DEVELOPMENT OF A MULTI-PLATFORM SIMULATION FOR A PNEUMATICALLY-ACTUATED QUADRUPED ROBOT

A Thesis
Presented to
The Academic Faculty

by

Hannes G. Daepp

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

Georgia Institute of Technology
December 2011

DEVELOPMENT OF A MULTI-PLATFORM SIMULATION FOR A PNEUMATICALLY-ACTUATED QUADRUPED ROBOT

Approved by:

Dr. Wayne J. Book, Advisor
Department of Mechanical Engineering
Georgia Institute of Technology

Dr. Jun Ueda
Department of Mechanical Engineering
Georgia Institute of Technology

Dr. Christiaan Paredis
Department of Mechanical Engineering
Georgia Institute of Technology

Date Approved: 4 November 2011

To my Grandfather,

David M. Gorkin

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Wayne Book, and my committee members, Dr. Chris Paredis and Dr. Jun Ueda. This project also would not have been possible without the aid of Michael Valente, who constructed and redesigned the mechanical components of the test platforms used for validation of the simulation model. Additionally, I would like to express my gratitude to friends, staff, and fellow students for their help, suggestions, and encouragement, especially JD Huggins, Ryder Winck, Brian Post, Mark Elton, Heather Humphreys, Josh Schultz, Alek Kerzhner, Longke Wang, and Aaron Enes. Finally, I would like to thank the Center for Compact and Efficient Fluid Power and the National Defense Science and Engineering Graduate Fellowship Team/Office of Naval Research for providing the financial support that made the thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Meaning
B_{00}	Transformation Matrix from local robot origin to shoulder
Δ	Distance from local robot origin to leg pair
x_{offset}	Offset of leg shoulder from robot thorax
d_1	Orthogonal distance from shoulder to coxa
a_1	Length of coxa
a_2	Length of femur
a_3	Length of tibia
x_{offset}	Offset of leg shoulder from robot thorax
ϕ	Rotation of 0 or 90° about the local robot origin
θ_0	Angle of rotation of robot shoulder
θ_1	Rotation of Alpha joint
θ_2	Rotation of Beta Joint
θ_3	Rotation of Gamma Joint
m	Mass
$\dot{m}$	Mass flow
u	Input Voltage
A(u)	Orifice Area as a function of input voltage
c_d	Discharge coefficient
P	Pressure
P_u	Upstream Pressure
P_d	Downstream Pressure
P_{Supply}	Supply pressure
$P_{Chamber}$	Cylinder chamber pressure
P_{Cr}	Critical pressure ratio

Symbol	Meaning
T	Temperature
T_u	Upstream temperature
T_d	Downstream temperature
$Phi(P_d, P_u)$	Function within mass flow equation
x	Piston position
x_{abs}	Absolute piston position, which includes the dead space in the cap side
$\dot{x}$	Velocity of cylinder piston
C_1	Constant for mass flow that is a function of R and k
C_2	Constant for mass flow that is a function of R and k
R	Universal Gas Constant
k	Ratio of Specific Heats
c_p	Specific Heat
K_0	Constant
A_0	Constant
τ	Time constant
$F_{Friction}$	Friction force
F_C	Coulomb friction
F_S	Static friction, Stiction
C_V	Coefficient of viscous friction
μ	Coefficient of friction for dry friction model
N	Normal force
k_{tanh}	Tanh coefficient
v	Piston velocity
v_S	Sliding speed coefficient

Symbol	Meaning
P_{input}	Modeled varying supply and exhaust pressures for valve model
C_{Scale}	Function of pressure cutoffs and input pressures
K_{Press}	Function of pressure cutoffs
x_{offset}	Pressure model x-axis offset
y_{offset}	Pressure model y-axis offset
P_S	Actual supply pressure
P_E	Actual exhaust pressure
C_{HP}	High pressure cutoff voltage
C_{LP}	Low pressure cutoff voltage
l_1 - l_5	Constant distances used to find relationship of rotation angle to piston position in Chapter 5
ϕ_1	Angle constant
ϕ_2	Angle constant
λ_1	Varying angle used to find moment arm for force-torque conversions
θ_α, θ_β, θ_γ	Measured rotation of Alpha, Beta, and Gamma joints
γ_α, γ_β, γ_γ	Angle of Alpha, Beta, Gamma joints needed for Law of Cosines
F	Actuator force output
T_α, T_β, T_γ	Simulated actuator torque output

SUMMARY

Successful development of mechatronic systems requires a combination of targeted hardware and software design. The compact rescue robot (CRR), a quadruped pneumatically-actuated walking robot that seeks to use the benefits garnered from pneumatic power, is a prime example of such a system. A simulation has therefore been developed that models the dynamics of the robot and its interaction with the environment. However, development of an entirely new dynamic simulation specific to the system is not practical. Instead, the simulation combines a MATLAB/Simulink actuator simulation with a readily available C++ dynamics engine. This multi-platform approach results in additional incurred challenges due to the transfer of data between the platforms. As a result, the system developed here is designed in the fashion that provides the best balance of realistic behavior, model integrity, and practicality.

This thesis discusses development and testing of a simulation that uses an analytically derived valve model to enable a user to examine the impacts of pneumatic actuation on a walking robot. An actuator model is developed using classical fluid circuit modeling together with nonlinear area and pressure curves to model the valve and a Stribeck-Tanh model to characterize the effects of friction on the cylinder. The valve model is designed in Simulink and validated on a single Degree-of-Freedom test rig.

This actuator model is then interfaced with SrLib, a dynamics library that computes dynamics of the robot and interactions with the environment, and validated through comparisons with a CRR prototype. Conclusions are focused on the final

composition of the simulation, its performance and limitations, and the benefits it offers to the system as a whole.

CHAPTER I

INTRODUCTION

In the wake of catastrophic disasters, rescue teams are forced to deal with harsh terrain, limited resources, and minimal time for action. This is the type of scenario in which rescue robots are used. Though many researchers are working to enhance the role of robots in disaster recovery [19, 39, 40, 45], the focus is often placed on endurance and search [37] rather than actual rescue [45]. Additionally, most current rescue robots are electrically actuated and only capable of exerting significant force at the cost of high weight [37].

In 2006, in an attempt to create a more capable rescue robot, the Center for Compact and Efficient Fluid Power (CCEFP) created a testbed called the Compact Rescue Robot (CRR). The CRR combines the high force and power density of pneumatic actuation [9, 10, 44, 47, 52, 55] with compact, lightweight power sources [22, 43, 44] to produce a lightweight, quadruped robot capable of high force output during several hours of untethered operation. The robot, which uses legged motion because of its known effectiveness on difficult terrain [19], is tele-operated via a user interface with audio-visual and haptic feedback. Two prototypes have been constructed, one with two legs and another with four. However, while the two-legged version can only walk with the aid of training wheels, the four-legged robot is plagued by reliability issues. Both robots are also subject to typical hardware design constraints that limit radical design changes because of potentially catastrophic setbacks resulting from possible failure.

In 2008, in response to these challenges, a simulation was developed to provide an additional design tool to model the robot's behavior given position commands

[29]. However, the simulation represented joints as proportionally controlled servo motors capable of infinite torque, essentially limiting it to a kinematic simulation useful only for development of high level control strategies [13]. This thesis focuses on the development and implementation of an improved simulation that combines the simulated robot dynamics with a model of the pneumatic actuator dynamics, thereby providing a valuable new design tool to further CRR development.

1.1 Motivation

The simulation presents a basis for evaluation of fluid power in legged robotics by coupling modeling of pneumatic actuation and a dynamic environment simulation. Since the simulation of the robot and environmental interactions is solved in real-time, it is possible to examine altered dynamics and effectiveness of control schemes following system modification. Similarly, compact power sources can be tested by modeling their weights and observing the amount of time it takes for the robot's actuators to have consumed the total amount of provided fuel. In simulation, design changes can be made safely and easily, and the impacts can be seen almost immediately.

The simulation also provides flexibility in design of the operator interface: The ease of modification and testing on the simulation make it possible to safely alter the system design and operator interface hardware and software in parallel, quickly viewing the change in overall performance, and determining the right combination of parameters to encourage the best operator performance. Additionally, the benefits of haptic feedback are preserved by providing the operator with a sense of the impact of pneumatic actuation through force feedback.

1.2 Research Objectives

The simulation is constructed to be a balance of usability and performance. The goal is to produce a simulation that is easily modifiable using actual, physical parameters

specific to the actuator or robot, applicable across a range of situations, and implementable on existing, commonly available hardware and software. The simulation integrates an analytically derived model of a pneumatic actuator with the robot's overall dynamics, using the two-legged CRR prototype to verify the results.

The simulation performance must reflect the behavior of the system to ensure that it can be used as a substitute for the physical robot during the design process. However, perfect correspondence between simulation and reality is impossible. Instead, emphasis is placed on accuracy of the overall behavior of the pneumatic system, rather than on the accuracy of the individual actuators. The model is derived analytically rather than through system identification because of the ability to obtain a model that depends on known, measurable actuator and robot parameters, which can be edited to fit the simulation to different tasks and hardware modifications.

For convenience of construction, the simulation is composed of interfaced existing software packages. The actuator model is written in Simulink and implemented on a realtime computer running xPC target and connected, via a local network, to SrLib, a free and readily available dynamics library running in C++/openGL.

The overall objective is to produce a simulation that closely captures the effects of pneumatic actuation on behavior of the robot using a system that can be easily modified and conveniently constructed. Results will focus not only on the accuracy of the system, but also on the modeling and implementation decisions, the simulation's limitations, and suggested improvements.

CHAPTER II

BACKGROUND AND LITERATURE REVIEW

The design choices for the CRR are discussed in detail in [23] and [55], but much of the reasoning can be summarized by the review of the value of legged locomotion and pneumatic actuation discussed in this chapter. Additionally, an overview of pneumatic modeling approaches and simulation platforms provides a background and justification for the combination of tools used within this thesis.

2.1 Legged Locomotion

The success of legged motion in nature and on a range of terrains [19] has made it a popular research topic within robotics. Numerous efforts have been made to analyze legged motion and replicate it in engineering, with testbeds ranging from quadrupeds [16, 17, 50], to hexapods [11, 25, 57], to vehicles with eight legs [54] or more. A wide span of research applications has served to demonstrate the versatility of legged locomotion, using platforms like climbers for mudslides and earthquakes [12, 39], packhorses for military aid [42], and manipulators for maintenance of nuclear systems [33].

Most early research on legged systems focused on the construction of walking machines [25, 26, 49, 57] and the development of basic gaits [34, 35]. Later efforts studied the challenges that arise from gait implementation: avoidance of kinematic singularities [38], transitions between gait types [28], and ways to circumvent deadlock (the case where no further forward motion is possible) [18], among others.

The CRR has walked successfully in the past, using pre-programmed gaits [55] and a controller that makes use of gain scheduling to vary actuator control effort as the leg passes through different kinematic configurations. To represent this kind of

robot behavior, the actuator accuracy should be preserved throughout the switching process, a condition best achieved by an analytically derived actuator model that functions across an array of situations.

2.1.1 Pneumatic Actuation

Pneumatic actuation has been widely employed in industry [46] because of its many benefits, including power and force density, clean, safe actuation and low cost [9, 30, 52], especially compared to electric motors [8, 30]. These benefits can be enhanced through the use of high pressures, as exemplified by multiple CCEFP projects intended for future implementation on the CRR that are aimed at creating compact, lightweight power sources [22, 43, 44]. Pneumatic actuation has also been used in past legged robots [31, 32, 41], where its light weight and natural compliance aids in the overall design and behavior of walking machines.

However, the compressible and highly nonlinear nature of pneumatic systems has proven to be a limiting factor in the wider adaptation of pneumatics as an actuation choice. Researchers have struggled to find a simple control strategy such as PID or PVA control that produces high quality position tracking results without also requiring considerable mechanical modification of the actuator [30, 56]. Instead, a variety of advanced control techniques that are more computationally costly and sensor heavy have been tested, such as fuzzy state feedback [46], impedance control [58], neural networks [10], and adaptive control [8], though the best results have generally been achieved using sliding mode controllers [9]. Bone and Shu [9] offer a comprehensive review of the approaches and results of some of the most prominent efforts at pneumatic control. While many of these controllers demonstrate promising accuracy in tracking, most of them limit their validations to simple, single degree-of-freedom linear actuators. The CRR requires a controller that can maintain robustness despite the varying dynamics that result from a legged robot that can be in either

stance or swing, or the various applied forces that must be handled. This was dealt with previously by Guerriero through the use of pressure feedback and gain scheduling [23].

These limitations affect the overall performance of the robot and need to be appropriately understood in order to produce a robot capable of the accuracy and reliability necessary for search and rescue. The development of an actuator model makes it possible to not only observe the final position and pressure response, but also to examine how controllers and dynamics are affected by individual parameters in the actuator. Furthermore, the integration of robot dynamics will help help to guide control and interface design along a path specific to the desired application.

2.1.2 Pneumatic Modeling

A typical pneumatic actuator consists of a valve and cylinder. Several standard models have been developed, including ones by the Instrument Society of America (ISA) [51], National Fluid Power Association (NFPA) [27], and a similar standard model used in academia [6, 47, 52]. Each of these models follows the same basic principles, but varies primarily in its definition of modeling constants and associated assumptions. Researchers have also developed several non-traditional models, such as a high integrity mass-based model [52] or lower-order linearized models [56]. However, these approaches are primarily used for targeted usage in controls or to illustrate a particular point, and are not as well suited for general, open loop dynamic modeling.

While several papers have discussed the derivations and usage of models for pneumatic actuation [10, 27, 47, 51, 52], one of the most explicit and useful papers is a 2005 publication by Bone and Shu [47], in which they discuss the derivation of a model for a specific physical test setup in detail. The authors focus in particular on the matching of model data, though they replace the standard mass flow equations with a curve fit version of their own that matches the measured data more closely.

They also use a simple friction model, and discuss the derivation of stiction forces from the pressure measurements for use in a simple velocity Coulomb-viscous friction model.

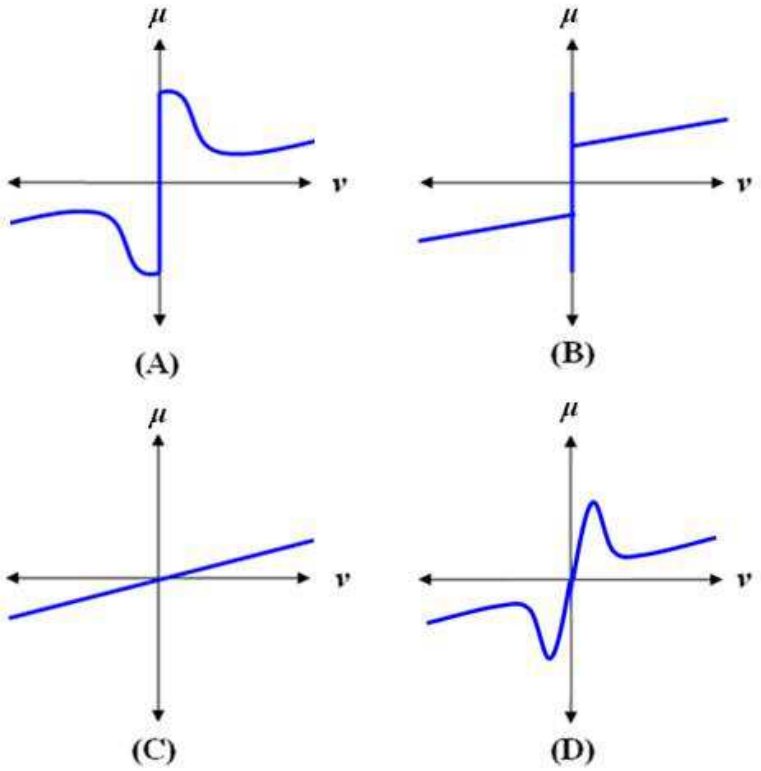

Figure 1: Friction models

Friction is a major area of concern in pneumatic modeling, caused by interactions between the piston seals and cylinder housing. Several approaches have been developed, ranging in complexity and accuracy [7, 47, 58]. The approach used by Bone and Shu is simple, but is difficult to use in practice because of the discontinuity that occurs at zero velocity. Andersson [7] provided an overview of challenges and potential solutions to the discontinuity and other problems, noting the trade-offs of realism and ease of modeling among different friction characterizations. Figure 1 shows several variations on velocity-dependent friction models, a topic that will be addressed in greater detail in section 4.2.2.

2.2 Simulation

Simulation is a popular tool in robotics development because it enables fast and effective parameter adjustment and design testing, low cost, increased safety and overall improvement of the decision process [14, 20, 21, 59]. Several established packages exist that allow modeling of different actuator types in large systems, such as Dymola, MSC-Adams, SimulationX, and Simscape/Simulink. Products like Dymola and SimulationX use the Modelica language to broadly define systems, providing considerable breadth of application [21]. Though many such simulation packages are limited to offline solutions, several commercially available tools exist that convert these non-realtime models to realtime using dedicated hardware [20, 21]. Modelica-based models can be run on a target using a special Dymola compiler [2], while Simulink programs can use Simulink Coder to run on xPC Target, MATLAB's realtime target operating system [4]. This thesis uses a Simulink based model running on a dedicated target PC. The model's structure is based on a set of generalized equations, together with customized variations to more accurately capture details like equivalent valve orifice area and cylinder friction. Simulink is also the program used to interface with and control the actual robot, and was thereby chosen over other options to maintain consistency and convenience of use.

Modeling of the robot dynamics and environmental interactions could also be done in programs like Dymola or Simulink, but these software packages have been shown to be ill-equipped for the dynamic challenges of walking and surface contacts [53]. Instead, several open source simulation platforms have been developed that are specifically aimed at creating robots from rigid body and joint libraries and modeling their behaviors [1, 24, 48].

One such platform is Seoul National University's Robotics Lab Library (SrLib) [24], which provides a solution in the form of a real-time multi-body rigid dynamics simulation with collision detection and contact behavior. Systems are created in

SrLib using links and joints from the library, then run in an environment composed of multiple systems. The simulation uses Lie Groups and recursive dynamics, coupled with a Projected Gauss Seidel (PGS) solver for Linear Complementarity Problems (LCP) for contact and friction modeling, to produce a solution [24].

SrLib was chosen because of its realtime operability, its modular structure, and the support network of SrLib developers located on the Georgia Tech campus when a version of the simulation without actuator dynamics was originally implemented in 2009. Additionally, SrLib is configured for TCP/IP communication, enabling it to interface easily with other platforms. A UDP connection across a local network was used to establish communication between the Simulink actuator model and SrLib robotics simulation, forming the complete dynamic simulation developed in this thesis.

CHAPTER III

SYSTEM OVERVIEW

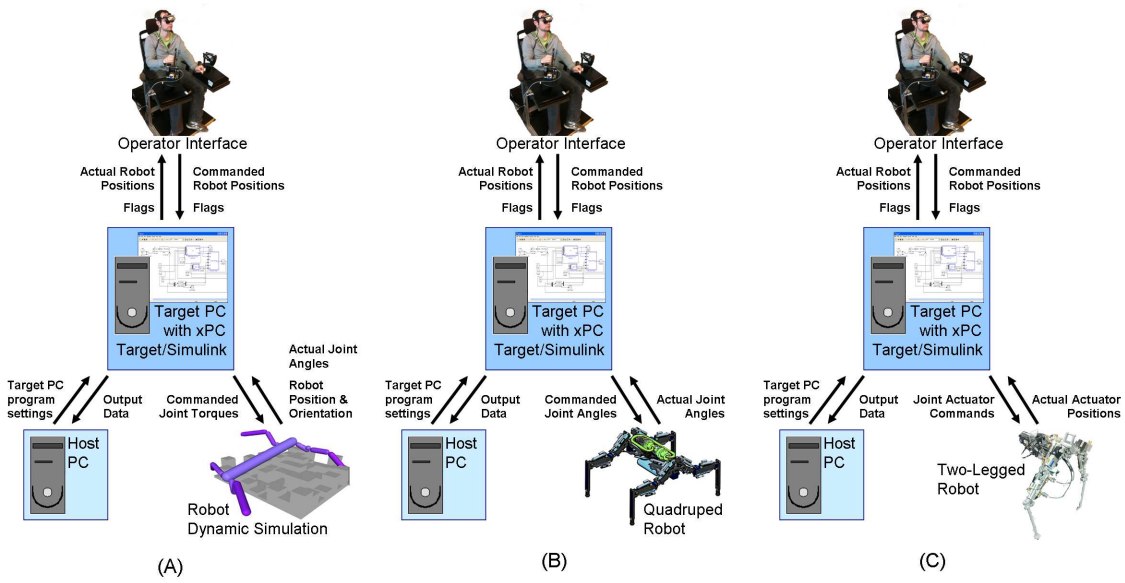

Figure 2: System Components with robot

On the physical testbed, depicted in Figures 2b and c, there are four primary components:

1. Operator Interface
2. Target PC and Simulink Software
3. Host PC for Target
4. Robot

Each of these components serves a specific function towards operation of the robot. The operator interface allows a user to manipulate and move the robot in free space.

The target PC contains the primary Simulink code that is used to translate commands from the operator to robot actuator commands and provided feedback to the operator. The host PC is used to run the target to collect data processed during experiments or other operations. The final component, the robot, receives commands from the operator and returns actual information such as position, joint angles, and orientation.

From a high level perspective, replacement of the physical robot with the actuator model and dynamic simulation is simple, since the commands sent to and from the robot are the same as those exchanged by the simulation. An overview of the components and their interaction is shown in Figure 2a and discussed in detail in the following sections.

3.1 Rescue Robot Testbed

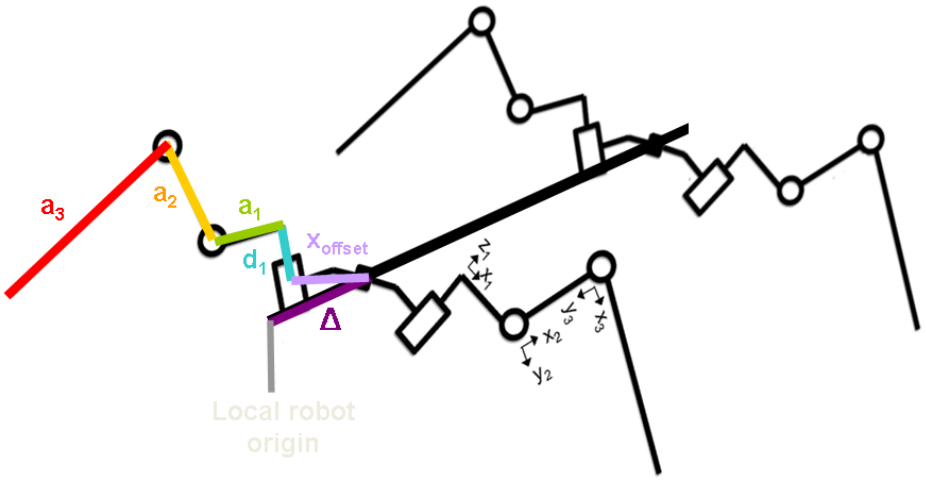

Figure 3: Robot diagram

Several versions of the robot have been developed for use in this system, including a four-legged prototype for mobility and manipulation testing, a simulated quadruped for design development, and a two-legged prototype for use with manipulation, simulation validation, and operator interface development. While the dimensions, actuation strategies, and purposes of these robots vary, they are related by the fact that they

are fundamentally of the same kinematic structure. That is, they each possess the form illustrated in Figure 3, such that each shoulder can be mapped from the local robot origin using the transformation

$$B_{00} = Rot_z(\phi)Trans_x(x_{offset})Trans_y(\Delta)Rot_y(\theta_0) \tag{3.1.1}$$

where ϕ is a rotation of 0 or 90° about the local robot origin. From the respective shoulders, then, each of the robot's legs can be described by the Denavit-Hartenberg parameters in Table 3.1.

Table 1: D-H Table for each leg. Asterisks mark a variable

Joint	θ	d	α	a
Shoulder	–	–	0	0
1	θ_1*	d_1	-90	a_1
2	θ_2*	0	0	a_2
3	θ_3*	0	0	a_3
4	0	0	–	–

The motivation for these kinematics is primarily based on past manipulator design and the overall goals of this particular testbed, as discussed extensively in theses by Guerriero and Wait [23, 55].

3.1.1 Quadruped Robot

Of the available hardware, the four-legged robot (Figure 4) is most relevant to eventual practical implementation. This robot uses 12 actuators controlled by independent microcontrollers that receive signals from a Target PC containing the primary Simulink software. The microcontrollers are found on Joint Control Units, or JCU's, located at each joint. Each JCU receives setpoints from the target and implements PVA controllers using position feedback from a rotary potentiometer – also processed by the JCU – to actuate the joint. The 12 JCU's are connected by a CANbus wire that is routed back to the target PC. This communication design results in a compact system that localizes low-level control tasks and is intended for eventual field use.

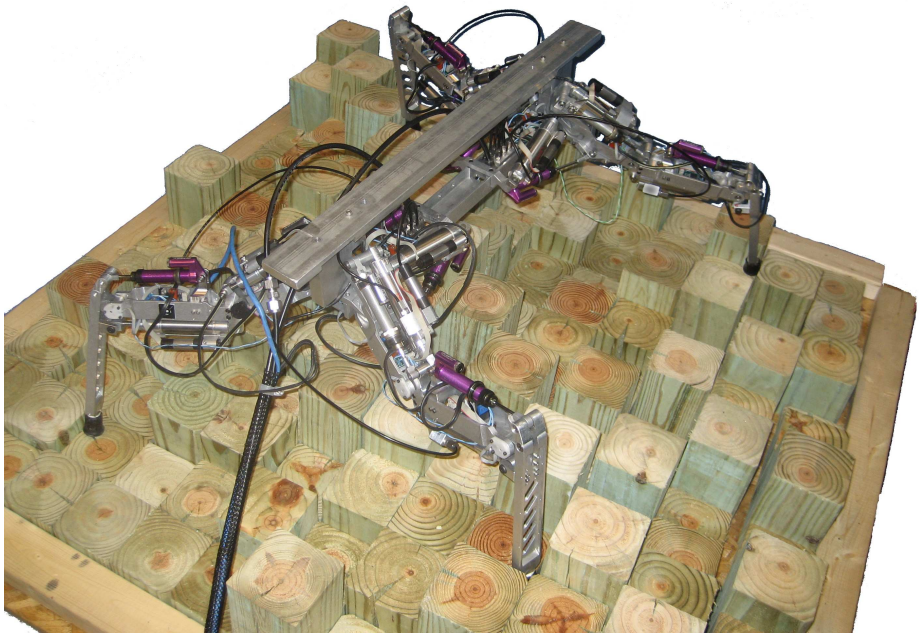

Figure 4: Quadruped robot resting atop an obstacle block

The actuation strategy is relatively simple: each joint features a cylinder-damper combination connected to a custom-built rotary valve. Whereas most pneumatic actuation strategies have required one or more pressure sensors in addition to position feedback, this approach minimizes the amount of required sensors, and functions by using the added effect of damping to help linearize the system and to raise the gain margin so that more control effort can be implemented [56]. While this strategy has been shown to improve control where only position feedback is used, its success pales when compared to more complicated techniques that utilize pressure and force loops. Accordingly, the other hardware available for use with the testbed makes use of a setup that is easily adaptable to more advanced control strategies.

3.1.2 Two-Legged Testbed

The two-legged testbed (Figure 5) was originally designed by Guerriero as an interim platform to test the feasibility of the operator interface. It has since developed into

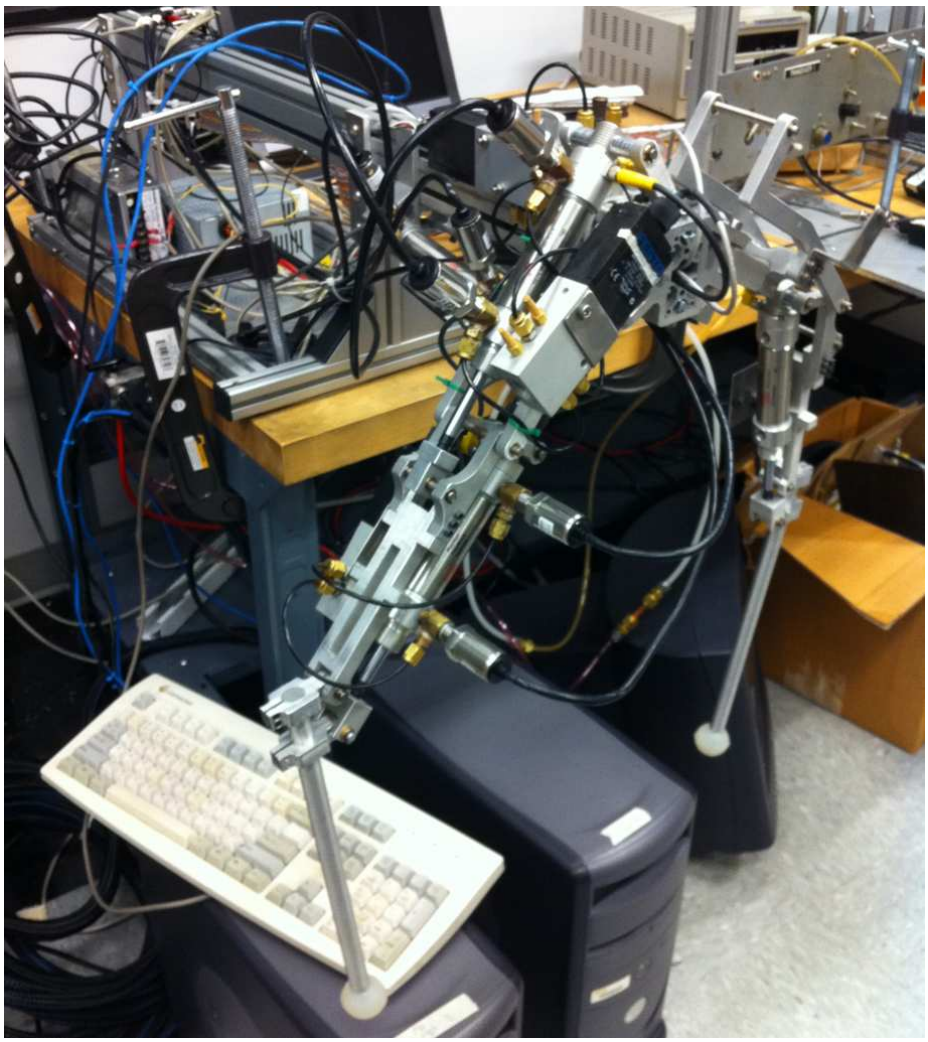

Figure 5: Two-legged robot

a system with which to study pneumatic control strategies for manipulation and tele-operation and to aid interface design. Unlike the quadruped prototype, the two-legged robot uses actuators without dampers, instead making use of position and pressure feedback. Valve input commands are issued from a target PC containing Simulink software that governs both high level and low level (joint control) tasks. This setup ensures a more modular approach than that of the quadruped, though it makes the system less inclined for practical implementation and more targeted towards laboratory testing of research goals.

3.2 Dynamic Simulation

The robot simulation consists of two parts: an actuator model that accepts a voltage input and outputs a command torque, and a C++/OpenGL model that accepts command torques from actuator models for each joint and provides robot dynamics (Figure 2a). The platforms interact via UDP connections sent between computers on a local network, as shown in Appendix D.

The simulated robot has the same dimensions and kinematics as the two-legged robot, but extended to a four-legged configuration. The simulated actuators were also modeled based on the physical ones found on the two-legged platform, which was available for use and more reliable for an extended time period than the quadruped prototype. Additionally, the two-legged robot uses valves and cylinders commonly found in industry and among pneumatic research testbeds [9, 47, 52], making its usage as simulation validation more relevant to a general audience.

Future application of the simulation to the quadruped robot is likely possible, though complicated due to the difference in actuator design. While the linear damper and reduced cylinder sizes would be relatively simple to model, the custom rotary valves, which possess different dynamics than the Festo MPYE-5 valves that were used, would require considerable extra effort to model. This type of redesign, while potentially valuable in examining the breadth of utility of the simulation, was not pursued because of the extensive data collection, reconsideration of key components required for its implementation, and inability to maintain the reliable behavior that would be required of the quadruped prototype.

3.3 Operator Interface

The robots described above all possess two or more legs, with the intent of achieving mobility and manipulation. To do so, they are coupled with an operator interface

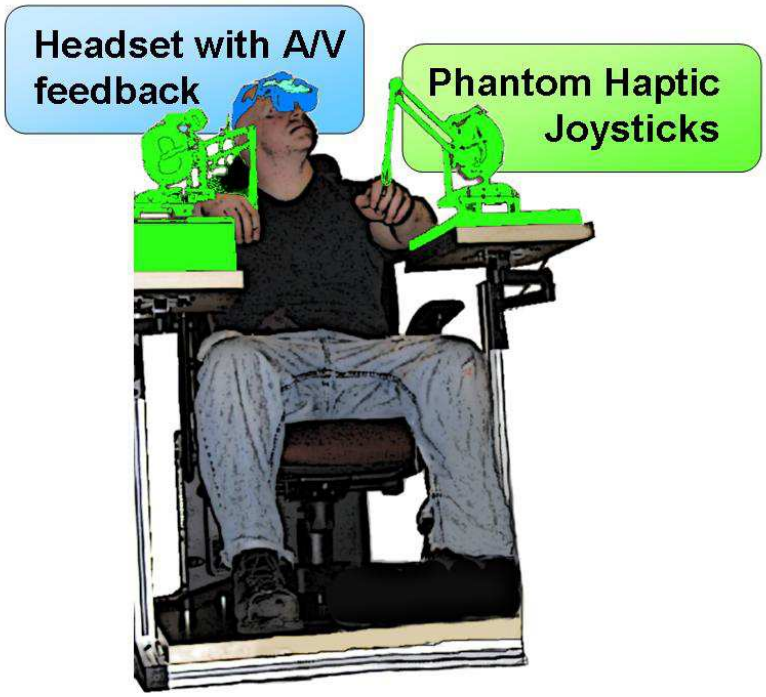

Figure 6: Operator Interface

(Figure 6) that inputs user commands from two Phantom haptic joysticks and returns audio-visual and haptic feedback. Each of the Phantom joysticks is capable of operating in six degrees of freedom, though only three are used. The motion of the end effectors of these joysticks is mapped to the end effectors of the robot via an offset and a scaling, and then converted to joint angle motions using inverse kinematics, as seen in the codes found in Appendix A. Additional user inputs from keyboard commands or other sources can be coded as numeric flags and transmitted for use as indicators or in state machines.

To map the robot's four legs to the operator, a high-level control strategy that combines haptic feedback with a constrained model-predictive controller was implemented. The controller used haptic feedback to guide placement of the front legs and provided automated rear leg motions, resulting in a gait the produced general forward motion without ever risking loss of static stability. The controller was tested on a

non-dynamic version of the simulation, in which the simulation simply accepted joint angles and used a proportional controller to actuate joints capable of exerting effectively infinite torque in a very short time period, resulting in nearly perfect dynamics. The results from these efforts where promising [13], but were limited in their value by a lack of realistic dynamics, another motivating factor for increased simulation realism.

CHAPTER IV

ACTUATOR SIMULATION DESIGN

A model for the actuator was developed using the "standard" academic model for a pneumatic actuator [6, 47, 52], based on a physical setup consisting of a valve and cylinder with position and pressure feedback. The model was designed and validated in MATLAB Simulink, then packaged as a single block that accepts voltage and returns output force. This block was in turn integrated with the complete dynamic simulation, leading to a full approximation of the system behavior that will be discussed in Chapter 5.

4.1 Simulink Actuator Model

The fundamental equations that define the actuator model consist of four primary components: a conversion of voltage input to orifice area, an equation for mass flow through an orifice, an energy balance to relate mass flow to change in pressure, and a force balance to derive output force from the pressure and friction components. The resulting model is a nonlinear, third order system as shown in Figure 7.

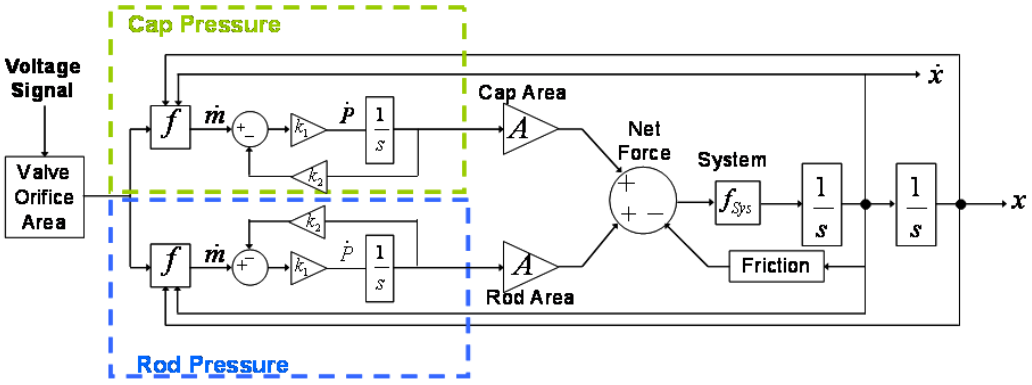

Figure 7: Actuator block diagram

4.1.1 Mass Flow Through an Orifice

Mass flow, $\dot{m}$, is typically calculated as a function of upstream and downstream pressure, orifice area, and a discharge coefficient, c_d, and several predefined constants, and is of the form

$$\dot{m} = A(u)c_d\Psi(P_d, P_u) \tag{4.1.1}$$

The choice of upstream and downstream pressure is dependent on the direction of flow, as defined by the valve command. For each chamber then, the mass flow can be defined as

$$\text{Charging (flow into the chamber): } P_d = P_{Supply}, P_u = P_{Chamber} \tag{4.1.2}$$

$$\text{Discharging (flow out of the chamber): } P_d = P_{Chamber}, P_u = P_{Atmosphere} \tag{4.1.3}$$

The function $\Psi(P_d, P_u)$ is a nonlinear piecewise-defined function that varies depending on the pressure ratio, which determines whether the system is experiencing choked or unchoked flow:

$$\text{Critical Pressure Ratio for air} \qquad P_{cr} = P_d/P_u = 0.528 \tag{4.1.4}$$

For $P_d/P_u > P_{cr}$ (un-choked flow):

$$\Psi(P_d, P_u) = C_1\frac{P_u}{\sqrt{T_u}}\left(\frac{P_d}{P_u}\right)^{1/k}\sqrt{1-\left(\frac{P_d}{P_u}\right)^{(k-1)/k}} \tag{4.1.5}$$

and for $P_d/P_u <= P_{cr}$ (choked flow):

$$\Psi(P_d, P_u) = C_2\frac{P_u}{\sqrt{T_u}} \tag{4.1.6}$$

where T_u and T_d refer to upstream and downstream temperature. These quantities are calculated using the ideal gas law and the instantaneous total mass and pressure in the cylinder

$$T = \frac{PAx_{abs}}{mR} \tag{4.1.7}$$

where A is the cross-sectional area of the chamber and x_{abs} is the absolute position of the rod, which includes the dead space inherent to each chamber of the cylinder. The constant terms C_1 and C_2 are functions of the universal gas constants R and the ratio of specific heats k, as shown in equation (4.1.8):

$$C_1 = \sqrt{\frac{2k}{R\,(k-1)}} \tag{4.1.8}$$

$$C_2 = \sqrt{\frac{k}{R\left(\frac{k+1}{2}\right)^{(k+1)/(k-1)}}} \tag{4.1.9}$$

For air, k = 1.4 and R = 287 J/Kg K.

