HAPTIC CONTROL AND OPERATOR-GUIDED GAIT COORDINATION OF A PNEUMATIC HEXAPEDAL RESCUE ROBOT

A Thesis
Presented to
The Academic Faculty

by

Brian A. Guerriero

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
George W. Woodruff School of Mechanical Engineering

Georgia Institute of Technology
August 2008

HAPTIC CONTROL AND OPERATOR-GUIDED GAIT

COORDINATION OF A PNEUMATIC HEXAPEDAL RESCUE ROBOT

Approved by:

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

Dr. Christian Paredis
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Harvey Lipkin
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: 6-20-2008

in loving memory of my mother,

Pamela J. Guerriero (1958-99).

Her strength and insight has been my inspiration and guiding light.

The wisdom and guidance, which she instilled in me, remains my foundation and personal ethic.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Wayne Book, for his support and guidance through the course of this project. His gave me an incredible opportunity, confidence and freedom to design and develop this robot platform.

JD Huggins also deserves my utmost gratitude for his assistance. His innate ability to analyze and troubleshoot problems saved me incalculable amounts of time and frustration. His advice and experience helped me throughout my design and fabrication phases, and his foresight was invaluable and significantly contributory to my success.

I would like to thank my committee members Dr. Harvey Lipkin and Dr. Chris Paredis for their time, advice, and insights throughout the progress of this project.

Dr. Haihong Zhu deserves special gratitude for his tireless efforts in providing this project with functional and reliable sensing technology.

Dr. Matt Kontz deserves special thanks for helping me get started here at Georgia Tech, laying the technical foundations for my xPC Target communications, and the PHANToM C++ codes, which were crucial time-saving tools.

I would like to thank Roman Shtylman for his assistance in modifying and optimizing my C++ codes. His work, on his own time, saved me weeks of debugging and recoding.

I would also like to give my gratitude to John Graham and his staff in the ME machine shop for their training, patience, and time.

Additionally I would like to thank my friends and colleagues who gave me advice and guidance who were not directly involved with this project, namely Matt Rogge, Jevawn Roberts, Renee Sutherland, Tom Groshans and all my IMDL lab mates, Mark Elton, Mohsin Waqar, Aaron Enes, Longke Wang, Heather Humphres, Brian Post, and Ryder Wyck.

Lastly, I would like to thank my family for their support, financial and moral.

I would like to acknowledge the corporate sponsors who donated parts and time to this project, Festo, Sentrinsic, Daman, and Enfield Technologies.

This work was funded through the National Science Foundation Center for Compact and Efficient Fluid Power, Grant EEC # 0540834.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A two-legged walking robot was designed, fabricated, and controlled through bilateral teleoperation via two PHANToM haptic devices. The Compact Rescue Crawler is a collaborative effort between Georgia Institute of Technology, Vanderbilt University, and North Carolina A&T working through the NSF Center for Compact and Efficient Fluid Power.

The Georgia Institute of Technology contributions to this pneumatic testbed are a haptically controlled two-legged robot, operator workstation, an augmented reality interface, and a guided-gait routine allowing a single operator to effectively control six legs while maneuvering through treacherous and unknown terrain. The two-legged vehicle was built and is teleoperated from a remote operator workstation. The guided-gait routine was designed, as well.

A force-based position controller coordinates 3D operator inputs into pneumatic cylinder stroke length commands and tracks position commands to within 10%. The controller tracks position in both free-space and ground contact scenarios, allowing the user to walk the robot remotely from the workstation and haptically feel the environment, and see the terrain through a head mounted display controlling an onboard PTZ camera.

CHAPTER 1

INTRODUCTION

1.1 CCEFP Background

The National Science Foundation Center for Compact and Efficient Fluid Power is an engineering research center focused on, as the name implies, improving the compactness, efficiency, and effectiveness of fluid power. Making efficient fluid power ubiquitous in our society allows high power density devices to be more commonplace. Improving fluid power effectiveness improves efficiency and unleashes the potential to save millions of dollars worldwide. Divided into three Thrusts and four Testbeds, the CCEFP research is managed across seven universities. The Testbeds not only serve as platforms and showcases for the technologies developed through the individual research Thrusts, but are also focal points for the new research required to achieve Testbed success.

Compactness, Efficiency, and Effectiveness are the three research Thrusts. Four testbeds are currently under development. Led by researchers at Purdue University, TB1 is an excavator testbed on which new developments in variable displacement pumps, throttleless valve control, and human factors research will be implemented.

Led by researchers at the University of Minnesota (UMN), TB3 is a small hybrid urban vehicle testbed on which

new open accumulator developments and other efficiency research will be implemented. New compact components developed through CCEFP research will also find a home on the small Urban Vehicle (sUV) testbed.

TB6: Fluid Power Assisted Ankle-Foot Orthoses, led by researchers at University of Illinois Urbana-Champaign (UIUC) seeks to revolutionize the orthoses currently available by integrating fluid power assistance and resistance. These orthoses will showcase research products in compactness and effectiveness.

Finally, TB4: Compact Rescue Crawler (CRC), led jointly by Vanderbilt University and Georgia Institute of Technology, is a revolutionary hexapedal search and rescue robot driven by hot-gas monopropellant. Harnessing new developments in chemofluidic actuation, control, and user interfaces, this testbed will eventually become an effective and powerful alternative to electric motor-driven search vehicles. The CRC also epitomizes the challenges of man-machine interaction prevalent in many fluid power applications and will lead to future opportunities for human-scale fluid power devices.

1.1.1 Collaboration

The research at Georgia Institute of Technology was part of a collaborative effort between Vanderbilt University, North Carolina Agricultural and Technical State University (NCAT), and Georgia Institute of Technology. Georgia Tech research is focused on the multimodal man-

machine interface and the haptic control of the robot legs [1].

Research at Vanderbilt is focused on chemofluidic actuation using decomposed H_2O_2 for a power source. Researchers are developing valves and actuators to control and harness the high temperature and pressure fluid produced through decomposition [2].

Vanderbilt research is also focusing on using an impedance controller to maneuver legs through a hexapedal tripod gait in which the operator will give simple commands to move the robot, i.e. “forward,” “right,” etc [3].

NCAT research is focused on the human factors areas related directly to TB4. The research covers a task analysis for the rescue mission, task analysis for the operator driving the CRC, and methods through which information should be quickly and effectively displayed to the operator.

1.2 Legged Mobility

Numerous advantages arise when legged locomotion is chosen over tracked or wheeled methods. A vehicle with redundant legs can alternatively use a spare leg as a manipulator as is often seen in nature (Figure 1.1).

Figure 1.1: An Ant Uses Redundant Legs to Manipulate its Environment

Legged platforms also display static stability when maintaining at least three points of ground contact, but can be much more maneuverable in unknown and hazardous terrain. Legs can step over obstacles, whereas tracks must rely on motor torque and traction to pull themselves and the weight of the entire vehicle over terrain.

In a search and rescue scenario, a legged vehicle can maneuver through, over, and under debris more nimbly than a comparably sized tracked vehicle. Due to the nature of legged locomotion, such methods have been difficult to realize because of the low speed and high joint torques necessary to exert force at the foot. Large motors and harmonic drives work well for industrial robots, but search and rescue vehicles must maintain a small profile while remaining strong and maneuverable.

Legged vehicles also leave a smaller footprint on the environment, providing advantages to the logging industry to prevent lasting imprints on the forest bed from road-building and tracked vehicles destroying the ground (Figure 1.2).

Figure 1.2: John Deere Legged Harvester

1.3 Research Objectives

The research objectives for the Testbed 4: Compact Rescue Crawler are threefold. First, the legged robot platform must be designed and fabricated. Secondly, the robot legs must be controlled effectively through a real-time onboard controller and remote workstation. Thirdly, a gait sequence must be designed to guide trailing legs over the terrain and obstacles which the operator avoided while guiding the front legs.

1.3.1 Testbed Design and Fabrication

Design requirements for the testbed design were fairly open due to the pioneering nature of the robot. The testbed was to have two functioning legs and be geometrically similar to the testbed already under development at Vanderbilt. The new Georgia Tech CRC Testbed remained very close in size to the CRC at Vanderbilt, with different actuators and valves. Range of motion was sacrificed slightly to employ prototype

actuators with embedded position sensing, and donated Festo proportional valves were used because of their availability, practicality, and reliability.

Significant amounts of time were spent analyzing SolidWorks models for mechanical interferences and optimizing the range of motion for each joint. Care was also taken while designing joints, selecting fittings, and designing rod ends for each actuator, ensuring the robot would be functional, compact, and easily maintainable.

1.3.2 Control and Interface

The control system for the robot was designed in Simulink and run real-time on an xPC Target computer with analog inputs and outputs. The overall control objective is to bilaterally teleoperate the robot through two PHANToM haptic devices. The robot feet positions are coordinated with the PHANToM endpoint positions. PHANToM inputs are transformed into joint angle commands which, in turn, are transformed into cylinder stroke length commands. The cylinder stroke lengths are position controlled by classical methods and newly developed non-contacting position sensors with added force control effort by pressure sensor feedback.

The overall control objective is to provide accurate, stable position tracking control of each leg in both free space and ground contact. Since the legs are haptically controlled by the operator, importance is placed on maintaining low tracking error in order to provide crisp

haptic feedback when an outside obstacle impedes leg motion and physically induces position error.

1.3.3 Gait Coordination

The objective of the gait planning portion of this project is to design a routine for commanding trajectories to the rear four legs of the robot when the operator is directly controlling the front pair. Since the operator can easily control two legs, and not six simultaneously, the gait coordinator must record the trajectories of the front legs, calculate the position of the trajectory in the global robot coordinates, and play the appropriate trajectory back through subsequent leg pairs to avoid known obstacles. This method of locomotion allows the operator to manually guide the robot through treacherous and unknown terrain without requiring simultaneous user control of all six legs simultaneously.

CHAPTER 2

RESEARCH BACKGROUND AND LITERATURE REVIEW

2.1 Pneumatic Control Research

In many respects, pneumatic actuators are excellent devices for producing smooth, reliable, and low-cost linear motion. Cylinders can be created in nearly any diameter to produce force for most applications. Powered by compressed air (or other gas), the flow rate of the fluid is controlled by valves. On-off control of pneumatic actuators is exceedingly easy, requiring only an inexpensive spool valve.

2.1.1 Servo Control

Precise position control of pneumatic actuators, however, is much more difficult to achieve. Two physical methods prevail in obtaining position control, pulse-width modulation of on-off solenoid valves, and proportional servo valves.

Pulse-width modulation (PWM) control of a hydraulic system was initially investigated by D. Boddy at Purdue University in 1966. The pneumatic system control and development was originally experimented in 1987 by T. Noritsugu [4], and later expanded upon van Varseveld and Bone in 1997 [5]. These systems provide fast, accurate, and inexpensive position control with precision comparable to that achieved through use of servo valves. In 1990 Kunt

and Singh at Ohio State University developed a linear time varying model for open loop PWM control of a pneumatic actuator [6]. This work was expanded in 2006 by Shen, Zhang, Barth, and Goldfarb at Vanderbilt University through development of a nonlinear model-based control structure [7]. These methods are novel in the respect that the solenoid valves employed are relatively inexpensive and very fast.

Proportional spool valves, however, are accurate, more traditional, and only one valve is required to regulate flow and direction into both chambers of the pneumatic cylinder. Position control techniques have been developed, tested and refined for myriad uses, from robotic legs to high precision positioning systems.

Both classical and modern control methods have been applied to pneumatic servomechanisms. While a simple PID controller may seem trivial, advances to the classical method have been put forth, such as incorporating differential pressure feedback into the control effort. Pressure feedback accompanying position feedback aids control, because through a flow control valve, pressure and flow rate (actuator velocity) are coupled. Wang, Pu, and Moore experimented with acceleration feedback rather than pressure feedback [8]. The main advantage was the lower sensor cost, where only one accelerometer is needed rather than two separate pressure sensors. They were able to use a velocity command feed-forward, null-offset compensation,

and acceleration damping feedback to supplement a PID controller, matching velocity trajectories well.

Chillari, Guccione and Muscato compared several control techniques applied to pneumatic actuators [9]. They compared PID control, fuzzy control, PID control with pressure feedback, Fuzzy control with pressure feedback, sliding mode control, and neuro-fuzzy control. Their results showed that fuzzy logic control yielded the best tracking and transient responses, but in the classical domain, the PID control with a gain scheduled differential pressure feedback performed better than the simple PID controller.

More advanced, modern control methods yield impressive position control results of pneumatic servo systems. Tanaka, Yamada, Shimizu, and Shibata developed an advanced method of multi-rate adaptive pole placement for pneumatic actuators [10]. Korondi and Gyeviki developed a robust sliding mode control for a pneumatic actuator [11]. They were able to achieve robust position control with only 3.8 mm steady state position error. Guvenc developed a discrete time model regulator using model inversion and PD control to achieve closed loop position control of a pneumatic actuator [12]. Energy saving techniques were implemented by Al-Dakkan, Barth, and Goldfarb using an additional proportional valve to re-route high pressure exhaust gasses back into the high-pressure chamber of the actuator [13]. They showed that using dynamic energy

constraints, energy savings of up to 45% could be achieved. This is a particularly poignant breakthrough with respect to a mobile, self powered rescue vehicle. Energy savings in a high-risk mission environment could be the difference between life and death for victims.

