automation in Construction*, vol. 9, pp. 421–435, 2000.

[12] Henke, R. W., *Introduction to Fluid Mechanics*. Reading, MA: Addison-Wesley Pub. Co., 1966.

[13] Henke, R. W., *Fluid Power Systems and Circuits*. Hydraulics & Pneumatics Magazine pub., 1983.

[14] Horie, T., Tanaka, K., Abe, N., and Taki, H., "Remote force control of robot using phantom haptic model and force sensor," in *Proceedings of the 4^{th} IEEE International Symposium on Assembly and Task Planning*, (Fukuoka, Japan), pp. 128–135, May 2001.

[15] John Deere & Co., *Technical Manual 4000 Series Compact Utility Tractor Attachments*, 1999. TM1763-01Jul99.

[16] Koivo, A. J., Thoma, M., Kocaoglan, E., and Andrade-Cetto, J., "Modeling and control of excavator dynamics during digging operation," *Journal of Aerospace Engineering*, pp. 10–18, January 1996.

[17] Kontz, M., "Haptic enhancement of operator capabilities in hydraulic equipment," Master's thesis, Georgia Institute of Technology, 2002.

[18] Krishnaswamy, K. and Li, P., "Multi degree of freedom passive teleoperation of a hydraulic backhoe using dynamic passive valve," in *Proceedings of the American Control Conference*, vol. 5, (Arlington, VA), pp. 3932–3937, 2001.

[19] Langreth, R., "Smart shovel." Popular Science Magazine, June 1992. p. 82-84, 108-109.

[20] Lawrence, P. D., Salcudean, S. E., Sepehri, N., Chan, D., Bachmann, S., Parker, N., Zhu, M., and Frenette, R., "Coordinated and force-feedback control of hydraulic excavators," in *4th International Symposium on Control and Information Sciences*, (Stanford, CA), pp. 181–194, Experimental Robotics IV, June 30 - July 2 1995.

[21] Lipkin, H. and Duffy, J., "Sir robert stawell ball and methodologies of modern screw theory," *Journal of Mechanical Engineering Science, Proc Instn Mech Engrs*, vol. 216, 2002. Part C.

[22] Love, L. J. and Book, W. J., "Force reflecting teleoperation with adaptive impedance control," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, pp. 159–165, February 2004. ISSN 1083-4419.

[23] Luengo, O. and Barrientos, A., "Telemanipulation and supervisory control of a backhoe excavator.," in *SPIE Conference on Telemanipulator and Telepresence Technologies V*, (Boston, MA), pp. 24–31, Nov. 1998.

[24] Margolis, D. and Shim, T., "Instability due to interacting hydraulic and mechanical dynamics in backhoes," *Journal of Dynamic Systems, Measurement, and Control*, vol. 25, pp. 497–504, September 2003.

[25] Merritt, H. E., *Hydraulic Control Systems*. New York: John Wiley & Sons, 1967.

[26] MOOG, Inc., *Servovalves and Servo-Proportional Valves Product Line Overview*. http://www.moog.com.

[27] Nguyen, Q. H., Ha, Q. P., Rye, D. C., and Durrant-Whyte, H. F., "Force/position tracking for electrohydraulic systems of a robotic excavator," in *Proceedings of the 39^{th} IEEE Conference on Decision and Control*, (Sydney, Australia), pp. 5224–5229, Dec. 2000.

[28] Papadopoulos, E., Mu, B., and Frenette, R., "On modeling, identification, and control of a heavy-duty electrohydraulic harvester manipulator," *IEEE/ASME Transactions on Mechantronics*, vol. 8, June 2003.

[29] Peshkin, M. A., Colgate, J. E., Wannasuphoprasit, W., Moore, C. A., Gillespie, R. B., and Akella, P., "Cobot architecture," *IEEE Transactions on Robotics and Automation*, vol. 17, August 2001.