4.1.2 Energy Balance

The previous equations govern the dynamics of the valve. The next step, an energy balance, defines the link that connects mass flow through the valve to changes in pressure within the chambers of the cylinder. Under an adiabatic assumption, which is generally acceptable for fast acting systems such as a pneumatic walking machines, the change in pressure in each chamber of the cylinder can be found using the ideal gas law:

$$\dot{P} = \frac{kRT\dot{m}}{x_{abs}A} - \frac{P\dot{x}}{x_{abs}}\left(\frac{kR}{c_p} + 1\right) \tag{4.1.10}$$

where c_p = 1012 J/Kg K is the specific heat of room temperature air. Equation (4.1.10) is applied to both sides of the cylinder and integrated to get rod-side and cap-side pressures .

4.1.3 Force Balance

Given the pressures in each side of the chamber, a force balance is used to determine the amount of force exerted by the actuator. Naturally, there are some losses that may affect the actual force experienced by the system. The most critical of these is friction, which is heavily dependent on the internal construction and lubrication of the cylinder, as well as the rates at which the system is excited, and will be discussed further in section 4.2.2.

The net force can then be summarized by equation (4.1.11):

$$F_{net} = P_{cap}A_{cap} - P_{rod}A_{rod} - P_{atm}A_{piston} - F_{friction} \tag{4.1.11}$$

where A_{cap} and A_{rod} refer to the cap-side and rod-side cylinder cross-sectional areas, A_{piston} refers to the area of the rod, and the pressures used are absolute.

4.2 Modeling Adjustments

Though seemingly straightforward, the equations introduced in sections 4.1.1 - 4.1.3 are complicated by key parameters that are difficult to model accurately and consistently. Notably, in the mass flow equation (eq. (4.1.1)), the discharge coefficient, c_d, and orifice area, $A(u)$, have been shown to be nonlinear functions of command voltage, and act as a scaling factor on the mass flow. In the force balance, the net output force is altered by friction, a nonlinear force that is best modeled as a velocity dependent function. These components were independently analyzed such that accurate models could be derived and integrated with the otherwise standard actuator model.

4.2.1 Data Collection from an Actuator Test Rig

In order to fit and validate the model, a test setup was required. The setup, shown in Figure 8, consists of a Festo MPYE-5 proportional directional control valve coupled with a 1.75" stroke Bimba PFC cylinder. The PFC model contains an internal linear potentiometer that uses a wiper in the rod to output the rod position. Three SSI Technologies pressure sensors located at the supply input to the valve and at the head of each chamber provided measurements for the supply, rod-side, and cap-side pressures. A force transducer, an ATI-ia Gamma model, was screwed to the end of the rod and modified to be able to carry additional mass using a threaded bar. For tests where the piston position needed to remain fixed, a spacer was placed between the mass attachment and the end of the cylinder, forcing the piston to remain at

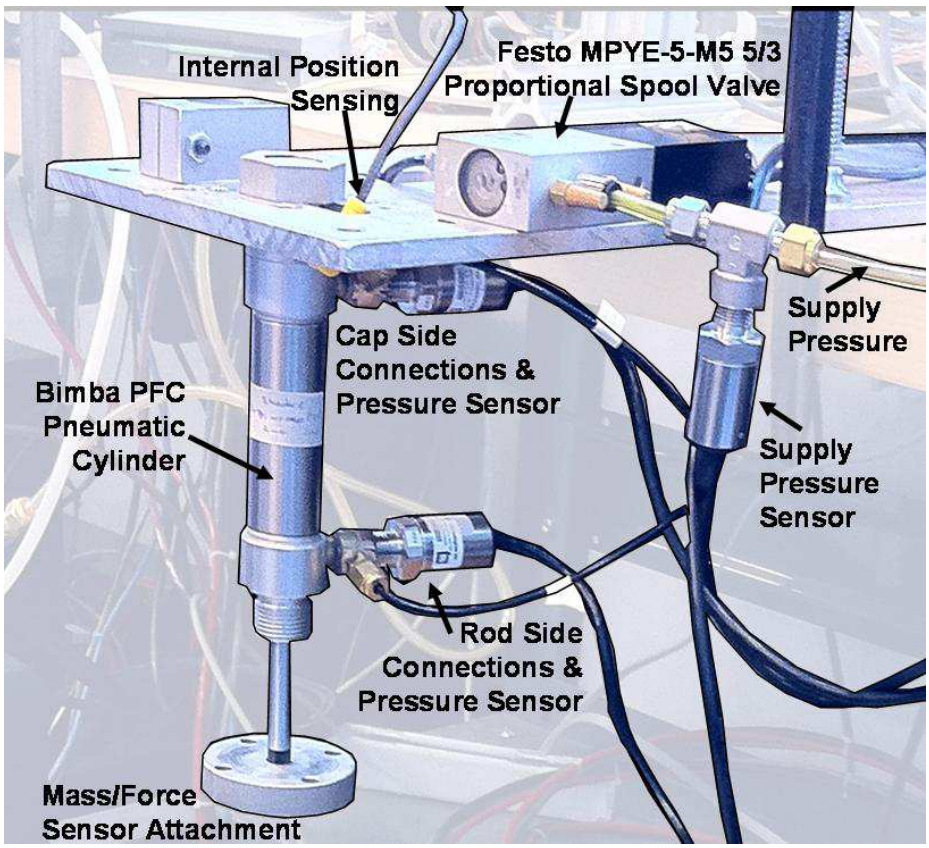

Figure 8: Test setup used to validate cylinder model. Note that the force sensor and mass attachment are not pictured here

its maximum extension throughout the test. Signal and operating power were provided by a separate power source, but the actuating commands and measured signals were sent and collected by a PC104 acting as the target PC for appropriate MATLAB/Simulink files. The PC104 uses two Diamond Systems cards to send and read signals – a DMM-32X-AT A/D card, and an RMM-1612-XT D/A card. Additionally, because the cylinder and valve were simply mounted to an aluminum board that was clamped to the table, the orientation could easily be changed by rotating the board adjusting the clamp configuration.

It is important here to specify some general terminology for use with the valve. The MPYE-5 series has a 0 to 10 V input range, referred to as the valve "input voltage", or "voltage command". Additionally, for some voltage-dependent signals in

this model, it was desirable to use a voltage range centered about zero, which spans -5 to 5 V and is referred to as the "centered voltage input". However, the valve flow is not actually centered midway through the range – instead, there is an offset voltage at which point flow through both ports of the valve is equal. This point in the voltage input command is called the valve offset. The specific voltage is called the "centered voltage offset" within the centered voltage command, or "voltage offset" within the regular valve voltage command range, and has the value of 0.25 V and 5.25 V for centered and un-centered versions, respectively. A detailed derivation will follow in section 4.2.3.

The Simulink program (Appendix A) provided the appropriate scaling and offset constants to convert the raw voltage signals to pressures, positions, and forces. However, even after these conversions, much of the data needed to be filtered to ensure that it could be worked with. Originally, low-pass filters were used to get rid of the higher frequency noise content, but this resulted in considerable time delay that became even more aggravated in the friction analysis, where it was necessary to also obtain and filter the velocity. Instead, the final analyses use the `filtfilt` command in MATLAB, which performs zero-phase digital filtering by processing the data offline in both the forward and reverse directions [3]. The filters used by `filtfilt` were Chebyshev type II Lowpass filters, designed using the MATLAB `filterbuilder` tool. The filters were chosen for each dataset by examining the frequency response using a signal analysis script documented in Appendix A.

4.2.2 Friction

Friction is a recurring topic in modeling and control of pneumatic cylinders. The primary challenge is its inherently nonlinear behavior, which is perhaps best described by breaking it down into several components that define the general form seen in Figure 9. Static friction, or stiction, represents the forces experienced when the

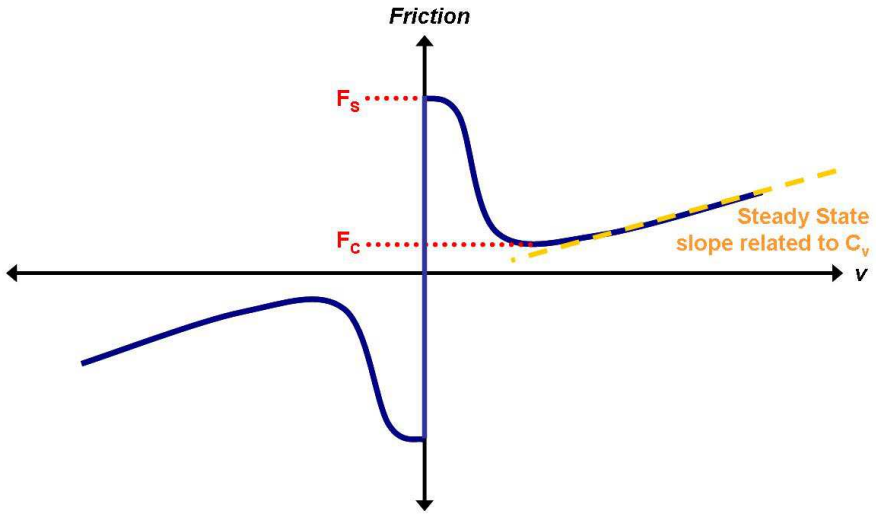

Figure 9: General form of friction

cylinder first begins to move, and is denoted by F_S in the figure. This typically results from the particular seal or lubrication used and the contact between the piston and the cylinder. Additionally, stiction may vary if taken at the end stops or from some midpoint, though it is typically regarded as one fixed magnitude. Once the rod begins to move, it experiences Coulomb (F_C) and Viscous ($C_V\dot{x}$) friction, resulting not only from the interaction of cylinder components, but also possibly from the compressibility of the gas.

Efforts by past researchers to model these effects vary in complexity. While many researchers have simply approximated friction with a viscous damping term [6, 7], several advanced models have been developed that looked at the behavior of two contacting surfaces on a more sophisticated level, examining the behavior of the contacts on a microscopic scale. Many of these models, such as LuGre and Stribeck, though popular, are of limited use due to the presence of a discontinuity at zero velocity. Alternative methods such as Dankowicz have been developed to provide accurate, continuous friction models for very small velocities [7]. Such precise models,

however, are used at the cost of complex development, consisting of detailed measurements, analysis and computation on a material level. This kind of precision is largely unnecessary for human-scale systems such as this testbed. Instead, more practical techniques developed by past researchers have been applied, using variations on established viscous and Coulomb friction models [7, 47] and with empirically derived components [6, 47].

4.2.2.1 Coulomb and Coulomb-Viscous Friction Models

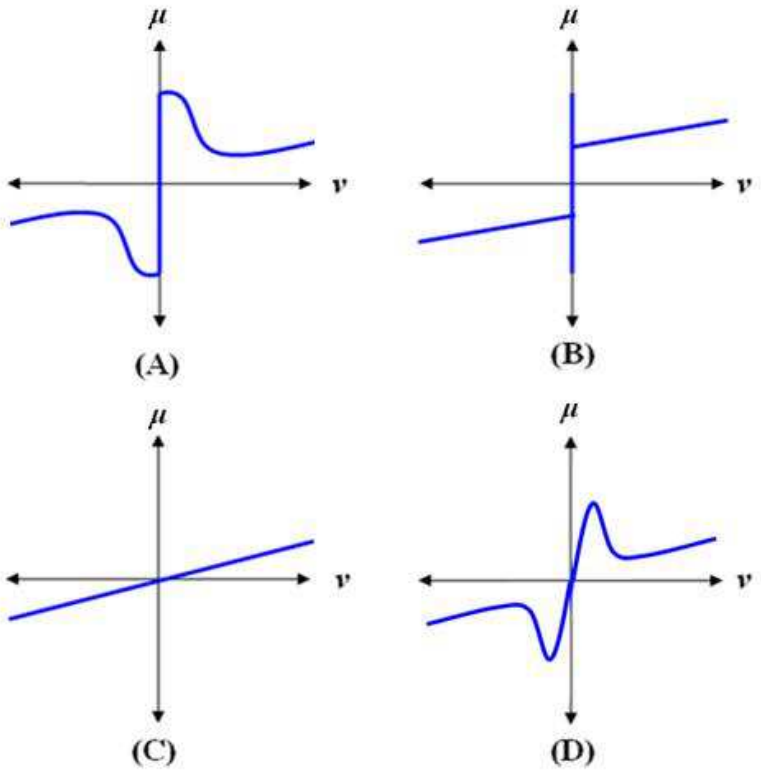

Figure 10: Overview of friction models

One of the simplest friction models is a basic Coulomb friction model, characterized by the equation

$$F_{Friction} = \begin{cases} F_C sign(\dot{x}) & \text{if } |\dot{x}| > 0 \\ F_{Cyl} & \text{if } \dot{x} = 0 \text{ and } F_{Cyl} < F_C \end{cases} \tag{4.2.1}$$

where F_{Cyl} is the net force the cylinder piston experiences due to chamber pressures and F_C is the Coulomb sliding friction force. For the system described in section 4.2.1,

$$F_C = F_{Cyl} + mg \tag{4.2.2}$$

and the Coulomb sliding friction force is defined by the coefficient of friction μ and the normal force N to be

$$F_C = \mu N \tag{4.2.3}$$

However, this model, often referred to as dry friction, does not account for the viscous component of friction. Instead, a different approach that combines Coulomb and viscous forces can be applied [47], resulting in a model similar in form to a simplified Stribeck Curve, as seen in Figure 10b. Assuming a constant normal force, N, this friction model can be defined as

$$F_{Friction} = \begin{cases} F_{Cyl} & \text{if } \dot{x} = 0 \text{ and } |F_{Cyl}| < F_S \\ F_S - F_{Cyl} & \text{if } \dot{x} = 0 \text{ and } |F_{Cyl}| > F_S \\ F_C sign(\dot{x}) - C_V \dot{x} & \text{if } |\dot{x}| > 0 \end{cases} \tag{4.2.4}$$

where the maximum stiction force, F_S, the Coulomb force, F_C, and the coefficient of viscous friction, C_V are determined experimentally as done in Shu and Bone [47] and discussed in section 4.2.2.4.

4.2.2.2 Viscous Friction Model

A simple and frequently employed alternative to the afore-mentioned models is the approximate friction with a single velocity dependent term (Figure 10c):

$$F_{Friction} = C_V \dot{x} \tag{4.2.5}$$

where C_V is the coefficient of viscous friction, found empirically. This friction model is certainly adequate for cases of low stiction and in some control applications, where model error in favor of simple design is both justified and expected, but is likely inappropriate for cases where stiction plays a significant role.

4.2.2.3 Stribeck-Tanh Model

To avoid the discontinuity incurred by more realistic friction models while still obtaining more accurate behavior than a sole viscous friction model can provide, a friction model that approximates the stiction region with a continuous segment was applied. Literature demonstrates several techniques for approximating this region [7], such as through the use of a scaled hyperbolic tangent function, or by substituting a steep, linear, velocity dependent friction curve that is saturated once the velocity becomes greater than some minimum value. The latter of these two was applied to an earlier cylinder [15], where it showed promising results. However, the model was limited by its piecewise continuity – edges occurring at the start of each defined region made data harder to fit.

Instead, the analysis was performed with a smoother continuously differentiable form that is based around a hyperbolic tangent curve, shown in Figure 10d and defined by equation (4.2.6):

$$F_{Friction} = \left[F_C + (F_S - F_C)\, e^{-(|v|/v_S)^i}\right] tanh(k_{tanh} v) + C_v v \tag{4.2.6}$$

where F_C and F_S represent Coulomb and static friction, respectively, v is the velocity, v_S is the sliding speed coefficient, i is the exponent, k_{tanh} is the tanh coefficient, and C_v is the coefficient of viscous frictions. While F_C, F_S, and C_v can be found experimentally, the other parameters are matched to the data based on the overall fit. Though the components interact to form a general friction curve, the contributions of each constant are generally focused on one individual aspect of the curve, as illustrated in Figure 11.

The Stiction and Coulomb friction components define the magnitudes of the curve at its respective friction maximum and post-stiction minimum. The sliding speed coefficient determines the velocity threshold at which stiction transitions to other forms of friction, and the exponent, i, alters the change in slope of this region as it

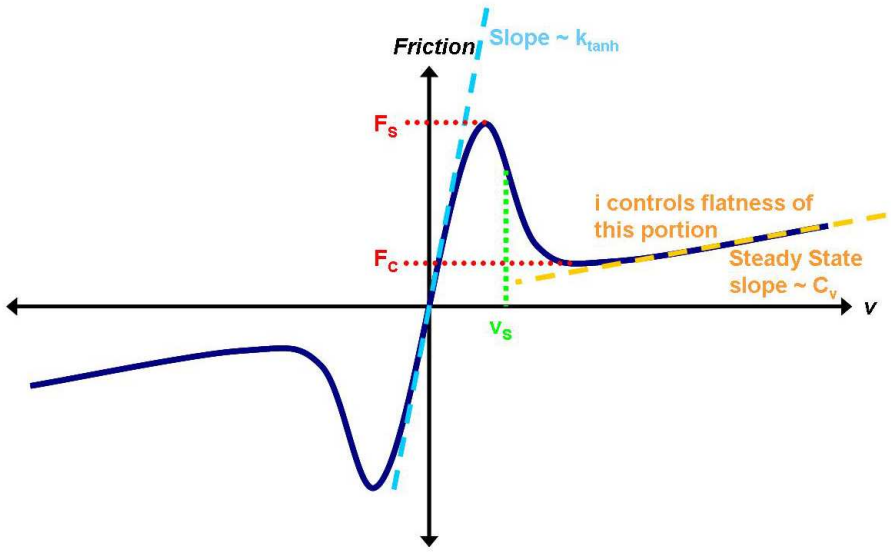

Figure 11: Contribution of coefficients to Stribeck-Tanh friction model

varies from post-stiction drop to steady state viscous and Coulomb friction. A higher k_{tanh} results in a steeper slope around zero velocity, between the friction peaks, and higher C_v raises the slope of friction in the steady state.

4.2.2.4 Friction Component Measurement

Certain friction parameters, such as the stiction, Coulomb friction, and viscous components can be measured by observing pressure and position curves. Looking at a typical open loop step, shown in Figure 12, it can be seen that initially, the pressures are in their steady state equilibrium for a particular voltage. Once the step command is received, the valve orifice opens and the pressures begin to change, eventually reaching a peak that corresponds to the time at which the rod first begins to move. This point is referred to in this thesis as a "stiction peak", and represents the pressures needed to first cause the rod to move. The associated stiction values can be found by taking the net force from the differential pressures and subtracting any offsets due to mass of the rod and its load. Similarly, the region after the peak represents the friction region governed by Coulomb and viscous friction. Since a step corresponds to a

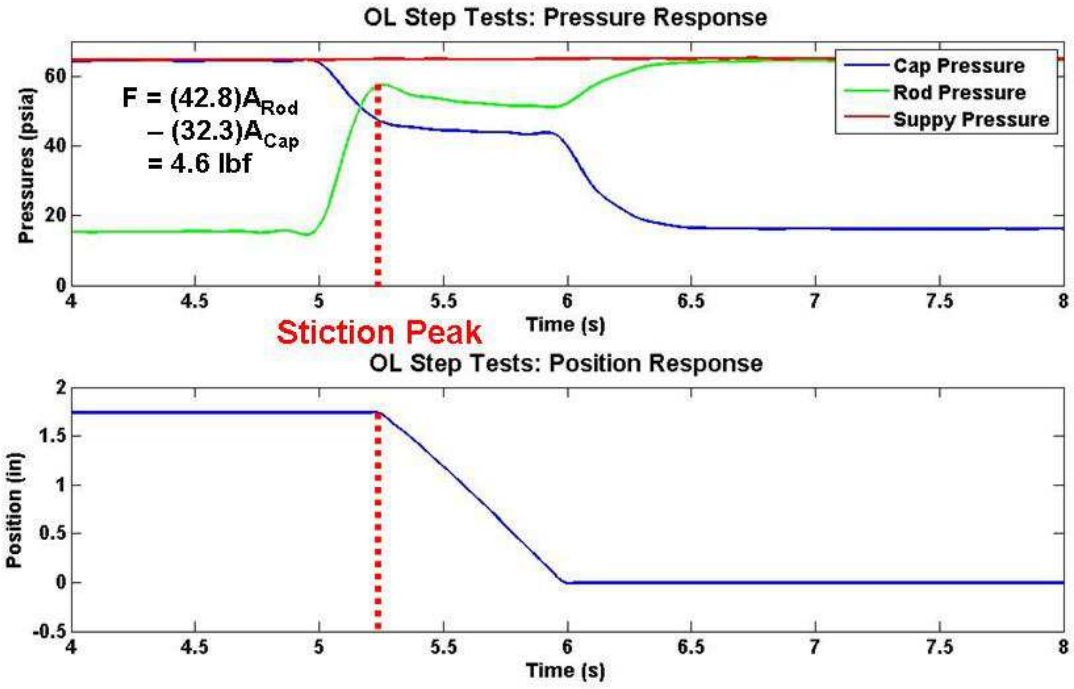

Figure 12: Determination of friction components from pressure curves

constant orifice opening, the pressure change should be relatively constant, resulting in an accordingly constant velocity. Thus, assuming a relatively simple friction model such as the Coulomb-Viscous model, the resulting equation describes the net force as a sum of forces with two unknowns, C_v, and F_C. By taking multiple measurements at different points with the same velocity, these equations can be solved simultaneously to find values for the Coulomb and Viscous friction components.

To determine parameters that fit the friction models introduced in section 4.2.2, open loop step tests were run at 20, 30, and 40 psig, in several orientations, and with different loads placed on the end. While an ideal test would begin at rest midway through the range of piston travel and end within the range as well, such tests are difficult to conduct repeatably because of the challenges of guaranteeing absolutely zero velocity with only an open loop valve command. Instead, most tests were run from endstop to endstop. In cases where the contact of the piston and seals with the

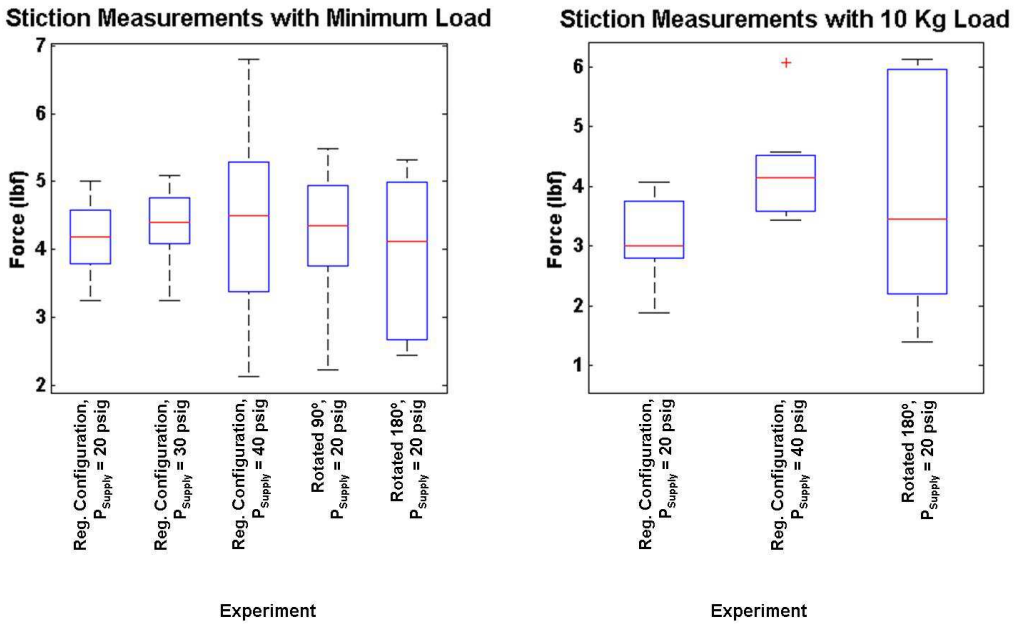

Figure 13: Stiction distribution. Red lines show the median value, the edges of the blue box mark the 25th and 75th percentiles, the dashed lines extend to the most extreme data points not considered outliers, and the outliers are plotted as red '+' symbols.

endstop is fundamentally different from elsewhere in the cylinder, these kind of tests would be of questionable value. However, the cylinders used are cushioned by dead space at either end, and motion of the rod is stopped by a small contact locally about the rod, not the entire piston surface area, meaning that these tests still produced credible results for an overall cylinder friction model. Stiction was measured for each of the cases using the methods derived earlier, and the results, shown in Figure 13, clearly illustrate that stiction falls between 3.5 and 5.0 lbf.

Measurements of C_v and F_C proved less consistent, in part due to the lack of useful data that could be collected with the given methods. Achievement of an accurate solution to the system of equations relied heavily on the assumption of a constant velocity step. In reality, however, the step test was marked by small oscillations before settling, albeit briefly, on a final constant velocity, producing varying results.

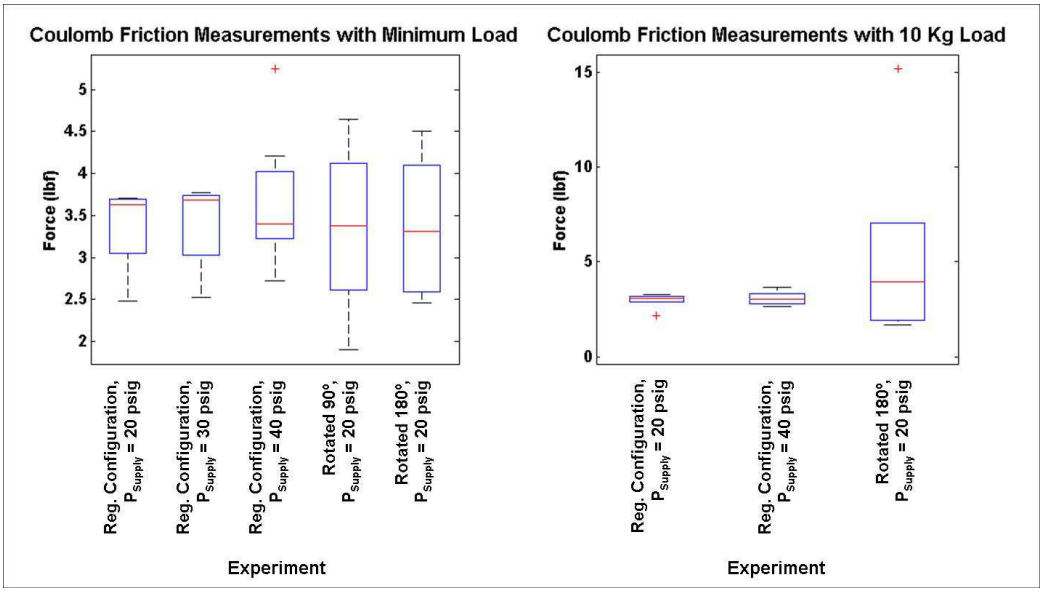

Figure 14: Friction distribution assuming that only Coulomb friction occurs when the cylinder is in motion. Red lines show the median value, the edges of the blue box mark the 25th and 75th percentiles, the dashed lines extend to the most extreme data points not considered outliers, and the outliers are plotted as red '+' symbols.

An overview of these values and their apparent variation can be seen in Table 4.2.2.4. However, by assuming that the Coulomb-viscous portion was dominated by F_C and setting C_V to zero, it was possible to obtain an estimate of the total friction using the same net force approach that had been used to obtain F_S. These results are displayed in Figure 14.

Despite the lack of trends among values determined by solving equations simultaneously, the friction estimates obtained looking at the sum of forces proved consistent and were instead used as a starting point by which to tune curves to match the observed behavior. It is worth noting that among the few values in the table that did provide plausible (non-negative), consistent results (represented by standard deviations considerably less than the average value), those that were calculated using a large mass attachment also showed values near this estimate.

To better understand the friction model that would be most appropriate for this cylinder, a force sensor was used to examine the exact forces experienced by a load on

Table 2: Tabulated Coulomb and Viscous Results

Load of 10 Kg				
Case	Avg C_V	Std Dev	Avg F_C	Std Dev
Up Reg 20 psig	2.26	7.37	0.42	2.53
Down Reg 20 psig	11.83	16.53	4.3	0.46
Up Reg 40 psig	-1.14	2.78	1.21	0.72
Down Reg 40 psig	-0.17	0.61	5.09	0.32
Minimum Load (0.25 Kg)				
Up Reg 20 psig	-1.1	6.4	14.7	2.95
Down Reg 20 psig	0.22	0.44	-6	1
Up Reg 40 psig	0.74	0.58	11.8	0.35
Down Reg 40 psig	1.05	11.1	-9.3	6.3

the rod. These forces were subtracted from the net force due to chamber pressures, providing the difference between these two quantities, which were then assumed to be friction forces. By taking these friction force datasets and plotting them against velocity, a general trend was established and the correlation to established friction models became clearer, aiding in the choice of an appropriate model. Similar results could be viewed on the time scale by plotting net force, friction force, and position.

Based on the results of these curves and conclusions from previous literature that friction in cylinders typically follows the form of a Stribeck curve, the Stribeck-Tanh curve was chosen for use with this cylinder and adjusted using the constants described in equation (4.2.6), as seen in Figure 15. The choice of constants was further driven through the use of coefficients that provided the best general accuracy across a range of step types, as seen in Figure 16. Focus in the Figure should be placed on the highlighted green regions, which show simulated and measured friction during several open loop step responses. The remaining regions (shaded in gray) occur when the piston is at an end stop, introducing deviations in the form of reaction forces. Since the reaction forces were not accounted for in the portion of the simulated force model, the grayed regions can be ignored. The final configuration used in this model was with $F_S = 4.5$ lbf, $F_C = 3$ lbf, $C_V = 0.5$, $i = 5$, $V_S = 0.1$, and $k_{tanh} = 40$.

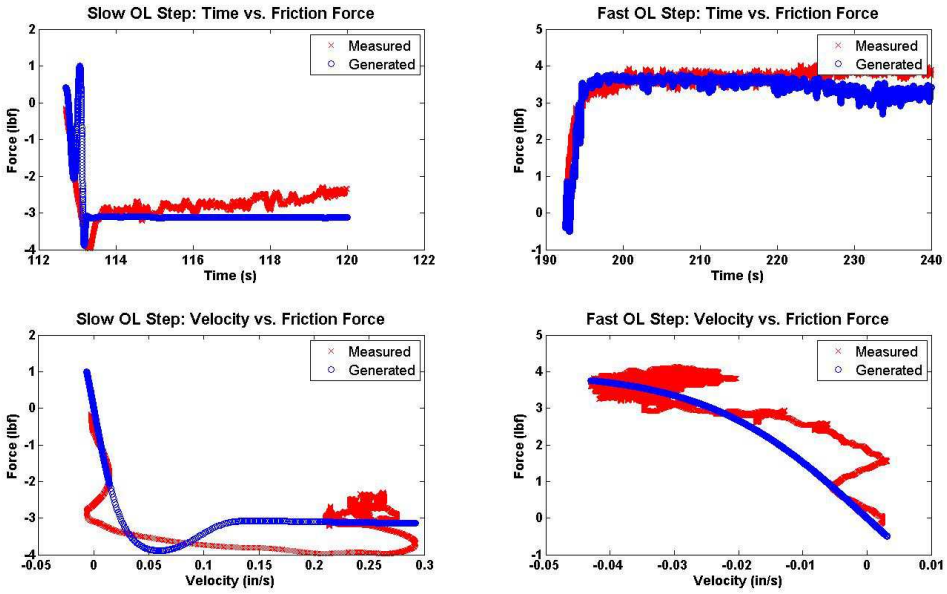

Figure 15: Fitting of Stribeck-Tanh curve to measured friction data

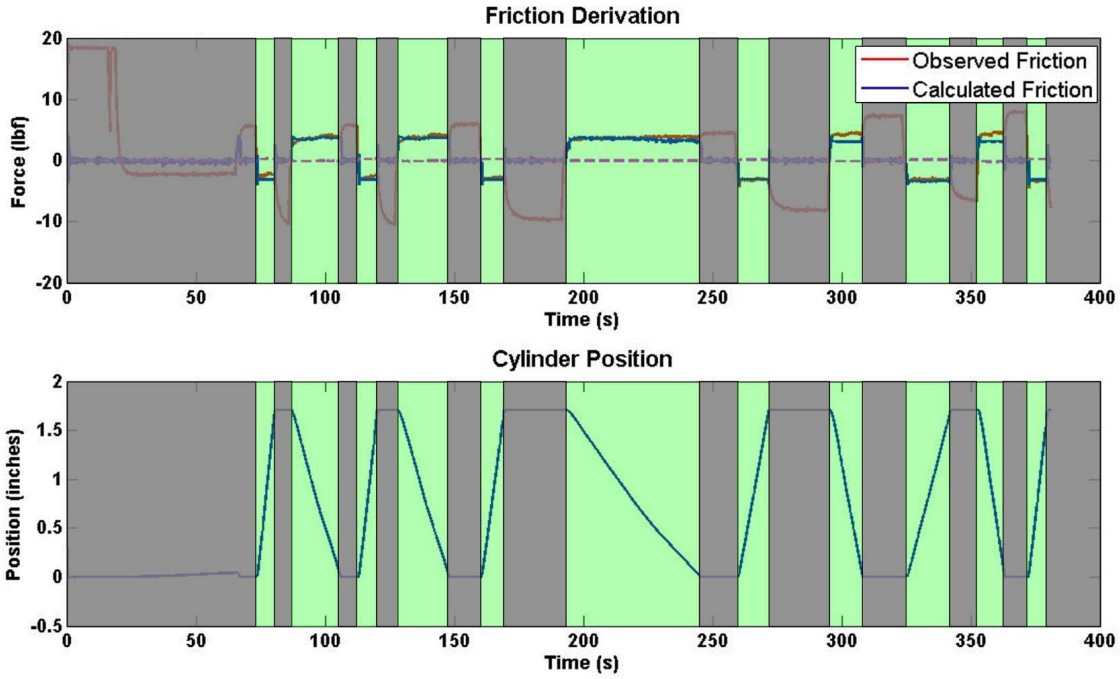

Figure 16: Performance of friction model for a range of step types. Regions highlighted in green are during cylinder motion, while those faded out are when the piston is at the limits of motion, at which point reaction forces come into play

4.2.2.5 Limitations

The friction model used in this thesis proved to be relatively accurate in its characterization of the effects of friction within a cylinder model, but it does possess limitations. First, since the model is based on an overall characterization of friction rather than an exact model of the actual component interaction that causes friction, a more general model is created that leaves some cases unmodeled. For example, the friction occurring at different points, particularly at the limits of piston motion, as opposed to the middle of the cylinder, may differ, but this is not reflected in the model. Additionally, as noted in [15], the use of a velocity dependent model that has zero friction at zero velocity can lead to poor stiction representation in the closed loop. This effect and its allowability are discussed in greater detail in the overall cylinder performance analysis in section 4.4.2.

4.2.3 Valve Model Fitting

As shown in equation (4.1.1) above, the valve is modeled as a flow through an orifice that scales with input command. This orifice area is additionally scaled by a discharge coefficient, c_d, a constant term used to approximate how the shape of the orifice affects mass flow throughput. An optimal approach would be to determine the exact geometry of the area as a function of input voltage and then use this known function together with a mass flow measurement to find the unknown discharge coefficient. However, an examination of the valve dynamics shows that finding this relationship would be rather difficult. While the valve spool moves linearly with input command (this has been verified by using an LVDT to measure the response), the internal orifices are comprised of five evenly spaced circular holes that can be covered or uncovered by the spool, making the relationship of area to voltage difficult to derive exactly without the use of high precision measuring tools or detailed valve drawings (these types of design specifications are not covered in the documentation that Festo

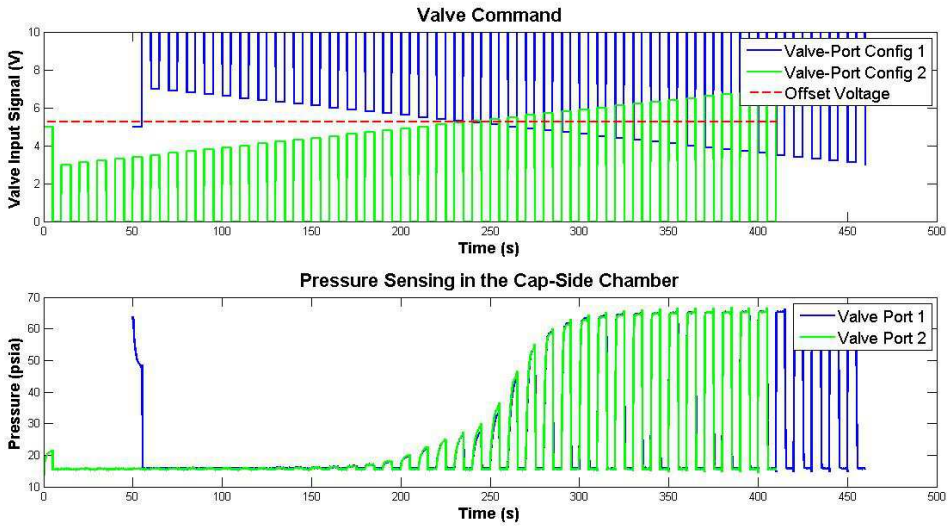

Figure 17: Matching of pressure curves to demonstrate existence of offset voltage at 5.25 V

provides). Instead, the discharge coefficient and orifice area were modeled as a single equivalent area that is a function of input voltage. This equivalent area can be approached in the same ways that most past researchers have approached their area models.

One of the first critical quantities is the offset voltage. Although the valve operates over a range of 0 to 10 V, because of its geometry, it is actually based around a voltage slightly above the center. This voltage is referred to as the valve offset and can be measured in several ways.

From a functional perspective, the valve offset is the input voltage at which flow is equalized in both ports of the valve. This number was determined by using a flow meter to check when the flow is equal from both ports of the valve, resulting in an offset voltage of 5.25 V, or centered voltage offset of 0.25 V. The value was verified by measuring the pressure change in a cylinder with a fixed length rod in one chamber, then switching the ports and measuring the pressure change again. Since the offset voltage is the center of the orifice area curve, flow around it should be symmetrical

near zero – that is, if the ports are switched and an equivalent negative voltage is sent to the valve, given the same chamber size, the same pressure change should be observed. By shifting the two pressure curves so that they lay atop one another and examining the base voltage needed to achieve this kind of symmetrical relationship, it was verified that the offset voltage was 5.25, as shown in Figure 17.

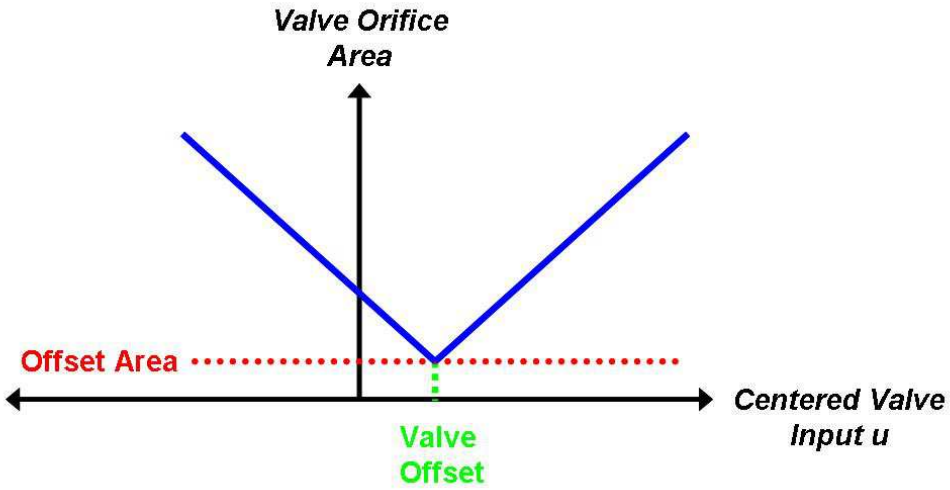

Figure 18: Linear area model with offset area

Once the offset voltage is known, an area model can be formed on a voltage scale. Many researchers simply model the area as a linear curve with an offset area and curves that meet at the valve offset [10], as illustrated in Figure 18. The offset area ensures that even when voltage is at its equilibrium position, flow through the valve can still occur. While such models fail to provide accurate behavioral effects at the inner and outer extremes, they do capture the general behavior and may suffice for certain tasks where either this operating range isn't as heavily used, or model error is acceptable and even assumed.