Recent research at Vanderbilt University, published by Goldfarb, Barth, Fite, Mitchell, Shields, Gogola, and Wehrmeyer provide control and implementation techniques for monopropellant based fluid power [1, 14-16]. The valve developed through their research is capable of controlling the flow of H_2O_2 decomposition gasses. These exhaust products essentially equate to a high quality steam. A 70% H_2O_2 solution exits the catalytic reactor as H_2O and O_2 at 450 degrees F, and 300 psi. These high temperatures and pressures exceed design constraints of any small, commercially available proportional valve.

Practical servo-pneumatic control is dependent on some level of actuator state feedback. Embedded position feedback has been traditionally difficult to integrate into fluid power actuators. Several commercial solutions exist to “piggyback” sensors onto cylinders, and the position feedback is accurate and reliable. A more novel solution is the capacitively-coupled resistance sensor developed by Zhu and Book from Georgia Institute of Technology [17]. This non-contacting displacement sensor can be compactly embedded in fluid power actuators yielding accurate position feedback through a small integrated sensor.

2.1.2 Robotic Applications

Pneumatic position control has been applied to robotic applications by several researchers. More specifically, legged robots have been controlled by complex control architectures allowing fluid gaits and upright walking. McKibben artificial muscles are usually extremely useful for emulating muscles with fluid power, but Muscato and Spampinato developed a five degree of freedom pneumatic leg with cylinders, capable of force interactions with the ground plane [18]. Their leg was controlled through a multi-level architecture and pre-programmed gaits.

Guihard, Gorce, and Fontaine developed a control architecture for a bipedal robot, SAPPHYR, designed to pull a wheeled cart [19]. This project demonstrated the leg to leg interactions coupled with adaptive pneumatic control and, again, pre-programmed gaits. They showed that pneumatic actuators make for effective leg actuators with the added advantage, for bipeds, that the compressible gas acts as a slight damper during foot contact. The compressibility and damping also causes a slight orientation shift as each foot sets down, a valuable insight.

The BIPMAN pneumatic bipedal platform, developed by Gorce, Vanel, and Guihard in France exhibits a very intricate control architecture [20-22]. Using supervisory controls based on biomechanical research, they control torso posture and orientation with the legs, just as a

human would. Dynamic impedance controllers control the force and stroke length of the leg actuators. BIPMAN is an impressive testbed, able to step over obstacles, incorporating biomechanical properties in its feet and joint structure.

In 1954, Denavit and Hartenberg developed a method for describing kinematics of serial links [23]. This method, using link and joint geometry to relate the tip position to the base, has become the basis for analyzing kinematics and dynamics of serial manipulators. Pieper in 1968 described the application of Denavit-Hartenberg parameters to the generalized serial robot [24]. They described the algorithm by which the end-effector position is described by the joint angles, and the inverse, in which the known end-effector position determines the possible joint angles of the manipulator.

2.2 Gait Research

Every insect walks with a certain gait. Gaits exhibited in nature are intuitive to the creature executing them, whether a bipedal gait performed by humans, or a hexapedal gait demonstrated as a stick insect moves nimbly over terrain (Figure 2.1).

Figure 2.1: Stick Insect *Carausius morosus*

The execution of hexapedal gaits in robots is commonly performed through central pattern generators or finite-state methods. In central pattern generated gaits, when the robot is commanded to walk forward, it simply plays its pre-planned forward walking gait, and the legs move the body forward. Finite-state planners execute pre-planned gaits based on the robot state. A certain gait can be planned for flat terrain, and another for stair climbing. Coordinated gaits are more autonomous gaits which control legs and body position with respect to a general high-level command (Figure 2.2).

Figure 2.2: Big Dog Military Robots

2.2.1 Coordinated Hexapedal Gaits

Cruse investigated the gait coordination and autonomy of the stick insect in 1996 [25]. He determined that the insect was kept stable by a tripod gait, which keeps at least three feet planted on the ground at all times. Coordinated gaits are neither pre-planned nor fixed. The *Carausius morosus* and *Obrimus asperrimus*, more commonly known as stick insects were the main foci of Cruse's analyses. He noted and analyzed leg trajectories and joint angles as the insects walked along varied surfaces. Cruse's later analysis [26] yielded WALKNET, an algorithm describing the autonomous gait of the stick insect. Through the simulated WALKNET routine, simple high-level commands such as "forward" can be used to automatically

move all legs in such a way that the body moves stably over smooth flat terrain.

Wait and Goldfarb expanded on the WALKNET routine in 2007 with research directly applicable to the Compact Rescue Crawler [3]. Their analysis, oriented to robot control, showed several drawbacks of the WALKNET routine, specifically its joint-space control rather than overall task space control. They modified WALKNET to maintain body height and ground contact, rather than joint angles, keeping feet in place should the footholds loosen or slip. They also added a yaw control feedback loop, controlling lateral stability and position during walking.

2.2.2 Other Gaits

Tripod gaits work well for hexapods moving under coordinated leg control, but other gaits exist with benefits and drawbacks. A centipede style gait isolates leg pairs and moves each in sequence. Torige, Noguchi, and Ishizawa showed how centipede leg movement acts as a wave based on the foot positions of previous segments [27]. Their robot tests of the centipede wave gait showed that distributed control architecture allowed for better leg control and the option to add more leg segments to the robot.

CHAPTER 3

TESTBED DESIGN

3.1 Leg Structure and Design

The two robot legs (Figure 3.1) developed through this project were constructed primarily from 6061 aluminum alloy. This strong, light metal was chosen due its high strength to weight ratio, and its ease of machinability.

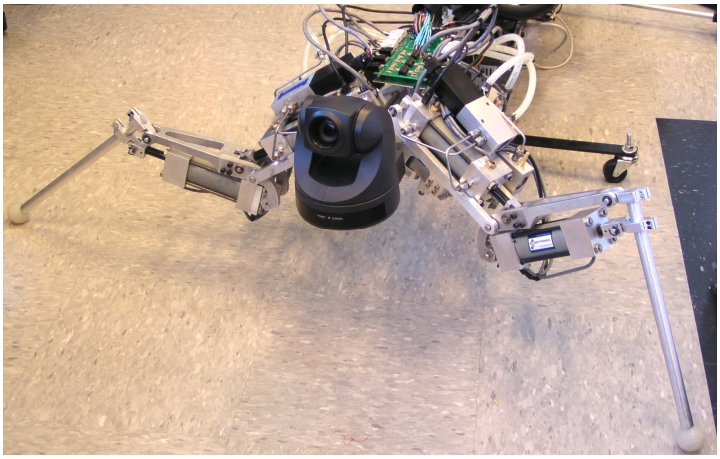

Figure 3.1: Two CRC Robot Legs

The support cart holds the rear of the robot, and physically emulates the support from the absent four rear legs. The support cart also acts as a mounting structure for the computers and power supply that drive the robot.

The main spine of the robot is a 48 in. beam of 80/20 1 in. square extrusion. The front and rear shoulder are fastened to the square extrusion via ¼-20 bolts received by

tee nuts. The square profile prevents the shoulder harnesses from rotating on the frame due to moments applied by the legs.

The shoulder harnesses were waterjet cut from 1 in. aluminum plate. The front shoulder harness provides a clevis mount for each shoulder. The pivot arm is constrained to 30 degrees below horizontal within the clevis by a 0.375 in. diameter steel pin. A needle bearing assembly is mounted inside each clevis arm, and a bronze thrust bearing is nested above and below the swing arm (Figure 3.2).

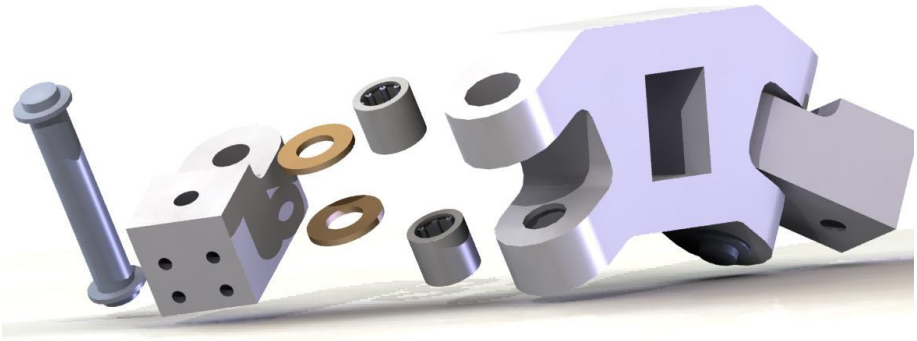

Figure 3.2: Front Shoulder Bearing Assembly

These four bearing surfaces on each side resist all moments applied by the leg on the shoulder harness. Due to the precision needle bearings, the assembly exhibits very little mechanical play.

The rear shoulder harness, mounted to the spine, is pinned to the rear of the swing cylinders, allowing them to

pivot as the leg is extended and retracted (Figure 3.3). The rear shoulder harness was waterjet cut from 1 in. aluminum plate.

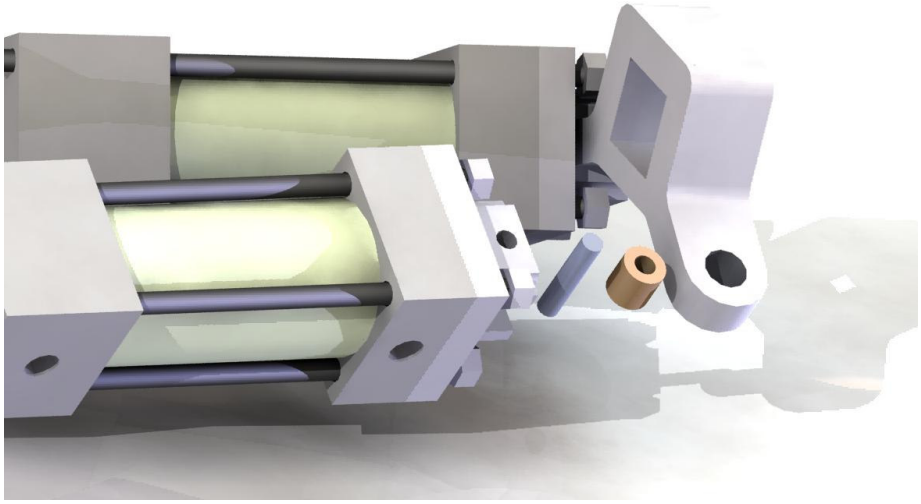

Figure 3.3: Rear Shoulder Assembly

The pivot arms, driven by swing cylinders L1 and R1, support the entire leg mechanisms. The pivot arms directly support the thrust cylinders L2 and R2, via the mid-leg arms. The mid-leg arms were waterjet cut from 0.375 in. aluminum plate. These arms support the rear of cylinders L2 and R2 and provide the pivot points for joints L2 and R2 (Figure 3.4).

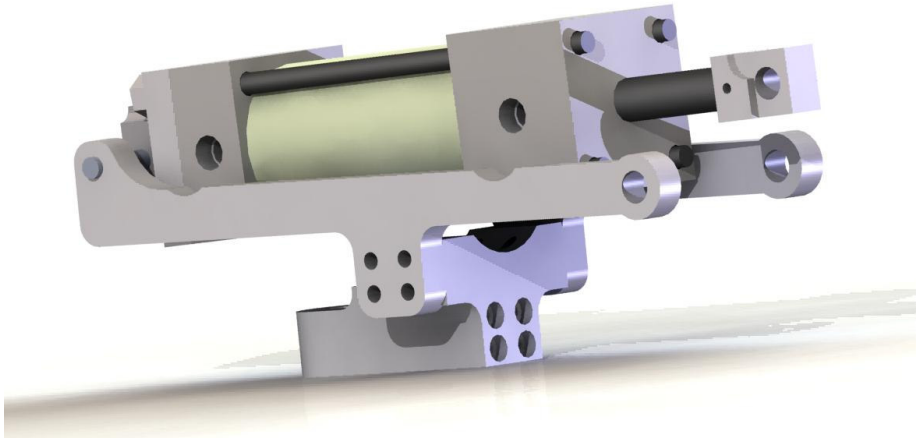

Figure 3.4: Mid-leg Assembly Cradling Cyl. L/R2

The lower-leg arms pivot on the mid-leg arms and cradle cylinders L3 and R3. These lower-leg arms are directly pinned to the rod ends of cylinders L2 and R2, and are responsible for supporting much of the robot weight. The main A-shaped piece of the lower-leg arm was waterjet cut from 1 in. aluminum plate, and the curved rear pieces were waterjet cut from 0.375 in. plate (Figure 3.5).

Figure 3.5: Lower Leg Assembly

The final link holding the foot and making ground contact is a 0.625 in. 12 in. long aluminum rod. This rod is held by two clamps allowing its length and range of motion to be adjusted (Figure 3.6). Length is adjusted simply by sliding the leg rod through the clamps and range of motion is adjusted by changing the distance between the clamps. A decrease in range of motion will allow the cylinder to apply more torque to the joint.