[30] Rossignac, J., Allen, M., Book, W. J., Glezer, A., Ebert-Uphoff, I., Shaw, C., Rosen, D., Askins, S., Bai, J., Bosscher, P., Gargus, J., Kim, B. M., Llamas, I., Nguyen, A., Yuan, G., and Zhu, H., "Finger sculpting with digital clay: 3d shape input and output through a computer-controlled real surface," in *Proceedings of Shape Modeling International 2003*, pp. 229–231, May 2003. Coll. of Comput., Georgia Inst. of Technol., Atlanta, GA, USA.

[31] Salcudean, S. E., Hashtrudi-Zaad, K., Tafazoli, S., DiMaio, S. P., and Reboulet, C., "Bilateral matched-impedance teleoperation with application to excavator control," *IEEE Control Systems*, pp. 29–37, December 1999.

[32] Salcudean, S. E., Tafazoli, S., Hashtrudi-Zaad, K., and Lawrence, P. D., "Evaluation of impedance and teleoperation control of a hydraulic mini-excavator," in *5th International Symposium on Control and Information Sciences*, (Barcelona, CA), pp. 229–240, Experimental Robotics V, June 1997.

[33] Salcudean, S. E., Tafazoli, S., Lawrence, P. D., and Chau, I., "Impedance control of a teleoperated mini excavator," in *International Conference on Advanced Robotics*, (Monterey, CA), pp. 19–25, July 7-9 1997.

[34] Sauer-Danfoss, *PVG 32 Proportional Valves Technical Information.* http://www.sauer-danfoss.com.

[35] Sciavicco, L. and B. Siciliano, B., *Modeling and Control of Robot Manipulators.* London: Springer-Verlag, 2nd ed., 2001.

[36] SensAble Technologies, Inc., *GHOST SDK API Reference Version 4.0*, 1996-2002. http://www.s ensable.com.

[37] Singh, S., "State of the art in automation of earthmoving," *Journal of Aerospace Engineering*, pp. 179–188, Oct. 1997.

[38] Tafazoli, S., Lawrence, P. D., and Salcudean, S. E., "Identification of inertial and friction parameters for excavator arms," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, vol. 15, pp. 966–971, October 1999.

[39] Tsai, L. W., *Robot Analysis*, pp. 395–403. New York: John Wiley & Sons, 1999.

[40] Tzafestas, C. S., "Whole-hand kinesthetic feedback and haptic perception in dextrous virtual manipulation," *IEEE Transactions on Systems, Man & Cybernetics, Part A (Systems & Humans)*, vol. 33, pp. 100–113, Jan. 2003.

[41] Vaha, P. K. and Skibiniewski, M. J., "Dynamic model of excavator," *Journal of Aerospace Engineering*, vol. 6, pp. 148–158, 1993.

[42] Yan, J. and Salcudean, S. E., "Teleoperation controller design using H_∞ optimization with application to motion–scaling," *IEEE Transactions on Control Systems Technology*, vol. 4, pp. 244–258, May 1996.

VITA

Joseph G. Frankel earned his Bachelor's degree in Mechanical Engineering from the University of Idaho in 2002 and his Master's degree in Mechanical Engineering from the Georgia Institute of Technology in 2004 with a focus in Controls and Automation. Joe returned to college as a non–traditional student and single parent of two boys in 1997 after building and operating a micro–brewery and restaurant in Moscow, Idaho from 1991 to 1998. During college he spent two summers as a design engineering inten at the Argonne National Laboratory–West in Idaho Falls, Idaho, and four summers as a research assistant, two at the University of Idaho working with acoustic aerosol separators, and two summers at Georgia Tech, one contructing custom laboratory equipment, and the last summer working with haptics/hydraulics research. Joe is a native of the Pacific Northwest, and in addition to engineering, enjoys many outdoor activities including moutaineering, downhill skiing, and cycling. After graduating from Georgia Tech, Joe plans to return to the Pacific Northwest to get married to his long–time fiancee, Tami Bishop, and work as a motion control engineer at Electro-Scientific Industries in Portland, Oregon, designing controls for consumer electronics manufacturing equipment.