The valve model's accuracy is further compromised by the fact that it assumes an exclusive connection to either the supply or exhaust pressure; however, this is not the case. The valve model used here (and commonly used in literature) technically models

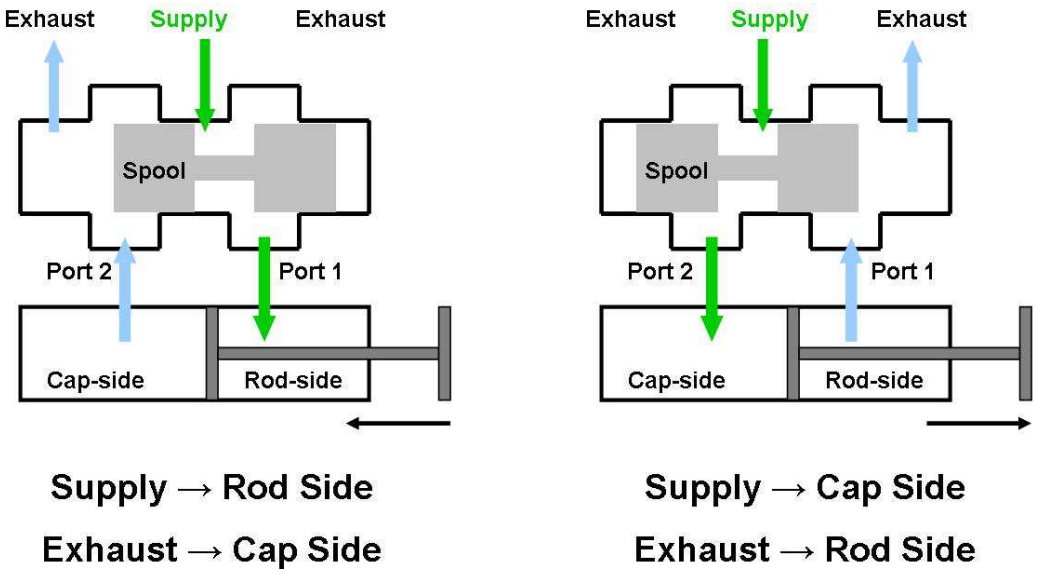

Figure 19: Overlapped valve model

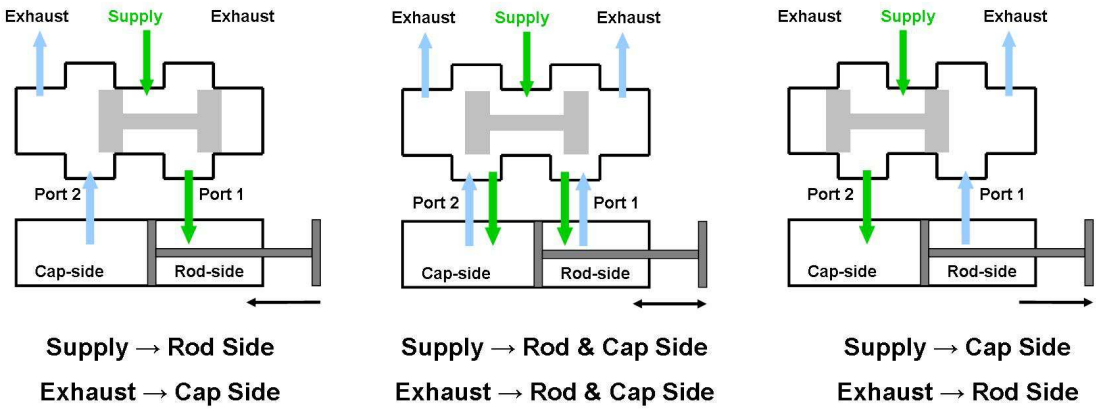

Figure 20: Underlapped valve model

an overlapped valve, meaning that the relationship of the spool to valve orifices has only two possible cases, as depicted in Figure 19. However, in reality, as discussed in [47], the valve is underlapped, meaning that the spool may not always cover both ports, leading to some loss of mass flow for input commands near the valve offset (Figure 20). The end result is that pressures in this region reach asymptotes that are neither supply nor exhaust, but rather are somewhere in between, as the cylinder is neither fully charging nor discharging.

This phenomenon has been discussed in other works, and is typically dealt with in one of several ways. Many researchers simply ignore it, especially those who are only using the model within a controller. However, the region spans approximately +/- 0.75 V about the valve offset, a region that has been shown to be very active in closed loop commands on this testbed, so its dynamics cannot be ignored.

To model this region, it is necessary to examine the factors that could cause pressure to stop changing before exhaust or supply pressures are reached. From equation (4.1.10), it can be seen that the case where $\dot{P}$ prematurely reaches zero can only be achieved in a few circumstances. For a nonzero velocity, the two terms could cancel, though this solution is unlikely to occur for any extended period of time since a nonzero velocity means a changing position, thus varying the terms. Instead, in the zero velocity case, $\dot{P}$ can only prematurely reach zero if $\dot{m}$ equivalently reaches zero. This approach is further confirmed by experiments showing that the pressure losses observed in cylinder actuation persist in step tests where piston position was fixed ($\dot{x} = 0$) during cylinder actuation.

Based on this analysis, several methods were found to model the loss of mass flow in the valve:

1. Assume a constant mass flow loss and offset the overall $\dot{m}$ function by this amount

2. Assume zero mass flow loss except in cases where cylinder does not fully charge

or discharge

3. Model the net mass flow as a sum of positive and negative flows through a more accurate valve orifice model

4. Ignore the transients and focus only on the steady state values by changing the magnitude of the input port pressures perceived by the valve

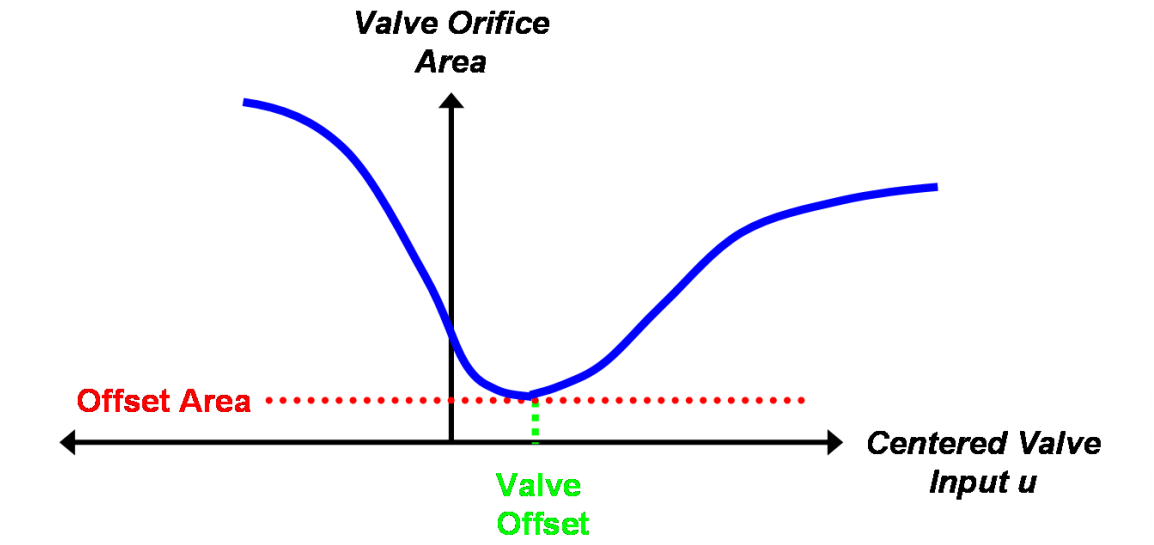

Figure 21: Valve orifice area for use with mass flow as a net sum of different mass flows

Each of these fitting strategies was attempted, tuning the values to get the best pressure and position correlation. Methods (1) and (2) both modeled the regions close to zero with reasonable accuracy, though they appeared to fail in some circumstances. The difference between the two caused differing values of equivalent orifice area, though these still followed the general form observed in the Figure 21. The primary concern was that, since mass flow depends on upstream and downstream pressure, it would vary for different supply pressures, and the losses would have to follow some sort of trend. In practice, however, trends were difficult to match, and the method appeared to be little more than a "fudge factor".

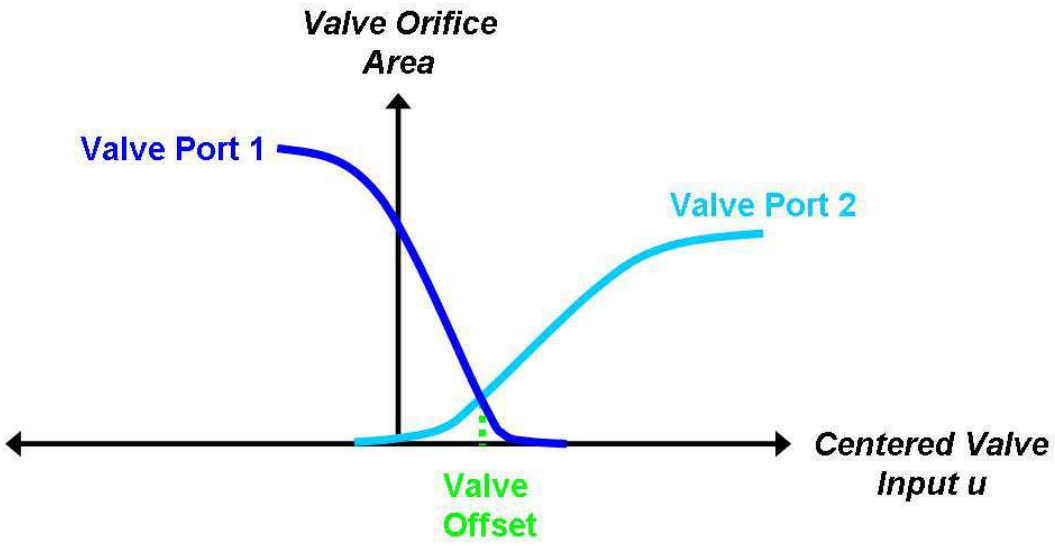

Figure 22: Valve orifice area for use with mass flow as a net sum of different mass flows

Instead, method (3) offers an approach that models mass flow losses. This method has been tried by previous researchers [47], who pointed out that the loss in mass flow can be written as a sum of positive and negative mass flows. The approach is based on the valve dynamics shown in Figure 20. It uses an equivalent orifice area model that extends the orifice area of each port beyond the valve offset (Figure 22), thereby eliminating the need for the assumption that each port is exclusively attached to the exhaust or supply pressure. Instead, the valve output port is open to some combination of both the supply and exhaust pressure, so that in the commands about the valve offset, mass flow loss is included in the modeling process. While this approach seems accurate in principle, past researchers who applied it have had only limited success in the affected regions [47]. In efforts to do the same on this model, the areas were modeled by using pressure curves from fixed-rod models to determine where the orifice area first began to open for each valve, then fitting an equivalent orifice area to that curve. The process was complicated, however, by the fact that even with a fixed piston position, the pressure dynamics were now coupled in the affected

regions, making it more difficult to accurately determine an appropriate equivalent area. Even after extensive efforts to find well-fitting matches, the approach only worked well for isolated cases and virtually no trends were observed in the affected regions. This approach might be more valuable if the actual area curves were known, rather than being fitted based on the system dynamics.

Method (4) takes a different approach, affecting mass flow by examining the steady state behavior. One way for the mass flow to be reduced to zero early is if the upstream and downstream pressures, P_u and P_d, are not equal to the actual supply and exhaust pressures. In a sense, this is the case when the spool is undersized or leakage is occurring. Accordingly, in this model, the valve orifice area was modeled as a nonlinear area starting at a given offset area occurring at the valve offset area, and the pressures were adjusted in the affected regions. Since the ports are always attached to only one input pressure and the upstream and downstream pressures always refer to the chamber pressure and one of the two input pressures, modification of an input pressure only affects one chamber pressure at any given time. Accordingly, the input pressures could be modified to be some pressure between the exhaust and supply – essentially a combination of the two – by matching adjusted input pressures to the steady state pressures. To ensure that these steady state pressures could be collected independently of the equivalent voltage area, the pressures were matched using tests with fixed piston position, such that the second term of equation (4.1.10) did not play a role in the derivation. If the equivalent orifice area was wrong, it would only scale the transient components of the pressure, and the steady state pressure curves would be unaffected, instead depending on the choice of input pressures. The pressures were then tabulated across a range of voltage inputs.

Based on the observed curves (Figure 23), while the pressures could be modeled as a piecewise continuous function that is linear in the affected regions and constant

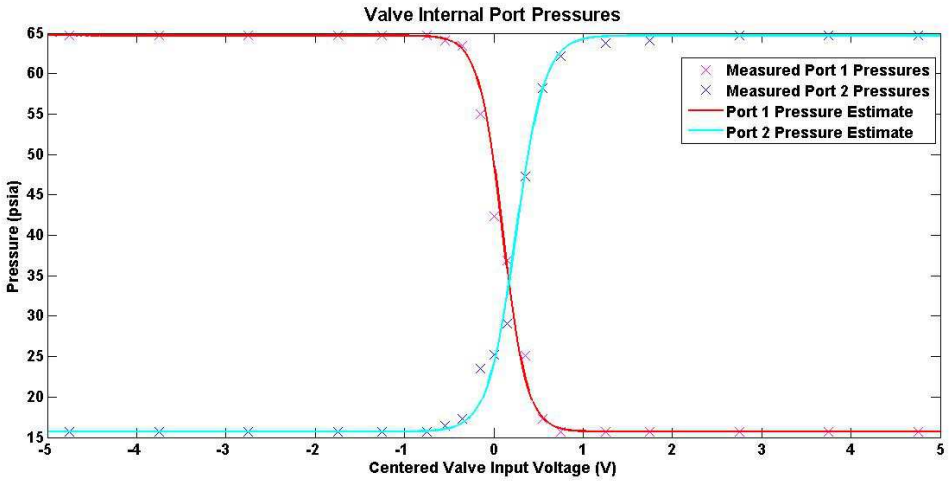

Figure 23: Input pressure trends

otherwise (such that the pressure is always a linear combination of the input pressures), a more accurate model can be formed by fitting the data to a scaled tanh curve with an offset. The resulting curve has the form

$$P_{input} = y_{offset} + C_{scale} tanh\left(K_{Press}\left(u - x_{offset}\right)\right) \tag{4.2.7}$$

where the constant coefficients can be defined as functions of key parameters for each curve: the supply pressure, P_S, the exhaust Pressure, P_e, the low Pressure cutoff, C_{LP} (the voltage/x-axis value at which curve transitions from the constant minimum pressure to a changing one), and the high pressure cutoff, C_{HP} (the voltage/x-axis value at which curve transitions from the constant maximum pressure to a changing one). The constants used in equation (4.2.7) can then be defined as

$$y_{offset} = \frac{P_S + P_E}{2} \tag{4.2.8}$$

$$x_{offset} = \frac{C_{HP} + C_{LP}}{2} \tag{4.2.9}$$

$$C_{scale} = sign(C_{HP} - C_{LP})\frac{P_S - P_E}{2} \tag{4.2.10}$$

$$K_{Press} = (2\pi)/(|C_{LP} - C_{HP}|) \tag{4.2.11}$$

Using these functions for input pressure, equivalent orifice areas were determined by matching simulation and experimental data. Though these curves were originally measured with the piston position fixed to avoid friction considerations, it was found that this data, while reasonably accurate, did not translate fully to non-fixed motion.

Instead, step tests were run on the (unconstrained) cylinder test rig, with each step consisting of a 10 second interval. The valve would first be given the maximum command in the direction opposing the step, where the magnitude of the step is added to 5 V to get the actual input signal (e.g., for a 0.75 V step, a centered valve command of -5, or actual centered voltage signal of 0, would be sent for the first 5 seconds). At the 5 second mark, the voltage command was given, at which point position and pressure trends were measured for the remaining 5 seconds.

These tests were executed for 30 and 50 psig supply pressures in the standard valve-chamber configuration using a series of command voltages. For each step, the simulation was executed with an equivalent area. This approximate equivalent area curve was implemented by setting a discharge coefficient in the setup file (see Appendix B) and assuming a linear scaling of area in the model. The product of the discharge coefficient and the area function corresponded to the equivalent orifice area for that voltage command. Tuning of the discharge coefficient as a function of voltage was done by hand, observing the measured and actual pressure and position dynamics and matching them as best as possible.

From a pressure perspective, the curves demonstrate several key features. They begin to move when the step command is first given, ascend at a certain rate to reach the stiction peak, then drop to a lower value and continue until the end stop is reached, at which point they expand again until the steady state values are reached, as shown earlier in Figure 12. Simulated pressures were matched with emphasis on the time and magnitude of the stiction peak, duration of the motion, and slope of the curves. Because pressures are difficult to match exactly, position was used as a

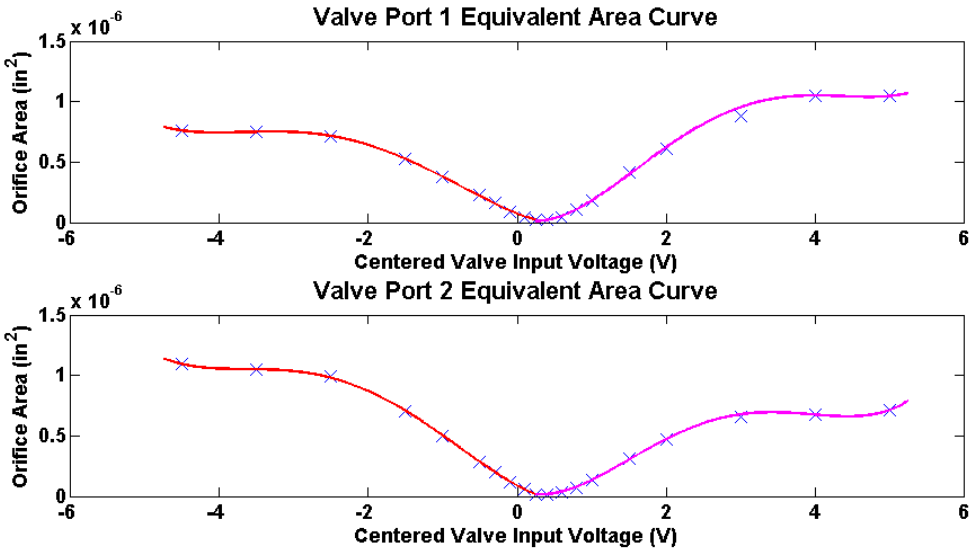

Figure 24: Equivalent area curves for 50 psig actuator model

final-check, ensuring that the motion began at the right time and had the right slope and duration. Since the user of this simulation will most likely experience the effects of fluid power in the form of effects on position and velocity, the usage of position as an error check is justified. The equivalent orifice areas from each step were then plotted against input voltage. Curves were fit to the data points, with the results shown in Figure 24.

4.2.3.1 Curve Fitting Methods

The process of fitting curves to the pressure and equivalent orifice area data points was done with the intent of minimizing error while keeping true to the expected forms described in literature and by the manufacturer. For the pressure curves, this was achieved using the hyperbolic tangent curve discussed above described in eq. (4.2.7). The equivalent area curves, by contrast, were fitted using fourth order polynomials to model each side of the area function centered near the valve offset. In addition to requiring a model that minimized error overall, it was critical that the model include the minimum area offset and exhibit good correlation in voltage commands about the

valve offset, where smaller differences in equivalent area have a more significant impact on overall pressure and position accuracy. Curves were found using the MATLAB functions `polyfit`, which allows a user to fit curves to specific function types. An m-file corresponding to these fitting actions can be found in Appendix B.

4.3 Experiments & Results

To validate the model, tests were run that encompassed both the general, open-loop behavior of the model, as well as the closed-loop tracking for tasks similar to those needed on this testbed. The first such set of tests was a series of open-loop step responses (Figures 25 and 26). These are easy to perform and provide valuable insight into the system characteristics. The accuracy of the friction model and pressure constants can be viewed by looking at the time it takes for the position to begin changing (or its resistance to change if the pressures are not high enough), the pressures needed to do so, and the duration of the step thereafter. Additionally, the steady state pressure helped to validate the models used earlier. Step tests were performed not only at the points used for curve fitting, but also at several other locations, demonstrating pressure and position dynamics that closely matched the actual observed behavior. As is evident from the figures, the only place where position correspondence deviates significantly is when the voltage command is 5.25 V, or exactly at the offset voltage. Even when fitting curves exactly, this region is difficult to model exactly using an overlapped model, and can be viewed as an outlier, particularly since the curves near it actuated at very small voltage inputs demonstrate good correlation of measured and simulated values.

While the behavioral similarities observed in the open-loop response are promising, it is desirable to use a more quantifiable approach to demonstrate correspondence between the simulated and actual responses. To do so, first order responses were fitted to the pressure curves, using Matlab's `fit` function to achieve the closest fit

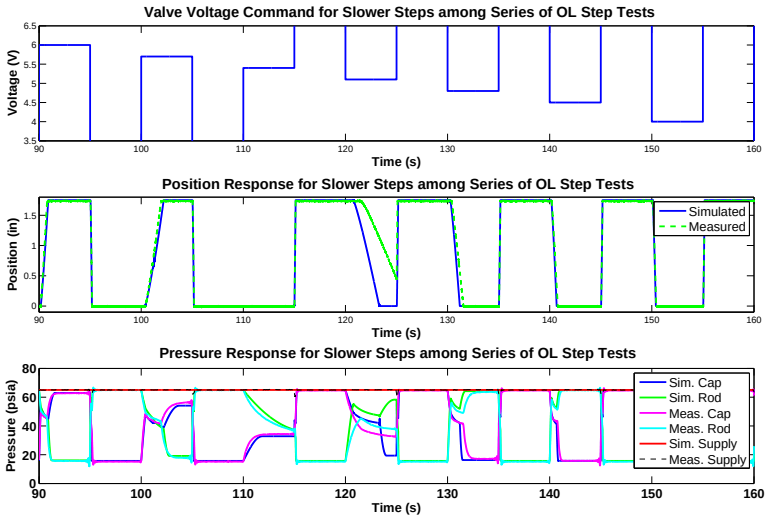

Figure 25: Selection of slower step tests used to validate the actuator model through its open loop response

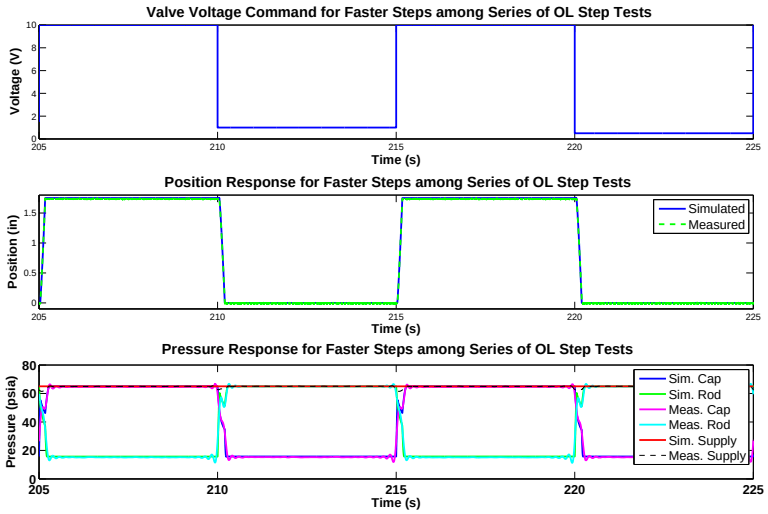

Figure 26: Selection of faster step tests used to validate the actuator model through its open loop response

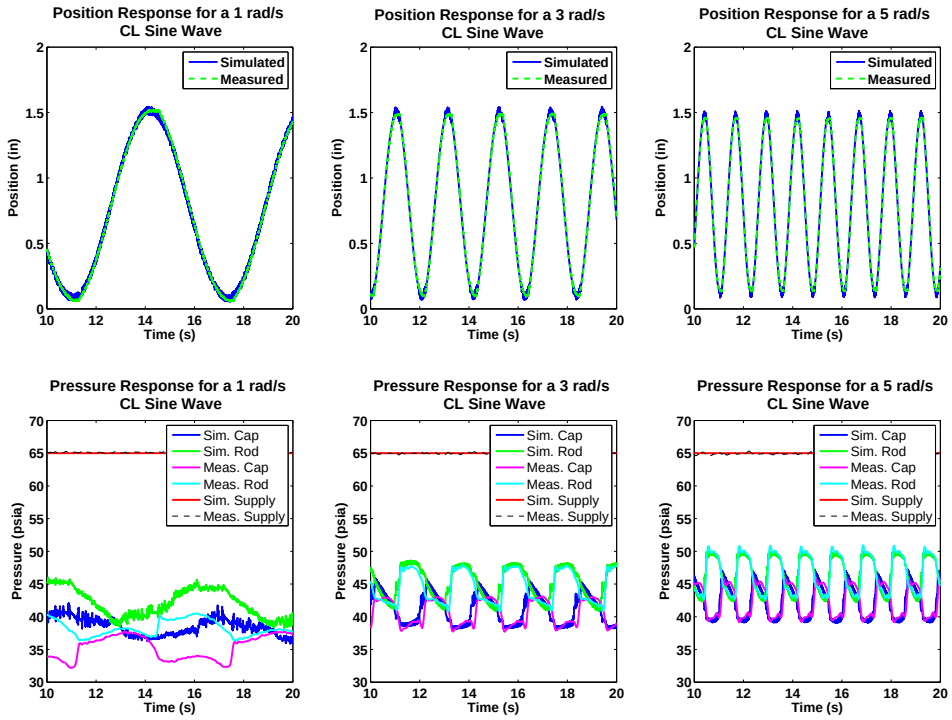

Figure 27: Selection of sine tracking tests used to validate the actuator model through its closed loop response

in the least squares sense. By matching individual cap side pressure steps to a first order response of the form

$$y(t) = K_0 - A_0 e^{(t/\tau)} \tag{4.3.1}$$

where K_0 and A_0 are constants and τ is the time constant, it was possible to numerically related multiple actual and simulated pressure responses through comparison of their respective time constants. Responses were fitted for several steps spanning the majority of the voltage range, producing fits with Coefficients of Determination (R^2 value) ranging from 0.8 - 0.95. Over seven steps, the time constant of the simulated systems consistently fell within 5 - 40 % of the time constants for the corresponding actual response. Better matches were exhibited for faster steps, where the time constants of the simulated responses differed by at most 15% from their respective

counterparts for the actual response.

Given accurate open-loop behavior, it would be expected that closed-loop results would return similarly good correlation. To ensure that the closed-loop curves provided a good representation of the type of behavior that might actually be required on this testbed, a series of sine waves of increasing frequency was used. On the actual robot, commands will be given by the user, generally resulting in a continuous but also progressively changing curve. Accordingly, several closed loop sine waves were provided that show close correlation except for very low velocities, (Figures 27 and 28), where the pressure curves had the right form, but not magnitude, an effect that will be discussed in section 4.4. The value of the simulated response can be further upheld through an examination of the lag with respect to the reference input showed differences of 0.02, 0.01, and 0.005 seconds between the simulated and actual lags, for tracking of 1, 3, and 5 rad/s sine waves, respectively. These values correspond to no more than 25% deviation from the lag of the actual system with respect to the reference input.

Additionally, testing of a closed loop step response (Figure 28) showed that the model possessed a similar rise time and settling time to the actual system, as well as oscillations characteristic to a third order system, as can be seen in the zoomed-in step response. In order to again demonstrate these similarities quantitatively, a superposition of a first order and a second order response was matched to several closed loop position responses, again using the `fit` function to determine the best fit in the least squares sense. Results showed that the first order response was generally the dominant component, achieving fits with R^2 values of 0.92 - 0.99 before any higher-order components were added to the fit. Additionally, the higher-order dynamics that were able to be matched by the function were generally of minimal significance and in some cases provided unrealistic values (e.g., negative damping), rendering them largely useless for comparative purposes. Instead, the time constants of the best

fit curve for position response was used to numerically demonstrate similarities in simulated and actual responses. Using the five steps seen in the figure, it was shown that the measured and simulated time constants differed by no more than 40%. These results can be improved by excluding the first step (at 6 seconds), which appears to be an outlier. The remaining time constants exhibit variation between the measured and actual systems of 5 - 25%. Furthermore, the average τ for the simulated response is 0.082, with a standard deviation of 0.003, while the average τ for actual responses was 0.085 with a standard deviation of 0.01. These average time constants, which are shown by their low standard deviations to be relatively consistent, differ by just 2%, further validating the accuracy of the simulation.

Since one of the main intended applications of this simulation is for use with an operator interface, the most critical measurements are those that will be directly perceived by the user: force, and more importantly, position. The Root Mean Squared Error (RMSE) of the simulated position with respect to the actual position provides a quantitative measure of the value of the simulation. RMSE was calculated over the full time span of the response and is intended to provide a versatile comparison among many response types, including steps, sines, and walking motions. It is complemented by the afore-mentioned analyses of specific responses that focus on similarities in characteristics of the response and on quantifiable numerical similarities of representative LTI systems for the simulated and actual models.

For the closed loop step response, RMSE was found to be 0.052 inches, less than 3% of the 1.75 inch stroke length, and likely within the desired tolerance of a user seeking primarily behavioral similarities in the response. This tolerance should be acceptable based on past human-machine interface studies, in which operators have been found to be capable of recognizing the transmission of approximately 3 bits of a given a signal, such as angular position [36]. Furthermore, a review of advanced controllers for positioning pneumatic cylinders [9] shows that while the best controllers

have an accuracy of 0.5mm, most fall within an accuracy range of 4 - 8 mm, mostly falling within 5 - 10% of the total operating range. The accuracy achieved by this simulation, which is equivalent to about 1mm of tolerance, is therefore likely already within the tolerance that most controllers will be able to provide.

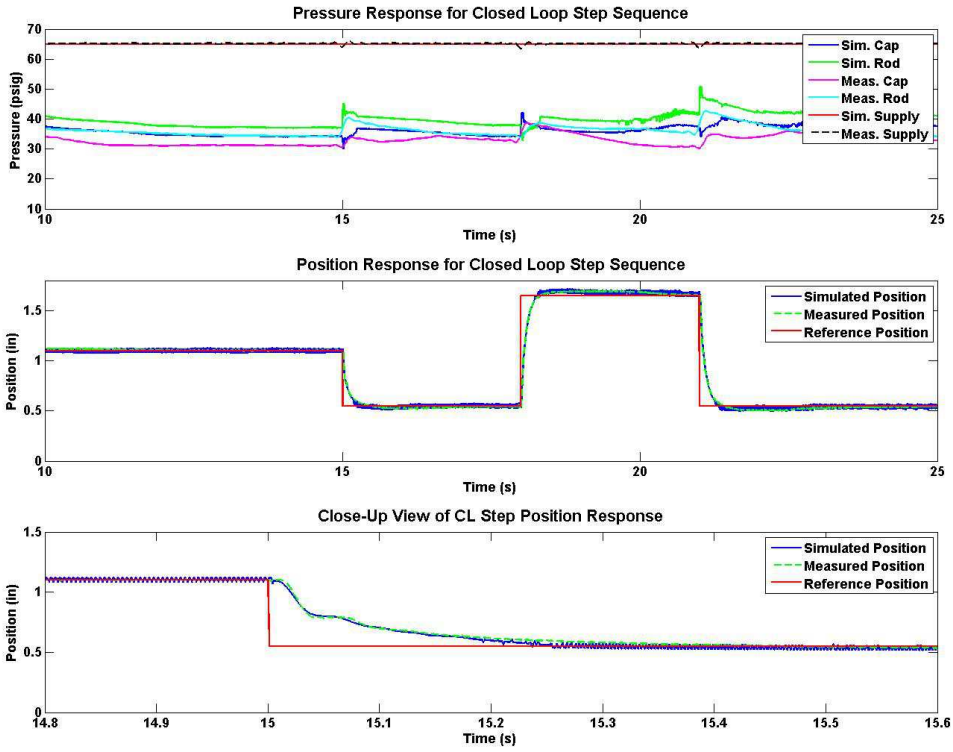

Figure 28: Selection of sine tracking tests used to validate the actuator model through its closed loop response

4.4 Limitations

Though the results of physical experiments largely validate the construction of the simulation, there are some discrepancies between the physical and simulated data resulting that must be addressed individually.

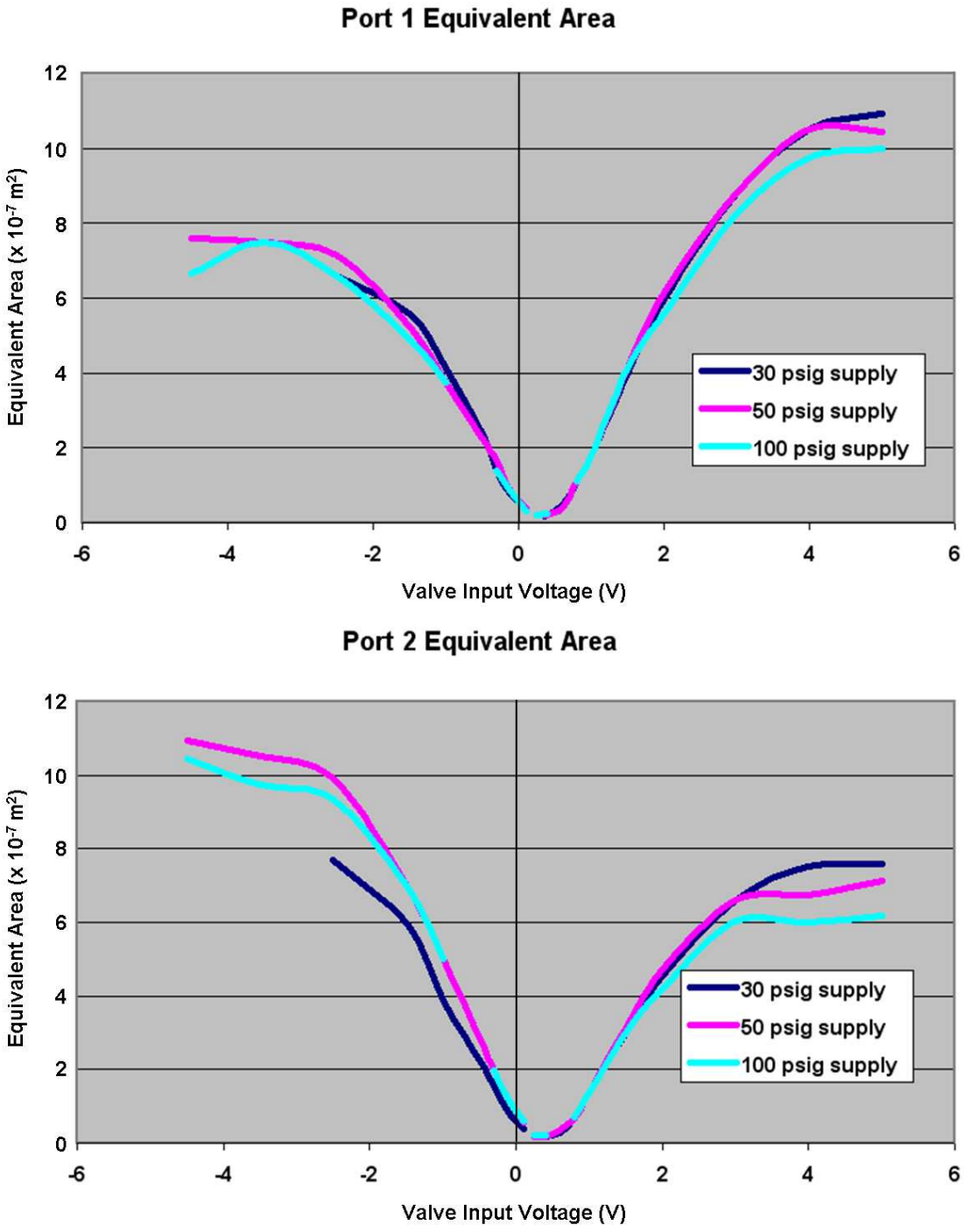

Figure 29: Effects of variations in actuator configuration and supply pressure on model accuracy

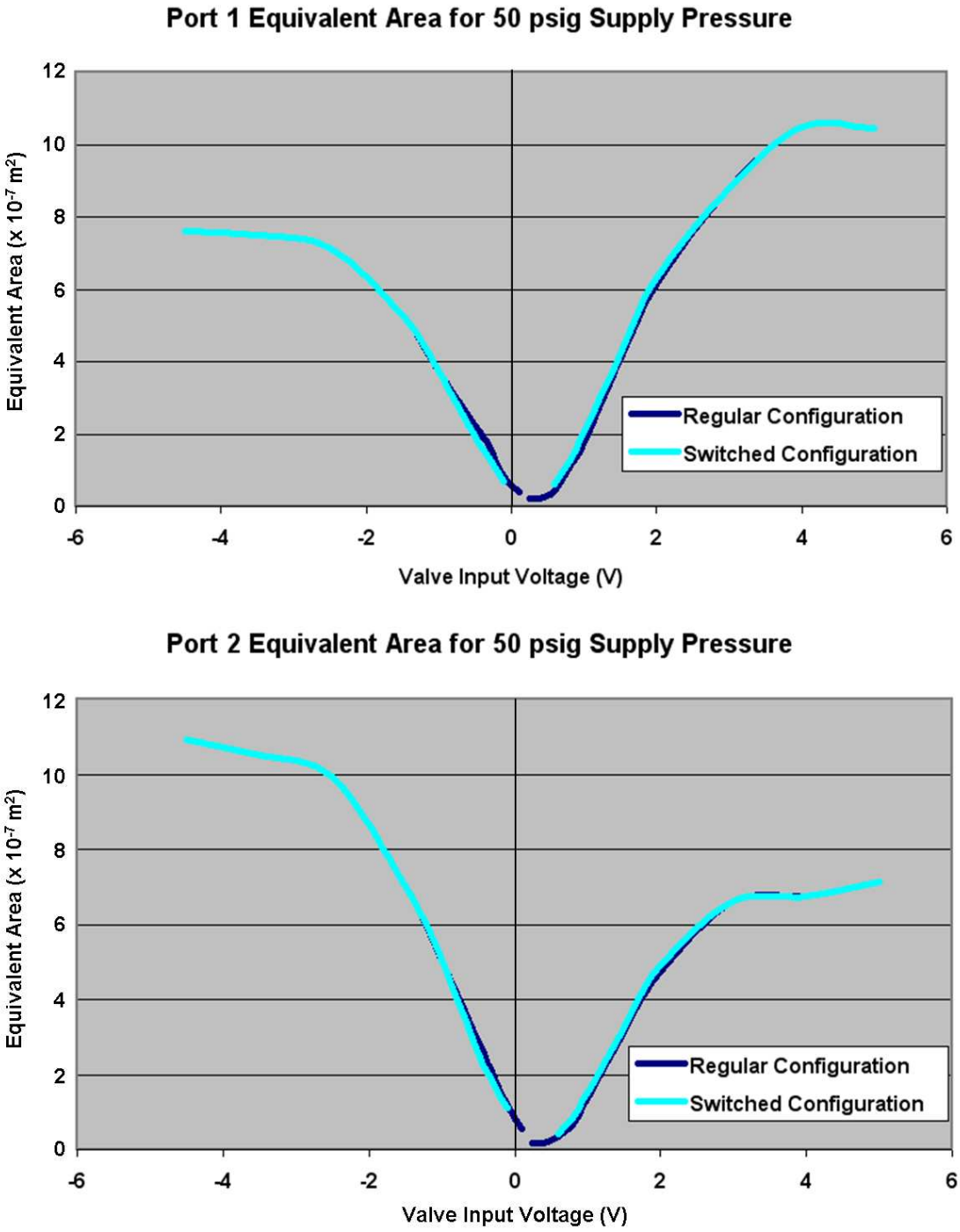

Figure 30: Effects of variations in actuator configuration and supply pressure on model accuracy

4.4.1 Robustness & Variability

To check the model's breadth of operation, tests were performed using several different supply pressures (Figure 29). While the overall performance remained consistent, tests that used air supplied at 30 psig exhibited some variation in equivalent area curves. The change in equivalent area largely makes sense, since mass flow is dependent on pressure, so a change in pressure proportions will affect the scaling, or equivalent orifice area, needed to get the same result. However, this restriction is likely not as important for the robot simulation because the robot is intended for higher pressures. It can be seen in the figure that as pressures rise, the changes in area trends become less pronounced. Additionally, the generality of the valve model was demonstrated by switching the connection to cylinder chamber from one valve port to the other and collecting equivalent area data. The results, illustrated in Figure 30, showed very similar equivalent areas.