The actual foot of the last link is a silicon rubber ball. This rubber ball exhibited the best traction properties to the waxed tile floor in the laboratory test environment, and was therefore used throughout development. The ball is fastened to the end of the link with one ¼-20 screw.

Figure 3.6: Ground Contact Foot and Final Link

3.1.1 Actuator Design and Construction

Pneumatic actuators developed for this project were custom-made by Sentrinsic for this application. The cylinders feature integrated position and pressure sensors. Each NFPA standard tie-rod style cylinder is identical, save for the rod ends. The main barrel is a composite wound polymer tube with aluminum endcaps. The cylinder bore is 1.5 in. and stroke length is 1.4 in. A clevis plate joins the rear of the cylinder to its associated pivot pin and protects the internal circuitry (Figure 3.7).

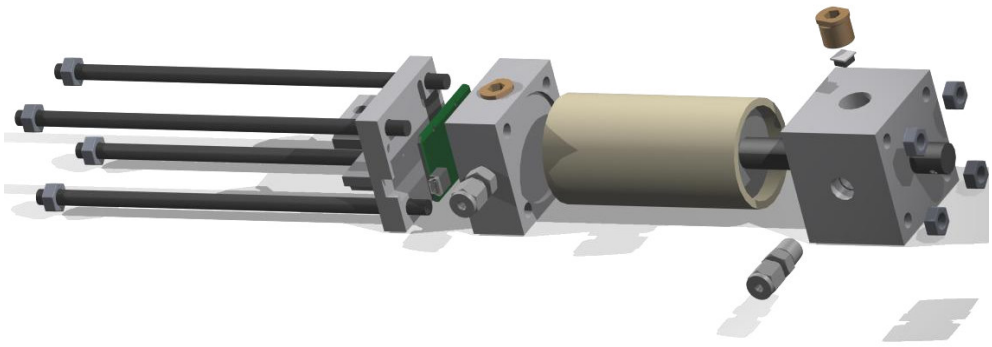

Figure 3.7: Cylinder Assembly

Four 0.25 in. tie rods clamp the clevis plate, endcaps, and barrel. Internal o-rings seal the junction between endcaps and barrel. The rod and piston use standard pneumatic lip seals. The piston rod is a 0.50 in. fiberglass rod fixed to the piston via a pin. The aluminum rod ends are pinned to the piston rod. The rod ends were milled from 6061 aluminum stock specific to each joint, with holes through which their connecting pins mount.

1/8 NPT threads were machined into one side each endcap for air fittings. 1/4 NPT threads were machined into one side of each endcap, 90 degrees from the air fittings for pressure sensors.

The fiberglass piston rods are ideal for this search and rescue application because they will not permanently deform from impacts. Steel piston rods, once bent, render the entire actuator useless. The light fiberglass rods will withstand impacts from debris without undergoing any permanent deformation. Forces large enough to destroy the thick fiberglass rods would surely ruin any similar steel piston rods.

The composite wound barrels, made by Polygon, can withstand much higher chamber pressures than could ever be provided through H_2O_2 decomposition. The composite material is also favorable for this application because it will resist denting. A dented steel barrel from debris impacts will significantly restrict piston motion, essentially crippling the robot.

Each pneumatic cylinder is referenced in this document by the joint it actuates, e.g. cylinder R1, cylinder L3. Each chamber is referenced *a* or *b* with respect to Figure 3.8 below.

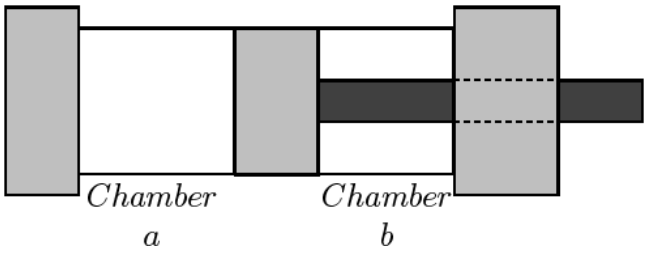

Figure 3.8: Cylinder Chamber Labeling Convention

3.2 Valves

Air flow rate into each chamber of each cylinder is controlled by a proportional directional spool valve. Air flow rate is proportional to spool position and direction from center. The FESTO MPYE-5-M5-010-B valves allow a maximum flow rate of 100 L/min [28].

The spool is held in it center position by two magnet springs. Each magnet is wrapped with a solenoid. At a 5V signal, the spool remains centered, and no air flows through the valve. As the signal increases, the spool moves proportionally as the current through the coils changes. At 0V or 10V, the spool orifice is completely open, allowing maximum air flow (Figure 3.9).

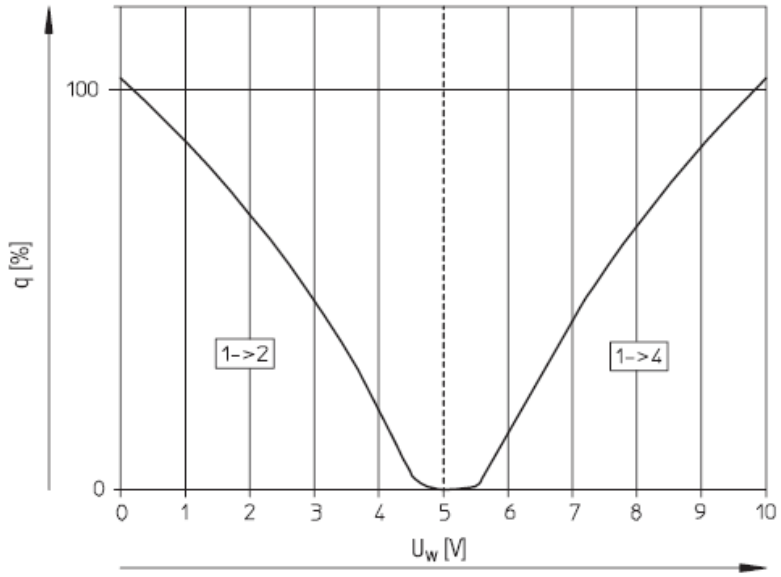

Figure 3.9: Flow Rate As Function of Setpoint Voltage U

The valves were piped on the robot to correspond flow direction to stroke direction. When a high signal (> 5V) is applied to the valve, air flow into the cylinder causes the actuator to extend. A low signal (< 5V) retracts the actuator.

3.2.1 Limitations

The Festo MPYE-5-M5-010-B valves can control flow up to 100 l/min through 5mm (10-32) fittings and provide spool position response up to 100 Hz [28]. Given that the cylinder bore is 1.5 inches and maximum stroke length is 1.4 inches, 100 L/min (101.7 in^3/s) translates into a maximum stroke speed of 57.56 in/s (Equation 3.1).

$$A_a = \frac{\pi d^2}{4} = 1.767in$$
$$\dot{x}_{\max} = \frac{\dot{V}_{\max}}{A_a} = \frac{101.7in^3/s}{1.767in^2} = 57.56in/s \quad (3.1)$$

57.56 in/s, or 4.8 ft/s is a stroke speed faster than the operator could command, and is likely outside the capabilities of the physical system.

Rather than analyzing a maximum stroke speed, the maximum frequency of operation is instead analyzed by calculating the maximum stroking frequency attainable with a 101.7 in^3/s gas flow rate (assuming no compression) (Equations 3.2 and 3.3).

$$V_{stroke} = V_a + V_b = A_a x_{\max} + A_b x_{\max}$$
$$V_{stroke} = 2.474in^3 + 2.199in^3 = 4.673in^3$$

(3.2-3)

$$freq_{stroke} = \frac{\dot{V}_{\max}}{V_{stroke}} = \frac{101.7in^3/s}{4.673in^3} = 21.8Hz$$

Each valve has the flow capacity to stroke a cylinder back and forth over 20 times per second. This poses absolutely no restrictions on design or control capabilities of the robot.

3.2.2 Plumbing

The main air supply is provided through a 0.25 in. Nylon 12 flexible tube connected to the main distribution manifold. Six flexible lines connect the manifold to each valve. The flexible lines designed for use with barbed fittings are braided Tygothane tubing, 0.125 in. ID, 0.375 in. OD, with a bend radius of only 0.5 in. 0.125 in. OD stainless steel tubing connects each valve to its cylinder on L1/R1 and L2/R2. Flexible tubing connects Valves L3/R3

to cylinders L3/R3. Such materials were chosen for their workability and pressure ratings. Each component must be able to withstand operating pressures of approximately 300 psi to conform to chemofluidic research ongoing at Vanderbilt University.

At cylinders L1/R1 and L2/R2, the valve is mounted directly to the cylinder. This close placement reduces the amount of compressibility exhibited by the metered air in the lines between valve and actuator. Valves L3/R3 are mounted close to valves L2/R2 and connected to cylinders L3/R3 by Tygothane tubing and 1/8 NPT barbed fittings. This positioning prevents the relatively heavy valve from adding to the load overhanging joint 2.

3.3 Sensors and Signals

Two types of sensors critical to pneumatic control were integrated into the CRC. Position sensors feedback cylinder stroke length to the controller, and pressure sensors feedback individual chamber pressures to the controller.

3.3.1 Position Sensors

Made by Sentrinsic and developed for the CRC project, the non-contacting Coupled-Capacitance Resistive Sensors (CCRS) measure piston position within the cylinders. Each cylinder houses its own circuit board, which outputs a 0-10V signal directly proportional to the distance from the piston to the bottom of the cylinder. This signal is

converted by the controller to stroke length of the rod end to the absolute minimum stroke length. The prototype cylinders used were designed specifically for the CRC and design flaws were rectified by improvements to the overall Sentrinsic design.

3.3.2 Pressure Sensors

Small absolute pressure sensors were integrated into the endcaps of each cylinder. These 250 psi MEMS sensors can withstand pressures up to three times the 250psi rating, and measure just 0.30 in. on each side. Sensors are model 1471-250AW made by Measurement Specialties for applications in medical devices and internal remote tire pressure measurement.

These sensors were installed on custom-made 1/4 NPT threaded plugs (Figure 3.10). The sensor housings were then sealed and installed into the aluminum cylinder endcaps.

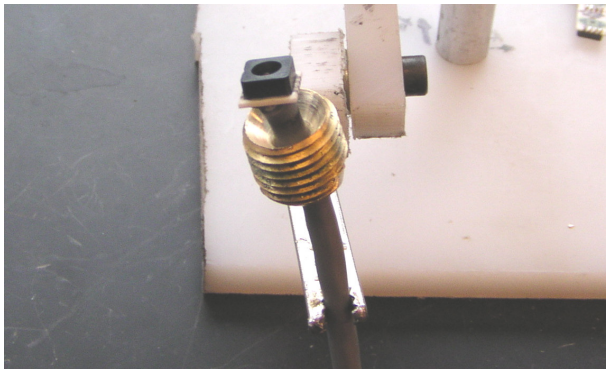

Figure 3.10: Pressure Sensor Assembly

Each sensor behaves like a strain gauge, measuring absolute pressure within each cylinder chamber. A 5V

potential is applied across the bridge, and the resultant output voltage is proportional to the absolute pressure applied. The output voltage is very small, rated 16 mV at the full 250 psi rating [Datasheet]. With a maximum full-scale voltage output span of 0.016V at 250psi, the linear scale of sensor output was determined as:

$$k_{sensor} = \frac{16mV}{250psi} = 0.064mV / psi \tag{3.4}$$

The projected sensor output at the maximum 300psi was estimated using this scale value k_{sensor}:

$$\Delta V_{press-\max} = 0.064mV / psi \cdot 300psi = 19.2mV \tag{3.5}$$

A safe value of 30mV was chosen for amplifier gain selection to allow for any variations in the sensor and any DC offsets. Since the maximum analog input value readable by the onboard A/D cards is 10V, the maximum pressure signal at 300 psi, when amplified, must remain below or at the 10V threshold. Therefore, the highest desirable gain was determined:

$$k_{op-amp} = \frac{V_{\max-amped}}{V_{\max-raw}} = \frac{10V}{0.030V} = 333.3 \tag{3.6}$$

To produce a readable signal, the sensor output voltages are each amplified through op-amps wired in a difference amplifier configuration (Figure 3.11).