4.4.2 Friction Effects

In a previous section, the friction model was derived and analyzed in terms of the forces at work. Throughout the validation process, friction was analyzed in terms of the affected components in the pressure response and the resistance to motion. While this works quite well in the open loop, in cases of very slow velocity, notedly in closed loop cases, the behavior is not as accurate. Whereas with an open loop step response, stiction will appear primarily as a prevention of motion until a specific pressure is reached, the closed loop behavior is not as simple. Instead, in areas of high friction, such as the point at which a closed loop sine wave changes direction, the position will appear to oscillate about some nearly still value. This occurrence results from the approximation of friction behavior near zero velocity – instead of a significant jump that would result in a discontinuity, a sloped line that is zero at zero velocity has been used, a fact noted in Book & Daepp [15], where similar friction and

valve models were used. Despite the fact that this position behavior is not accurate, it is acceptable because the velocity deviation is minimal and the position is accurate within a switching region.

4.4.3 System Bandwidth

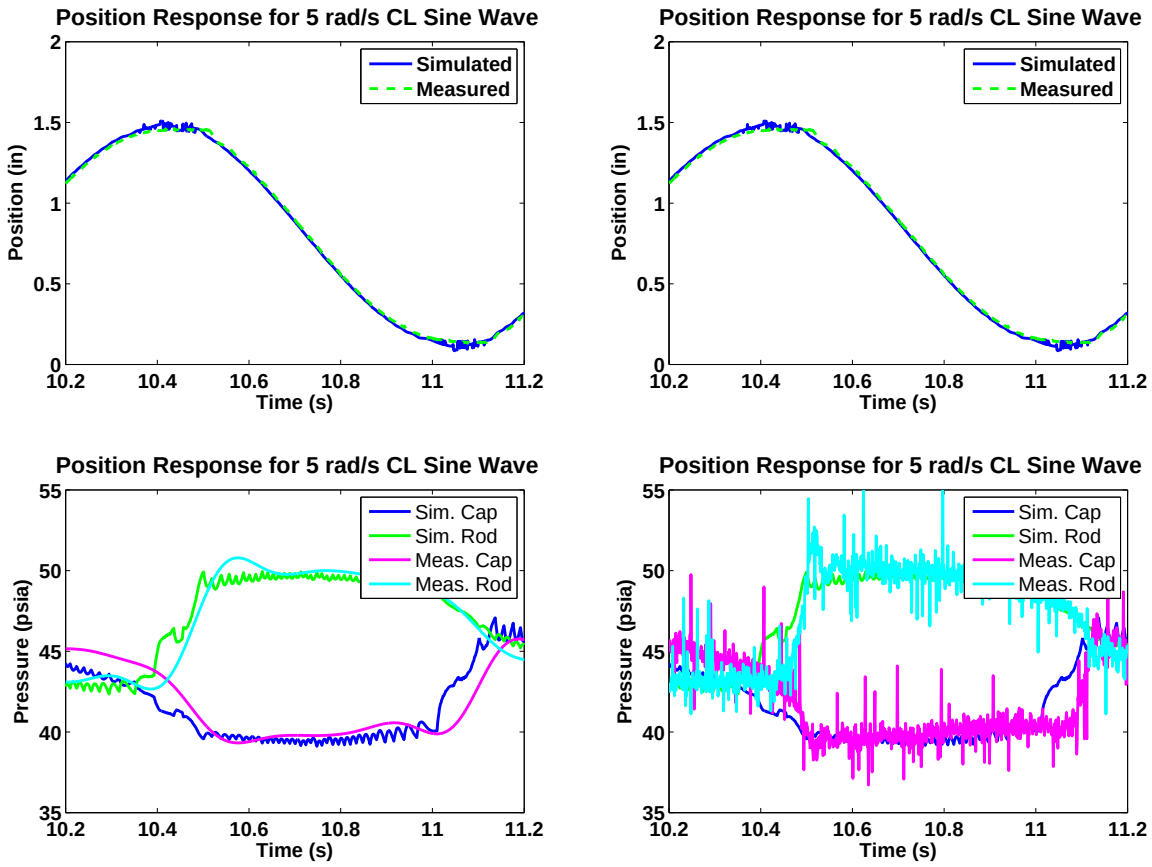

Figure 31: Left: Simulated closed loop pressure appears to have higher bandwidth than sensor pressure. Right: Unfiltered pressure data shows that this effect may be caused by sensor measurements and data processing.

The pressure trends observed in the closed loop responses largely match the actual pressure waves, but a closer look (bottom left plot of Figure 31) reveals that the model appears to have a higher bandwidth than the actual model, based the occurrence of small, high frequency variances in the simulated pressures that are not evident in

the original sensor pressures. However, this discrepancy is likely due to the methods by which the sensor dynamics were obtained. First, the sensor may not be able to update at the rate of change observed in the simulated model. Furthermore, the sensor data was filtered in order to get the smooth curves, likely eliminating some of these faster pressure dynamics. The bottom right plot in the figure shows what the unfiltered pressure data looks like, demonstrating that such small spikes may simply be unobservable with this set of pressure sensors and associated filtering.

CHAPTER V

DYNAMIC SIMULATION

In this chapter, the actuator model developed and validated in Chapter 4 will be implemented in a simulation of the robots dynamics and shown to realistically capture full dynamics of the robot behavior in an appropriately and conveniently configured system layout.

To do so, the completed Simulink actuator model is coupled with SrLib, a C++ and OpenGL simulation of the robot's dynamics and corresponding display of a model of the robot interacting with the environment, to provide a complete equivalent to the actual physical robot. SrLib allows users to build systems using a library of joints, components, and environments. The original robot configuration in SrLib was set up and defined without the inclusion of actuator dynamics by Ta Kim, as noted in her special project report [29] and with the help of Dr. Frank Park and Jaeyoung Haan, who first created SrLib [24]. The codes were later modified to more accurately represent the physical robot and include consideration of the actuator dynamics, as modeled in this thesis.

5.1 SrLib Structure

SrLib uses a hierarchical structure to define components and link together with the elements available in the library. Figure 32 outlines the general structure of the program. At the highest level, the processes in SrLib fall into one of two stages: they either contribute to the initialization of the configuration, environment, and interface, or they loop continuously following initialization, stopping only once the program is terminated. To fully understand this diagram, it is necessary to provide descriptions of several key files, functions, and processes involved. Further information and code

excerpts from these files can be found in Appendix C.

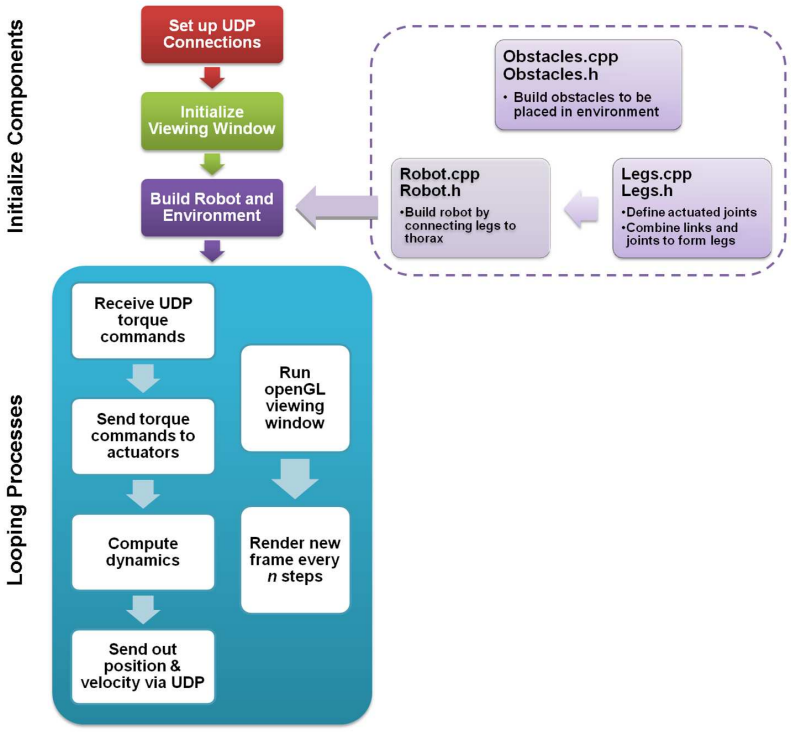

Figure 32: Diagram of SrLib process structure

`Leg.cpp` and `Leg.h` are the most basic complete files of the SrLib robot structure. The legs consist of separate rigid elements, the coxa, tibia, and femur, known as "links", and modeled as cylindrical capsules with a fixed diameter, and color, as illustrated in Figure 33. Mass and inertia properties are either specified explicitly or calculated internally by providing SrLib with a density for the link. Each link has specified end points and is connected to a joint at some local Cartesian relation to these specified points. Several kinds of joints exist, though here only weld joints, which fix two links together, and rotary joints, are used. Rotary joints accept torque commands and provide single-axis rotation – essentially like an actuated pin connection. `Leg.cpp` also contains the code that directs torque commands to the appropriate joint. In the original, non-dynamic version of SrLib, position commands were provided instead and a simple proportional controller determined the actual

torque needed to maintain these positions, with torque limits set by the user in an optional command.

The next files in the robot structure are `robot.cpp` and `robot.h`. These files take multiple `Leg.cpp` instances and connected them to a thorax, again using single-axis rotary joints for the moving shoulder connections, and using rigid weld joints everywhere else. A point on the thorax is set as the robot origin, to be used for placement and orientation elsewhere in the code.

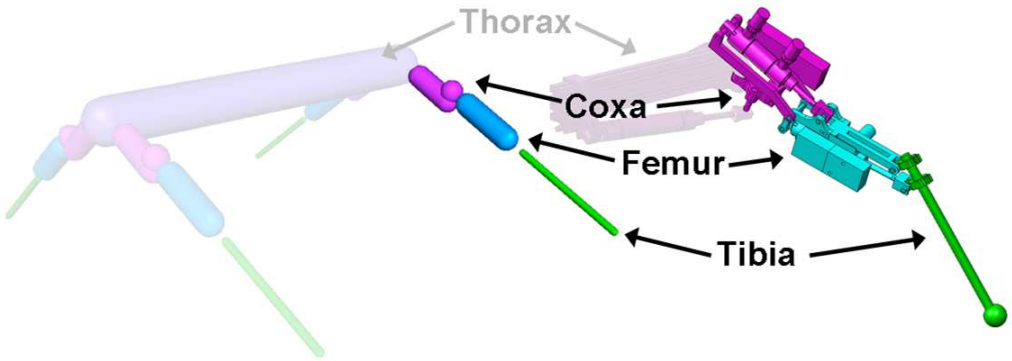

Figure 33: Robot leg structure, shown in SrLib (left) and on an a Solidworks model of the actual robot. Corresponding colors show how real components contribute to modeled SrLib links

The robot is placed in an environment in `main.cpp`, using Cartesian coordinates to specify the location of the robot origin and Euler angles to provide a starting orientation. From the *main()* function within `main.cpp`, the most high-level function in the code, the simulation, robot, obstacles, and other objects are initialized, and the simulation is run.

Two other functions of significant interest are the *send* and *receive* functions, also located in `main.cpp` and the associated header files. The *send* function specifies outgoing UDP message content and converts it to the appropriate formats. Though message content can be modified depending on the situation, the intended format for typical (non-debugging) application uses 12 messages to provide the angles of

the joints, 3 messages for the robot's displacement from the global origin, and 3 messages to provide Euler Angles describing the robot's orientation. Similarly, the *receive* function accepts UDP messages in a standard form and extracts the desired information.

Following initialization of the robot and viewer, the bulk of the simulation consists of sending and receiving data, computing the dynamics, and updating the viewer. However, the dynamics simulation is not entirely independent of the openGL codes needed for the viewer. Instead, features such as center of mass tracking and object interaction modeled within openGL are used by the dynamics engine. Though these functions are used continuously, the viewing window is updated only as often as specified by the user. In other words, the two processes are not entirely separate, but the actual rendering is not necessary for the success of the simulation.

The serial arrangement of functions (Receive-Compute Dynamics-Send) in this thread enforces a pseudo-realtime environment as far as processing power allows; that is, the simulation will only compute dynamics and send data back if it has received a joint command, which can only occur if it has received a UDP message packet (from the actuator model). As long as the total time needed for SrLib to run these functions is less than the time step used by the Simulink actuator model and there are no inconsistencies (varying lags, pauses) in the rate at which data is passed between the host and target, this arrangement will allow SrLib to perform like a realtime system, with a clock that matches that of the target.

5.2 Actuator Model Integration

SrLib is connected to the actuator model using UDP communication to transmit data. Validations are performed on a single leg by executing open and closed loop commands to each joint and recording the system response.

While angles are measured directly in simulation, the physical two-legged robot

actually uses linear potentiometers built into the cylinders to measure leg motions. Since knowledge of the joint angles is necessary for inverse kinematics and for integration with SrLib, it was necessary to determine the relationship between piston displacement and angular position of the corresponding joint. The actuator models also require velocity in addition to position feedback, leading to equations for linear velocity based on the angular position and velocity provided by from SrLib. Equations for this conversion were derived using the parameters defined in Figures 34, 35, and 36. On each joint, θ is the measured angular displacement used in simulation and for inverse kinematics (the arrows define the positive direction), while x is the cylinder piston displacement.

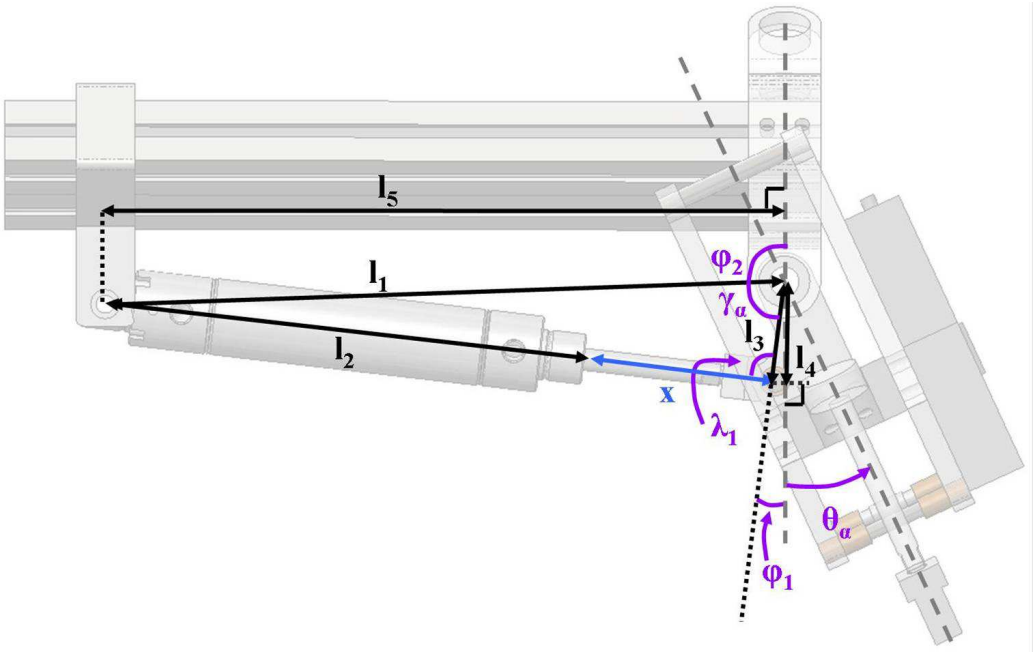

Figure 34: Geometry of an Alpha joint. The cylinder and associated sensors have been removed to provide a clearer picture of the relevant joint geometry, though the piston is still shown

For the alpha joint, the relationship of x to γ_α can be obtained by applying the

Law of Cosines:

$$x = -l_2 + \sqrt{{l_1}^2 + {l_3}^2 - 2l_1 l_3 cos(\gamma_\alpha)}$$
$$\dot{x} = \frac{2l_1 l_3 sin(\gamma_\alpha)\dot{\gamma_\alpha}}{2({l_1}^2 + {l_3}^2 - 2l_1 l_3 cos(\gamma_\alpha))^{1/2}} \tag{5.2.1}$$

where γ_α and $\dot{\gamma_\alpha}$ are defined as

$$\gamma_\alpha = 180\,^\circ - \phi_2 - \phi_1 + \theta_\alpha$$
$$\dot{\gamma_\alpha} = \dot{\theta_\alpha}$$
$$\phi_1 = cos^{-1}(l_4/l_3) \tag{5.2.2}$$
$$\phi_2 = 180\,^\circ - sin^{-1}(l_5/l_1)$$

where θ_α has a range of -47 to 90 °. Similar derivations follow for the other two joints. In the case of the beta joint:

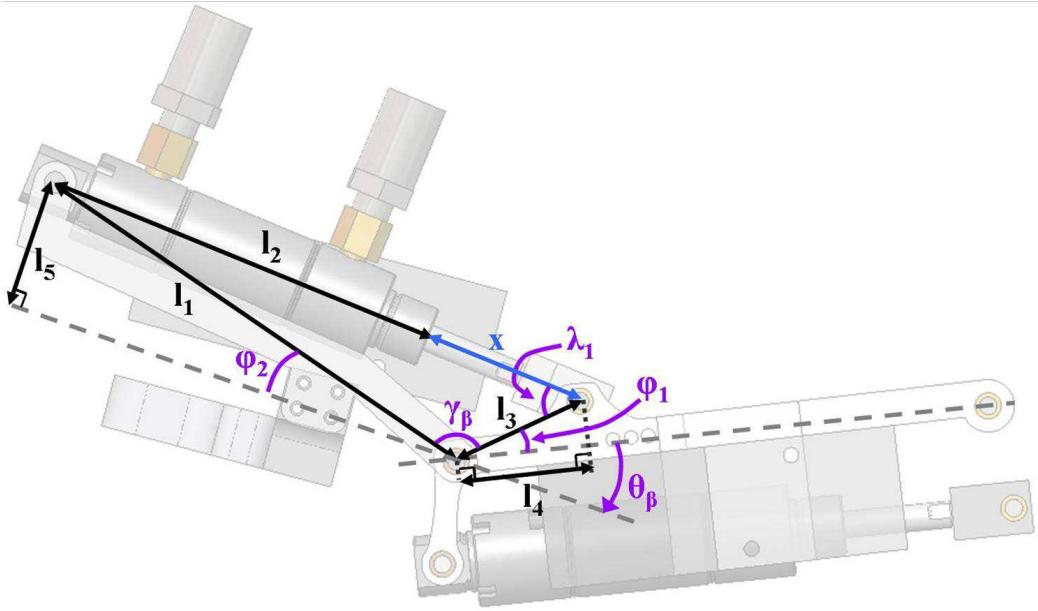

Figure 35: Geometry of a Beta joint

$$x = -l_2 + \sqrt{{l_1}^2 + {l_3}^2 - 2l_1 l_3 cos(\gamma_\beta)}$$
$$\dot{x} = \frac{2l_1 l_3 sin(\gamma_\beta)\dot{\gamma_\beta}}{2({l_1}^2 + {l_3}^2 - 2l_1 l_3 cos(\gamma_\beta))^{1/2}} \tag{5.2.3}$$

$$
\begin{aligned}
\gamma_\beta &= 180\,^\circ - \phi_2 - \phi_1 - (-\theta_\beta) \\
\dot{\gamma_\beta} &= \dot{\theta_\beta} \\
\phi_1 &= cos^{-1}(l_4/l_3) \\
\phi_2 &= sin^{-1}(l_5/l_1)
\end{aligned}
\tag{5.2.4}
$$

where θ_β has a range of -1 to -67 °. For the gamma joint:

$$
\begin{aligned}
x &= -l_2 + \sqrt{{l_1}^2 + {l_3}^2 - 2l_1 l_3 cos(\gamma_\gamma)} \\
\dot{x} &= \frac{2l_1 l_3 sin(\gamma_\gamma)\dot{\gamma_\gamma}}{2({l_1}^2 + {l_3}^2 - 2l_1 l_3 cos(\gamma_\gamma))^{1/2}}
\end{aligned}
\tag{5.2.5}
$$

$$
\begin{aligned}
\gamma_\gamma &= 180\,^\circ - \phi_2 - \theta_\gamma \\
\dot{\gamma_\gamma} &= -\dot{\theta_\gamma} \\
\phi_2 &= sin^{-1}(l_4/l_1)
\end{aligned}
\tag{5.2.6}
$$

where θ_γ has a range of 17 to 95 °.

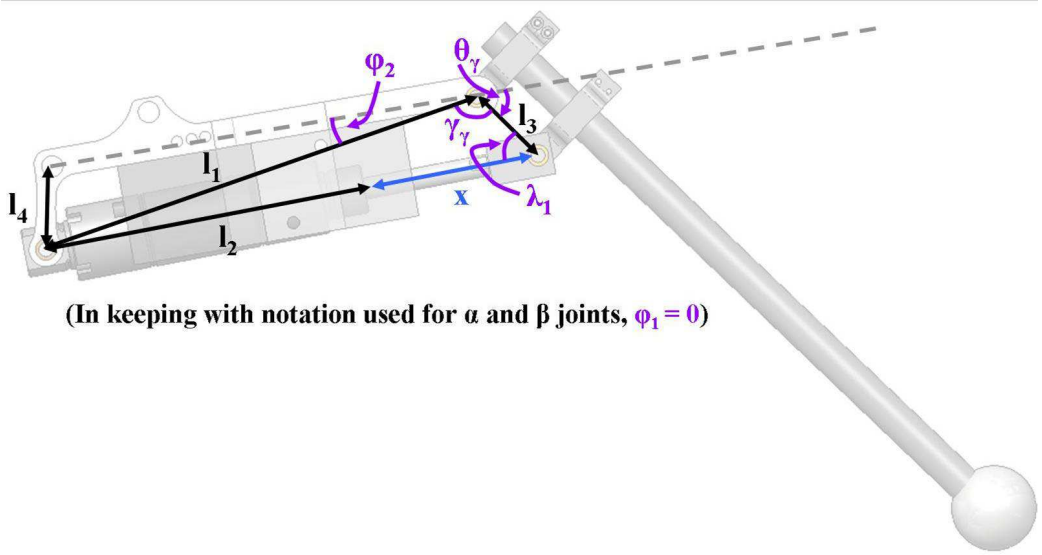

Figure 36: Geometry of a Gamma joint

To ensure realism of the angle to piston extension equations, the derived relationships were compared to simple linear versions found by measuring the extents with a protractor and interpolating between the extents.

Another necessary conversion is introduced as the cylinder model, which produces a net output force, is combined with SrLib, where each joint accepts torque commands about the pivoting joint. Using the instantaneous joint geometry displayed in Figures 34, 35, and 36, and variables introduced in equations (5.2.2), (5.2.4), and (5.2.6), the required torques are defined in equations (5.2.7) (Alpha joint), (5.2.8) (Beta joint), and (5.2.9) (Gamma joint):

$$T_\alpha = l_3 F sin(\lambda_1) = l_3 F sin(sin^{-1}(\frac{l_1 sin(\gamma_\alpha)}{l_2 + x})) = \frac{l_3 F l_1 sin(\gamma_\alpha)}{l_2 + x} \tag{5.2.7}$$

$$T_\beta = l_3 F sin(\lambda_1) = l_3 F sin(sin^{-1}(\frac{l_1 sin(\gamma_\beta)}{l_2 + x})) = \frac{l_3 F l_1 sin(\beta\alpha)}{l_2 + x} \tag{5.2.8}$$

$$T_\gamma = l_3 F sin(\lambda_1) = l_3 F sin(sin^{-1}(\frac{l_1 sin(\gamma_\gamma)}{l_2 + x})) = \frac{l_3 F l_1 sin(\gamma_\gamma)}{l_2 + x} \tag{5.2.9}$$

5.3 Simulink Model

The Simulink model expands upon the actuator model developed in Chapter 4, resulting in a file that, together with SrLib, can simulate one 3-DoF leg (see Appendix C for file and associated scripts). The core components consist of actuator models customized to each joint and blocks for angle and torque conversions discussed in section 5.2. Additionally, inputs are provided to each actuator in the form of predefined or recorded signals, and can be fed through a controller that corresponds exactly to the one used on the actual robot for closed-loop tests. All important signals are recorded and saved on the target object. Other key components include a network communication strategy appropriate for realtime operation, and several analysis and debugging tools.

5.3.1 Analysis and Debugging Tools

One of the largest areas of concern in the integration of non-realtime SrLib and the realtime actuator model was the potential mismatch of simulation clocks. In order to analyze this potential limitation, the actuator simulation on the target was outfitted

with a block that timestamped packets and sent them to SrLib, where they were received and returned to the target by the next send thread. These packets were then recorded at the time they were received again by the target, documenting the overall travel time as well as any notable pauses in communication. Additionally, the time of the SrLib clock was passed back to the target so that the coordination of the realtime xPC target clock and the non-realtime SrLib clock could be studied more closely.

The packet passing analysis was combined with a program known as Wireshark [5] to analyze incoming and outgoing signal data. Wireshark allows users to monitor network ports and track all incoming and outgoing packets, including their type, origin, destination, and content, among other features. Together, these tools provide the operator with a thorough understanding of simulation synchronization. The packet passing tools can be used to see if a problem exists and how it affects the simulation, while Wireshark allows the user to see what component is causing problems. However, Wireshark should not be run continuously since it uses up system memory and could adversely affect simulation performance.

5.3.2 Network Setup

In typical xPC Target configurations, the target communicates, first and foremost, with the host PC, as discussed in Chapter 3. On systems running xPC Target, other UDP connections such as communication with the Phantoms or SrLib could until recently (2011) only occur over the same network card on the target side. This type of configuration was initially implemented within the Simulink model used for validation, but was found to result in models clouded with high frequency oscillations. Using the debugging tools noted in the previous system, it was shown that these problems were due to pauses up to 100 ms in communication between SrLib and xPC Target, caused by the fact that xPC Target does not give UDP realtime priority, instead running it as a background process with a buffer. These observations are shown in Figure 37: by

filtering the data so that only UDP packets were displayed, it could be clearly shown that the simulation started running when the first UDP packet was received from the target, or at 9.231 seconds. Looking at the results of the timestamped packet transfer, an initial large pause was found at about 0.2 seconds – in Wireshark, this instant can be clearly seen at 9.452 seconds, when SrLib returns a message to the target (Figure 37). This packet, no. 957, is the last packet to be sent for about 90 ms (SrLib cannot respond unless given a packet due to the serial structure discussed in section 5.1). Similar pauses can be matched to Wireshark data at 1.85 s, 11.05 s, and so forth. Since SrLib is incapable of sending a message unless it first receives one, these occurrences would imply that the target is failing to send messages.

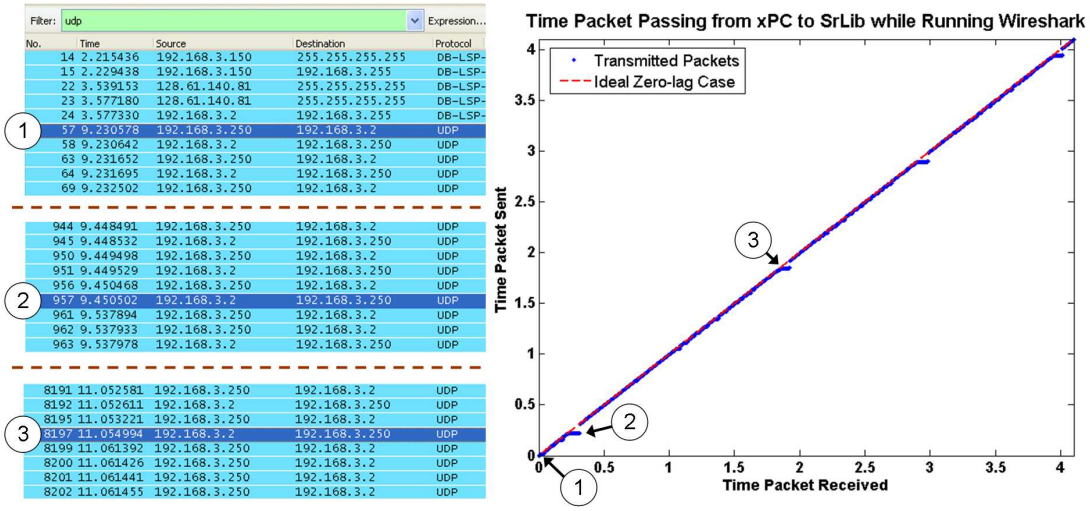

Figure 37: Pauses in plots can be tracked in Wireshark

Fortunately, the newest release of MATLAB contains realtime UDP blocks that use a separate, dedicated network card to ensure the non-host/target communication is given realtime priority. Following an upgrade to the most recent release, these blocks were installed, using a separate local network running across a switch. Tests run using the packet passing tools showed that delays have been eliminated and that the new configuration resulted in a total travel time of 3 - 4 ms. Host/Target

communication was maintained with a crossover cable.

5.4 Validation Platform Configuration

Several configurations of machines were tested to achieve better performance from the target and, in particular, from the non-realtime SrLib Simulation. Using packet passing tools and Wireshark, it was found that SrLib was capable of running within the desired time step on all machines, and the operating requirements for the target PC were relatively minimal. Instead, the primary differentiating factor among the choice of configuration was the effect of viewing window rendering on the system's overall performance. As shown in Figure 32, the viewing window updates every n steps, where n is set by the user. Every time and update occurs, the frame is redrawn, temporarily pausing all other SrLib activity. The duration of these pauses varies by choice of SrLib machine. The best results were seen when the same computer was used for both SrLib and as a host, appearing to depend more on the overall amount of physical memory than the processing power or memory priority allocated to it. Based on these tests, the two machine configuration (target, host/SrLib) was employed in further validation tests. Appendix D provides configuration and operation guides for both this case and a more general one.

During tests, processes on the SrLib/Host computer were kept to a minimum, since spikes in activity due to Windows' non-realtime nature could pause SrLib in the same way that rendering updates can affect the program.

The final configuration featured a target hosting the three-actuator Simulink model, UDP to the host and SrLib over independent networks/network cards, and an SrLib model of a quadruped robot with three actuated joints (the others were simulated but uncommanded) with a base that was fixed in space so that the front right leg could be effectively simulated and compared with an equivalent implementation on the actual two-legged robot.

5.5 Experiments & Results

Tests were performed on each joint individually and on all three simultaneously using prerecorded data from a gait-like motion. The tests varied somewhat from those performed in section 4.3. Whereas the open loop tests noted in section 4.3 had been aimed primarily at validating actuator performance in for small equivalent orifice areas and to show open-loop friction accuracy, the focus of tests in this section was primarily on validation of the actuator dynamics in context of their arrangement within the leg and together with leg dynamics. A variety of closed-loop tests were performed using the same controllers found on the physical model and focused on showing clear similarity between simulated and actual responses, using parameters such as rise time, overshoot, settling time, and correspondence in the effects of higher-order dynamics. The simulation's utility was further examined by performing tests to analyze the computational demands of individual simulation components.

5.5.1 Open Loop Experiments

Despite the fact that open loop tests were not formally documented, they were used to ensure that the cylinder behaved appropriately, e.g., to ensure that a given voltage corresponded to the appropriate cylinder direction at realistic extension velocity and cylinder pressures. Each actuator model was checked to ensure that it was properly configured.

5.5.2 Closed Loop Experiments

Closed loop tests were performed on each individual joint while the others were held steady, as well as on all three simultaneously. In the individual tests, the joints not tested were first extended fully using a fixed open loop valve command. This motion served not only to standardize the tests, but also to ensure that the simulation had essentially the same initial conditions as the physical testbed. Because of this initialization phase, comparison data was selected beginning at the 10 second mark.

For each test, the RMSE of the simulated position with respect to the actual position response provided an instant measure of simulation success. Further understanding was achieved by examining similarities in simulated and actual responses to different types of closed loop commands.

Step responses were used as the primary tool for analyzing the accuracy of individual joints. With visually apparent parameters like rise time, overshoot, and settling time, step responses provide a strong sense of whether or not two sets of dynamics (physical and simulated) are alike. However, a challenge with step responses is that their introduction of discontinuities may be too severe and not representative of any motion that the robot would realistically receive from a user. Therefore, trapezoidal profiles were additionally used in cases where the step command appeared too severe. Trapezoidal commands provide a way of keeping the parameters used to compare the physical and simulated systems, but with more gradual commands. Additionally, slow and fast sines were used to provide a different type of comparison, focusing on effects of friction at the curve peak and response of the system to a constantly changing command likely representative of the actual Phantom command.

Following examination of each individual joint, tests were run that showed the system's response to a user generated walking motion. Using the Phantom joysticks, an operator manipulated the leg in a cyclic walking motion several times, beginning with several slow steps, followed by several of medium duration, and concluding with a series of fast walking motions. Closed loop angle paths for each joint were recorded and played back in simulation, thereby providing a comparative data representative of the actual desired simulation behavior. The measured position data from the actual robot shown here is unfiltered, while the pressure data was filtered for signal clarity, using the filters introduced in section 4.2.1 and documented in Appendix A.

The tests that follow are introduced in increasing complexity, beginning with the joint closest to the end effector, and concluding with all three joints actuated

simultaneously.

5.5.2.1 Gamma Joint

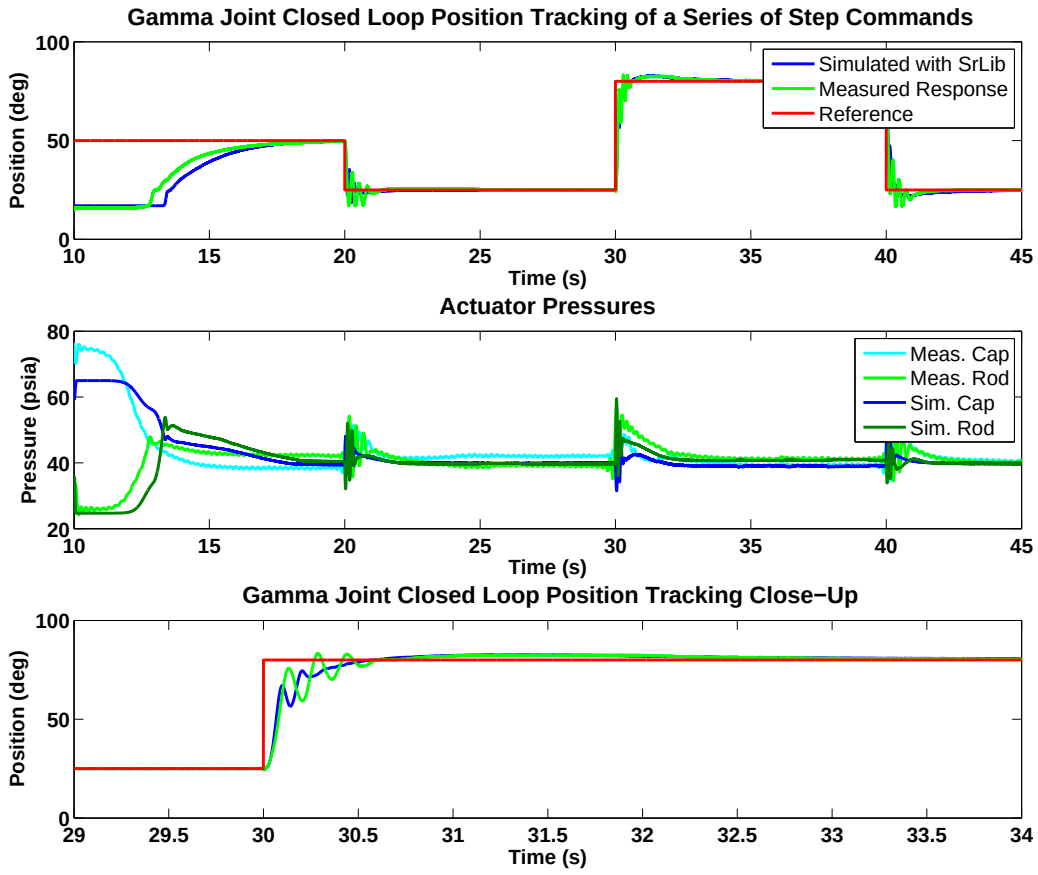

Figure 38: Closed loop tracking of a series of steps for the Gamma joint

Figure 38 shows simulated and actual responses to a series of step commands spanning most of the range of the Gamma joint (17 to 95°). The Root Mean Squared Error (RMSE) of the simulated to actual position response was found to be 2.6°, less than 3.5% of the total range of the joint. The level of accuracy is evident from the top plot, which clearly shows that the simulated response closely matches that of the actual system. A close-up of one of the steps in this response, shown in the bottom plot, displays the key similarities: the responses begin at the same time following the initial command and with the same slope, have the same overshoot, and converge

together to the same steady state value. Some of these parameters can be easily quantified, such as rise time, overshoot, and settling time. Rise time shows that while the general response is good, there is still some apparent variation – over a course of step responses, rise time ranged from practically zero error to almost 50%, indicated by an average error of 0.13s with a 0.19s standard deviation over the course of steps ranging from 0.11s to 1.3s rise times. However, other behavioral measurements were much more constant: over the course of three step responses, overshoot differed by 0.4 - 5°, or about 6% of the total operating range, while error in settling time of the simulated responses remained within 15, 15, and 50% of the settling time of the actual responses. In the last case, the response was marked by a very slow final, practically linear slope, such that the simulated response took an additional 0.66 seconds to go from under 10% to within 5% of the final value, resulting in a markedly higher error in settling time. Further verification of model accuracy is demonstrated by the presence of oscillations resulting from the third-order dynamics of the actuator model that occur in the simulated response. These also occur in the actual response at the same times, though amplitude and phase varies slightly. Furthermore, the simulated behavior of the cap side and rod side pressures (relative to each other) matches those of the actual system, showing correspondence at multiple integration levels of the third order system and reinforcing the claim that the both the simulated actuator and modeled robot dynamics are behaving realistically.