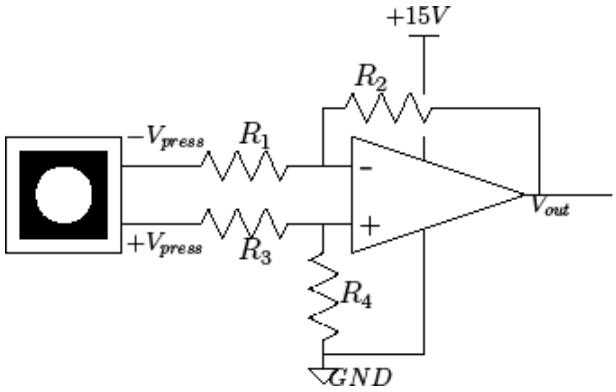

Figure 3.11: Difference Amplifier for Pressure Sensors

The op-amps are set to a gain of 340 V/V, with R_1 = R_3 = 1.5 kOhm, and R_2 = R_4 = 510 kOhm. This gain is calculated by applying the simple difference amplifier gain formula (Equation 3.7), where V_{press} is the voltage difference generated directly by the pressure sensor.

$$V_{out} = \frac{R_3}{R_1}(V_2 - V_1) = \frac{510k\Omega}{1.5k\Omega}(\Delta V_{press}) = 340 \cdot \Delta V_{press} \qquad (3.7)$$

Several factors played into the selection of 340V/V as the op-amp gain. Firstly, high impedance was desired to prevent any high currents from passing through the pressure sensors and op-amps. Secondly, the 510 kOhm and 1.5 kOhm resistors are very common, and large quantities were quickly obtained at the ME Electric Shop. Third, since the maximum analog input value readable by the onboard A/D cards is 10V, the maximum pressure signal at 300psi, when amplified, must remain below or at the 10V threshold. The high-rail op-amp voltage was not yet determined at the time of the design, but it was known to be 12-15 VDC since it

was to share the excitation voltage line with the position sensors.

3.3.3 Signal Routing Board

The op-amps that amplify the pressure sensor signals were integrated into a custom printed circuit board (PCB). This PCB routes all the control input signals to the valves from the main analog output wire harness, and routes all the feedback signals to the main analog input harnesses.

The two pressure sensor leads from each cylinder terminate in an eight-pin MOLEX 90142-0008 header plug. Each plug mates into a physically shielded, latched header mounted directly to the PCB. Of the eight wires to each cylinder, two carry a +5 VDC supply, two carry a ground connection, two carry a pressure signal potential from the sensor in the rod-side chamber p_b, and two carry a pressure signal potential from the base-side chamber p_a.

Each Texas Instruments LM3900N single-supply op-amp chip contains four independent amplifiers. The board was routed such that each chip amplifies four pressure sensors for two identical cylinders, i.e. R1a, R1b, L1a, and L1b. The high rail of the op-amps is a 15 VDC supply line shared with the position sensors. The pressure sensor amplifier output header consists of 12 shielded wires sending the high impedance signals directly to the analog inputs of the onboard controller.

The position sensor signals exit each cylinder through a standard mini-USB plug. Four wires are routed through

the shielded USB wire, +15 VDC, +6 VDC, 0-10 VDC signal, and ground. The six USB wires connect to six shielded header plugs at the rear of the board. The MOLEX 50-57-9404 latching header plugs are intuitively ordered and labeled to prevent crossed signals and installation errors. The PCB routes each position signal to a shielded six-wire header leaving the board through a harness and going directly to the analog inputs of the onboard controller. Each wire carries the positive signal value, and all sensors share a common ground.

Valve control inputs enter the signal routing board through a 12-wire header directly from the analog outputs of the onboard controller. The six control signals each consist of two wires carrying the +/- potential generated by the analog output card. From the header wires, each signal pair is routed to the sides of the board where they terminate in a 09-91-0400 MOLEX four-pin header. The valve control wires utilize four connections, +24 VDC, +Signal, -Signal, and ground from the signal board to each valve.

A six-wire power header connects the signal board to incoming voltage supply. The board uses +6 VDC, +15 VDC, and +24 VDC and a common ground for most components and sensors. The pressure sensor supply is an isolated +5VDC and ground connection. Without this isolated connection, the pressure sensors become coupled to the ground (low rail) of the op-amps, effectively bypassing the op-amps.

The signal routing board is mounted to a thin aluminum back plane via small screws and standoffs. The aluminum back plane is mounted to the spine of the CRC between the front and rear shoulder harnesses via two long standoffs. This placement centralizes the board, provides clearance for the shoulder swing, and allows for easy access and troubleshooting.

3.4 Denavit-Hartenberg Parameters

Each leg is modeled as a 3 degree of freedom serial robot. Using such a model, the joint angles can be related to the foot endpoint using Denavit-Hartenberg parameters (DH parameters) and forward and inverse displacement analyses [29].

3.4.1 Link Lengths and Joint Offsets

Due to the nature of the robot design, only three link lengths (a_1, a_2, a_3) and one joint offset (d_1) must be determined for accurate displacement analysis. Per DH practice, the robot was drawn and labeled as illustrated below in Figure 3.12.

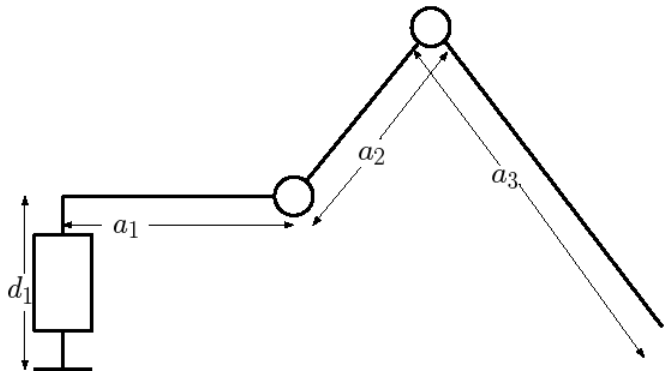

Figure 3.12: CRC Link Lengths and Joint Offset

Joint offset d_1 is the axial distance along Joint 1 from the base (shoulder) to the axis of Joint 2. Link length a_1 is the distance from Joint 1 to Joint 2 along Link 1. Link length a_2 is the distance from Joint 2 to Joint 3 along Link 2. Link length a_3 is the distance from Joint 3 to the endpoint of Link 3.

Link lengths and joint offsets were measured accurately using the SolidWorks model of the leg as described below. Since all links were fabricated directly from these drawings, they are considered accurate representations of the physical robot (units are inches).

Joint offset d_1 was measured from the midpoint of the front shoulder clevis to Joint 2 along the shoulder axis (Joint 1) (Figure 3.13). Offset d_1 = 1.608 inches.

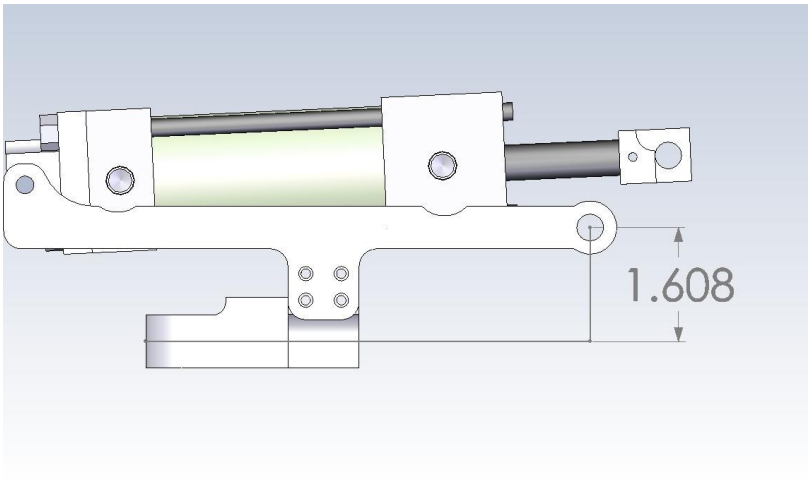

Figure 3.13: Measurement of Joint Offset d_1

Link length a_1 was measured from the center of the shoulder pin to the center of the pin of Joint 2. The link length is measured perpendicular to the joint offset d_1 as shown below in Figure 3.14. Link length a_1 = 5.75 inches.

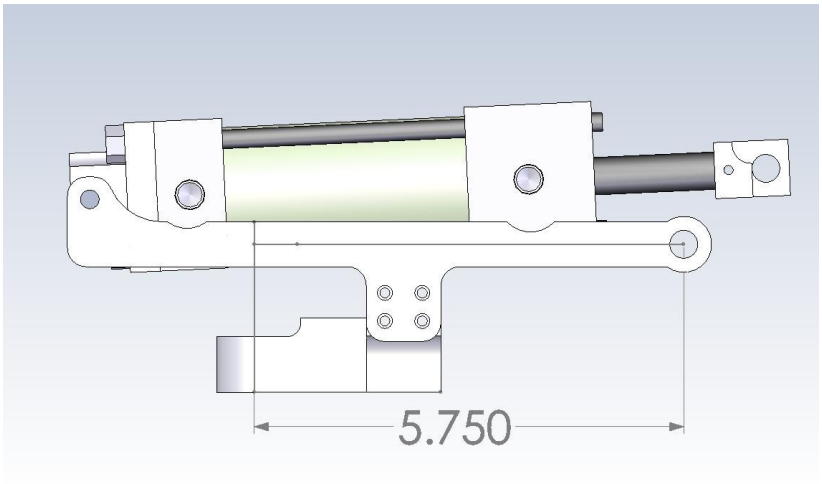

Figure 3.14: Measurement of Link Length a_1

Link length a_2 was determined by measuring the straight-line spacing between Joint 2 and Joint 3 as shown below in Figure 3.15. Link length a_2 = 6.828 inches.

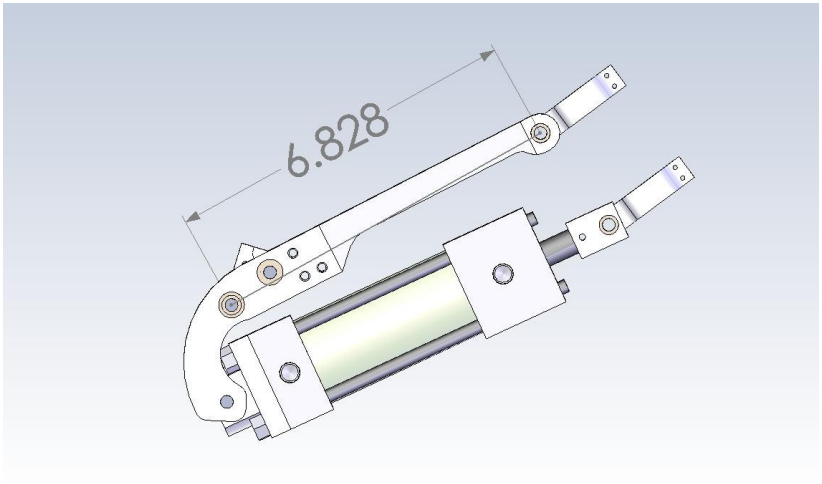

Figure 3.15: Measurement of Link Length a_2

The final link length a_3 was measured from the Joint 3 axis to the end of the manipulator (foot). The measurement is shown below in Figure 3.16 where a_3 = 12 inches.

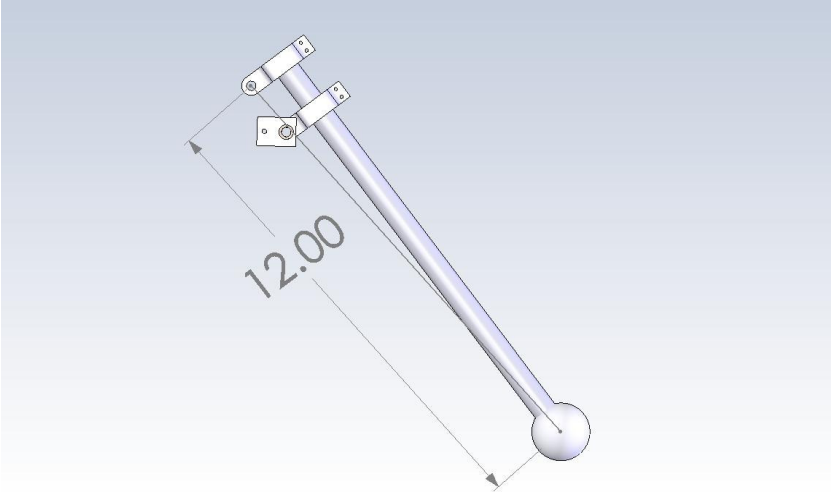

Figure 3.16: Measurement of Link Length a_3

3.4.2 Joint Notation

Serial joints on each leg are denoted in a manner consistent with Denavit-Hartenberg conventions (Figure 3.17). Starting at the base of the serial manipulator, the first shoulder joint is Joint 1, the second is Joint 2, and the final joint is Joint 3.

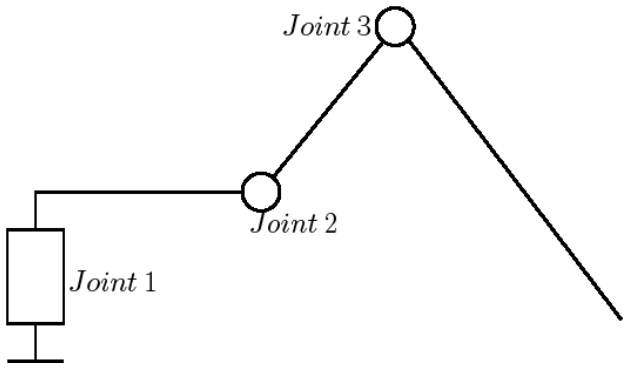

Figure 3.17: Joints of Serial Manipulator

In reference, each joint is preceded by a letter denoting the leg to which it belongs. The right leg consists of Joints R1, R2, and R3, and the left leg consists of Joints L1, L2, and L3 (Figure 3.18).

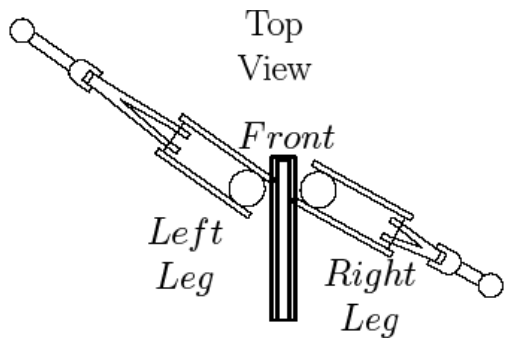

Figure 3.18: Leg Side Naming Convention

3.4.3 Origins and Coordinates

Each link on the serial manipulator uses its own coordinate system and axes as per standard DH convention. The base of the manipulator is origin O_0. Each successive origin is placed on a joint axis with the z coordinate along the joint axis and the x coordinate along the link length (Figure 3.19).

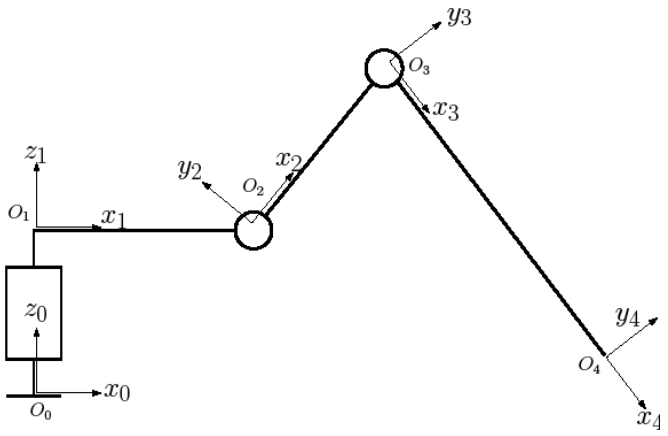

Figure 3.19: DH Origins and Coordinates

3.4.4 Joint Angle Convention

As per standard DH convention, each joint angle is measured about the joint axis, *z*, at each origin. The angle θ_i is measured in a positive direction at the *i*th joint from x_{i-1} to x_i. In this manner, each joint angle is standardized and measurable in its particular coordinate system regardless of the joint angles on other links (Figure 3.20).

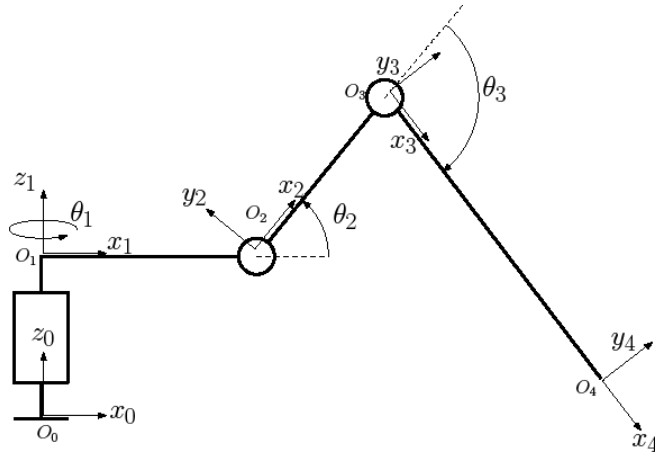

Figure 3.20: DH Angle Conventions

3.5 Force and Torque Analysis

The force generated by a pneumatic actuator is converted to joint torque by a fixed lever length from the rod end to the joint pin. Each joint must be capable of applying enough torque to enable the robot to complete its mission. Joints R1 and L1 must be able to either pull or push the robot on flat terrain and up and down obstacles. Joints R2 and L2 must be able to supply torque enough to counter the weight of the robot and lift the body from the ground. Joints R3 and L3 must provide stabilizing lateral forces through the feet.

Joint torques, in this particular system, are dependent on the direction in which the actuator is applying force. Since the rod area of one chamber decreases the available pressure area, the pull stroke is weaker than the push stroke:

$$F_a = p_a A_a = p_a \left(\frac{\pi d_{piston}^2}{4} \right) = 1.767 in^2 \left(p_a \right)$$
$$F_b = p_b A_b = p_b \left(\frac{\pi \left(d_{piston}^2 - d_{rod}^2 \right)}{4} \right) = 1.571 in^2 \left(p_b \right) \quad (3.8)$$

With a maximum supply pressure p_s of 300 psi, the maximum push force is calculated as

$$F_a = p_s A_a = 300 psi \cdot 1.767 in^2 = 530.1 lbf \quad (3.9)$$

and the maximum pull force is calculated as

$$F_b = p_s A_b = 300 psi \cdot 1.571 in^2 = 471.2 lbf \quad (3.10)$$

3.5.1 Joints R1 and L1

Actuators R1 and L1 must provide the torque to Joints R1 and L1 to physically pull the robot forward during a stance phase. The basic joint geometry is shown below (Figure 3.21).

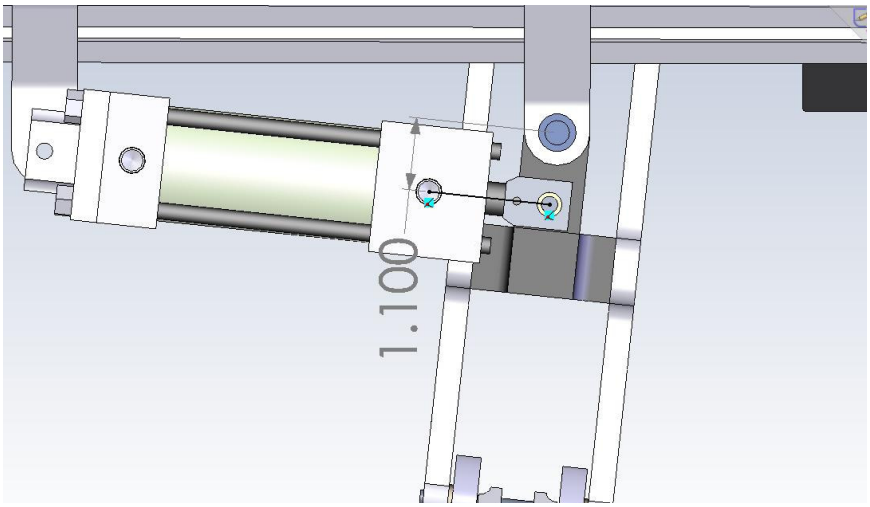

Figure 3.21: Max Moment Arm Joint 1 (Bottom View)

When Link 1 is perpendicular to Cylinder 1, the maximum torque is applied to Joint 1 (Equation 3.11).

$$\tau_{1\max} = F_1 r_1 = \left(p_a A_a - p_b A_b \right) r_1 \qquad (3.11)$$

The maximum joint torque will occur when pressure in chamber a p_a is maximum, zero pressure in chamber b p_b and when the moment arm r_1 is at its maximum of 1.100 inches.

$$\tau_{1\max} = \left(p_a A_a \right) r_1 = \left(300\,psi \cdot 1.767 in^2 \right) 1.100 in = 583.1 in \cdot lbf \qquad (3.12)$$

The worst possible case for Joint 1 torque occurs when the leg is swung fully forward and Cylinder 1 is pulling at

the shortest moment arm during the stance phase (Figure 3.22).

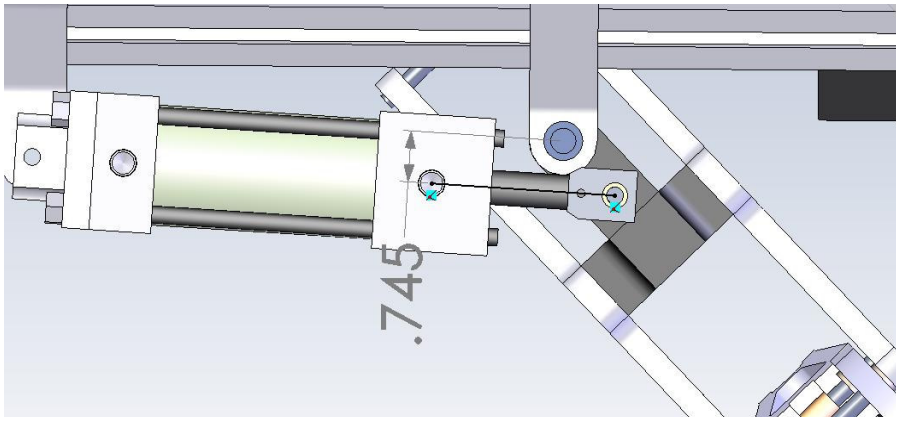

Figure 3.22: Shortest Moment Arm on Joint 1 (Bottom)

In this common case, the maximum torque applied to the manipulator by Joint 1 is

$$\tau_{1\max} = \left(p_a A_a\right) r_1 = \left(300\,psi \cdot 1.571 in^2\right) 0.745 in = 351.1 in \cdot lbf \quad (3.13)$$

The 351.1 in-lbf applied by Joint 1 in this pulling scenario must exceed the torque required to overcome gravity while the robot is climbing.

The actual pulling force applied at the foot of the robot is calculated by measuring the distance from the foot to Joint 1. Figure 3.23 shows the worst case, in which the foot is at its furthest point from Joint 1, creating the largest moment arm.

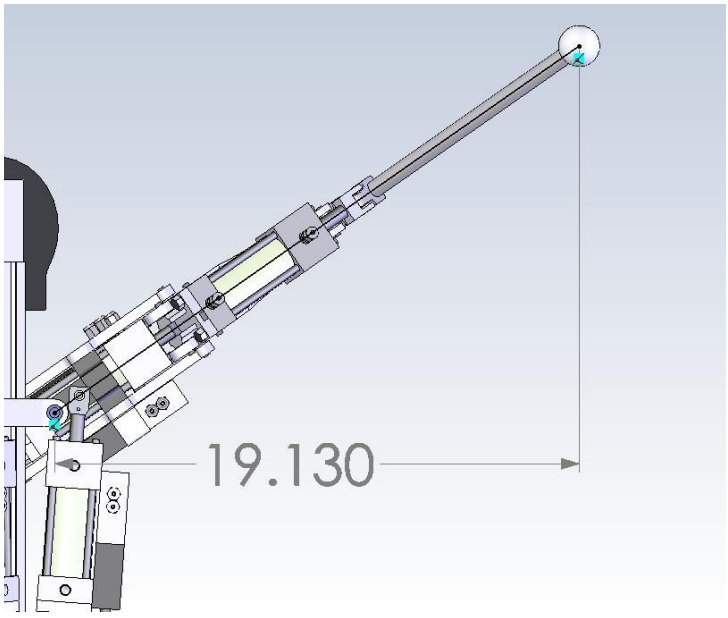

Figure 3.23: Maximum Foot Extension (Bottom)

The maximum forward pulling force Joint 1 can generate in this configuration, assuming zero foot slippage, is

$$F_{pull} = \frac{\tau_{\max}}{r_{foot}} = \frac{351.1in \cdot lbf}{19.130in} = 18.35lbf \qquad (3.14)$$

18.35 lbf maximum pulling force, per leg, at the weakest leg configuration is more than sufficient for most conceivable mission parameters in which this robot may find itself.

3.5.2 Joints R2 and L2

Joint 2 of each leg experiences the most extreme cases of torque demand. During a stance phase, Joint 2 provides most of the torque required to suspend the entire weight of the robot, and must be able to do so for all foot positions. During swing phases, Joint 2 supports the overhanging load of links 2 and 3, providing torque in the

opposite direction. In addition to the load disparity between swing and stance phases, the moment arm by which Cyl. 2 applies torque changes greatly through the range of Joint 2 angles (Figure 3.24). 11.2 degrees is the angular offset of the cylinder rod end from the true link ray. This offset is necessary to provide the cylinder rod clearance over the actual joint pin at full extension. The moment arm calculation (Equation 3.15) uses the distance between the joint pin and the rod end pin, 0.975 in. and the angular offset. Note that in this configuration, the joint angle will always be negative.

$$a = 0.975in \cdot \sin\left(180° + \left(\theta_2 - 11.2°\right)\right) \tag{3.15}$$

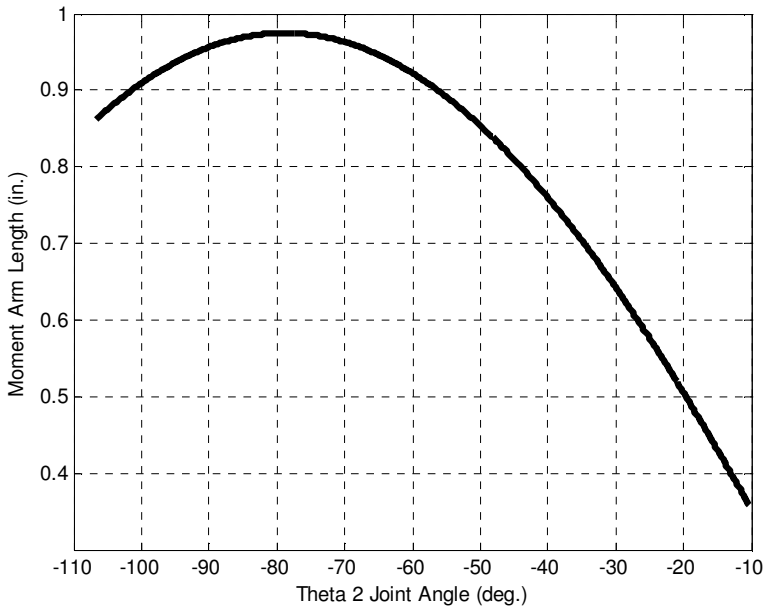

Figure 3.24: Moment Arm Length vs. Joint 2 Angle

To produce sufficient joint torque, the force applied by Cylinder 2 through the moment arm must be adequate to

lift the weight of the robot at the lowest mechanical advantage. The lowest mechanical advantage is experienced at a joint angle of -10.3 degrees, or full extension of Cylinder 2 (Figure 3.25).