While overall performance of the simulated Gamma joint is good, it is worth noting from the pressure plot that the measured rod-side pressure at first spikes to approximately 75 psia, a value unachievable in simulation since the supply is set at 50 psig, or approximately 65 psia. On the physical system, this was enforced by a regulator, as with the simple model from Chapter 4. However, the regulator does not function in an ideal manner, likely due to the distance between the regulator and the actuators, the multitude of actuators using the same air supply, and the

inherent inaccuracies of the manual regulator. Such effects were left unmodeled for this simulation and seemed to do little to adversely affect the results, but can be seen periodically in simulation results, as in Figure 38.

The variation in oscillation amplitude is likely due to some unmodeled effects such as joint friction. A less steep command, such as a trapezoidal profile, limits the effects of these approximations on the response and also provides a command trajectory closer to one that could actually be provided by the user (Phantom commands will not cause such a rapid change in command unless there is significant delay between sampled points). This is shown in Figure 39. The response from the trapezoidal profile also demonstrates an effect likely due to friction – at the start of each command, the response takes a moment to actually respond, waiting until the pressures build up to provide enough force for initial motion.

While the step and trapezoidal profiles show a strong correlation between simulated and actual dynamics of the actuator integrated into the robot, they don't fully represent the type of behavior that is representative of the bulk of what the robot will be asked to do: walk. This type of motion is frequently changing at different rates. A sinusoidal command, depicted in Figure 40, is more representative of this situation.

Simulated position response has an RMSE value of 1.5° with respect to the actual response, confirming the accuracy shown in the figure. Lag of the response signal with respect to the reference varies, but is generally greater than 0.15s for the simulated response and approximately 0.2 seconds for the actual response, confirming an accuracy within a 25% tolerance. Relative relationships of cap and rod side pressure are again similar to the actual behaviors, and third order dynamics are still evident in the position response. However, the effect of friction at the peaks of the sine wave is not truly captured. Instead, slow piston motions never completely stop and therefore never fall within the velocity range in which stiction would be applied long enough

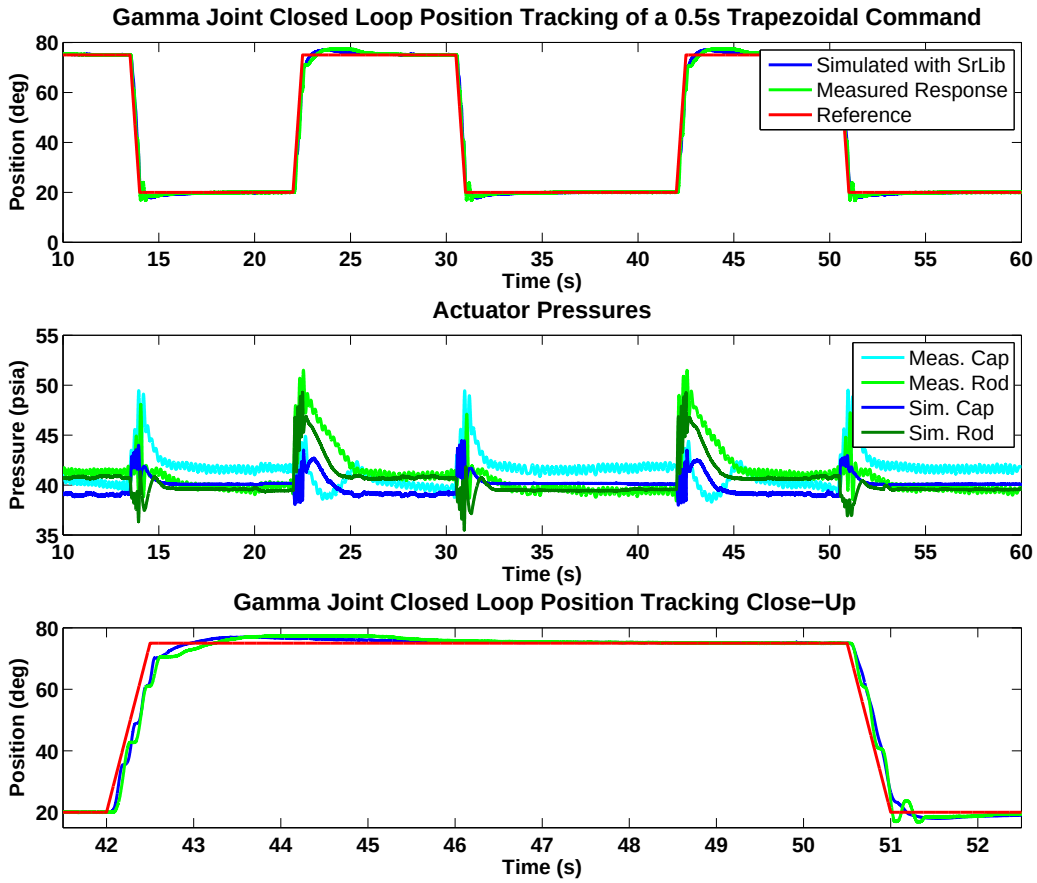

Figure 39: Closed loop tracking of a series of trapezoidal profiles for the Gamma joint

for the effects to be apparent, as discussed with regard to the simple actuator model in section 4.4.2.

5.5.2.2 Beta Joint

Like the Gamma joint, the Beta joint was initially subjected to a sequence of step commands across its range of operation (Figure 41). As seen in the figure and verified by a position RMSE of approximately 4°, or about 6% of this joint's operating range, the simulated response did not match the actual trajectory as closely as was seen with the Gamma joint, shown by the clear deviations of the simulated settling and rise time from those of the actual response (they both differed by a factor of 2 on average). Even

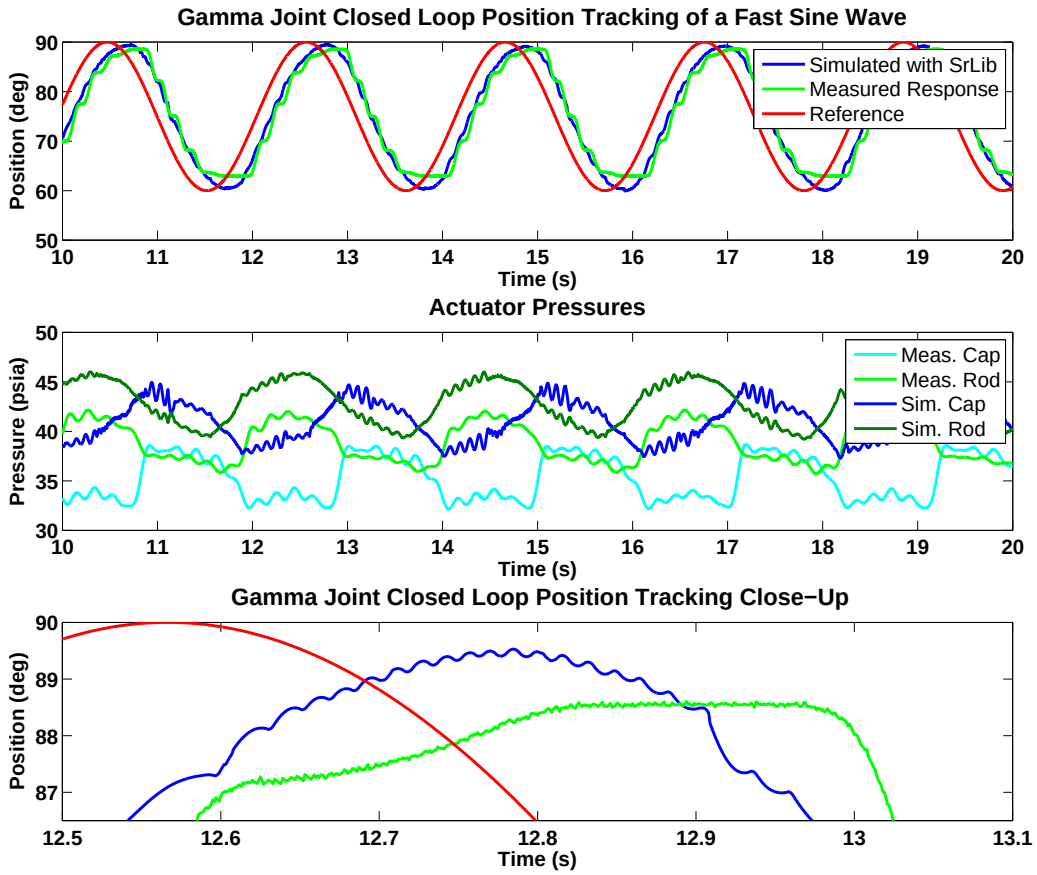

Figure 40: Closed loop tracking of a sine wave for the Gamma joint

in these cases, the response was still reasonable for a pneumatic actuator (it looked a lot like a slower version of the Gamma response) and matched many of the core parameters. It can be seen from the close-up that while rise time was not captured correctly, overshoot, slope, and the elements of third-order actuator dynamics were evident in both reality and simulation. Relative cap-to-rod side pressure relationships also behaved realistically.

A fast sine wave (Figure 42) was again used to provide a command similar to one that a user might generate, showing considerably better overall accuracy in simulated position response than was seen with step commands, indicated by its RMSE of 2° and lag of the simulated and actual responses with respect to the reference of 0.16 and 0.2 seconds, respectively. The observed behavior was similar to that seen for a

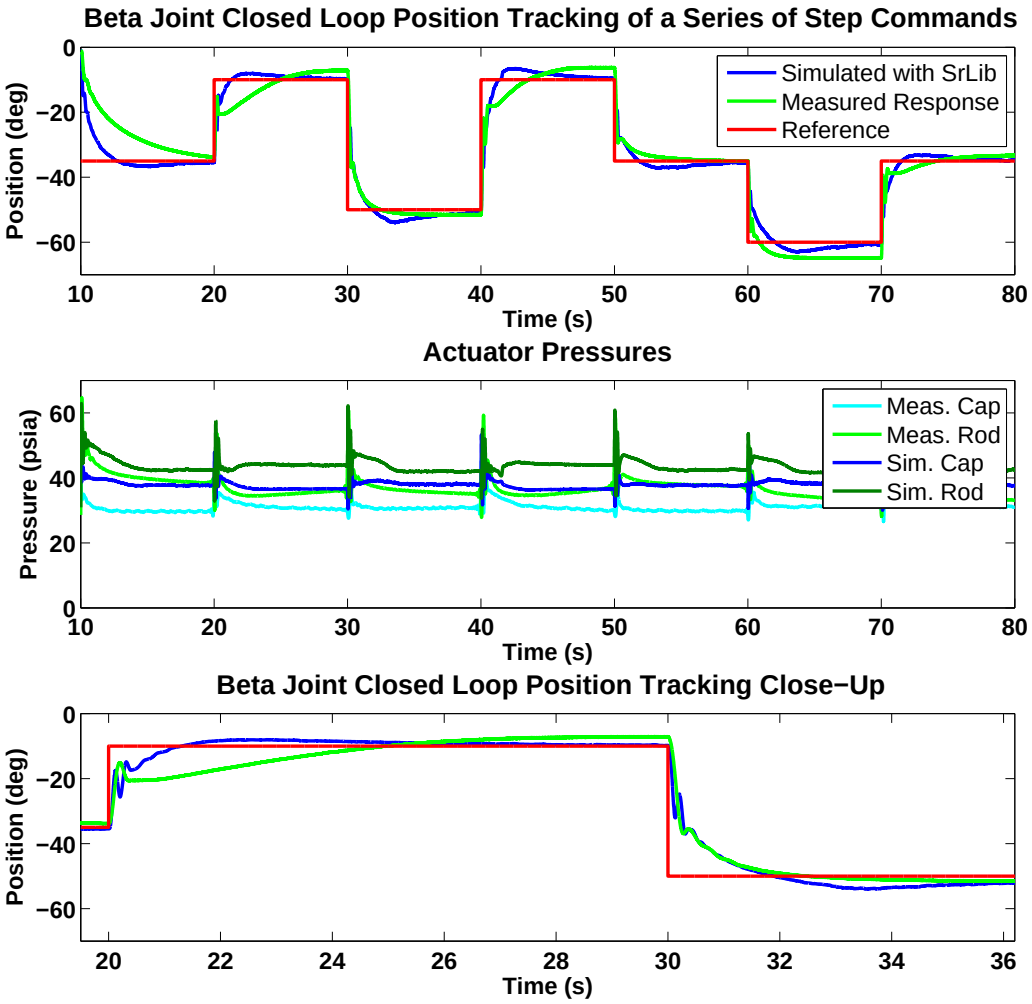

Figure 41: Closed loop tracking of a series of steps for the Beta joint

Gamma joint: while the simulated response closely matched that of the actual robot in form and dynamics, it failed to accurately capture the effects of friction at the sine peaks, again casting doubt on the utility of the advanced friction model within this type of simulation.

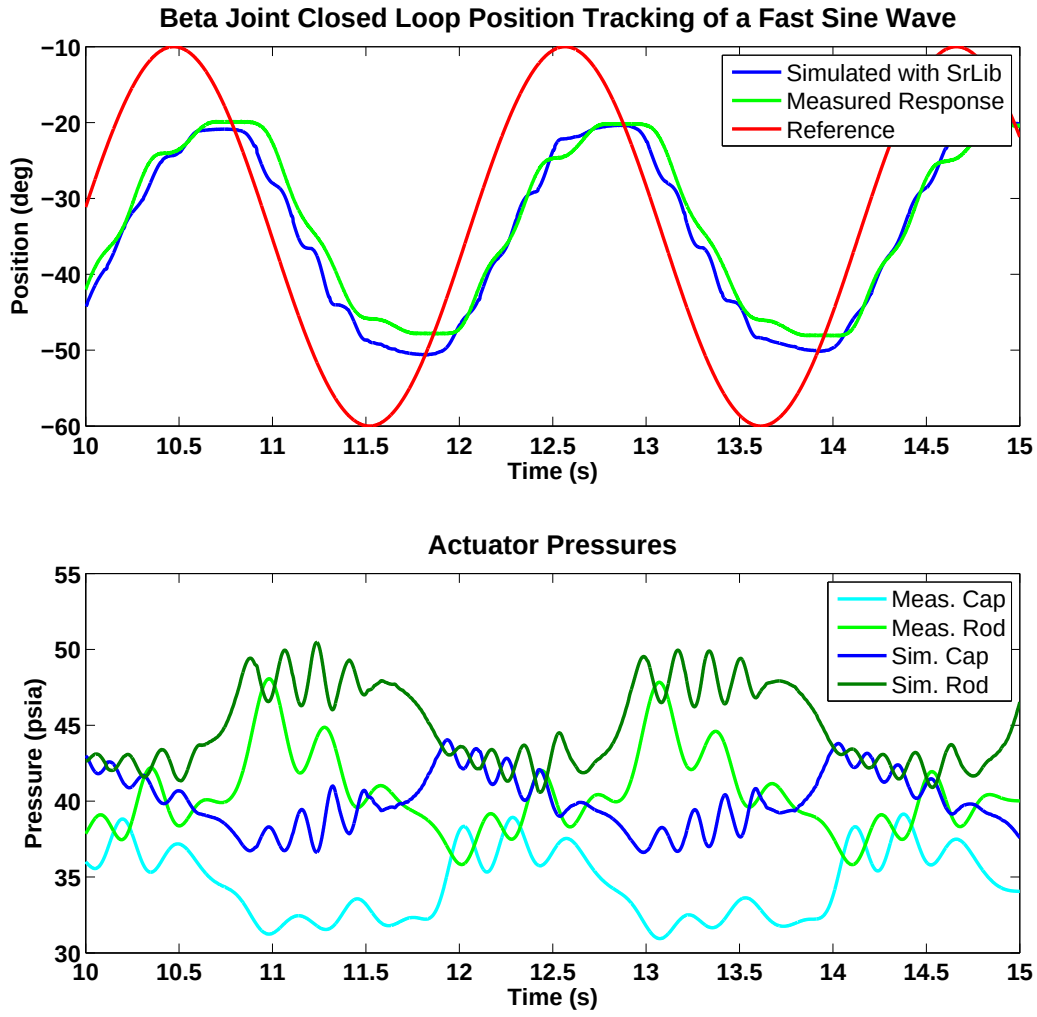

Figure 42: Closed loop tracking of a fast sine wave for the Beta joint

5.5.2.3 Alpha Joint

Figure 43 shows the response of the Alpha joint to a series of step tests. In addition to adding robot dynamics, this joint actually uses a cylinder with 3.5 inches of stroke length, making it approximately 2 inches longer than the other cylinders, a parameter that was adjusted in the setup file. The simulation and tests both show a tendency to lightly damped oscillations or limit cycling with the gains that were initially used. Instead, in later full leg tests, the gain used on the controller in simulation was reduced to provide a smoother alpha joint response, as will be seen in the next section.

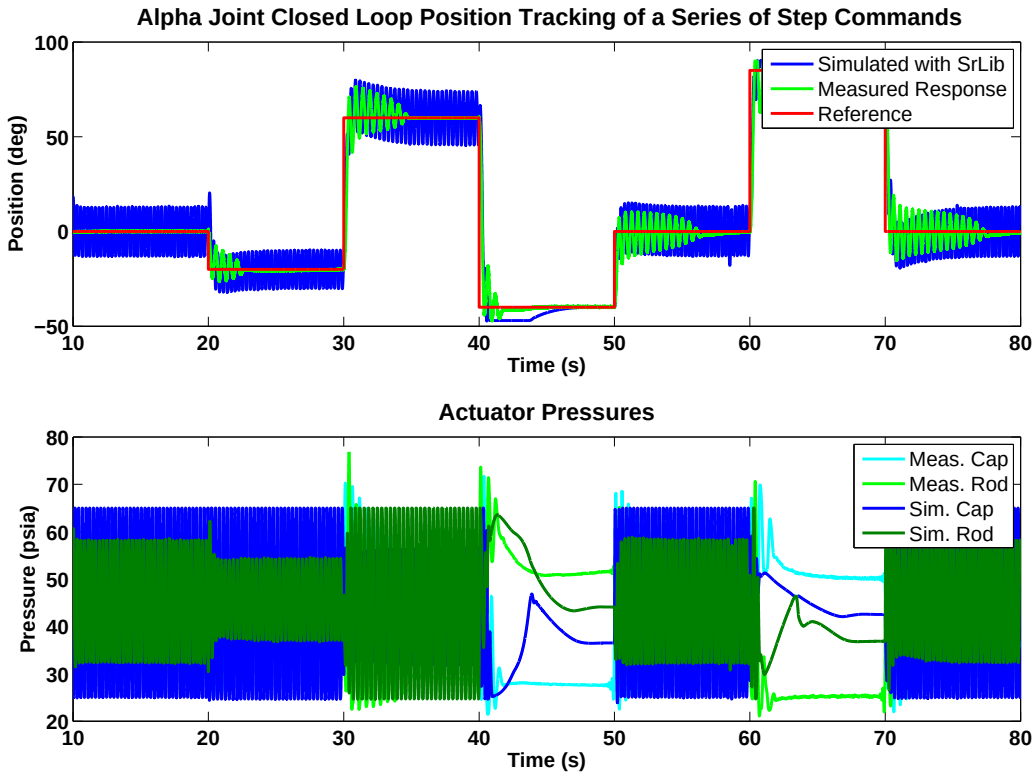

Figure 43: Closed loop tracking of a series of steps for the Alpha joint

5.5.2.4 Leg Commands

Figure 44 shows the actuator position tracking and pressure responses to a walking motion. This motion can be described as moving the end effector in a cyclic fashion starting from the rear bottom of its workspace, arcing upwards and then back down at the front bottom, and then dragging it back again, as is done in walking motions. The figure shows strong correlation between simulated and measured results for the Gamma and Beta joints, which both have simulated position RMSE values under 2.5°, but the oscillations in the Alpha response observed in section 5.5.2.3 continue to be problematic, contributing towards a high RMSE in simulated position of 10.2°.

Since the purpose of the simulation is to provide actuator behaviors that are realistic and accurate but should at least be reasonable, adjustments were made to the alpha actuator controller to obtain motions closer to those observed on the actual

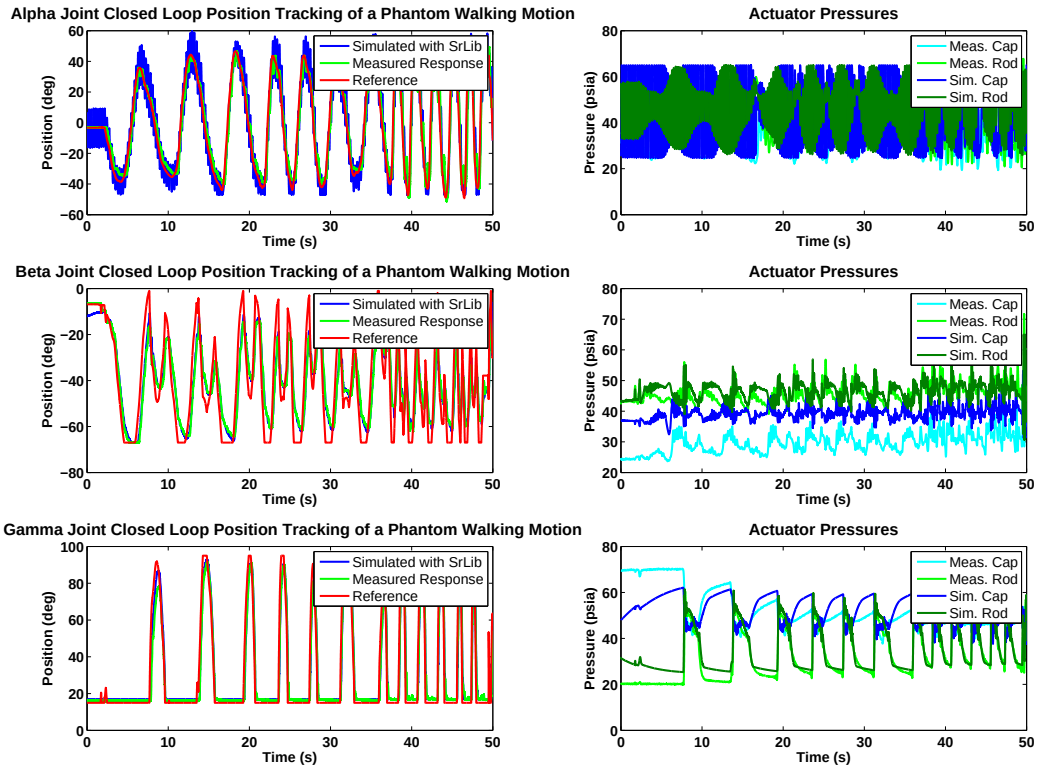

Figure 44: Joint tracking of a characteristic leg walking motion

testbed. As noted in the previous section, the response of the alpha joint is not unrealistic; rather, it appears to simply overshoot too heavily, resulting in continued oscillations. Accordingly, the overall gain on the PID controller was reduced by 25%, resulting in the response seen in Figure 45.

The new alpha joint position response has an RMSE of 7.3°, reduced from its previous value. However, this value is likely made larger by phase delay in the response and does not provide a complete picture. The figure shows the full impact of reduced gains on alpha joint performance. It can be seen that the alpha joint still possesses some of the initial oscillations observed in the actual result, but is much smoother than with the original controller. Beta and Gamma joints match both position and pressure closely, having RMSE's of 2° and 2.5°, respectively, showing also that the simulation is robust to disturbances, as Beta and Gamma act along the same axis. This result demonstrates that this simulation is capable of providing realistic behavior

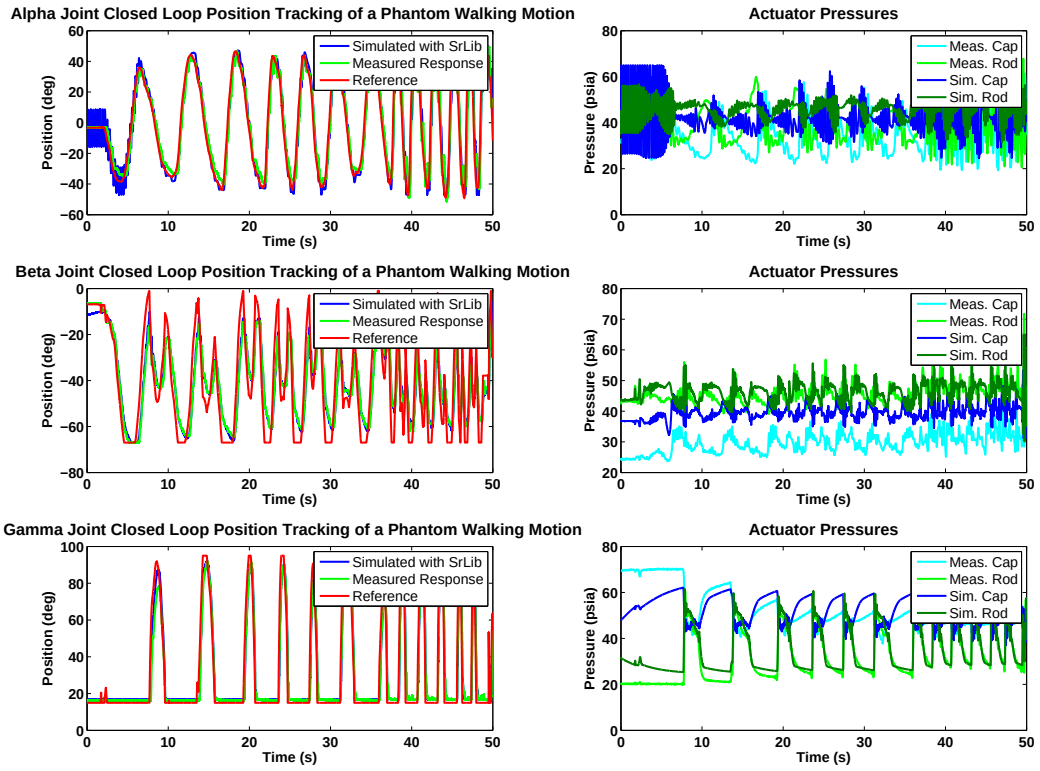

Figure 45: Joint tracking of a characteristic leg walking motion using an Alpha joint controller with a reduced gain

representative of a multi-DoF pneumatically actuated leg across a correctly configured network of simulation platforms.

5.5.3 Computational Demands of Individual Simulation Components

One of the goals of this thesis is to examine not only the plausibility of a dynamic simulation that includes the effects of pneumatic actuation, but also its utility. The actuator model used in Chapters 4 and 5 emphasizes detail, making use of fitted equivalent orifice areas, supply and exhaust pressure curves, and a complex friction model. However, the necessity of this level of accuracy is questionable, especially considering behaviors such as those seen in Figures 40 and 42, where it was shown that friction did not accurately model the situation. The following analysis examines the computational cost of several key simulation components, using a Simulink model

containing just enough components to model the Gamma Joint in simulation, provide a variety of inputs, control the signal with a PID controller, collect data, and debug network connections.

The metric used to analyze computational cost is known as task execution time, or TET. MATLAB allows the user to record a vector of TETs for each execution cycle when the target is run. TETs can be reduced by using a target with a faster processing power, but relative to an individual computer, individual model TETs provide a good measurement of how large or computationally demanding each model is. By eliminating individual components of the simulation and comparing the average TET of the reduced model with that of the original, it was possible to show, to some extent, the contribution of individual computations to the overall simulation execution time.

Table 5.5.3 shows the TETs for different models. Since the TET can vary depending on choice of input, one input type was used for all open loop tests and another for all closed loop tests. Standard deviations of the TET are shown in the third column, and are generally about 5%, meaning that the most significant reductions are those considerably greater than 5%.

First, an open loop test was used to be able to remove the controller without severely affecting performance. The controller was then reinstated and used for closed loop tests.

In the fourth and fifth entries, the removal of advanced degree and torque conversions replace the geometric relations derived in section 5.2 with simple linear relationships to map stroke length to joint angle.

In the sixth entry in the table, the Stribeck-Tanh friction model is removed and replaced with a simple, velocity-depended friction term, using a coefficient that had been derived in past cylinder models [15].

The next removal examines the impact of the underlapped valve model, which

uses fitted equivalent area and exhaust/supply pressure curves. This block was first replaced with a simpler block that used a linear equivalent area and maximum exhaust and supply pressures, and then modified so that the equivalent area and pressure curves could be independently removed. It can be seen here that removing individual components is not exactly equal to replacing to whole block, thereby showing that the actual structure of the block diagram has an impact on its performance as well.

Finally, the contributions of extra tools are examined, first as a block, then individually.

Table 3: Impact of removing components on the TET of the Gamma joint actuator model

Case	Average TET (x 10^{-5} s)	TET Standard Deviation (x 10^{-5} s)	Change from Complete Model (x 10^{-5} s)	Percentage Reduction
Open Loop Tests				
Complete Model	3.7260	0.1830	–	–
No Controller	3.6819	0.1793	-0.0441	1.2
Closed Loop Tests				
Complete Model	3.7498	0.2225	–	–
No Advanced Degree Conversion	3.7606	0.1914	0.0108	-0.3
No Advanced Torque Conversion	3.7117	0.1902	-0.0381	1.0
Simple Viscous Friction instead of Stribeck-Tanh	3.3255	0.0205	-0.0424	11.3
Simple Valve Model	2.8471	0.1875	-0.9027	24.1
Simple Valve Model: Simple Equivalent Area	3.2728	0.2032	-0.4770	12.7
Simple Valve Model: Simple Pressure Estimates	3.4525	0.2237	-0.2973	7.9
No Extra Tools	3.4584	0.2243	-0.2914	7.8
No Extra Tools: Packet Transfer Analysis	3.7272	0.2285	-0.0226	0.6
No Extra Tools: Multiple Input Choices	3.6211	0.2322	-0.1287	3.4
No Extra Tools: Scopes	3.7238	0.2266	-0.0261	0.7
No Extra Tools: Recorded Output Signals	3.6325	0.1767	-0.1173	3.13

Based on these results, it can be shown that the most significant adjustable TET costs result from the advanced friction model (11%), detailed valve model (24%), and extra analysis tools (8%). Provided with a simulation that has been effectively validated, the extra analysis tools can be largely reduced by limiting recorded data to a few critical signals, eliminating scopes and extra input options, and using packet transfer analysis only when something changes with the network configuration.

However, the friction and valve model present a potential a compromise. The primary goal of this valve model is to accurately simulate behavior where the equivalent orifice area is very small, or equivalently, when the valve command is operating in the

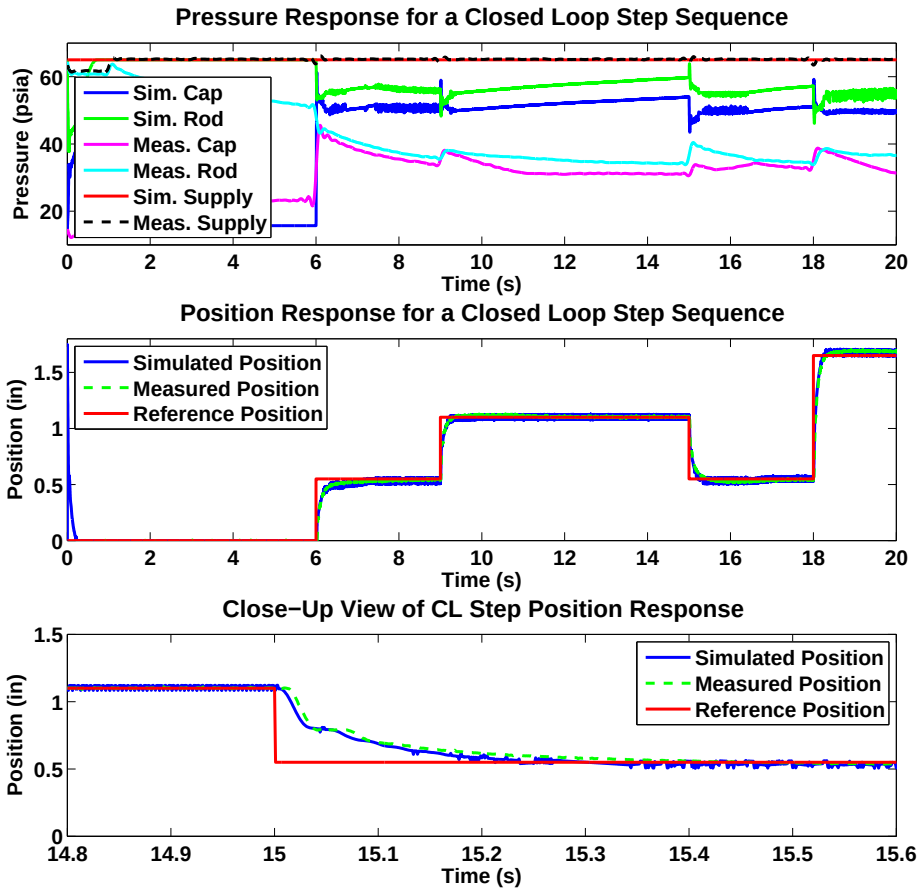

Figure 46: Closed loop tracking of a series of steps for the Gamma joint using a linear equivalent orifice area and fixed model supply and exhaust pressures

middle of the range, a region that is frequently accessed in this particular simulation. Indeed, running the simple Simulink actuator model from Chapter 4 using a fixed discharge coefficient and linear orifice area with the parameters from [15] instead of fitted orifice area curves, the differences are quite clear (Figure 46). While overall performance is similar to that of the advanced valve model, noted by the similar RMSE value of 0.049 inches, behavior at small orifice areas is noticeably poorly represented, as evidenced by the inability of the model to ever reach a fixed position during a step response. Extension of this simplified model to a Gamma joint in the full dynamic simulation (Figure 47) aggravates the problem, resulting in further loss of simulation accuracy due to increased oscillations at certain steps and less correlation between

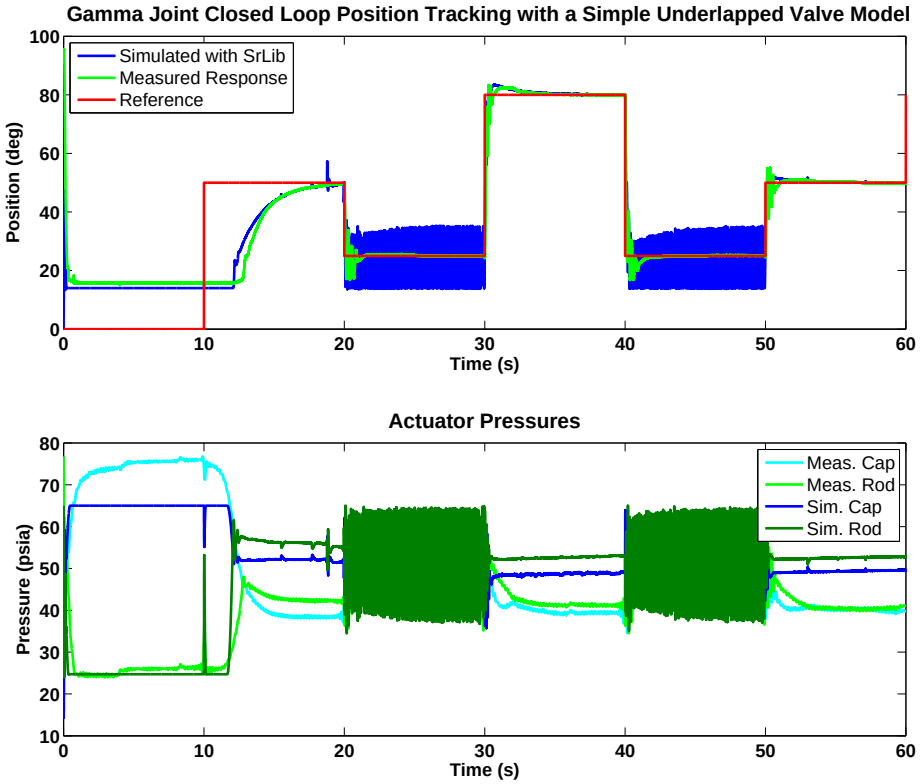

Figure 47: Closed loop tracking of a series of steps for the Gamma joint using a linear equivalent orifice area and fixed model supply and exhaust pressures

simulated and actual pressure behavior.

Friction provides a similar design choice. Figure 48 shows a test using the simple, viscous friction term applied on the simple actuator model from Chapter 4, demonstrating performance that is not quite as accurate as with the Stribeck-Tanh model, but certainly acceptable for general application, having a similar RMSE of 0.046 inches. The primary difference from the previous results using an advanced friction model is the lack of high frequency oscillations at steady-state, as observed with a Stribeck-Tanh model in place, and immediate motion of the simulated step response, which begins to move 0.01 seconds before the measured response, due to the lack of a stiction model. Overall, the results have similar rise and settling times to the actual performance, and even demonstrate some of the same oscillations resulting

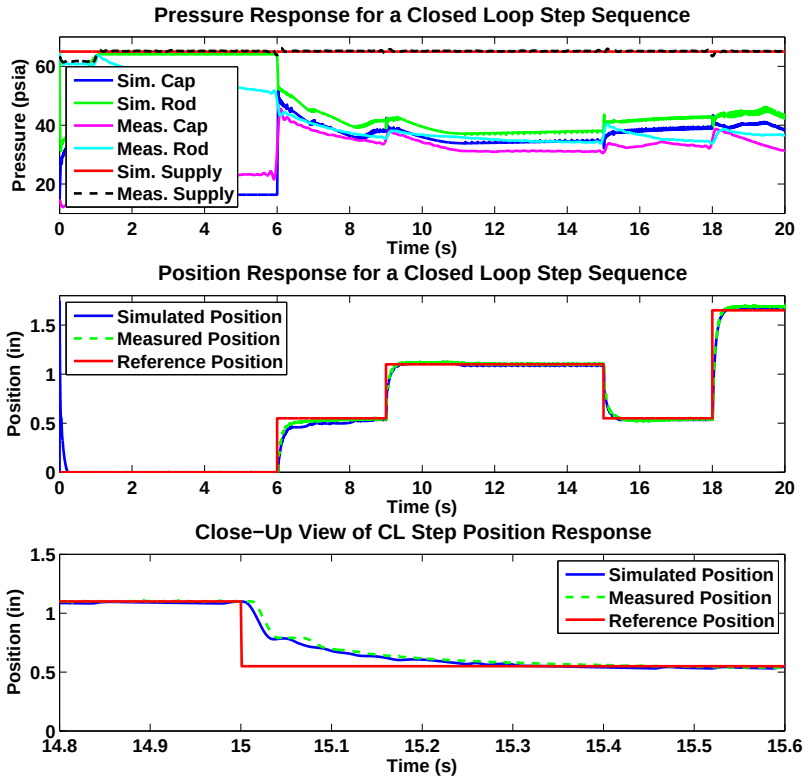

Figure 48: Closed loop tracking of a series of steps for a vertical actuator modeled in Simulink only (Chapter 4 model) using a simple viscous friction model

from third-order dynamics, albeit with slightly varying amplitudes and time of occurrence. However, when the same model is applied to a Gamma joint in the full dynamic simulation, performance suffers, providing heavily oscillatory and therefore unacceptable results, as seen in Figure 49. This variation in performance is likely due, in part, to the increasingly fragile nature of the simulation as it becomes dependent on more system components and their interactions. However, these effects alone are unlikely to cause the poor performance seen in the figure, given the acceptable results seen with the simple model. Instead, complications may arise with SrLib's solver as a viscous damping term replaces the advanced friction model.

Different solvers are known to have different operating regions; systems whose dynamics fall outside of these regions cannot be numerically solved with the given integrator. It is possible that the change in friction model results in a system that

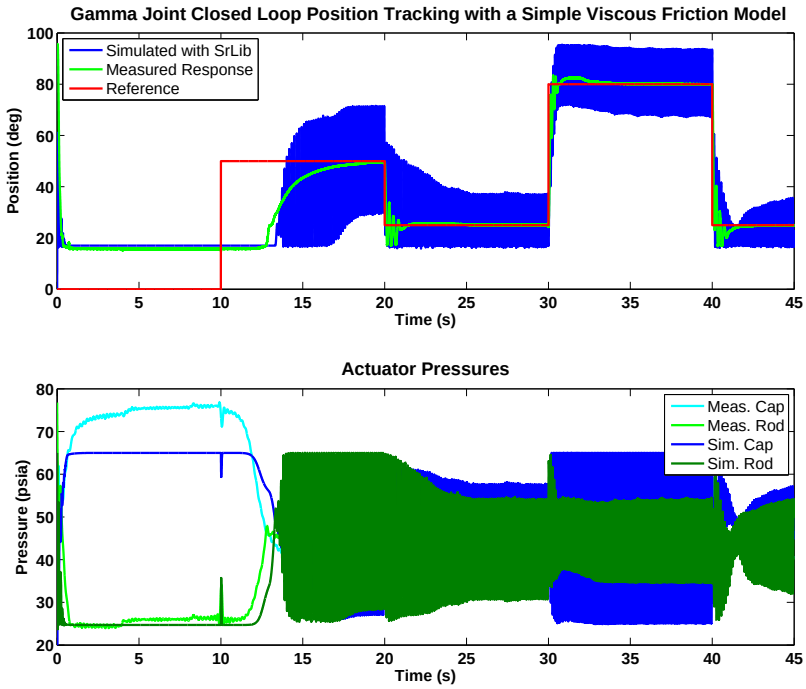

Figure 49: Closed loop tracking of a series of steps for the Gamma joint using a simple viscous friction model

is numerically unstable with given integration configuration. This problem might be avoided by moving damping to SrLib, where it might be modeled in a different fashion. However, while SrLib technically supports a damping term, tests have shown that modifying its value fails to affect simulation results. Even if such a term is successfully added to SrLib, it may still be insufficient to produce a stable solution, and may require further modifications to SrLib or a similar dynamics engine.

Using a viscous term as a friction model, performance will be less accurate than with the Stribeck-Tanh model, but the results from the simple actuator model nonetheless show that the simpler friction model, implemented correctly, could be used to reduce computational needs and complexity while maintaining acceptable performance that follows general behavioral trends.

5.5.4 Summary of Performance and Limitations

Overall, the simulation performed admirably, closely matching the actual Beta and Gamma joint behaviors both in step and sine tests and in the actual leg test subject to disturbances from other joints, and showing similarities in the alpha joint performance when using a reduced controller gain. An overview of component contribution to computation cost showed that the friction and equivalent area models could be reduced in complexity in exchange for faster processing, though simulation accuracy would likely suffer. However, the simulation's current accuracy is not perfect and subject to improvement, due to a variety of approximations and unmodeled effects.

First, minor effects such as joint/bearing friction, or resistance caused by electrical and air supply lines are assumed to be negligible. Additionally, friction within the cylinder has been extensively modeled, though it approximates the stiction region using a steep slope. Furthermore, if a cylinder piston is subjected to high side forces leading to reaction forces not along the piston axis within the cylinder, as could occur with the Beta joint, the resulting friction and resistance would be unaccounted for. These effects, however, generally contribute little to the overall simulation performance and should probably be left unmodeled or lumped into more general parameters.

One primary area of concern is the diminishing accuracy of the step response progressing along the arm from Gamma to Alpha joint. Since the actuator model was verified independently of SrLib, assuming that the associated algebraic equations and joint parameters have been correctly defined, the problem must be due either to integration with SrLib, sensitivity to the accuracy of the model, or robot modeling within the dynamics library. Some error is likely due to the method used to calculate the inertia of simulated robot's components. For the gamma joint, inertia of the femur was computed within SrLib as a thin rod with a density of 2.7 g/cm^3 (the density of aluminum), essentially an exact model of the actual system. The gamma joint is

also the most accurate of the joints in simulation. The inertia model used for the link actuated by the Beta joint, the tibia, was computed by using an oversize capsule and a low density approximation of 0.7 g/cm^3, thereby estimating several centered but spaced out components with one solid item. Results for this joint showed clear similarities, but deviated in the overshoot. Finally, the Alpha joint, which directly actuates the coxa link, demonstrated a response plagued with oscillations. The coxa is modeled as two cylindrical capsules with a density lower than aluminum, when in fact, it is one reasonably asymmetric component. Similarly, the joint actuation is modeled as a torque, when it is actually a force acting on a joint. Though the torque has been modeled with high accuracy, SrLib is capable of modeling prismatic joints that could more accurately capture forces and would also allow the user to include the cylinder in inertia calculations.

However, Phantom tracking tests using three different inertia models showed minimal performance changes. Figure 50 shows alpha joint response for (top to bottom) the model used in earlier tests in this thesis, an imported inertia found in SolidWorks, and a slightly more realistic SrLib model. Though none of these models are perfect (they all assume a fixed inertia despite the moving piston and are lacking in detail) the minimal performance changes cast doubt on inertia as the primary source of model error. Though the inertia will definitely contribute to overall accuracy when all else is correctly defined, as was seen with derivation of Beta joint inertias, another factor may be causing the error currently visible in simulated alpha joint response.

A further possible reason for the Alpha joint's lack of accurate response may be that the alpha joint dynamics lie outside the region that can be accurately calculated by the solver, either in Simulink or SrLib, as was discussed with regard to viscous friction in section 5.5.3. Both Simulink and SrLib used Runge-Kutta based solvers: Simulink was configured to use `ode5`, a 5th order Dormand-Prince integrator, while SrLib used an integrator that combined Euler and 4th order Runge-Kutta to achieve

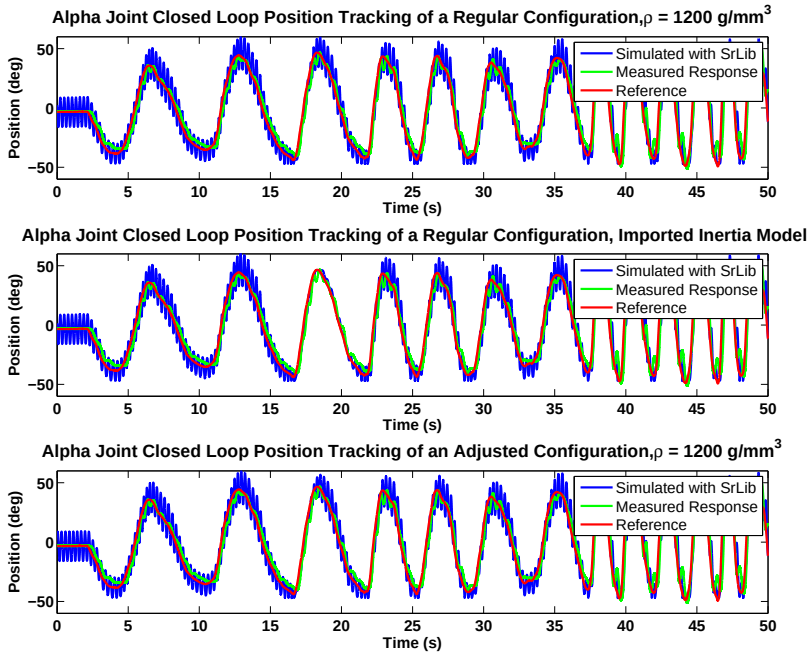

Figure 50: Closed loop tracking of Phantom commands to the alpha joint for three different inertia models

speed and accuracy. Simulink provides an array of available explicit solvers of increasing order, but contains just one implicit solver, which is intended for use with systems that possess stiff dynamics. Based on tests with the simple single DoF model, `Ode5` was found to produce accurate results, and was therefore used throughout the remainder of simulation iterations. Simulink's lone implicit solver, `ode14x`, which uses an extrapolation-based method, was also tested in its default configuration, but exceeded the computational ability of the target, resulting in a CPU overload. Other configurations may be possible that allow the implicit solver to work within the computational limits of this system. Since results from simulation show that solver choice may in fact play a key role in the overall good performance and versatile application of the completed dynamic simulation, a more thorough review of the used and available solvers, their capabilities, and their applicability to this system may be required.

Even if the solver is generally appropriate for the system dynamics, the sensitivity to accuracy of the model could be the cause of poor responses like that seen with

the alpha joint. It is evident from the measured responses that even the physical alpha joint, with its current controller, is critically stable – a region that is notedly difficult to model. In fact, the model's ability to predict a region of instability is already a credit to its accuracy. The observed oscillatory behavior might simply be due to numerical instabilities resulting from the model's in ability to accurately model a system already plagued by physical instabilities.

Another known unmodeled effect, first mentioned in section 5.5.2.1 is the assumption that each actuator's air flow is independent of the others. On the testbed, a regulator initializes the supply pressure to 50 psig, but the air then travels through several feet of tubing and is distributed among the valves and cylinders. Since air is compressible and subject to dynamic effects, some pressure behaviors observed on the actual robot are unobtainable in this simulation. However, this approximation did not appear to be a significant contributing factor and is likely not worth the computational effort to include in the model.

Finally, valve and friction approximations were shown to have a large impact on total task execution time, a metric that will become more critical as other elements are introduced into the simulation and it is expanded from having three to twelve actuator models (one for each leg). In its current form, the simulation runs at 1000 Hz, but actuator models have been shown to be able to run as slowly as 640 Hz while maintaining accuracy, corresponding to a step size of 0.0015625 s. The task execution times discussed in section 5.5.3 were for a simulation consisting solely of a Gamma joint; those models with all three joints typically have an average TET closer to 13 x 10^{-5} s. As more and more components are added to the Simulink model, the choice between accuracy and computational efficiency may become more dire, and based on results above, friction and valve modeling are clear components whose impact could be reduced. Additionally, while both components provide a desirable degree of accuracy across a range of inputs, their derivation was considerably detailed and

lengthy.

5.6 Improvements

Based on the performance summarized in the previous section, the simulation is subject to several areas of improvement. While there are a variety of items that could be changed, there are several within the actuator model and SrLib, as well as within the configuration itself, that would likely have the most significant effect on simulation quality.

5.6.1 Robot Simulation Parameters and Target Configuration

First, links and their respective inertias should be updated with more accurate models. While capsules are appropriate for the Beta and Gamma joint, the Alpha joint actually rotates about a point that is not the end of the component, and is considerably more asymmetric than a capsule implies. Inertias can be changed in two ways: in SrLib, or using external software to find a moment of inertia that can then be assigned to a specific link via an SrLib command. Internally, the links would need to be modeled as combinations of primitive shapes – capsules, boxes, spheres, etc. – with appropriate densities and connected by weld joints. Alternatively, the link shape could simply be approximated, much like the tibia in the current version, and then assigned inertias based on measurements from Solidworks or a similar modeling program. The inertial accuracy of the links could be further improved by replacing the torque approximation with a physical model of the actuator working as a prismatic joint along a single axis.

Second, unmodeled frictions and resistances could be lumped into a general damping term applied to the joint. This would need to occur in SrLib following receipt of torque commands.

Finally, the simulation needs to be expanded to include all twelve joints required

to accurately simulated a quadruped robot. Functionally, this requirement is straightforward; the current actuator models and their respective controllers and input files need to be replicated within a single Simulink file. Additionally, inputs should be replaced with angles derived from inputs of the Phantom joysticks using the inverse kinematics blocks used on the two-legged prototype, and the simulation should be outfitted with a higher order control scheme to allow the operator to guide each of the four legs.

The addition of these programs, however, will result in a complete model that is considerably more computationally demanding. An easy way to deal with this challenge is to use a target high in memory and processing power – the one used here had 2.1 GHz and 256mb RAM, and sufficed for at least three joints. Additionally, the friction and valve models could be reduced in complexity and accuracy, as discussed earlier, in exchange for a reduced overall TET.

5.6.2 Dynamics Library Design and Configuration

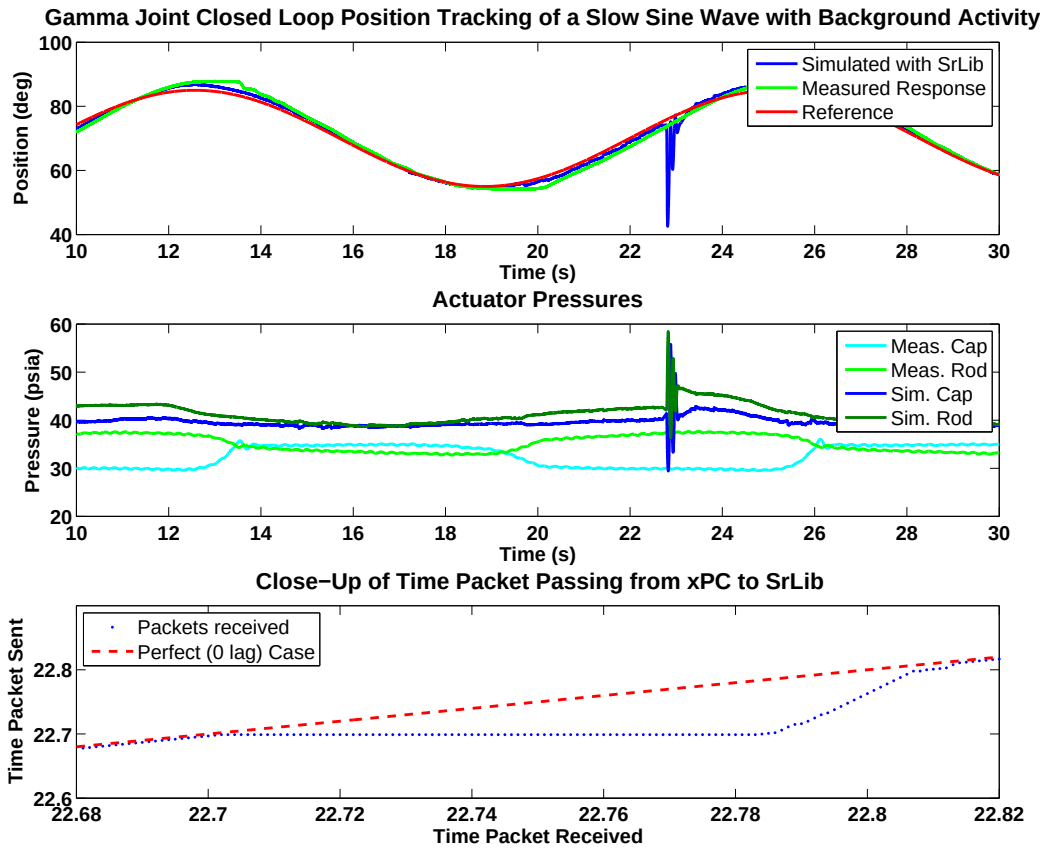

Figure 51: Errors in simulated response cause by pauses in SrLib operation

While SrLib's non-realtime nature has been largely irrelevant in the tests performed here, its good performance is not guaranteed. Figure 51 shows a case of a closed loop sine wave test for a gamma joint where the computer recorded a burst of activity mid-test, causing a packet delay and providing erroneous results. Such occurrences are rare – in the 50 most recent tests it happened once, but they cannot be easily predicted. One way to minimize the likelihood of such activity would be to change the Simulation configuration to use a fast computer with plenty of memory to run SrLib, and ensure that SrLib is the only major task running on that computer. Better still, the system could be ported to Linux or another real-time system and run, either

with SrLib or with a different dynamics engine, in real-time.

Another non-realtime effect is the occasional pauses caused by rendering a frame. With the computer configuration rendering frequency used for this thesis, this effect was not a significant concern, though the additional pauses of up to 1 ms were still present. Using separate threads for rendering and dynamics, the two processes could run in sync with each other, rather than alternating. In practice, this change would be somewhat complicated, since the dynamics engine actively uses many openGL features, and would likely require and additionally, entirely separate rendering process. However, it could improve the utility of the simulation by providing a more versatile system capable of running on a wider array of hardware configurations.

Finally, though lag was not a significant issue in recent tests, the 3-4 ms round trip time could become problematic in some circumstances, and may have contributed to friction inaccuracies due to velocity sampling. Such delays could be avoided by simply moving the entire simulation to one target PC, using MATLAB's mex features to interface SrLib's C++ code with the actuator model. Instead of sending a torque and feeding back position and velocity, position data would be sent to a viewer in a one-way connection. This type of setup would be ideal, as it would remove any dependence of the simulation on network communication while still maintaining a modular dynamics simulation with a viewing option.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This thesis has discussed the development and testing of a simulation for a pneumatically powered quadruped robot. The simulation combines an analytically derived actuator model with a dynamics library to provide a system that realistically portrays the effects of pneumatic actuation on user control of the robot. Results have demonstrated the simulation's performance and the impact of its design and configuration requirements on its overall convenience and applicability.

A two-legged prototype of the robot and a simple actuator test platform were used to validate the simulated actuation results at several stages of its development, using response characteristics such as overshoot, settling time, rise time, and correspondence to higher-order dynamics as indicators of similarity. Additionally, walking and manipulation-like tests run on a single leg of the robot were closely matched in simulation, again using similarities in position tracking and relative pressure behavior to validate the simulation's realism.

Simulation accuracy was achieved by combing classical fluid circuit modeling with advanced valve and friction models.

6.1 Impact of Advanced Valve Model

The advanced valve design was based on an overlapped valve model that utilized an equivalent orifice area curve and equivalent supply and exhaust pressure curve as functions of input voltage. While this valve model contributed to very accurate behavior, it was found to come at a high computational price. Tests of contributions of individual model components to total task execution time (TET) in section 5.5.3 demonstrated that the advanced valve model consumed as much as 24% of the total

TET. Tests using a simple valve model assuming linear orifice area scaling and a constant discharge coefficient proved to exhibit only a slight loss of accuracy for fast-moving dynamics, but fared poorly when attempting to model slow-moving behaviors, which are representative of valve performance near the offset voltage. While this loss in accuracy appeared to be of minimal importance in the simple actuator model, expansion to the complete dynamic simulation aggravated the result. Thus, despite its high computational needs, the advanced valve model is a necessary component for simulation accuracy.

6.2 Impact of Advanced Friction Model

Friction was shown to present a similar challenge. A Stribeck-Tanh model was used because it employs several empirically derivable parameters to provide a continuous function that includes consideration of static, Coulomb, and viscous friction, thereby ensuring a friction model that is both relatively accurate and easy to implement. However, like the advanced valve model, it was shown in section 5.5.3 that the friction model resulted in a high computational cost, contributing up to 11% of the overall TET. Tests using a simple viscous friction model showed that the impacts on accuracy of the simple actuator model were minimal, but the complete dynamic simulation performed poorly. This result, however, was likely due to a misplacement of the damping term; it would be preferable to comput the viscous friction in SrLib rather than in Simulink. While SrLib technically supports a damping term, tests using the parameter showed no noticeable damping effect on the associated actuators.

Additionally, the close correspondence to reality demonstrated on the simple actuator model using only a viscous friction term suggests that even if viscous friction alone is too basic for the complete dynamic simulation, a friction model likely exists that requires less overall computation than the Stribeck-Tanh model but could still provide the desired degree of accuracy in the full dynamic simulation.

6.3 Network and Configuration Requirements

Strict enforcement of the realtime environment was found to be one of the key factors in success of the simulation. The xPC target operating system was coupled with realtime UDP over a separate dedicated network card to enforce realtime communication between the target, host, and SrLib computer. Additionally, SrLib's serial structure, discussed in section 5.1, ensured that as long as it was run on a computer able to compute a cycle within the computing speed of the corresponding target PC (1000 Hz), SrLib would compute a cycle within an equivalent time period, effectively enforcing a realtime environment. Despite this structure, SrLib's non-realtime nature can still harm simulation performance. As pointed out in section 5.6.2, other activities on the SrLib computer could cause delays as large as 80 ms, creating significant oscillations that could potentially render the entire simulation unstable. Overall, however, the simulation was found to be successful on a network with approximately 3-4 ms of total travel time.

6.4 SrLib Approximations

The robot was represented in SrLib as a set of cylindrical capsules of assigned density actuated by rotary joints, using the instantaneous robot geometry to convert actuator forces to simulated joint torques. Results showed that the behavior of the dynamics-equipped simulation very closely matched that of the prototype for the Gamma joint, but began to deviate as more links were introduced. The Beta joint exhibited reasonable behavior as a step test and excellent behavior within the single leg swing, and the Alpha joint tended towards limit cycles in either case.

Since the goal of the overall simulation is to achieve behavior that is realistic in the sense that it is reasonable for a pneumatic system, the oscillations were eliminated by reducing the gain margin on the simulated Alpha joint controller by 25%, thus stabilizing the closed loop dynamics and providing a complete simulated leg swing

motion that closely matched both the tracking and relative pressure measurements of each of the corresponding joints on the robot prototype.

Several errors and approximations could have accumulated to contribute to the observed deviation in performance of the Alpha and Beta joints, but the primary suspect is the estimated inertia. The inertia of the tibia, the sole link actuated by the Gamma joint, was calculated as an aluminum cylinder – a near-perfect representation of the actual link, contributing to the strong similarities between simulated and actual Gamma joint behavior. The inertias of the links actuated by the Alpha and Beta joints, however, are actually based on much more asymmetric geometry than the cylindrical capsules used to estimate their inertia would imply. This assumption is acceptable for the Beta joint, but it warrants improvement for the Alpha joint, requiring either a better model in SrLib, or an assigned inertia based on an externally calculated model.

6.5 Overall Actuator Model Accuracy

Disregarding the computational costs incurred by the advanced valve and friction models, the completed actuator model proved to be very successful, useful for simulation, robot design, and potential future controller design. Tests on the simple actuator model and Gamma joint showed results that very closely matched position tracking and relative pressure behaviors, even for valve input voltages near the voltage offset, a range that has traditionally been challenging for researchers to match.

Similarly, though friction could not be accurately modeled in regions where the actuator moves very slowly but never actually stops (e.g., sine waves), its effects on step responses (in the form of the stiction peak) are well modeled.

Furthermore, joint accuracy has been maintained when subjected to disturbances and coupled dynamics, as shown by the continued accuracy of the Gamma and Beta joints as they are run simultaneously.

6.6 Future Improvements

The dynamic simulation developed in this thesis succeeds in closely portraying the impact of pneumatic power on a robot that walks and manipulates. However, to fully simulate a walking, pneumatically actuated robot in a fashion that can be conveniently implemented, several improvements still need to be made.

6.6.1 Expansion to 12 Joints

First, the associated Simulink file needs to be expanded from three joints to twelve. Since this will result in a higher overall TET, it may be necessary to use simplified valve and friction models, though faster hardware or distribution of the Simulink model among multiple computers on a local network might also suffice. To control the robot, higher level controllers such as those mentioned in [29] or [13] must be added to the file.

6.6.2 Robot Modeling

As noted in section 6.4, the robot's links, and particularly their inertias, are not characterized as accurately as they could be. An improved SrLib robot model that more accurately captures the masses and dimensions of the robot arm with respect to each joint should substantially improve the overall simulation's performance. This could be achieved either by designing a more detailed robot within SrLib, or by calculating the inertias of each link in Solidworks or a similar programming and then assigning the resulting inertia values to the corresponding links in SrLib. The overall model complexity could be further reduced by replacing the current rotary joints with prismatic joints representative of the pneumatic cylinders. This change would eliminate the need for torque conversions (angle conversions would still be necessary for the inverse kinematics) and thus also reduce the number of approximations used in the model. Additionally, modeling the actuators directly could help improve inertia models by including the effects of varying piston position on the inertia of each individual

link, currently modeled as a non-deformable rigid body.

6.6.3 Solver Review

In this thesis, the Simulink portions of the simulation used `ode5`, a 5th order Dormand-Prince (Runge-Kutta based) solver that is one of the available options found in Simulink, while SrLib used an algorithm that combines Euler and 4th order Runge-Kutta approaches. The choice of solver was not heavily debated in simulation implementation, but proved to be a possibly critical factor in the success of certain elements, such as viscous friction or alpha joint dynamics, where it may contribute strongly to the accuracy of the solution. An analysis of the available solvers – both those that are available and those that were used – and their applicability to the system dynamics of the model, especially these noted error-prone regions, would significantly improve the simulation's overall value and potentially solve some of the errors that are not yet fully explainable.

6.6.4 Improved Simulation Framework

It has been shown in this thesis that the configuration of SrLib, linked via a local network with minimal delay to SrLib, is a sufficient configuration to achieve the desired realistic performance of a robot with simulated pneumatic actuators. However, the reliability of the simulation could be improved through several configuration changes. First, SrLib could be replaced by an equivalent realtime simulation, or placed on a machine running only SrLib, to minimize opportunity for potential disruptions. Second, the friction models could be implemented in SrLib rather than in Simulink to ensure that force summations take into account as much of the overall system dynamics as possible.

Even without these changes, this simulation nonetheless provides a practically implementable framework for a model that closely characterizes how the choice of pneumatic actuation affects performance of a walking robot, thereby providing a

valuable design tool to the continued development of the compact rescue robot.

APPENDIX A

VALIDATION PLATFORM SOFTWARE AND ANALYSIS FILES

This section provides an overview of the files needed to run analyses on the simple test rig (Section 4.2.1) and two-legged robot prototype (Section 3.1.2). Each test platform requires a Simulink file, a script used to collect and process signals for analysis, and a set of filters used in data processing. Since the analysis scripts are practically identical for the two test platforms, only one script is shown. The filters are the same for both test platforms and are defined in A.3.

A.1 Simulink Model Files

For each of the following Simulink files, the first image shows the overall structure of the model, while the second image shows the composition of the Hardware Interface block, depicted in yellow in the overall structure.

A.1.1 Single Degree of Freedom Test Platform

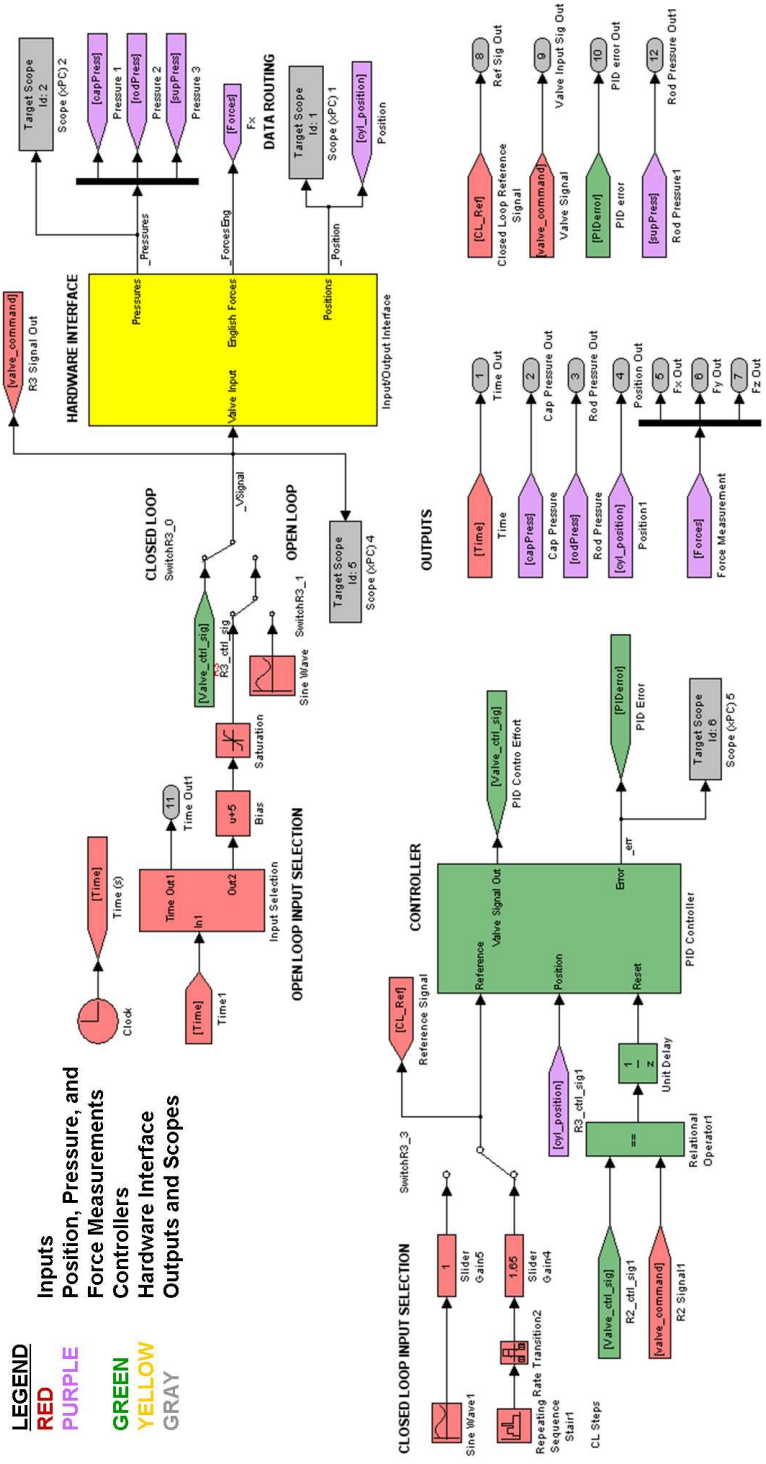

Figure 52: Simulink file for use with the single degree-of-freedom test rig

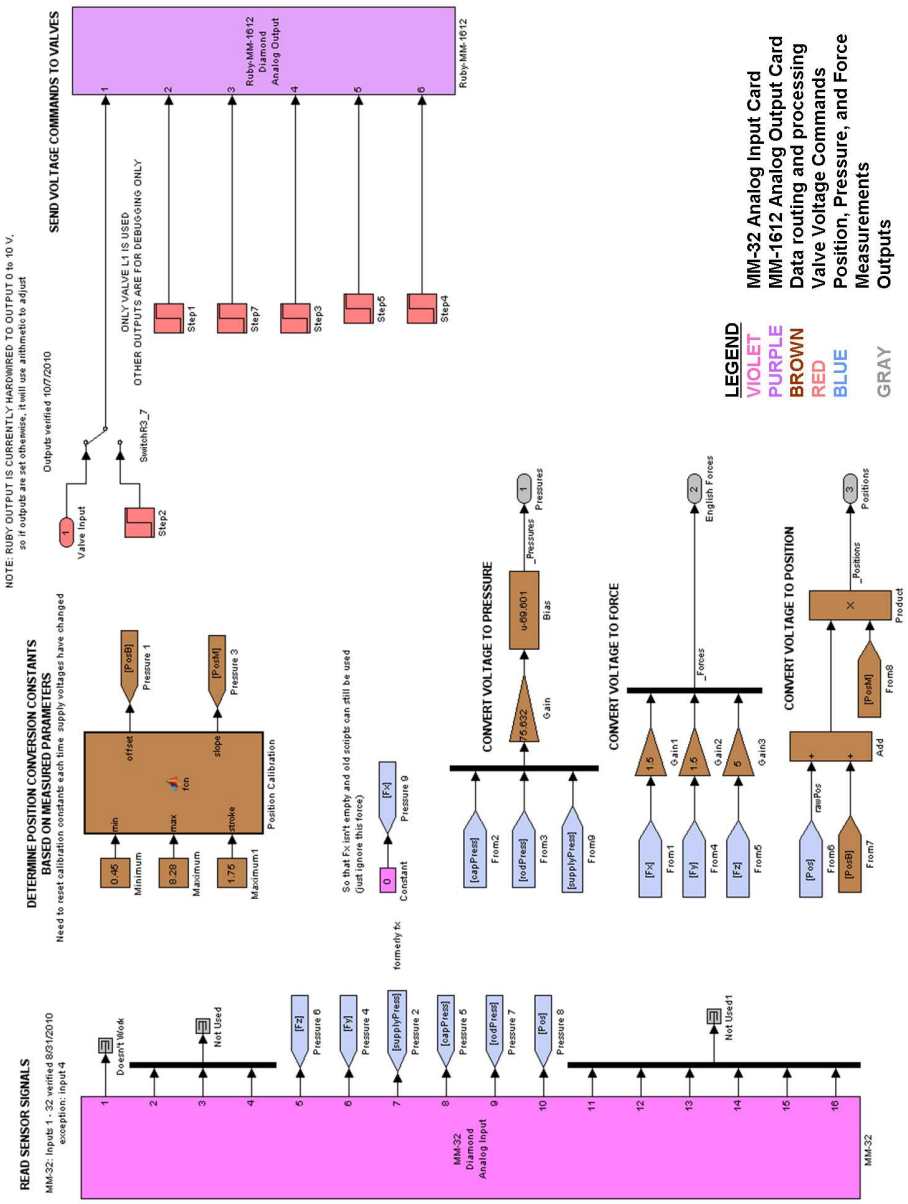

Figure 53: Subsystem of the Simulink file for use with the single degree-of-freedom test rig corresponding to the yellow "Hardware and Controllers" block in Figure 52

A.1.2 Two-Legged Prototype

This Simulink file has two main sections: (1) The forward and inverse kinematics used to convert Phantom commands to actuator angles (discussed in detail in [29] and to some extent [23]) and (2) the controllers and hardware interfacing tools. Figure 54

shows the framework of the entire file, while Figure 55 displays how the controllers and commanded inputs were interfaced with the hardware through Simulink.

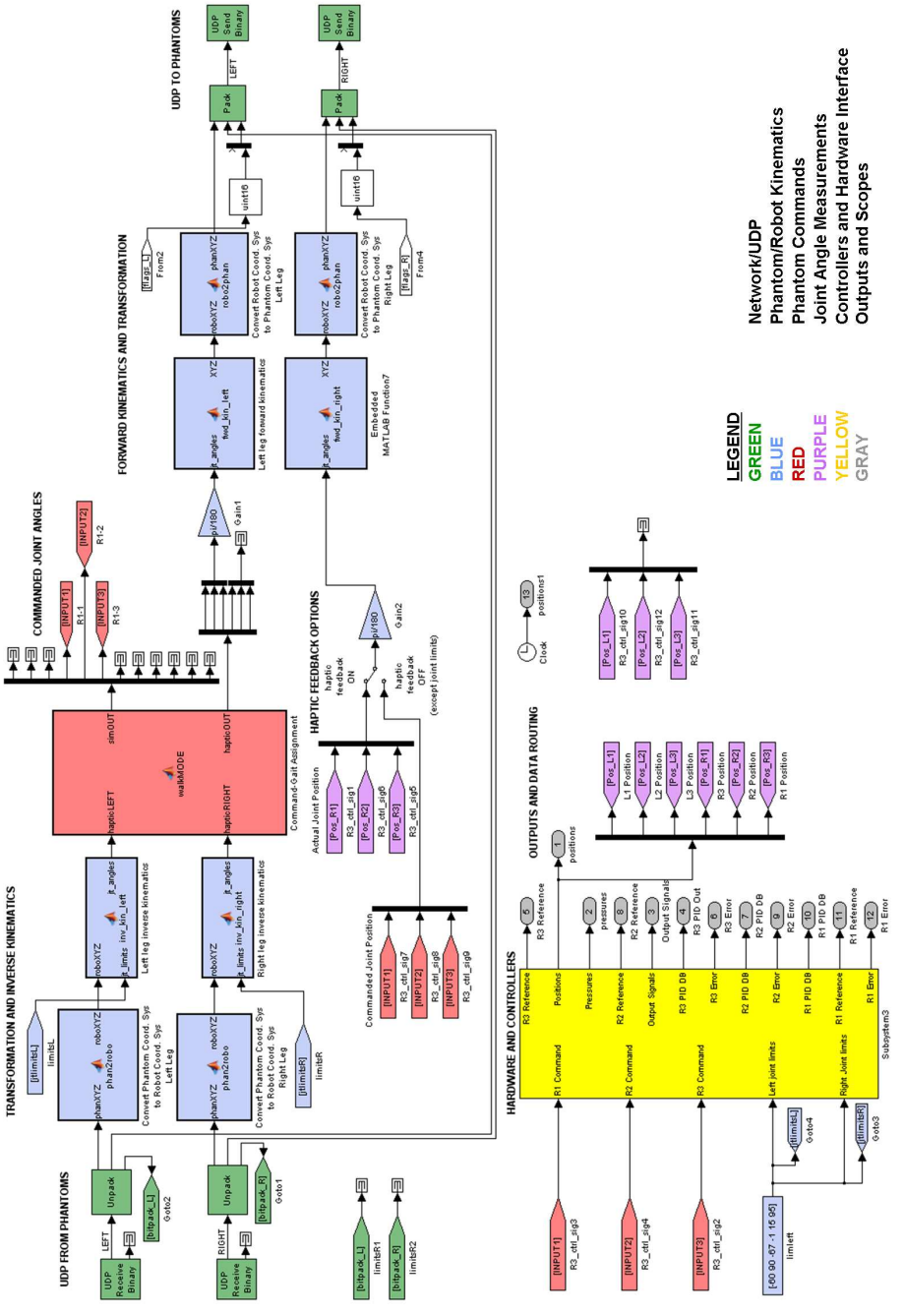

Figure 54: Simulink file for use with the two-legged CRR prototype

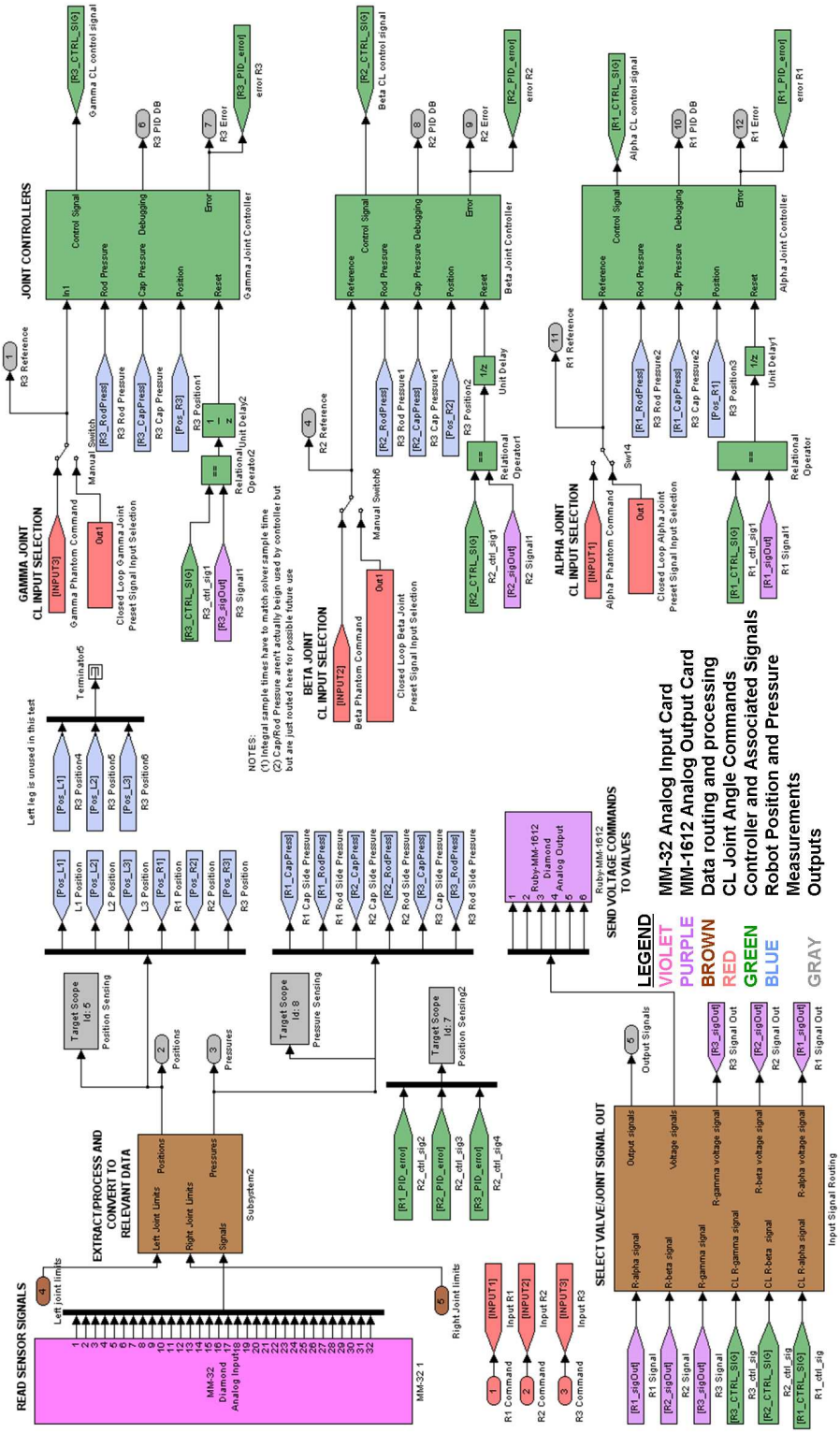

Figure 55: Subsystem of the Simulink file for use with the two-legged CRR prototype corresponding to the yellow Hardware Interfacing block in Figure 54