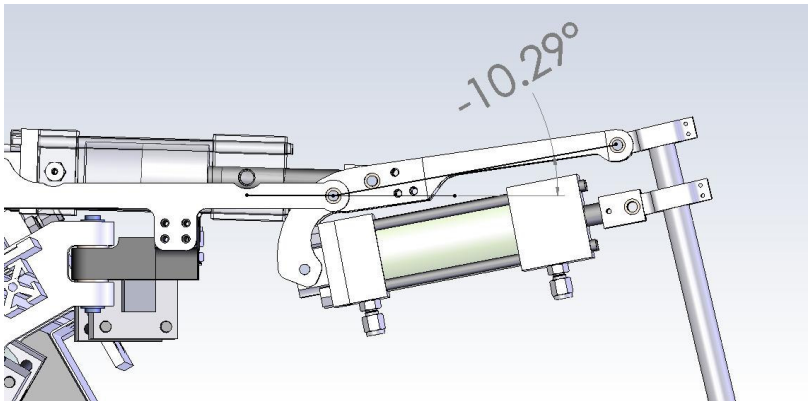

Figure 3.25: Cylinder 2 at Full Extension/Lowest Mechanical Advantage

Through Equations 3.16 and 3.17, the moment arm r_{min} and available joint torque τ_{2max} at this extreme configuration are calculated based on the joint geometry.

$$r_{\min} = 0.975in \cdot \sin\left(180° + \left(-10.3° - 11.2°\right)\right) = 0.357in \qquad (3.16)$$

$$\tau_{2\max} = p_a A_a r_{\min} = 300psi \cdot 1.767in^2 \cdot 0.357in = 189.2in \cdot lbf \qquad (3.17)$$

The torque required to lift the robot depends on the distance from the foot to the spine and the moment arm created. The largest moment is produced when Cylinder 3 is fully extended and the foot is splayed outward (Figure 3.26).

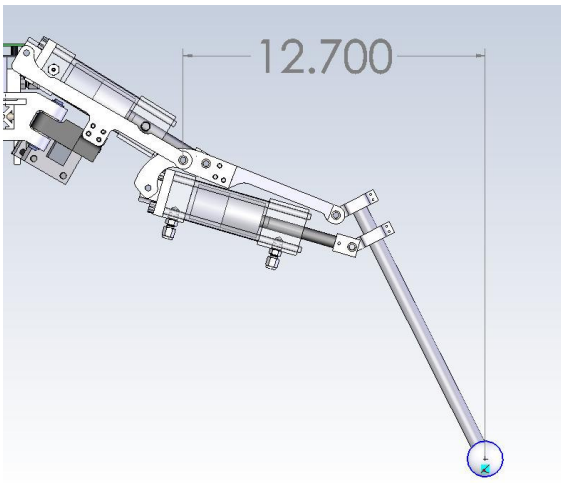

Figure 3.26: Maximum Moment Arm About Joint 2

Applying maximum torque against the maximum moment arm of 12.70 inches, the maximum lifting force per leg in this extreme case is evaluated in Equation 3.18.

$$F_{lift} = \frac{\tau_{\max}}{a_{\max}} = \frac{189.2in \cdot lbf}{12.70in} = 14.9lbf \qquad (3.18)$$

Nearly 15 lbf of lifting force per leg at the configuration with the least mechanical advantage is well within range of required forces for effective operation of the robot, assuming the robot weighs less than 90 lbf.

The most mechanical advantage occurs when the joint angle is at -78.8 degrees (-90 + 11.2 degrees) and the actuator applies force to the full 0.975 inch moment arm about Joint 2 (Figure 3.27).

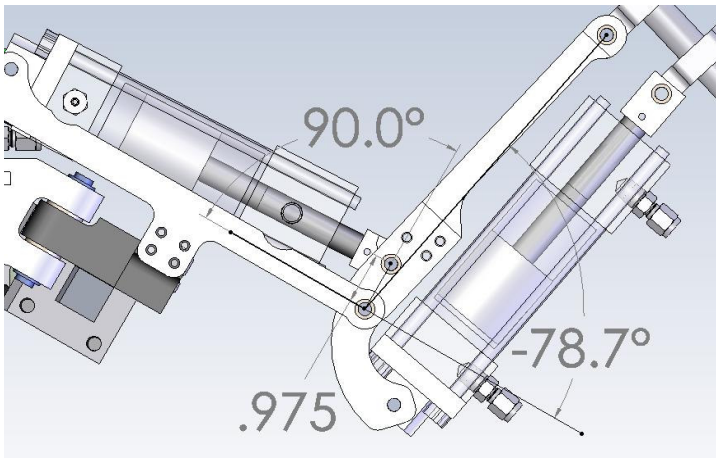

Figure 3.27: Joint 2 Largest Moment Arm

The maximum joint torque generated at Joint 2 is therefore 516.8 in-lbf (Equation 3.19).

$$\tau_{2\max} = p_a A_a r_{\max} = 300 psi \cdot 1.767 in^2 \cdot 0.975 in = 516.8 in \cdot lbf \qquad (3.19)$$

The maximum lifting force per leg is evaluated while Joint 2 is at -78.7 degrees and Cylinder 3 is fully retracted, producing the smallest possible moment arm about Joint 2 at the highest force configuration (Figure 3.28).

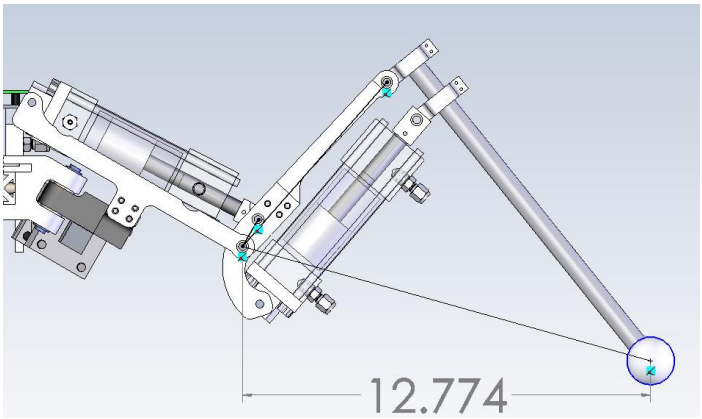

Figure 3.28: Shortest Moment Arm About Joint 2 at Highest Force Configuration

The moment arm about Joint 2, 12.77 inches, combined with the highest possible joint torque, each leg can produce a respectable downward lifting force at the foot of 40.5 lbf (Equation 3.20).

$$F_{lift} = \frac{\tau_{2\max}}{r_{\min}} = \frac{516.8in \cdot lbf}{12.77in} = 40.5lbf \qquad (3.20)$$

Multiplied by six legs, this highest possible lifting force will allow the robot to thrust its body upwards with almost 250 lbf of force. Such high forces could be useful for lifting fallen objects off a pinned victim or carrying extra tools and fuel into a mission.

3.5.3 Joints R3 and L3

Joint 3 on each leg produces torque to provide lateral stability and thrust relative to the robot spine. The distance between Joint 3 and the point at which the actuator applies force is variable. By loosening the leg rod clamps and sliding the clamps along the leg rod, the moment arm can be adjusted. For this project, the moment arm was adjusted to bring Cylinder 3 as close to Link 3 as possible without inducing mechanical interferences. This configuration reduces the amount of available lateral force at the foot, but provides the best range of motion for Link 3. The overall length of Link 3 is also adjustable by sliding the leg rod up through both rod clamps. Figure 3.29 shows the configuration used throughout this project and experiments.

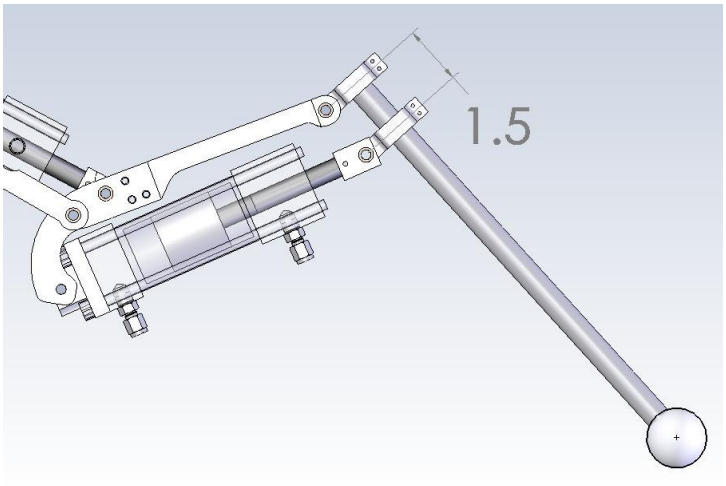

Figure 3.29: Link 3 Configuration

As with Joint 1, Joint 3 is at its weakest configuration when Cylinder 3 is fully extended and pulling, Joint 3 at 38.8 degrees. The moment arm is shortest in this scenario at 1.10 inches (Figure 3.30).

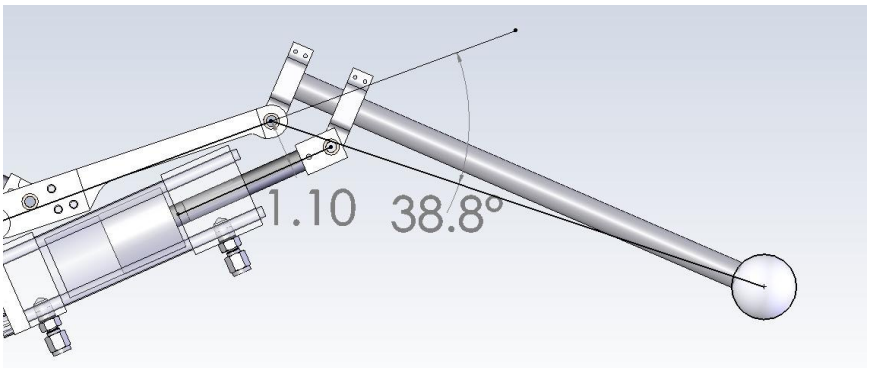

Figure 3.30: Shortest Moment Arm Aout Joint 3

The maximum torque available at Joint 3 in this configuration is 518.4 in-lbf (Equation 3.21)

$$\tau_{3\max} = p_b A_b r_3 = 300\,psi \cdot 1.571 in^2 \cdot 1.10 in = 518.4 in \cdot lbf \qquad (3.21)$$

The largest moment about Joint 3 applied by Link 3 is experienced when Joint 3 is perpendicular to the direction of the lateral force, employing the full length of Link 3 as the moment arm (Figure 3.31).

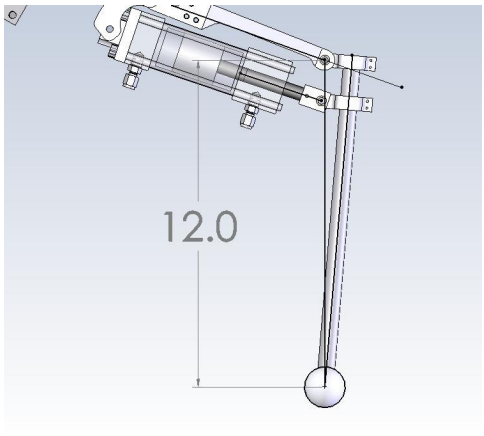

Figure 3.31: Largest Moment Arm about Joint 3

The 12.0 inch moment arm about Joint 3 will, in the most extreme case, apply 43.2 lbf of lateral pulling force (Equation 3.22).

$$F_{pull} = \frac{\tau_{3\max}}{r_{3\max}} = \frac{518.4in \cdot lbf}{12.0in} = 43.2lbf \qquad (3.22)$$

This amount of lateral force application in the most extreme case should be more than sufficient to stabilize, maneuver and manipulate the environment.

CHAPTER 4

TRANSFORMATIONS

Total system control is attained by individually controlling stroke lengths of actuators by sending control input signals to the proportional pneumatic spool valves. Input signals are generated from transforming operator hand motions into three-dimensional command vectors which are transformed into stroke length commands.