A.2 Analysis Files for Data Collected using the Validation Platforms

Variations of the following script were run to collect and process data from tests on the validation platforms:

```
%% M-file to collect data and analyze results
%Last updated October 2011 for use with Simulink 2011

%% Part 1: Collect data
% data = tg.OutputLog;
% close all;

positions = data(:,1:6);        %Positions
pressures = data(:,7:12)+14.7;  %Pressures
Rsignals = data(:,13:15);       %Reference Signals

tvecB = data(:,40); %Time
psi2Pa = 6894.76;   %Conversion Factor

%% Define Filters
load BimbaTools_v2;     %load saved filter data

[fp_num, fp_den] = tf(f_press);     %Pressure Filter
[fpos_num, fpos_den] = tf(f_pos);   %Position Filter (often unused)
[fv_num, fv_den] = tf(f_v);         %Velocity Filter
[ff_num, ff_den] = tf(f_F);         %Force Filter (used for simple ...
   validation platform)

%% Define working variables and filter appropriately
posB_raw = positions;   %Positions
posB_filt = filtfilt(fpos_num, fpos_den, posB_raw);
```

```
26
27 velB_raw = gradient(posB_filt,.001);   %Velocities
28 velB_filt = filtfilt(fv_num, fv_den,velB_raw);
29
30 pressB_raw = pressures;   %Pressures
31 pressB_filt = filtfilt(fp_num, fp_den,pressB_raw);
```

A.3 Filter Design

The filters referenced in section A.2 were defined using the `filterbuilder` tool in MATLAB. Values were chosen by running a signal analysis script on samples of the corresponding signal type (pressure, position, etc.) and observing the results. The signal analysis script is shown below and followed by screenshots of the filter definitions (Figure 56).

```
1  %% M-file to collect data and analyze results
2  %Last updated October 2011 for use with Simulink 2011
3
4  %% Part 1: Collect data
5  % data = tg.OutputLog;
6  % close all;
7
8  positions = data(:,1:6);        %Positions
9  pressures = data(:,7:12)+14.7;  %Pressures
10 Rsignals = data(:,13:15);       %Reference Signals
11
12 tvecB = data(:,40); %Time
13 psi2Pa = 6894.76;   %Conversion Factor
14
15 %% Define Filters
16 load BimbaTools_v2;     %load saved filter data
```

```
17
18 [fp_num, fp_den] = tf(f_press);     %Pressure Filter
19 [fpos_num, fpos_den] = tf(f_pos);   %Position Filter (often unused)
20 [fv_num, fv_den] = tf(f_v);         %Velocity Filter
21 [ff_num, ff_den] = tf(f_F);         %Force Filter (used for simple ...
       validation platform)
22
23 %% Define working variables and filter appropriately
24 posB_raw = positions;   %Positions
25 posB_filt = filtfilt(fpos_num, fpos_den, posB_raw);
26
27 velB_raw = gradient(posB_filt,.001);   %Velocities
28 velB_filt = filtfilt(fv_num, fv_den,velB_raw);
29
30 pressB_raw = pressures;   %Pressures
31 pressB_filt = filtfilt(fp_num, fp_den,pressB_raw);
```

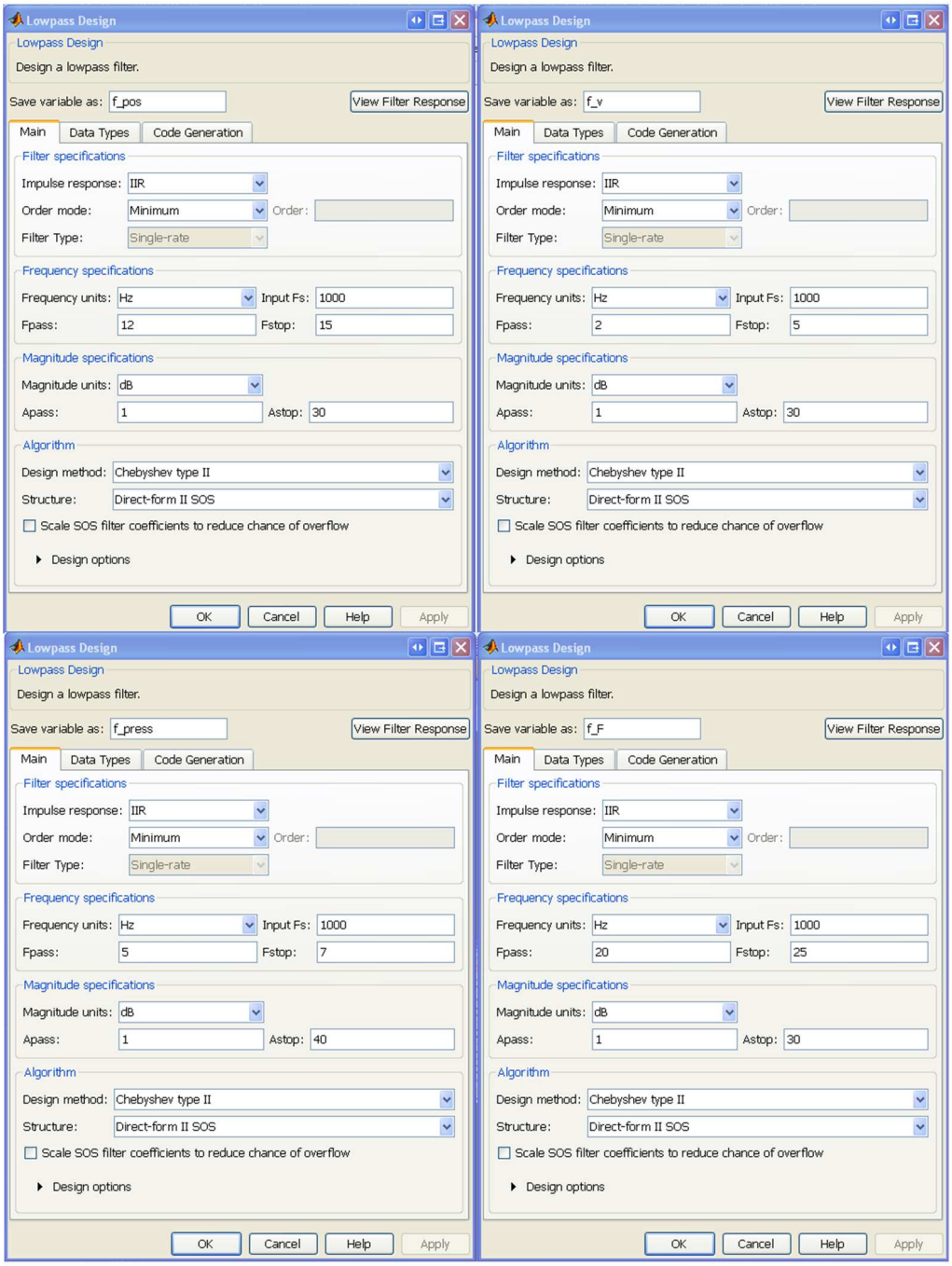

Figure 56: Filters used for data analysis. Clockwise from top left: Position, Velocity, Force, and Pressure

APPENDIX B

SIMPLE ACTUATOR MODEL COMPONENTS

B.1 Simple Actuator Model Simulink File

The simple actuator model was used in Chapter 4 to model an individual valve and cylinder moving a piston and attached mass. Two files are needed to run the model. First, a setup file is run that initializes the constants used throughout the simulation. Second, a Simulink file is run that is composed of the actuator model itself, a PID controller, a system model of the single degree-of-freedom test rig, and inputs and outputs as necessary, shown in Figure 57. Details on the framework for the actuator model are shown in figures 58, corresponding to the equations discussed in sections 4.1 and 4.2.3. An example code from one of the valve ports is used to show how the details from section 4.2.3 were integrated into the overlying code structure, and Figure 59 illustrates the composition of the friction model, which allows users to choose between a Stribeck-Tanh and a viscous model, using equations introduced in section 4.2.2. In each of the Simulink files, blocks are color-coded by contribution to the simulation, as described by the corresponding legends found in the figures below.

B.1.1 Setup File

The following script provides an example setup file used to validate the simple actuator model in its vertical, downward facing orientation, with an attached mass of 0.2 kg and air supplied at 50 psig.

```
1 %% Bimba Setup Version 2.0
2 %Created April 24, 2011
```

```
%Last updated October 2011

%% Prep workspace if desired
% clear all;
% close all;

%% General Constants

k=1.4;          %for air

cp=1.012*10^3;  %J/(kg*K) room temp air 1% humidity

g=9.81;         %m/s^2

%R=8.314;       %universal gas constant J/(K*mol)
% when divided my air molar mass (29g/mol)
R=287;          %J/(kg*K)

Ts=298;         %Kelvin (25 deg C)

%To be converted:
Ps=50.3;        %supply air, psig
Pe=15.7;        %exhaust air (room), psi

Ps=Ps+14.7;     %psi

%% Conversion factors
in2m = .0254;
psi2Pa = 6894.76;

%% Valve Dynamics
choke_ratio=.528;   %pressure ratio, if passed will generate choking
```

```
C1=sqrt(2*k/(R*(k-1))); %Unchoked Flow
C2=sqrt(k/(R*((k+1)/2)^((k+1)/(k-1)))); %Choked Flow

%Discharge coefficient models for linear cd tests
% cdC = [0 0.09];  %[slope offset]
% cdR = [0 0.12];  %[slope offset]
% cd = 0.2;        %slope

%% Cylinder Dynamics
rod=0.375*in2m;     %rod diameter, m
bore=1.0625*in2m;   %bore diameter, m
stroke=1.75*in2m;   %stroke length, m
m = 0.2;            %mass of attachment, kg
rds = 0.018;        %rod side dead space, m (measured: 0.018)
cds = 0.012;        %cap side dead space, m

Aslider = 0.0045*.0025;     %Slider area, m^2
Ap = pi*bore^2/4 - Aslider; %Piston side area, m^2
Arod = pi*rod^2/4;          %Area of Rod, m^2
Ar = Ap-Arod;               %Rod side area, m^2

%% Friction Models
b=70;                       %Damping constant for viscous model
alpha=1;

%Stribeck-Tanh friction model (all in lbf)
Fsf = 4.5;  %Static friction
FcfU = 3;   %Coulomb friction (upward movements)
FcfD = 3;   %Coulomb friction (downward movements)
Cv = 0.5;   %Coefficient of viscous friction
Ktanh = 40; %Tanh coefficient
ii = 5;     %exponent
```

```
67 Vs = 0.1;   %Sliding speed coefficient
68
69 zVel = Vs;  %Actually used in friction model as Vs, so not strictly ...
       zero velocity threshold
70 % zPos = 0.04;
71
72 %% Flags within test rig dynamic model
73 velset=1;   %To reset velocity integrator
74 accset=5;   %To reset acceleration integrator
75
76 %% Simulation Constants
77 dt=.001; %Time step, secs
78 Ps=Ps*psi2Pa;   %Supply pressure, Pa
79 Pe=Pe*psi2Pa;   %Exhaust pressure, Pa
80
81 %% Initial Conditions
82 startPos = stroke;
83 rodP0 = Ps;
84 capP0 = 14*psi2Pa;
```

B.1.2 Simulink File

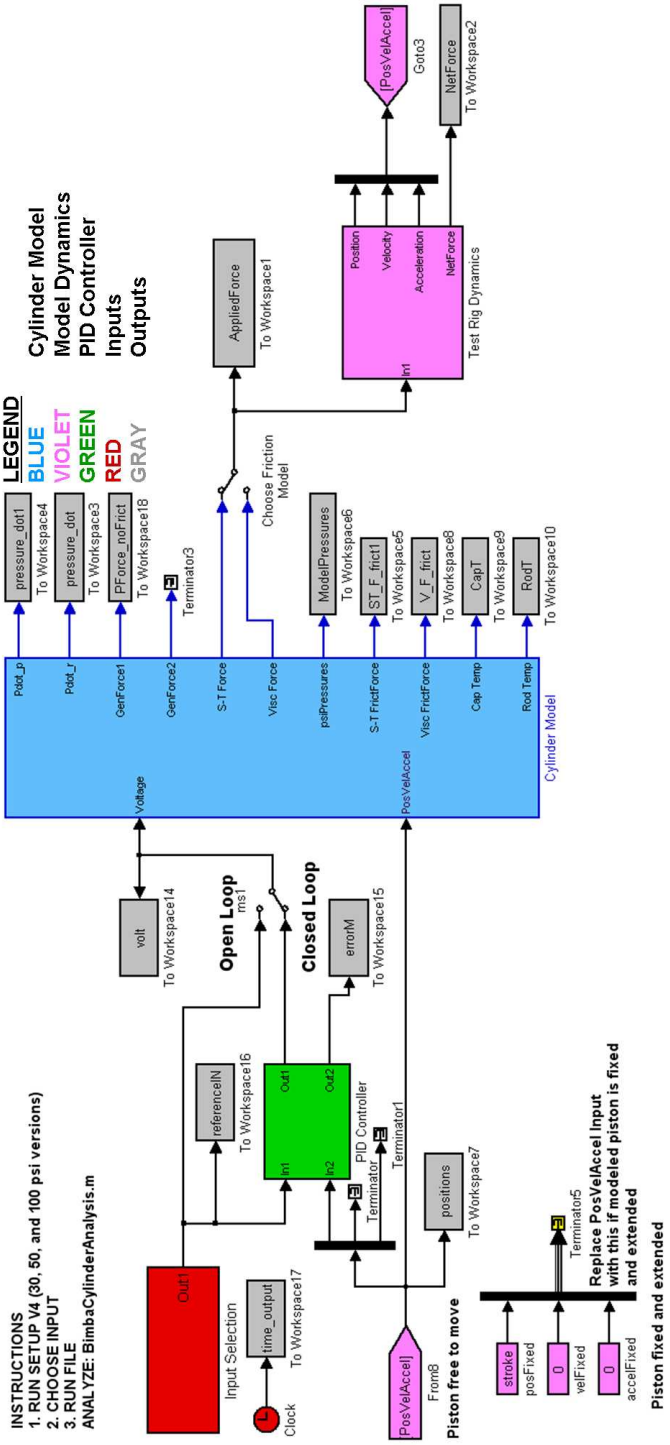

Figure 57: Simulink actuator model, controller, and system dynamics

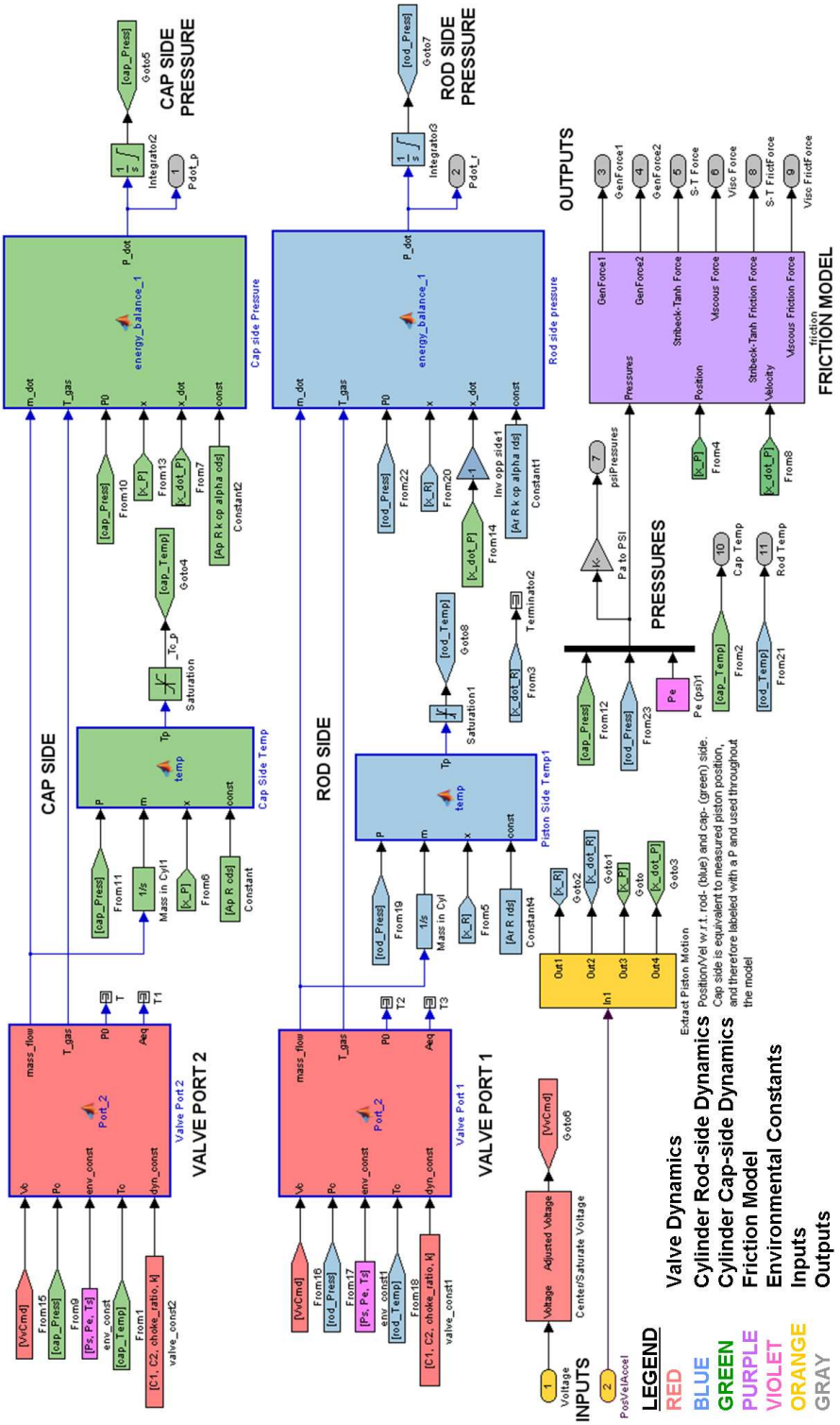

Figure 58: Valve and cylinder modeling inside the "Cylinder Model" block of the Simulink model shown in Figure 57

A sample script from the valve port 2 block shows how the equivalent pressure and area curves from section 4.2.3 were interfaced with the mass flow equations used in valve modeling:

```
function [mass_flow, T_gas, P0, Aeq]= Port_2(Vc, Pc, env_const, Tc, ...
    dyn_const)

%Vc = adjusted valve input voltage
%A = Valve orifice Area, m^2
%Pc = Cylinder piston side air pressure
%Tc = Cylinder piston side Air temperature

% Dynamics constants C1 C2 choke_ratio k Cd
C1=dyn_const(1);  %Unchoked flow
C2=dyn_const(2);  %Choked flow
choke_ratio=dyn_const(3); %Choke ratio
k=dyn_const(4);   %k constant for air

% global Ps Pe Ts
Ps=env_const(1);    %Supply pressure
Pe=env_const(2);    %Exhaust Pressure
Ts=env_const(3);        %Supply Temperature

%Define remaining pressure curve parameters
%tanh model
Hpc = 1.25;    %High Pressure x (adjusted) cutoff
Lpc = -.75;     %Low Pressure x (adjusted) cutoff

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Use equivalent Pressure based on curve derivation (in english units)
psi2Pa = 6894.76;
```

```
27
28 Ps_eng = Ps/psi2Pa;
29 Pe_eng = Pe/psi2Pa;
30 x_offset = (Hpc + Lpc)/2;
31 y_offset = (Ps_eng + Pe_eng)/2;
32 scaling = sign(Hpc - Lpc)*(Ps_eng - Pe_eng)/2;
33 m = 2*pi/abs(Lpc - Hpc);
34
35 P0_eng = y_offset + scaling*tanh(m*(Vc-x_offset));
36 P0 = P0_eng*psi2Pa;
37
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 %% Use equivalent area that combines Discharge Coefficient and Area
40
41 %Port 2 equivalent area curves from Area_Relations.xls
42 if(Vc>0)
43 %     polyf = [1.33e-8 -1.321e-7 3.687e-7 -5.24e-8 1.80e-8];  ...
      %weighted fit
44     polyf = [1.32e-8 -1.302e-7 3.608e-7 -4.33e-8 1.42e-8];  %reg. ...
          polyfit
45     Aeq = polyf(1)*Vc^4 + polyf(2)*Vc^3 +polyf(3)*Vc^2 + polyf(4)*Vc ...
          + polyf(5);
46 else    %positive area signifies supply air connected
47 %     polyf = [9.3e-9 9.17e-8 2.261e-7 -2.281e-7 1.80e-8];  ...
      %weighted fit
48     polyf = [9.4e-9 9.35e-8 2.326e-7 -2.225e-7 1.87e-8];  %reg. polyfit
49     Aeq = polyf(1)*Vc^4 + polyf(2)*Vc^3 +polyf(3)*Vc^2 + polyf(4)*Vc ...
          + polyf(5);
50 end
51
52 if(P0>Pc)   %flow into cylinder
53     Pu=P0;
```

```
54      Pd=Pc;
55      T=Ts;
56      direction=1;
57  else        %flow out of cylinder
58      Pu=Pc;
59      Pd=P0;
60      T=Tc;
61      direction=-1;
62  end
63
64  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
65  %% Calculate f based on choked or unchoked flow
66
67  if(Pd/Pu>choke_ratio)   %NOT choked
68      f=C1*Pu/sqrt(T)*(Pd/Pu)^(1/k)*sqrt(1-(Pd/Pu)^((k-1)/k));
69  else                    %CHOKED
70      f=C2*Pu/sqrt(T);
71  end
72
73  %% Calculate mass flow
74  mass_flow=Aeq*f;
75
76  %Assign direction
77  mass_flow = mass_flow*direction;
78  T_gas=T;
```

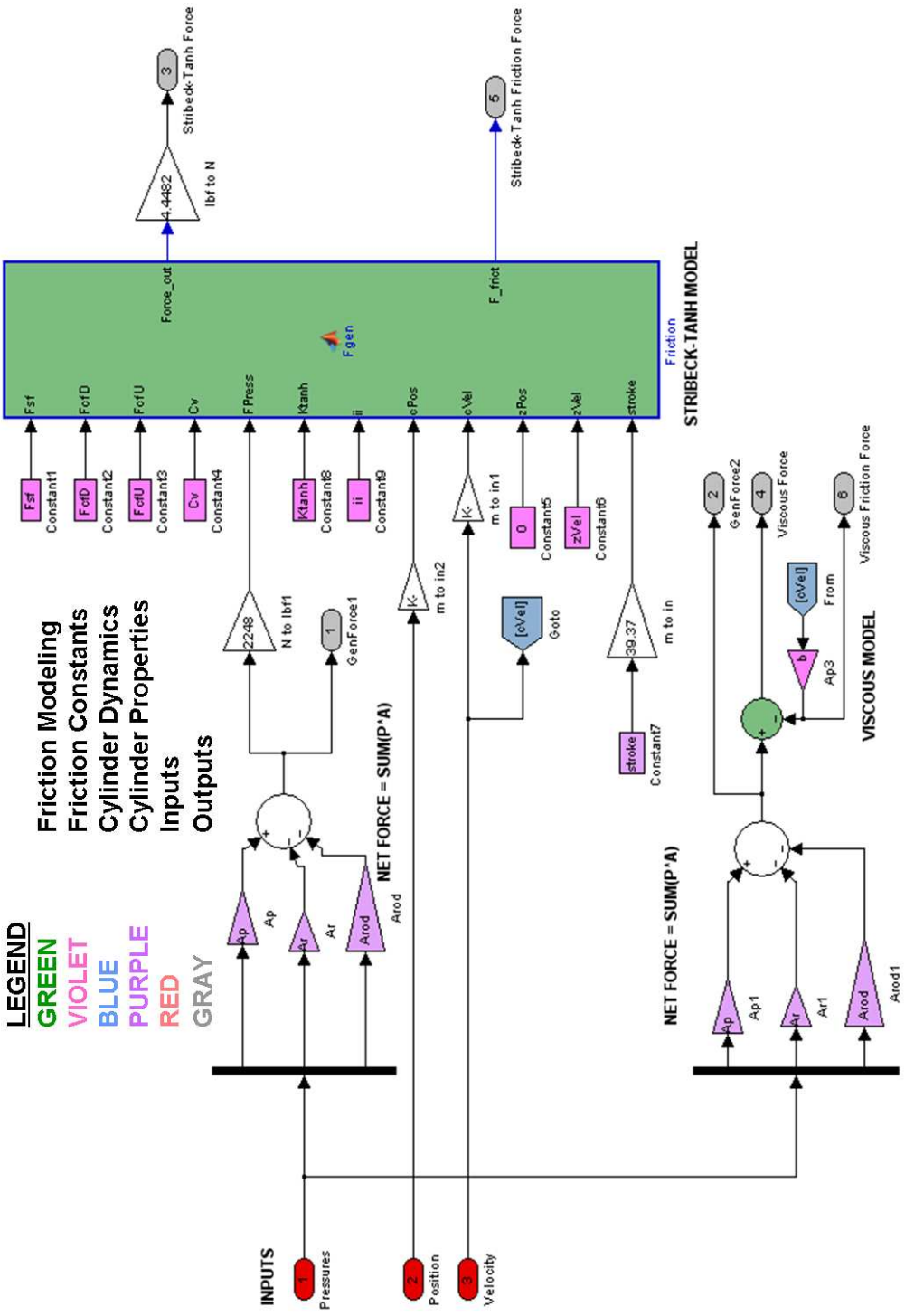

Figure 59: Friction model within the Simulink actuator model

B.2 Equivalent Area Curve Fitting Tools

The following script was used to generate the equivalent area and pressure curves discussed in section 4.2.3.

```
%% Find polynomial curve fits for areas
% close all;

%adjusted voltage input
x = zeros(19,1);
x(1:9,1) = flipud(-[0.15 0.35 0.55 0.75 1.25 1.75 2.75 3.75 4.75]');
x(10:19,1) = [0 0.15 0.35 0.55 0.75 1.25 1.75 2.75 3.75 4.75]';

%50 psig supply, standard config
cd2 = [0.115 .14 .18 .2 .2 .19 .18 .17 .19 .09 .06 .05 .06 .09 .125 ...
    .135 .12 .09 .075]';
cd1 = [.08 .1 .13 .15 .15 .15 .15 .12 .13 .11 .07 .06 .1 .12 .165 ...
    .175 .16 .14 .11]';
AL = zeros(19,1); AL(10,1) = 2e-7;
estA = abs(0.002*0.005/5*x);

Aeq1 = cd1.*(estA + AL);
Aeq2 = cd2.*(estA + AL);

xnegfit = x(1:10,1);
xposfit = x(10:19,1);
Aeq1negfit = Aeq1(1:10,1);
Aeq1posfit = Aeq1(10:19,1);
Aeq2negfit = Aeq2(1:10,1);
Aeq2posfit = Aeq2(10:19,1);

%% Use polyfit to get fourth order polynomials
```

```
P1neg = polyfit(xnegfit,Aeq1negfit,4);
P1pos = polyfit(xposfit,Aeq1posfit,4);

P2neg = polyfit(xnegfit,Aeq2negfit,4);
P2pos = polyfit(xposfit,Aeq2posfit,4);

% Plot results
AL = Aeq1(10,1);

%trendline curves
xIneg = -5:0.01:0;
xIpos = 0:0.01:5;
Aeq1neg = P1neg(1)*xIneg.^4 + P1neg(2)*xIneg.^3 + P1neg(3)*xIneg.^2 ...
    + P1neg(4)*xIneg + P1neg(5);
Aeq1pos = P1pos(1)*xIpos.^4 + P1pos(2)*xIpos.^3 + P1pos(3)*xIpos.^2 ...
    + P1pos(4)*xIpos + P1pos(5);

Aeq2neg = P2neg(1)*xIneg.^4 + P2neg(2)*xIneg.^3 + P2neg(3)*xIneg.^2 ...
    + P2neg(4)*xIneg + P2neg(5);
Aeq2pos = P2pos(1)*xIpos.^4 + P2pos(2)*xIpos.^3 + P2pos(3)*xIpos.^2 ...
    + P2pos(4)*xIpos + P2pos(5);


%% Use weighted fourth order polynomials with other options
% %Set options
% %Exclude outlier for s1
% s1 = fitoptions('Weights',[2 2 2 2 2 1 1 1 1 1]',...
%     'Method','LinearLeastSquares',...
%     'Robust','LAR');
% s2 = fitoptions('Exclude',[0 0 0 0 0 0 0 1 0 0]',...
%     'Weights',[2 2 2 2 2 1 1 1 1 1]',...
%     'Method','LinearLeastSquares',...
```

```
%     'Robust','LAR');
% s3 = s1;
% s4 = s1;
%
% cf1 = fit(xnegfit,Aeq1negfit,'poly4',s1);
% cf2 = fit(xposfit,Aeq1posfit,'poly4',s2);
% cf3 = fit(xnegfit,Aeq2negfit,'poly4',s3);
% cf4 = fit(xposfit,Aeq2posfit,'poly4',s4);
%
% %Set up trendline coefficients
% xIneg = -5:0.01:0;
% xIpos = 0:0.01:5;
% P1neg(1) = cf1.p1; P1neg(2) = cf1.p2; P1neg(3) = cf1.p3; P1neg(4) ...
    = cf1.p4; P1neg(5) = cf1.p5;
% P1pos(1) = cf2.p1; P1pos(2) = cf2.p2; P1pos(3) = cf2.p3; P1pos(4) ...
    = cf2.p4; P1pos(5) = cf2.p5;
% P2neg(1) = cf3.p1; P2neg(2) = cf3.p2; P2neg(3) = cf3.p3; P2neg(4) ...
    = cf3.p4; P2neg(5) = cf3.p5;
% P2pos(1) = cf4.p1; P2pos(2) = cf4.p2; P2pos(3) = cf4.p3; P2pos(4) ...
    = cf4.p4; P2pos(5) = cf4.p5;
%
% %Define trendline curves
% Aeq1neg = P1neg(1)*xIneg.^4 + P1neg(2)*xIneg.^3 + ...
    P1neg(3)*xIneg.^2 + P1neg(4)*xIneg + P1neg(5);
% Aeq1pos = P1pos(1)*xIpos.^4 + P1pos(2)*xIpos.^3 + ...
    P1pos(3)*xIpos.^2 + P1pos(4)*xIpos + P1pos(5);
%
% Aeq2neg = P2neg(1)*xIneg.^4 + P2neg(2)*xIneg.^3 + ...
    P2neg(3)*xIneg.^2 + P2neg(4)*xIneg + P2neg(5);
% Aeq2pos = P2pos(1)*xIpos.^4 + P2pos(2)*xIpos.^3 + ...
    P2pos(3)*xIpos.^2 + P2pos(4)*xIpos + P2pos(5);
```

APPENDIX C

DYNAMIC SIMULATION COMPONENTS

The dynamic simulation is constructed by interfacing a Simulink file containing the actuator model and associated inputs, outputs, analysis and control tools with SrLib, a dynamics library that builds a robot and its environment and models the dynamics of their interactions. This section provides an overview of the framework and key files required for each of these components.

C.1 MATLAB/Simulink Actuator Models

The Simulink file that was interfaced with SrLib contains three individual actuators that represent the Alpha, Beta, and Gamma joint of a leg. This file, displayed in figures 60 and 61, uses the individual actuator model referenced in Appendix B.1 with some minor modifications: (1) Many outputs used primarily for debugging and analysis have been removed, and (2) cylinder and joint parameters have been redefined to become joint-specific constants, as visible in the associated setup file provided in section C.1.1.

In addition to the actuator models, the Simulink file contains blocks to configure the local network settings and establish UDP communication with SrLib, as well as several blocks for data processing, inputs, outputs, conversion tools, debugging, and analysis. These are labeled and color coded, as specified by the legends within the corresponding figures.

C.1.1 Setup File

A setup script must be run before compiling the Simulink file. The following example is for a system with 50 psig supply air.

```
%% Bimba Setup for 50 psi SrLib Tests
%Created April 24, 2011
%Last updated October 2011

%% Prep workspace if desired
% clear all;
% close all;

%% Run time constants
dt = .001;
udprate = dt;

%% Load required pre-recorded signals
%Load the recorded signals from Phantom tests
LoadRecordedSignals;

%% General Constants
k=1.4;          %for air

cp=1.012*10^3;  %J/(kg*K) room temp air 1% humidity

g=9.81;         %m/s^2

%R=8.314;       %universal gas constant J/(K*mol)
% when divided my air molar mass (29g/mol)
R=287;          %J/(kg*K)

Ts=298;         %Kelvin (25 deg C)

%To be converted:
Ps=50.3;        %supply air, psig
Pe=24.7;        %exhaust air (room - measured), psi
```

```
33
34 Ps=Ps+14.7;     %psig
35
36 %% Conversion factors
37 in2m = .0254;
38 psi2Pa = 6894.76;
39
40 %% Valve Dynamics
41 choke_ratio=.528;       %pressure ratio, if passed will generate choking
42 C1=sqrt(2*k/(R*(k-1))); %Unchoked
43 C2=sqrt(k/(R*((k+1)/2)^((k+1)/(k-1)))); %Choked
44
45 cdC = [0 0.09];  %Unused linear cd model
46 cdR = [0 0.12];  %Unused linear cd model
47
48 alpha=1;
49
50 %% Cylinder Dimensions
51
52 %alpha joint
53 a_rod=0.375*in2m;   %rod diameter, m
54 a_bore=1.0625*in2m; %bore diameter, m
55 a_stroke=3.75*in2m; %stroke length, m
56 a_rds = 0.018;      %rod side dead space, m
57 a_cds = 0.012;      %cap side dead space, m
58
59 a_Aslider = 0.0045*.0025;           %Slider area, m^2
60 a_Ap = pi*a_bore^2/4 - a_Aslider;   %Piston side area, m^2
61 a_Arod = pi*a_rod^2/4;              %Area of Rod, m^2
62 a_Ar = a_Ap-a_Arod;                 %Rod side area, m^2
63
64 %beta joint
```

```
65 b_rod=0.375*in2m;   %rod diameter, m
66 b_bore=1.0625*in2m; %bore diameter, m
67 b_stroke=1.75*in2m; %stroke length, m
68 b_rds = 0.018;      %rod side dead space, m
69 b_cds = 0.012;      %cap side dead space, m
70
71 b_Aslider = 0.0045*.0025;           %Slider area, m^2
72 b_Ap = pi*b_bore^2/4 - b_Aslider;   %Piston side area, m^2
73 b_Arod = pi*b_rod^2/4;              %Area of Rod, m^2
74 b_Ar = b_Ap-b_Arod;                 %Rod side area, m^2
75
76
77 %gamma joint
78 g_rod=0.375*in2m;   %rod diameter, m
79 g_bore=1.0625*in2m; %bore diameter, m
80 g_stroke=1.75*in2m; %stroke length, m
81 g_rds = 0.018;      %rod side dead space, m
82 g_cds = 0.012;      %cap side dead space, m
83
84 g_Aslider = 0.0045*.0025;           %Slider area, m^2
85 g_Ap = pi*g_bore^2/4 - g_Aslider;   %Piston side area, m^2
86 g_Arod = pi*g_rod^2/4;              %Area of Rod, m^2
87 g_Ar = g_Ap-g_Arod;                 %Rod side area, m^2
88
89 %mass attachments for simple tests (not used with SrLib)
90 a_m = 0.2; %mass of attachment, kg
91 b_m = 0.2; %mass of attachment, kg
92 g_m = 0.2; %mass of attachment, kg
93
94 %% Cylinder Friction
95 %Viscous Friction model
96 a_b=70;              %Damping constant
```

```
97  b_b=70;                %Damping constant
98  g_b=70;                %Damping constant
99
100 %Stribeck-Tanh model (all in lbf)
101 Fsf = 4.5;  %Static Friction
102 FcfU = 3;   %Coulomb Friction (Inwards)
103 FcfD = 3;   %Coulomb Friction (Outwards)
104 Cv = 0.5;   %Coefficient of Viscous Friction
105 Ktanh = 40; %Tanh Coefficient
106 ii = 5;     %Exponent
107 Vs = 0.1;   %Sliding Speed Coefficoent
108
109 zVel = Vs;  %Actually used in friction model as Vs, so not strictly ...
       zero velocity threshold
110 zPos = 0.04;%Unused, formerly similar to Vs
111
112 %% Flags if needed
113 velset=1;   %To reset velocity integrator
114 accset=5;   %To reset acceleration integrator
115
116 %% Simulation Const
117 Ps=Ps*psi2Pa;   %Supply pressure, Pa
118 Pe=Pe*psi2Pa;   %Exhaust pressure, Pa
119
120 %% Initial Conditions
121 a_startPos = a_stroke;
122 b_startPos = b_stroke;
123 g_startPos = g_stroke;
124 rodP0 = Ps;
125 capP0 = 14*psi2Pa;
```