Each control time-step, the actual leg position is calculated and compared to the commanded position. The error between the two is displayed to the operator as a haptic force in the direction of the position error.

4.1 System Layout

The robot control system consists of three computers networked together via User Datagram Protocol (UDP). The three computers are an onboard PC104+ form-factor computer, a MATLAB host computer, and a computer at the operator workstation receiving PHANToM input commands and sending the data over UDP (Figure 4.1).

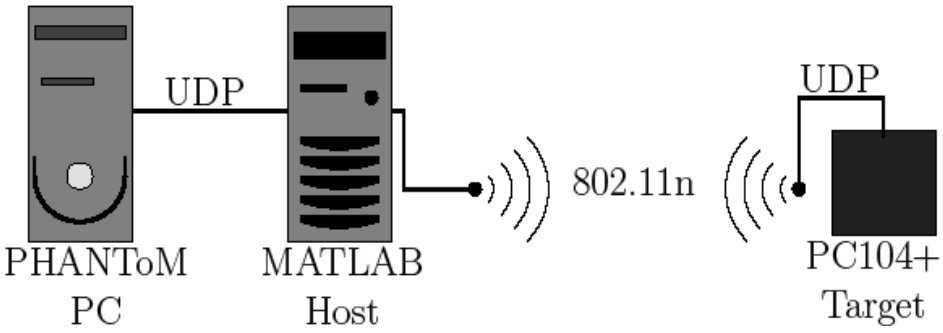

Figure 4.1: Computer Network

4.1.1 PHANToM PC

The PHANToM PC is a standard Dell Workstation that runs only Windows XP and C++ code for PHANToM control (Appendix C). PHANToM inputs are sent via UDP to the MATLAB host PC. PHANToM position values are also returned to the PHANToM PC via UDP from calculations performed on the robot. The C++ code running on the PHANToM PC calculates the required force to display to the user and sends the data to the PHANToMs.

The PHANToM PC also records input trajectories for storage, smoothing and later use with the gait coordinator.

4.1.2 MATLAB Host PC

The MATLAB Host PC acts as both a server and a high-end workstation. Simulink control diagrams are compiled and linked on the MATLAB Host and uploaded to the onboard xPC Target computer. The Host PC runs a high-end dual core Intel CPU, and has 2 GB RAM for rapid compilation of large Simulink control diagrams.

The MATLAB Host PC acts as a local server by forwarding the UDP packets to and from the PHANToM PC to the onboard PC104+ target PC. The packets are sent through a pair of Netgear WNHDE111 802.11n wireless bridges. The wireless bridges allow the robot to be untethered from the server and code uploader.

4.1.3 Target PC

The Target PC is a small, low-power computer housed onboard the robot itself. This computer is a PC104+ form-factor stack of 3 boards housed inside an aluminum box mounted to the robot spine. The Target PC is used solely for running the real-time controller compiled and uploaded by the Host PC. The real-time controller runs directly on the Target PC CPU at 1ms time-steps (1 kHz).

The three boards of the Target PC are a main CPU module, an analog to digital card (ADC), and a digital to analog card (DAC). Each board fits the standard PC104 standard dimensions of 4.6 in. x 3.8 in. The cards stack together via an 8-bit ISA bus header, a 16-bit ISA bus header, and a 32-bit PCI bus header. The three busses allow for interoperability between manufacturers and assembly standards.

The CPU board is an Arbor Computing Em104P-i8523 module with an Intel Celeron 600MHz CPU and a 512MB SO-DIMM RAM chip. The onboard Ethernet chipset is Intel 82562ET, which is compatible with the Simulink xPC Target upload protocol.

The ADC card, under the CPU board, is a Diamond Systems DMM-32X-AT card. The DMM-32X-AT reads up to 32 single-ended analog inputs, or 16 differential inputs. The card is configured to base address 0x300 and reads single-ended inputs from 0-10V.

The DAC card, under the ADC at the bottom of the stack, is a Diamond Systems RMM-1612-XT card. The RMM-1612-XT outputs up to 16 12-bit analog signals at 0-10V. The card is configured to a base address of 0x280.

4.2 Control Input Transformation

The operator input to the CRC system is a three-dimensional vector generated by each PHANToM haptic device. The vector, in input task space (x,y,z) is converted to joint space (θ_1, θ_2, θ_3) by an inverse displacement algorithm evaluating joint angles each time-step.

4.2.1 Input Task Space to Robot Space

The operator physically commands foot position of each leg through two PHANToM haptic devices. Each PHANToM has three degrees of freedom and sends data out in the form of a 3 dimensional position vector each time-step (1 ms). Each PHANToM output is a vector in millimeters from the PHANToM origin (set arbitrarily when the device is initialized) to the device endpoint. The coordinates of the input space to which the vector is referenced are shown below in Figure 4.2. Looking at the front of the PHANToM device, x is to the right, y is up, and z is inward.

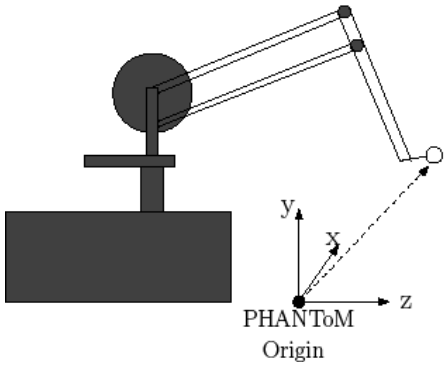

Figure 4.2: PHANToM Coordinates

The input vector is scaled, orthogonally transformed and then rotated by 30 degrees to match the downward angle of the shoulders on the robot.

Using standard Denavit-Hartenberg coordinates for the base of the leg, the PHANToM input vector coordinates must be orthogonally transformed to properly correspond to the leg coordinates (Figure 4.3).

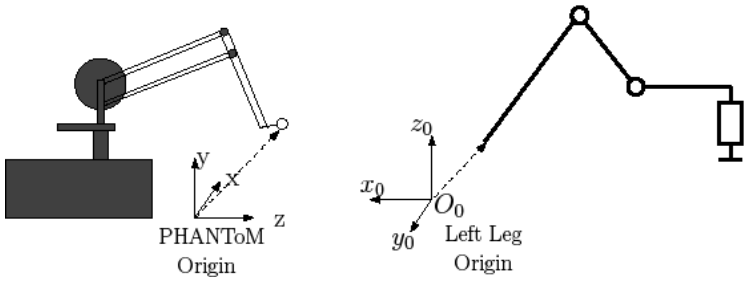

Figure 4.3: Left PHANToM Input and Leg Coordinates

The coordinate transform matrix is applied to the PHANToM input vector p_{input} to transform it into workspace coordinates p_{left} (Equation 4.1).

$$\begin{bmatrix} x_{left} \\ y_{left} \\ z_{left} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{input} \\ y_{input} \\ z_{input} \end{bmatrix} \tag{4.1}$$

The input vector in the leg workspace p_{left} is then rotated +30 degrees about the y-axis to properly match the downward leg angle yielding the rotated vector p_{rot} (Equation 4.2), (Figure 4.4).

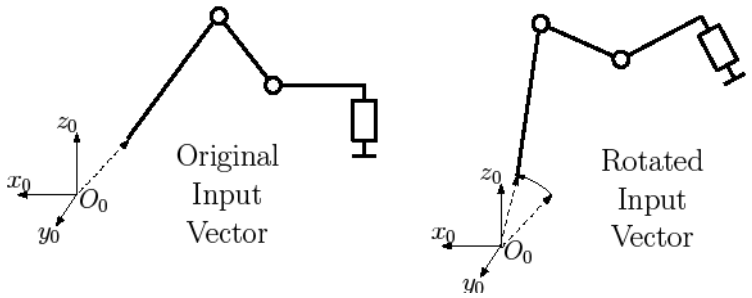

Figure 4.4: Rotated Input Vector

$$\begin{bmatrix} x_{rot} \\ y_{rot} \\ z_{rot} \end{bmatrix} = \begin{bmatrix} \cos 30 & 0 & -\sin 30 \\ 0 & 1 & 0 \\ \sin 30 & 0 & \cos 30 \end{bmatrix} \begin{bmatrix} x_{left} \\ y_{left} \\ z_{left} \end{bmatrix} \tag{4.2}$$

Finally, the new transformed input vector p_{rot} is converted from mm to inches and scaled up by a factor of 2 so that the physical PHANToM input workspace will encompass all areas within the leg workspace (Equation 4.3).

$${}_0p_{com} = \begin{bmatrix} {}_0x_{com} \\ {}_0y_{com} \\ {}_0z_{com} \end{bmatrix} = 2\left(\frac{1in}{25.4mm}\right) \begin{bmatrix} x_{rot} \\ y_{rot} \\ z_{rot} \end{bmatrix} \tag{4.3}$$

The same procedure is performed on the right leg to transform the PHANToM input vector into a usable vector in the leg workspace. First the PHANToM input vector coordinates are orthogonally transformed to match the coordinates of the leg workspace (Figure 4.5), (Equation 4.4).

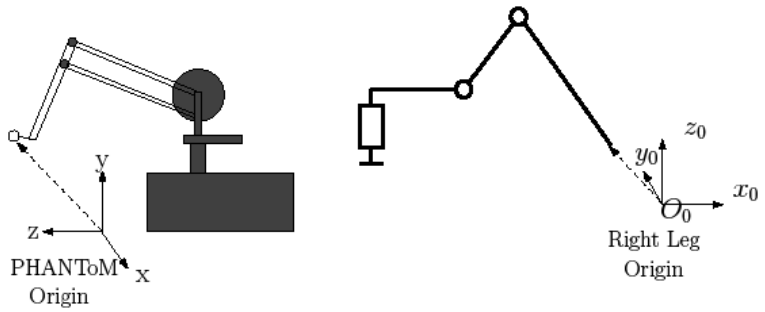

Figure 4.5: Right PHANToM Input and Leg Coordinates

$$\begin{bmatrix} x_{right} \\ y_{right} \\ z_{right} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{input} \\ y_{input} \\ z_{input} \end{bmatrix} \quad (4.4)$$

The +30 degree rotation about the y-axis and scaling for the right leg is identical to the procedure performed on the left.

The complete matrix $A_{phan-leg}$ transforming the PHANToM input vector to the one which is used for joint angle evaluation is shown below in Equation 4.5.

$$A_{phan-leg} = \left(\frac{2}{25.4}\right) \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \cos 30 & 0 & -\sin 30 \\ 0 & 1 & 0 \\ \sin 30 & 0 & \cos 30 \end{bmatrix} = \begin{bmatrix} -0.039 & 0 & -0.068 \\ -0.068 & 0 & 0.039 \\ 0 & 0.079 & 0 \end{bmatrix} \quad (4.5)$$

4.2.2 Leg Space to Joint Space

The transformed and rotated input vector ${}_0p_{com}$ relates commanded foot position from an arbitrary origin within the leg workspace. Joint angle calculation, however, requires a commanded position vector from the robot base, or the base origin of the D-H model ${}_0p_{04}$. Since the origin in the leg workspace is arbitrary, it is set at a point where the PHANToM workspace is able to reach every point in the leg workspace without reaching a physical motion limit.

Tests revealed that a satisfactory origin *O* placement from the leg origin O_0 (base) is 25 inches along the x-axis and -10 inches along the z axis, creating vector ${}_0p_{offset}$ (Figure 4.6).

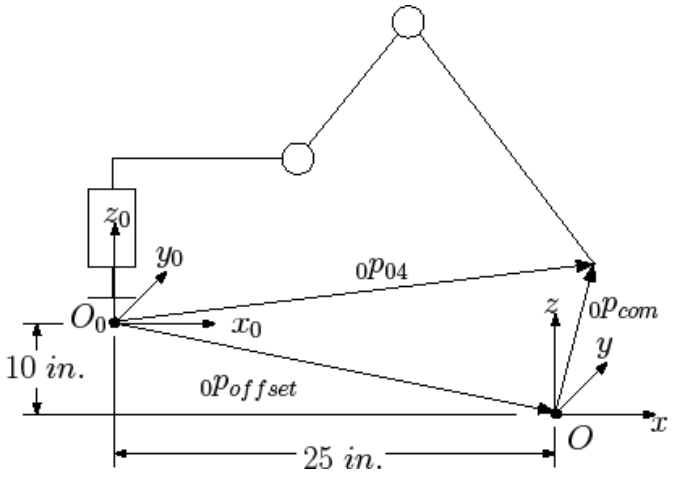

Figure 4.6: Origin *o* Placement Relative to Leg Base

Knowing the commanded foot position ${}_0p_{com}$ and the origin offset vector ${}_0p_{offset}$, the vector from the base to the foot ${}_0p_{04}$ can be found by simple vector addition (Equation 4.6).

$$ {}_0p_{04com} = {}_0p_{com} + {}_0p_{offset} \qquad (4.6) $$

The vector ${}_0p_{04}$ is used for directly evaluating the joint angles necessary to achieve the desired endpoint position. With this 3 degree of freedom serial manipulator, 4 solutions emerge from the inverse displacement algorithm. Two solutions emerge for Joint 1, and two solutions emerge for the evaluation of Joint 3 from each solution of Joint 1.