C.1.2 Simulink File

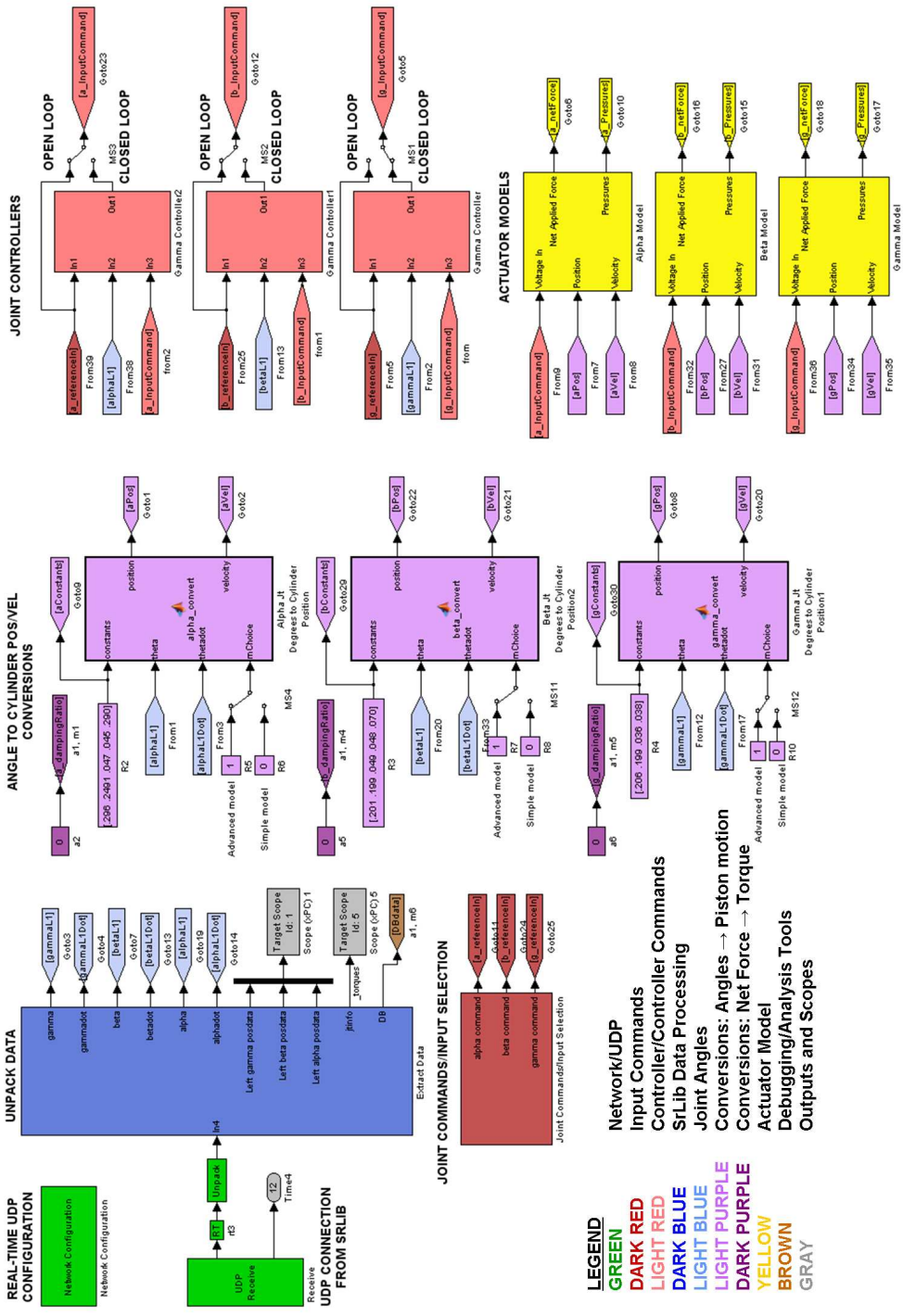

Figure 60: First half of the Simulink file used together with SrLib to simulate an entire CRR leg

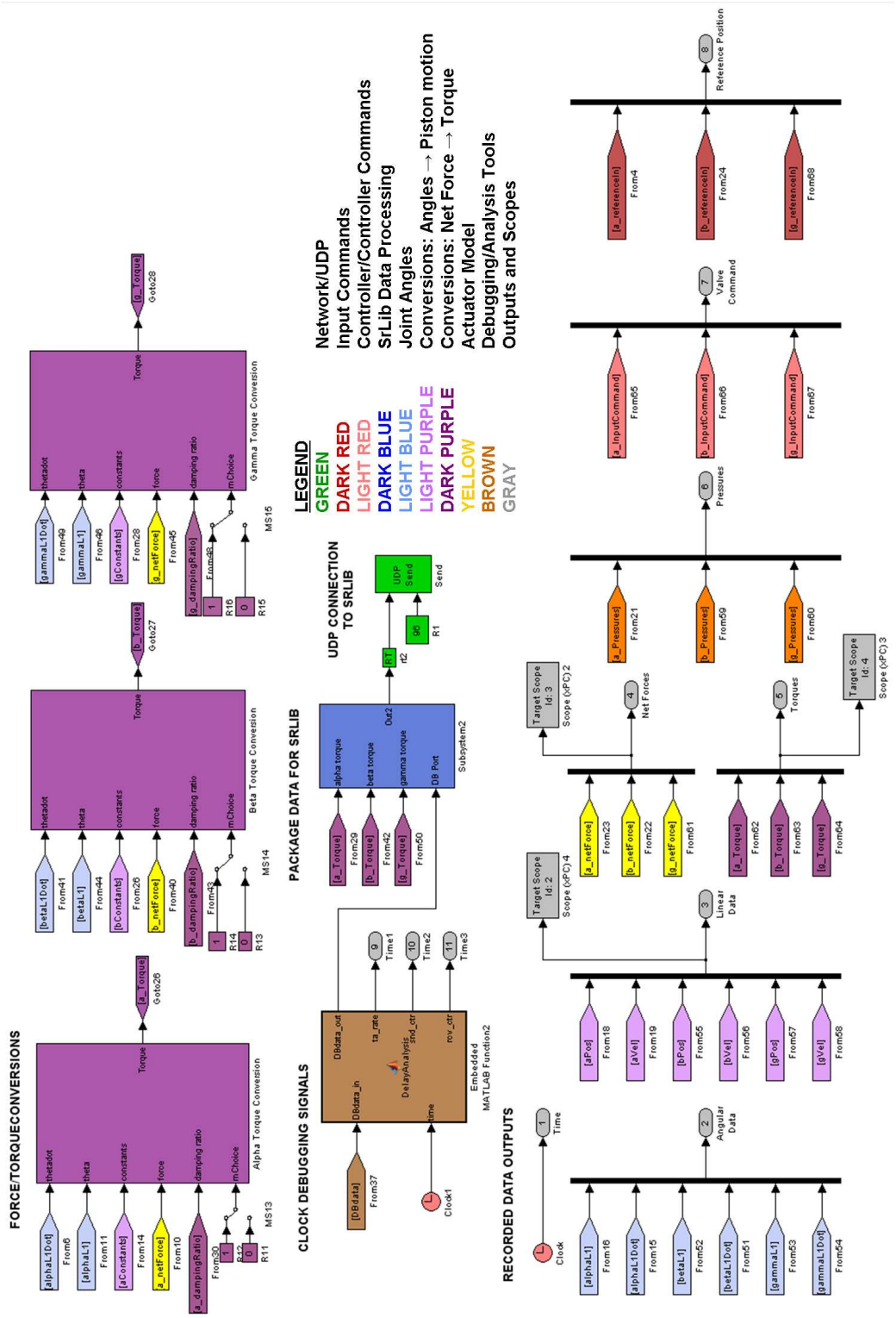

Figure 61: Second half of the Simulink file used together with SrLib to simulate an entire CRR leg

C.2 SrLib Components

Though SrLib consists of many components, its basic structure is illustrated in Figure 62 and discussed in greater detail in section 5.1. Example code from the files mentioned in section 5.1, which are largely specific to the CRR, is provided here. SrLib is also available for free download online at http://sourceforge.net/projects/srlib/ or by contacting Jaeyoung Haan [24].

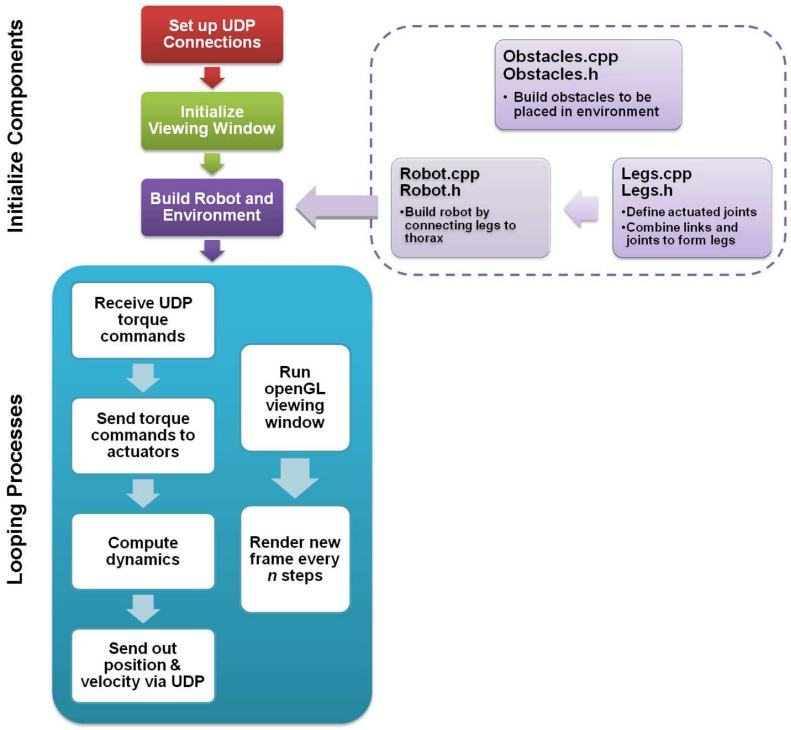

Figure 62: Diagram of SrLib process structure

C.2.1 Leg Model

The following excerpt from `Leg.cpp` shows how joint actuation, link geometry and properties are assigned, and a leg is constructed as a result.

```
1 void Leg::constructPart()  {
2     m_Coxa1.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
```

```
3       m_Coxa1.GetGeomInfo().SetDimension(0.05, 0.0961);
4       m_Coxa1.GetGeomInfo().SetColor(0.8, 0.2, 1);
5
6
7       //inertia for Coxa(1)
8       m_Coxa1.UpdateInertia(1200);
9
10      m_Coxa1Collision.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
11      m_Coxa1Collision.GetGeomInfo().SetDimension(0.05, 0.0961);
12      m_Coxa1Collision.SetLocalFrame(SE3());
13      m_Coxa1.AddCollision(&m_Coxa1Collision);
14
15      m_Coxa2.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
16      m_Coxa2.GetGeomInfo().SetDimension(0.05, 0.0);
17      m_Coxa2.GetGeomInfo().SetColor(0.8, 0.2, 1);
18      m_Coxa2.UpdateInertia(300); //Inertia for Coxa(2)
19
20      m_Coxa2Collision.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
21      m_Coxa2Collision.GetGeomInfo().SetDimension(0.05, 0.0);
22      m_Coxa2Collision.SetLocalFrame(SE3());
23      m_Coxa2.AddCollision(&m_Coxa2Collision);
24
25      m_Femur.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
26      m_Femur.GetGeomInfo().SetDimension(0.05, 0.123);        // ...
            Diameter, depth.
27      m_Femur.GetGeomInfo().SetColor(0, 0.6, 1);
28
29      //inertia for Femur
30      m_Femur.UpdateInertia(700);
31
32      m_FemurCollision.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
```

```
33      m_FemurCollision.GetGeomInfo().SetDimension(0.05, 0.123);   // ...
            Diameter
34      m_FemurCollision.SetLocalFrame(SE3());
35      m_Femur.AddCollision(&m_FemurCollision);
36
37      m_Tibia.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
38      m_Tibia.GetGeomInfo().SetDimension(0.015, 0.255);   // Diameter, ...
            depth.
39      m_Tibia.GetGeomInfo().SetColor(0, 0.8, 0);
40
41      //Tibia calculates its own inertia
42      m_Tibia.UpdateInertia(2700); //density in g/mm^3
43      m_Tibia.SetFriction(3.0);
44      m_TibiaCollision.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
45      m_TibiaCollision.GetGeomInfo().SetDimension(0.015, 0.255);  // ...
            Diameter
46      m_TibiaCollision.SetLocalFrame(SE3());
47      m_Tibia.AddCollision(&m_TibiaCollision);
48
49
50      //Alpha Joint
51      m_J[0].SetActType(srJoint::TORQUE);
52      m_J[0].SetPositionLimit(-47.0, 90.0); // in degree
53      m_J[0].SetChildLink(&m_Coxa1);
54      m_J[0].SetChildLinkFrame(EulerZYX(Vec3(0.0, SR_PI/2, ...
            0.0),Vec3(0.0, 0.0, 0.0731)));
55
56      m_Coxa.SetParentLink(&m_Coxa1);
57      m_Coxa.SetChildLink(&m_Coxa2);
58      m_Coxa.SetParentLinkFrame(EulerZYX(Vec3(0.0, 0.0, 0.0),Vec3(0.0, ...
            0.0, -0.0481)));
```

```
59      m_Coxa.SetChildLinkFrame(EulerZYX(Vec3(0.0, -SR_PI/2, ...
            0.0),Vec3(0.0, 0.0, -0.025)));
60
61      //Beta Joint
62      m_J[1].SetActType(srJoint::TORQUE);
63      m_J[1].SetPositionLimit(-67.0, -1); // in degree
64      m_J[1].SetParentLink(&m_Coxa2);
65      m_J[1].SetChildLink(&m_Femur);
66      m_J[1].SetParentLinkFrame(EulerZYX(Vec3(0.0, 0.0, ...
            -SR_PI/2),Vec3(0.025, 0.0, 0.0)));
67      m_J[1].SetChildLinkFrame(EulerZYX(Vec3(0.0, SR_PI/2, ...
            0.0),Vec3(0.0, 0.0, 0.087)));
68
69      //Gamma Joint
70      m_J[2].SetActType(srJoint::TORQUE);
71      m_J[2].SetPositionLimit(17.0, 95.0); // in degree
72      m_J[2].SetParentLink(&m_Femur);
73      m_J[2].SetChildLink(&m_Tibia);
74      m_J[2].SetParentLinkFrame(EulerZYX(Vec3(0.0, SR_PI/2, ...
            0.0),Vec3(0.0, 0.0, -0.087)));
75      m_J[2].SetChildLinkFrame(EulerZYX(Vec3(0, SR_PI/2, 0),Vec3(0, ...
            0.0, 0.152)));
76  }
```

C.2.2 Robot Model

The primary portion of the code from `Robot.cpp` pieces together the robot using the legs from `Leg.cpp` and the newly defined thorax link, as shown in the sample code below.

```
1  srSystem* RescueRobot::BuildRobot(SE3 T) {
2      //Build left front leg
```

```
3      L1.constructPart();
4      L1.m_J[0].SetParentLink(&m_Thorax);      //Attach to thorax
5      L1.m_J[0].SetParentLinkFrame(EulerZYX(Vec3(-4*SR_PI/6, 0.0, ...
           -SR_PI/2),Vec3(-0.0086, -0.046, 0.375)));
6
7      //Right front leg
8      R1.constructPart();
9      R1.m_J[0].SetParentLink(&m_Thorax);     //Attach to thorax
10     R1.m_J[0].SetParentLinkFrame(EulerZYX(Vec3(4*SR_PI/6, 0.0, ...
           SR_PI/2),Vec3(-0.0086, 0.046, 0.375)));
11
12     //Left rear leg
13     L2.constructPart();
14     L2.m_J[0].SetParentLink(&m_Thorax);     //Attach to thorax
15     L2.m_J[0].SetParentLinkFrame(EulerZYX(Vec3(-4*SR_PI/6, 0.0, ...
           -SR_PI/2),Vec3(-0.0086, -0.046, -0.375)));
16
17     //Right rear leg
18     R2.constructPart();
19     R2.m_J[0].SetParentLink(&m_Thorax);     //Attach to thorax
20     R2.m_J[0].SetParentLinkFrame(EulerZYX(Vec3(4*SR_PI/6, 0.0, ...
           SR_PI/2),Vec3(-0.0086, 0.046, -0.375)));
21
22     //Thorax properties
23     m_Thorax.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
24     m_Thorax.GetGeomInfo().SetDimension(0.094, 0.75);
25     m_Thorax.GetGeomInfo().SetColor(0.5, 0.4, 0.9);
26     m_Thorax.UpdateInertia(10000);
27     m_ThoraxCollision.GetGeomInfo().SetShape(srGeometryInfo::CAPSULE);
28     m_ThoraxCollision.GetGeomInfo().SetDimension(0.094, 0.74);
29     m_ThoraxCollision.SetLocalFrame(SE3());
30     m_Thorax.AddCollision(&m_ThoraxCollision);
```

```
31
32      //Set thorax as base and fix in space
33      m_Thorax.SetFrame(T);
34      this->SetBaseLink(&m_Thorax);
35      this->SetBaseLinkType(srSystem::FIXED);
36      this->SetSelfCollision(false);
37
38      return this;
39  }
```

C.2.3 Environment Objects

`Obstacle.cpp` is used to define the types of objects that the robot interacts with. While many types of environment objects are defined in the complete file, the following example script for a ramp provides an idea of how the environment is constructed.

```
1  srSystem*   rampRoll::BuildObstacle(SE3 T) {
2      //Build a ramp
3      ramp.GetGeomInfo().SetShape(srGeometryInfo::BOX);
4      ramp.GetGeomInfo().SetDimension(1.2, 1.2, 0.019);
5      ramp.UpdateInertia();
6      ramp.SetFriction(3.0);
7      rampCollision.GetGeomInfo().SetShape(srGeometryInfo::BOX);
8      rampCollision.GetGeomInfo().SetDimension(1.2, 1.2, 0.019);
9      rampCollision.SetLocalFrame(SE3());
10     ramp.AddCollision(&rampCollision);
11
12     //Set base link
13     ramp.SetFrame(T);
14     this->SetBaseLink(&ramp);
15     this->SetBaseLinkType(FIXED);
16
```

```
17      return this;
18  }
```

C.2.4 Execution

`Main.cpp` shows the broader construction of SrLib. Several portions of the script are shown here, illustrating the functions most relevant to the topics discussed in this thesis. First, the `main()` function is used to run the program:

```
1  int main(int argc, char **argv)
2  {
3      // setup up incoming UDP channel from simulation
4      //Content removed due to network privacy concerns,
5      //but this is where IP and ports to use are defined
6
7      // STEP 1: Viewer initialization
8      // srSimpleViewer render scene using OpenGL GLUT.
9      // ...OpenGL GLUT need argc and argv for initializing.
10     gViewer.Init(&argc, argv, "Compact Rescue Robot");
11
12     // STEP 2: Robot Modeling
13     User_Modeling();
14
15     // STEP 3: Simulation setting
16     User_SimulationSetting();
17
18     // STEP 4: Run window view.
19     gViewer.Run();
20
21     return 0;
22 }
```

Within `main()`, two sections of interest are the User Modeling section, which defines the robot and environment discussed earlier in this appendix, and the User Control Loop, which specifies the serial structure that runs continuously once the basic framework has been initialized:

```
void User_Modeling()
{
    SE3 T;

    // System 1: Create Ground
    gSpace.AddSystem(gGround.BuildGround());
    gGround.m_Ground->SetFriction(5.0);

    // System 2, 3: Create Robot and Obstacles
    gSpace.AddSystem(robot.BuildRobot(EulerZYX(Vec3(SR_PI, -SR_PI/2, ...
        0.0), Vec3(0.0, 0.0, 0.8))));
    gSpace.AddSystem(map1.BuildObstacle(EulerZYX(Vec3(0.0, 0.0, ...
        0.0),Vec3(1.0, 3.4-0.05, 0.0)),"HALF RANDOM"));

    ////- Space
    // Set simulation time step.
    gSpace.SetTimestep(0.001);
    // Set gravity
    gSpace.SetGravity(0.0, 0.0, -9.8);  //-9.8 in lst set
    // Set number of sub-step for rendering
    gSpace.SetNumberofSubstepForRendering(100);
}
```

```
// Every simulation step, this function will work.
void User_CBFunc_ControlLoop()
```

```
{
    ReceiveNewData();

    robot.L1.SetHapCommand(thetaL11, thetaL12, thetaL13);
    robot.L2.SetHapCommand(thetaL21, thetaL22, thetaL23);
    robot.R2.SetHapCommand(thetaR21, thetaR22, thetaR23);

    //R1 receives explicit torque commands
    robot.R1.m_J[0].m_State.m_rCommand = thetaL11;
    robot.R1.m_J[1].m_State.m_rCommand = thetaL12;
    robot.R1.m_J[2].m_State.m_rCommand = thetaL13;

    //For position rather than torque commands
    //robot.PositionControl(0.001);
    SendNewData();
}
```

Finally, the `Send()` and `Receive()` functions show how data is shared with MATLAB/Simulink.

```
void SendNewData()
{
    udp_send_num++;

    //Joint position, velocity
    msgtemp.L11 = robot.R1.m_J[2].m_State.m_rValue[0];  //Gamma position
    msgtemp.L12 = robot.R1.m_J[2].m_State.m_rValue[1];  //Gamma velocity
    msgtemp.L13 = robot.R1.m_J[1].m_State.m_rValue[0];  //Beta position
    msgtemp.R11 = robot.R1.m_J[1].m_State.m_rValue[1];  //Beta velocity
    msgtemp.R12 = robot.R1.m_J[0].m_State.m_rValue[0];  //Alpha position
    msgtemp.R13 = robot.R1.m_J[0].m_State.m_rValue[1];  //Alpha velocity
    msgtemp.L21 = 0;
```

```
    msgtemp.L22 = 0;
    msgtemp.L23 = 0;
    msgtemp.R21 = 0;
    msgtemp.R22 = 0;
    msgtemp.R23 = 0;

    //Robot orientation
    SO3 temp = robot.m_Thorax.GetOrientation();
    msgtemp.ori11 = temp[0];
    msgtemp.ori12 = temp[1];
    msgtemp.ori13 = temp[2];
    msgtemp.ori21 = temp[3];
    msgtemp.ori22 = temp[4];
    msgtemp.ori23 = temp[5];

    //Debugging data
    msgtemp.ori31 = time_packet;
    msgtemp.ori32 = udp_send_num;
    msgtemp.ori33 = gSpace.m_Simulation_Time;

    sockH2C->send((char *) &msgtemp, sizeof(msgtemp));

}
```

```
void ReceiveNewData()
{
    sockS2D->recv((char *) &msg, sizeof(msg));

    thetaL11 = msg.L11; //Alpha Torque
    thetaL12 = msg.L12; //Beta Torque
    thetaL13 = msg.L13; //Gamma Torque
```

```
8
9       //Joint Angles
10      thetaR11 = msg.R11;
11      thetaR12 = msg.R12;
12      thetaR13 = msg.R13;
13      thetaL21 = msg.L21;
14      thetaL22 = msg.L22;
15      thetaL23 = msg.L23;
16
17      thetaR21 = 0;
18
19      //Debugging:
20      time_packet = msg.R21;
21      thetaR22 = 0;
22      thetaR23 = 0;
23      udp_recv_num++;
24  }
```

APPENDIX D

OPERATOR MANUAL

The following sections describe the basic technical configuration and procedure required to run the simulation and associate two-legged prototype.

D.1 Dynamic Simulation

The choice of hardware for use with the dynamic simulation is quite flexible. The computers are constrained primarily by the performance demands, as noted in Chapter 5. Additionally, the host and SrLib computer must each contain an Ethernet card, and the target PC must contain two. While the type of card does not matter for the SrLib PC, the host and target must use xPC Target compatible Ethernet cards.

Once hardware has been chosen, the network settings on each machine must be properly configured. Figure 63 shows a general case that also includes inputs from the Phantoms, while Figure 64 shows the case used for validation in this thesis, excluding Phantoms and using a single computer as both a host and to run SrLib.

The configurations shown in Figures 63 and 64 use a local switch operating on the 192.168.3.xxx network and host/target communication across the 30.30.30.xxx network. xxx is referred to with placeholders X0 - X3 and Y0-Y1, respectively, having value from 0 to 255.

The following steps must then be run to operate the simulation:

1. Set up the network and boot disk configuration as shown in Figure 63/64.

2. On the Host PC, open an appropriate Simulink file.

3. On the Phantom PC, open an appropriate Phantom control file.

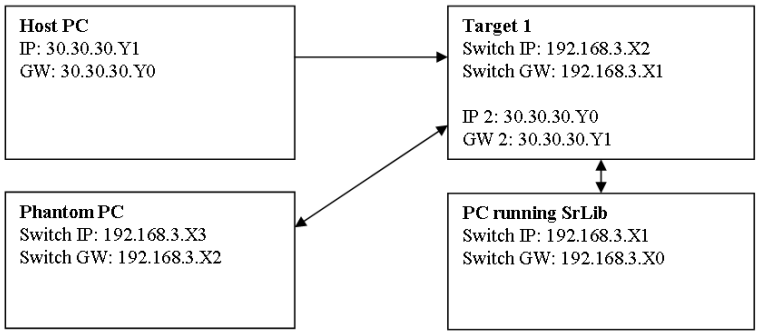

Figure 63: Dynamic simulation network configuration - general case. GW = "Gateway".

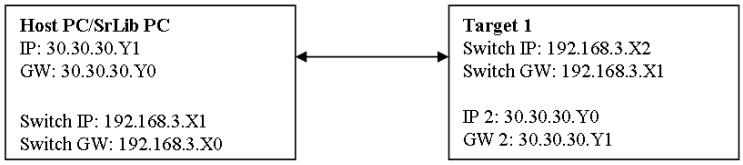

Figure 64: Dynamic simulation network configuration - arrangement used for validation in this thesis. GW = "Gateway".

4. On the PC running SrLib, open the appropriate SrLib file.

5. On the Host PC, click "incremental build" to compile and download the Simulink file to the target. Once it has completed downloading (this can be observed from the target monitor or the MATLAB screen), wait a few seconds to ensure that it has fully loaded (otherwise, MATLAB may return an error and force a target reboot).

6. Once it has downloaded, click "Connect to Target" and wait for the play button to light up. Click play to run.

7. Turn on the Phantoms. Do this by clicking "Go" or hitting F5 on the keyboard while in PosMode.dsw on the Phantom PC.

8. Run the simulation by hitting "play" on the Host PC. Once it is open, the

mouse can be used to rotate the view or adjust the zoom. To begin simulation, hit"p" on the PC running SrLib.

The order of steps 2-5 and 6 - 8 can be varied, though the phantoms tend to move about if turned on before the simulation is begun. For systems where the phantom is not used, ignore steps 3 and 7.

D.2 Two-Legged Prototype

The configuration shown in Figure 65 uses a local switch operating on the 192.168.3.xxx network, where xxx is referred to with placeholders X0 - X3, having value from 0 to 255.

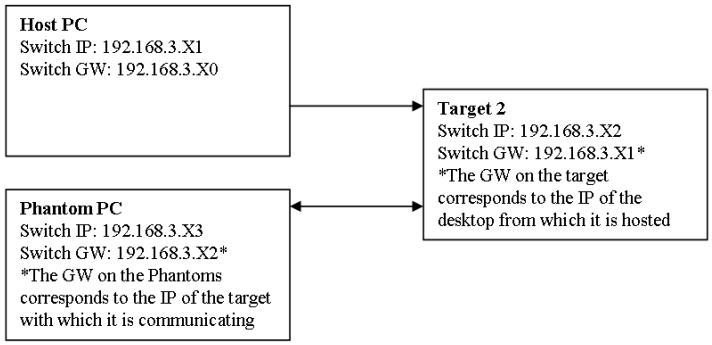

Figure 65: Two-legged prototype network configuration - general case. GW = "Gateway".

1. Set up the network and boot disk configuration as shown above.

2. Open an appropriate Simulink file on the Host PC for use with the target.

3. On the Phantom PC, open an appropriate Phantom control file.

4. On the Host PC, click "incremental build" to compile and download the Simulink file to the target. Once it has completed downloading (this can be observed from the target monitor or the MATLAB screen), wait a few seconds to ensure that

it has fully loaded (otherwise, MATLAB may return an error and force a target reboot).

5. Once it has downloaded, click "Connect to Target" and wait for the play button to light up.

6. Click "Play". The program is now running, but further configuration is still necessary.

7. Turn on the Phantoms. Do this by clicking "Go" or hitting F5 on the keyboard while in PosMode.dsw on the Phantom PC. The program can now be operated.

REFERENCES

[1] "Gazebo," 2005. http://playerstage.sourceforge.net/gazebo/gazebo.html. Last accessed: October 8, 2011.

[2] "Dymola: Code and model export," 2011. http://www.3ds.com/products/catia/portfolio/dymola/dymola-product-line/code-and-model-export/. Last accessed: October 8, 2011.

[3] "filtfilt: Zero-phase digital filtering," 2011. http://www.mathworks.com/help/toolbox/signal/ref/filtfilt.html. Last accessed: October 4, 2011.

[4] "Simulink coder," 2011. http://www.mathworks.com/products/simulink-coder/index.html. Last accessed: October 22, 2011.

[5] "Wireshark," 2011. http://www.wireshark.org/. Last accessed: October 18, 2011.

[6] Al-Dakkan, K. A., Barth, E. J., and Goldfarb, M., "Dynamic constraint-based energy-saving control of pneumatic servo systems," *Transactions of the ASME. Journal of Dynamic Systems, Measurement and Control*, vol. 128, pp. 655–62, 2006.

[7] Andersson, S., Soderberg, A., and Bjorklund, S., "Friction models for sliding dry, boundary and mixed lubricated contacts," *Tribology International*, vol. 40, no. 4, pp. 580–587, 2007.

[8] Bobrow, J. E. and Jabbari, F., "Adaptive pneumatic force actuation and position control," *Transactions of the ASME. Journal of Dynamic Systems, Measurement and Control*, vol. 113, pp. 267–72, 1991.

[9] Bone, G. M. and Shu, N., "Experimental comparison of position tracking control algorithms for pneumatic cylinder actuators," *IEEE/ASME Transactions on Mechatronics*, vol. 12, pp. 557–61, 2007.

[10] Carneiro, J. F. and de Almeida, F. G., "Modeling pneumatic servo-valves using neural networks," in *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pp. 790–795, 2006.

[11] Celaya, E. and Porta, J. M., "A control structure for the locomotion of a legged robot on difficult terrain," *IEEE Robotics and Automation Magazine*, vol. 5, pp. 43–51, 1998.

[12] Cepolina, F., Moronti, M., Sanguinet, M., Zoppi, M., and Molfino, R. M., "Roboclimber versus landslides: design and realization of a heavy-duty robot for teleoperated consolidation of rocky walls," *IEEE Robotics & Automation Magazine*, vol. 13, pp. 23–31, 2006.

[13] Chipalkatty, R., Daepp, H. G., Egerstedt, M., and Book, W., "Human-in-the-loop: Mpc for shared control of a quadruped rescue robot," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 25 - 30, 2011 2011.

[14] Corominas Murtra, A., Mirats Tur, J. M., Sandoval, O., and Sanfeliu, A., "Real-time software for mobile robot simulation and experimentation in cooperative environments," in *1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots, SIMPAR 2008, November 3, 2008 - November 6, 2008*, vol. 5325 LNAI of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 135–146, Springer Verlag, 2008.

[15] Daepp, H. G. and Book, W., "Modeling and simulation of a pneumatically-actuated rescue robot," in *52nd National Conference on Fluid Power*, 2011.

[16] de Santos, P. G., Estremera, J., Garcia, E., and Armada, M., "Including joint torques and power consumption in the stability margin of walking robots," *Autonomous Robots*, vol. 18, pp. 43–57, 2005.

[17] De Santos, P. G., Galvez, J. A., Estremera, J., and Garcia, E., "Sil04: a true walking robot for the comparative study of walking machine techniques," *IEEE Robotics and Automation Magazine*, vol. 10, pp. 23–32, 2003.

[18] Ding, Y. and Scharf, E., "Deadlock avoidance for a quadruped with free gait," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, 8-13 May 1994*, vol. 1 of *Proceedings 1994 IEEE International Conference on Robotics and Automation (Cat. No.94CH3375-3)*, pp. 143–8, IEEE Comput. Soc. Press, 1994.

[19] Driewer, F., Baier, H., and Schilling, K., "Robot-human rescue teams: a user requirements analysis," *Advanced Robotics*, vol. 19, pp. 819–38, 2005.

[20] Ferretti, G., Gritti, M., Magnani, G., Rizzi, G., and Rocco, P., "Real-time simulation of modelica models under linux / rtai," in *4th International Modelica Conference*, pp. 359–365, 2005.

[21] Ferretti, G., Magnani, G., Porrati, P., Rizzi, G., Rocco, P., and Rusconi, A., "Real-time simulation of a space robotic arm," in *Workshop on robot simulators at the IEEE/RSJ International Conference on Intelligent RObots and Systems*, 2008.

[22] Goldfarb, M., Barth, E. J., Gogola, M. A., and Wehrmeyer, J. A., "Design and energetic characterization of a liquid-propellant-powered actuator for self-powered robots," *Mechatronics, IEEE/ASME Transactions on*, vol. 8, no. 2, pp. 254–262, 2003.

[23] Guerriero, B., *Haptic Control and Operator-Guided Gait Coordination of a Pneumatic Hexapedal Rescue Robot*. PhD thesis, Georgia Institute of Technology, 2008.

[24] Haan, J., "Srlib user manual," tech. rep., Seoul National University, 2009.

[25] Halme, A., Kartikainen, K., and Karkkainen, K., "Terrain adaptive motion and free gait of a six-legged walking machine," *Control Engineering Practice*, vol. 2, pp. 273–9, 1994.

[26] Hirose, S., "A study of design and control of a quadruped walking vehicle," *International Journal of Robotics Research*, vol. 3, pp. 113–33, 1984.

[27] Hong, I. and Tessmann, R., "The dynamic analysis of pneumatic systems using hypneu," in *Proceedings of the National Conference on Fluid Power*, vol. 47, pp. 61–70, National Fluid Power Association, 1996.

[28] Jian-Nan, L. and Shin-Min, S., "Modeling gait transitions of quadrupeds and their generalization with cmac neural networks," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 32, pp. 177–89, 2002.

[29] Kim, T. Y., "Simulation and control of a four-legged rescue robot," tech. rep., Georgia Institute of Technology, December 2009.

[30] Kriegsmann, M., "Servocontrol with pneumatic actuators," *Machine Design*, vol. 79, pp. 78–80, 2007.

[31] Luk, B. L., Collie, A. A., and Billingsley, J., "Robug ii: An intelligent wall climbing robot," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 2342–2347 vol.3, 1991.

[32] Luk, B. L., Collie, A. A., Piefort, V., and Virk, G. S., "Robug iii: a tele-operated climbing and walking robot," in *Control '96, UKACC International Conference on (Conf. Publ. No. 427)*, vol. 1, pp. 347–352 vol.1, 1996.

[33] Luk, B. L., Liu, K. P., Collie, A. A., Cooke, D. S., and Chen, S., "Tele-operated climbing and mobile service robots for remote inspection and maintenance in nuclear industry," *Industrial Robot*, vol. 33, pp. 194–204, 2006.

[34] McGhee, R. B. and Frank, A. A., "On the stability properties of quadruped creeping gaits," *Mathematical Biosciences*, vol. 3, pp. 331–52, 1968.

[35] McGhee, R. B. and Iswandhi, G. I., "Adaptive locomotion of a multilegged robot over rough terrain," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-9, pp. 176–82, 1979.

[36] McRae, A., "Channel capacity in absolute judgment tasks: An artifact of information bias?," *Psychological Bulletin*, vol. 73, pp. 112–121, 1970.

[37] Messina, E., Jacoff, A., Scholtz, J., Schlenoff, C., Huang, H.-M., Lytle, A., and Blitch, J., "Statement of requirements for urban search and rescue robot performance standards," technical report, National Institute of Standards and Technology Department of Homeland Security, 2005.

[38] Mistry, M., Nakanishi, J., and Schaal, S., "Task space control with prioritization for balance and locomotion," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, 29 Oct.-2 Nov. 2007*, 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 331–8, IEEE, 2007.

[39] Moosavian, S. A. A., Kalantari, A., Semsarilar, H., Aboosaeedan, E., and Mihankhah, E., "Resquake: a tele-operative rescue robot," *Journal of Mechanical Design*, vol. 131, p. 081005 (11 pp.), 2009.

[40] Murray, C. J., "Robotic lifesaver [battlefield extraction assist robot]," *Design News*, vol. 61, pp. 46–50, 2006.

[41] Nelson, G. M. and Quinn, R. D., "Posture control of a cockroach-like robot," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 1, pp. 157–162 vol.1, 1998.

[42] Raibert, M., "Bigdog, the rough-terrain quadruped robot," in *17th World Congress, International Federation of Automatic Control, IFAC, July 6, 2008 - July 11, 2008*, vol. 17 of *IFAC Proceedings Volumes (IFAC-PapersOnline)*, Elsevier, 2008.

[43] Riofrio, J. A., Al-Dakkan, K., Hofacker, M. E., and Barth, E. J., "Control-based design of free-piston stirling engines," in *American Control Conference, 2008*, pp. 1533–1538, 2008.

[44] Riofrio, J. A. and Barth, E. J., "Design of a free piston pneumatic compressor as a mobile robot power supply," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 235–240, 2005.

[45] Schneider, D., "Robin murphy roboticist to the rescue," *Spectrum, IEEE*, vol. 46, no. 2, pp. 36–37, 2009.

[46] Schulte, H. and Hahn, H., "Fuzzy state feedback gain scheduling control of servo-pneumatic actuators," *Control Engineering Practice*, vol. 12, pp. 639–50, 2004.

[47] Shu, N. and Bone, G. M., "Development of a nonlinear dynamic model for a servo pneumatic positioning system," in *Mechatronics and Automation, 2005 IEEE International Conference*, vol. 1, pp. 43–48 Vol. 1, 2005.

[48] Smith, R., "Ode: Open dynamics engine," 2006. http://www.ode.org. Last accessed: October 8, 2011.

[49] Song, S.-M. and Waldron, K. J., *Machines that walk : the adaptive suspension vehicle*. MIT Press series in artificial intelligence, Cambridge, Mass.: MIT Press, 1989.

[50] Sukhatme, G. S., "The design and control of a prototype quadruped microrover," *Autonomous Robots*, vol. 4, pp. 211–20, 1997.

[51] Thomas, J. H., "Proper valve size helps determine flow," in *Control Engineering Online*, Control Engineering Online, 2000.

[52] Thomas, M. B. and Maul, G. P., "Considerations on a mass-based system representation of a pneumatic cylinder," *Journal of Fluids Engineering, Transactions of the ASME*, vol. 131, pp. 0411011–04110110, 2009.

[53] Thompson, S. C., Young, N., Thiets, R., Groshans, T. M., and Paredis, C. J. J., "Modeling and simulation in the design of a lunar rover suspension with extensibility to martian applications," in *2007 AIAA Modeling and Simulation Technologies Conference, August 20, 2007 - August 23, 2007*, vol. 2 of *Collection of Technical Papers - 2007 AIAA Modeling and Simulation Technologies Conference*, pp. 682–695, American Institute of Aeronautics and Astronautics Inc., 2007.

[54] Urwin-Wright, S., Sanders, D., and Chen, S., "Terrain prediction for an eight-legged robot," *Journal of Robotic Systems*, vol. 19, no. Copyright 2002, IEE, pp. 91–8, 2002.

[55] Wait, K. W., *The Use of Pneumatic Actuation to Address Shortcomings Concerning Normalized Output Power in State of the Art Mobile Robotics*. PhD thesis, Vanderbilt University, 2010.

[56] Wait, K. W. and Goldfarb, M., "Enhanced performance and stability in pneumatic servosystems with supplemental mechanical damping," *Journal of Dynamic Systems, Measurement and Control*, vol. 132, p. 041012 (8 pp.), 2010.

[57] Waldron, K. J. and McGhee, R. B., "The adaptive suspension vehicle," *IEEE Control Systems Magazine*, vol. 6, pp. 7–12, 1986.

[58] Zhu, Y. and Barth, E. J., "Impedance control of a pneumatic actuator for contact tasks," in *2005 IEEE International Conference on Robotics and Automation, April 18, 2005 - April 22, 2005*, vol. 2005 of *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 987–992, Institute of Electrical and Electronics Engineers Inc., 2005.

[59] Zlajpah, L., "Simulation in robotics," *Mathematics and Computers in Simulation*, vol. 79, pp. 879–897, 2008.