The inverse displacement analysis herein is based on the generalized method of analyzing the first 3 joints of a Puma robot [29] since both the Puma and the CRC legs have very similar joint structures.

Only one set of angle solutions is a possible configuration for this robot, alleviating the need to solve for multiple joint solutions simultaneously.

The inverse displacement algorithm functions are drawn in a Simulink diagram by their orders of operations through which each solution is computed. The Simulink diagrams are located in Appendix B. Each 1 ms time-step, the joint angle solutions are updated based on the new control input received via UDP from the PHANToM controllers.

Two of these algorithms run simultaneously, one for each leg. Since both legs are identical and have identical D-H parameters and coordinates, both algorithms are identical, and denoted by Joint 1, Joint 2, Joint 3 rather than the leg specific notation R1, R2, R3, etc.

Linear Trigonometric Equations are solved throughout the inverse displacement algorithm. The solution method taken from [29] is found in Appendix A.

4.2.2.1 Joint 1

The shoulder pivot angle of Joint 1 is solved from the endpoint vector ${}_0p_{04}$ first by using the vector ${}_1p_{14}$, from O_1 to O_4 at the endpoint in coordinate frame 1 (Equation 4.7).

$$\begin{bmatrix} ({}_1p_{14})_x \\ ({}_1p_{14})_y \\ ({}_1p_{14})_z \end{bmatrix} = \begin{bmatrix} ({}_0p_{04})_x \cos\theta_1 + ({}_0p_{04})_y \sin\theta_1 \\ ({}_0p_{04})_y \cos\theta_1 - ({}_0p_{04})_x \sin\theta_1 \\ ({}_0p_{04})_z - d_1 \end{bmatrix}$$
$$\begin{bmatrix} ({}_1p_{14})_x \\ ({}_1p_{14})_y \\ ({}_1p_{14})_z \end{bmatrix} = \begin{bmatrix} a_1 + a_2\cos\theta_2 + a_3(\cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3) \\ d_2 + d_3 \\ -a_2\sin\theta_2 - a_3(\cos\theta_2\sin\theta_3 - \sin\theta_2\cos\theta_3) \end{bmatrix} \quad (4.7)$$

Since the middle term, $({}_1p_{14})_y$ contains only the variable for Joint 1 as a linear trigonometric equation and ${}_0p_{04}$, this can be easily rearranged and solved for θ_1 (Equation 4.8).

$$({}_0p_{04})_y \cos\theta_1 - ({}_0p_{04})_x \sin\theta_1 - d_2 - d_3 = 0 \quad (4.8)$$

The linear trigonometric equation solution yields two sine and cosine pairs (Equation 4.9).

$$\cos\theta^{(\pm)} = \frac{ad \mp b\sqrt{a^2+b^2-d^2}}{a^2+b^2}$$
$$\sin\theta^{(\mp)} = \frac{bd \pm b\sqrt{a^2+b^2-d^2}}{a^2+b^2} \quad (4.9)$$

Where a, b, and d are the sine and cosine coefficients from Equation 4.8 (Equation 4.10).

$$
\begin{aligned}
a &= \left({}_0p_{04}\right)_y \\
b &= -\left({}_0p_{04}\right)_y \\
d &= d_2 + d_3 = 0
\end{aligned}
\tag{4.10}
$$

Rewriting and simplifying Equation 4.9, the sine and cosine pairs are solved for the positive solution, corresponding to the first angle solution for Joint 1:

$$
\begin{aligned}
\cos\theta_1^{(+)} &= \frac{-b}{\sqrt{a^2+b^2}} = \frac{\left({}_0p_{04}\right)_x}{\sqrt{\left({}_0p_{04}\right)_y^2 + \left({}_0p_{04}\right)_x^2}} \\
\sin\theta_1^{(+)} &= \frac{a}{\sqrt{a^2+b^2}} = \frac{\left({}_0p_{04}\right)_y}{\sqrt{\left({}_0p_{04}\right)_y^2 + \left(-{}_0p_{04}\right)_x^2}}
\end{aligned}
\tag{4.11}
$$

The ATAN2 function of MATLAB is used with the sine and cosine solutions to θ_1. This finds the correct angle corresponding to the sine and cosine values while considering signs of both (Equation 4.12).

$$
\theta_1^+ = ATAN2\left(\sin\theta_1^+, \cos\theta_1^+\right) \tag{4.12}
$$

In the linear trigonometric equation, the d term is zero, this implies that the robot is in a displacement singularity at Joint 1. This scenario would occur when the wrist, or in this case, foot, passes over or under the axis of Joint 1. This event physically cannot occur on this particular serial robot.

Because of this serial link configuration and its absence of joint offsets d_2 and d_3, the evaluation of Joint 1 can be greatly simplified (Figure 4.7).

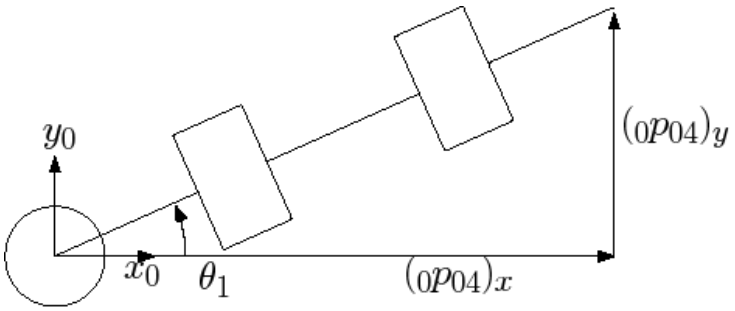

Figure 4.7: Top View of Joint 1, d_2 and d_3 = 0

With this configuration, θ_1 can be simply expressed as the inverse tangent of the *x* and *y* components of the input vector $_0p_{04}$ (Equation 4.13).

$$\theta_1 = ATAN2\left(\left({}_0p_{04}\right)_y, \left({}_0p_{04}\right)_x\right) \qquad (4.13)$$

4.2.2.2 Joint 3

Once a single solution for Joint 1 has been evaluated, the angle is used in the evaluation of Joint 3. Since only one solution from Joint 1 is chosen, (the other is a physically impossible configuration), only two possible solutions for Joint 3 emerge. Only one solution for Joint 3 will be chosen.

Initially, to solve for θ_3, the vector $_1p_{14}$ must be found from the input vector $_0p_{04}$. Vector $_1p_{14}$ is the vector from O_1 to O_4 as seen from the reference frame of the coordinates of O_1 (Figure 4.8).

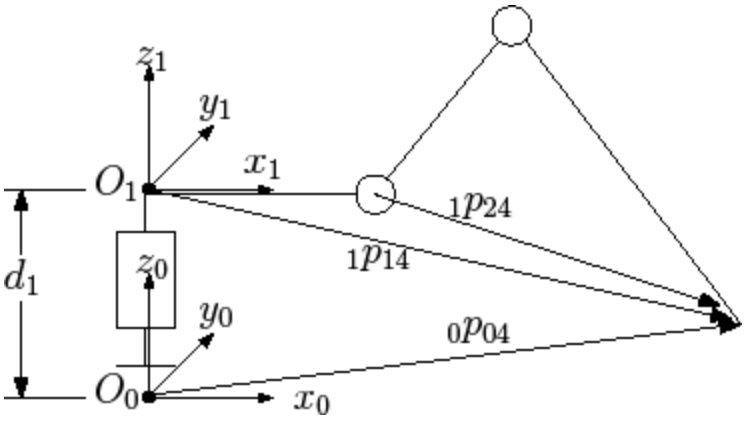

Figure 4.8: Vectors Needed for Joint 3 Evaluation

Vector $_1p_{14}$ can be evaluated directly from Equation 4.7 above:

$$\begin{bmatrix} (_1p_{14})_x \\ (_1p_{14})_y \\ (_1p_{14})_z \end{bmatrix} = \begin{bmatrix} (_0p_{04})_x \cos\theta_1 + (_0p_{04})_y \sin\theta_1 \\ (_0p_{04})_y \cos\theta_1 - (_0p_{04})_x \sin\theta_1 \\ (_0p_{04})_z - d_1 \end{bmatrix} \tag{4.14}$$

The vector $_1p_{12}$ is expressed in terms of the D-H values:

$$_1p_{12} = a_1x_1 + d_2y_2 = a_1x_1 \tag{4.15}$$

Vector addition shows that

$$_1p_{14} = {}_1p_{12} + {}_1p_{24} \tag{4.16}$$

Rewriting 4.16, $_1p_{24}$ can be expressed as

$$\begin{bmatrix} (_1p_{24})_x \\ (_1p_{24})_y \\ (_1p_{24})_z \end{bmatrix} = \begin{bmatrix} (_1p_{14})_x \\ (_1p_{14})_y \\ (_1p_{14})_z \end{bmatrix} - \begin{bmatrix} (_1p_{12})_x \\ (_1p_{12})_y \\ (_1p_{12})_z \end{bmatrix} = \begin{bmatrix} (_1p_{14})_x - a_1 \\ (_1p_{14})_y \\ (_1p_{14})_z \end{bmatrix} = \begin{bmatrix} (_1p_{14})_x - 5.75in \\ (_1p_{14})_y \\ (_1p_{14})_z \end{bmatrix} \tag{4.17}$$

Substituting $_1p_{14}$ from Equation 4.7 into Equation 4.17 yields

$$\begin{bmatrix} ({}_1p_{24})_x \\ ({}_1p_{24})_y \\ ({}_1p_{24})_z \end{bmatrix} = \begin{bmatrix} a_1 + a_2\cos\theta_2 + a_3(\cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3) - a_1 \\ d_2 + d_3 \\ -a_2\cos\theta_2 - a_3(\cos\theta_2\sin\theta_3 + \sin\theta_2\cos\theta_3) \end{bmatrix}$$

$$\begin{bmatrix} ({}_1p_{24})_x \\ ({}_1p_{24})_y \\ ({}_1p_{24})_z \end{bmatrix} = \begin{bmatrix} a_2\cos\theta_2 + a_3(\cos\theta_2\cos\theta_3 - \sin\theta_2\sin\theta_3) \\ 0 \\ -a_2\cos\theta_2 - a_3(\cos\theta_2\sin\theta_3 + \sin\theta_2\cos\theta_3) \end{bmatrix} \quad (4.18)$$

The unknown θ_2 term can be eliminated by squaring and summing the x and z components of ${}_1p_{24}$ (Equation 4.19).

$$2a_2a_3\cos\theta_3 + a_2^2 + a_3^2 - ({}_1p_{24})_x^2 - ({}_1p_{24})_z^2 = 0 \quad (4.19)$$

Equation 4.19 is another linear trigonometric equation with coefficients

$$\begin{aligned} a &= 2a_2a_3 \\ b &= 0 \\ d &= \left(a_2^2 + a_3^2 - ({}_1p_{24})_x^2 - ({}_1p_{24})_z^2\right) \end{aligned} \quad (4.20)$$

Again, this will yield two sine-cosine pairs from which two angle solutions emerge. Results have shown that the second solution is the one which yields an achievable joint angle command. The coefficients from Equation 4.20 above are then substituted into Equation 4.9 for evaluation and simplified:

$$\begin{aligned} \cos\theta_3^{\pm} &= \frac{ad \mp 0}{a^2} = \frac{d}{a} \\ \sin\theta_3^{\pm} &= \frac{0 \pm a\sqrt{a^2 - d^2}}{a^2} = \frac{\pm\sqrt{a^2 - d^2}}{a} \end{aligned} \quad (4.21)$$

$$\cos\theta_3^- = \frac{\left(190.62in^2 - \left({}_1p_{24}\right)_x^2 - \left({}_1p_{24}\right)_z^2\right)}{163.87in^2}$$

$$\sin\theta_3^- = \frac{-\sqrt{\left(163.87in^2\right)^2 - \left(190.62in^2 - \left({}_1p_{24}\right)_x^2 - \left({}_1p_{24}\right)_z^2\right)^2}}{163.87in^2} \quad (4.22)$$

As with the Joint 1 solution, the two sine-cosine values are input into the ATAN2 MATLAB function to yield one corresponding joint angle (Equation 4.23).

$$\theta_3 = ATAN2\left(\sin\theta_3^-, \cos\theta_3^-\right) \quad (4.23)$$

4.2.2.3 Joint 2

Once joint angles θ_1 and θ_3 have been evaluated from the PHANToM input vector, Joint 2 is solved based on the solutions for Joint 1 and Joint 3.

First, the *x* and *z* components of Equation 4.18 are rearranged into matrix form, pulling out the unknown θ_2 term:

$$\begin{bmatrix} \left({}_1p_{24}\right)_x \\ \left({}_1p_{24}\right)_z \end{bmatrix} = \begin{bmatrix} a_2 + a_3\cos\theta_3 & -a_3\sin\theta_3 \\ -a_2 - a_3\sin\theta_3 & -a_3\cos\theta_3 \end{bmatrix} \begin{bmatrix} \cos\theta_2 \\ \sin\theta_2 \end{bmatrix} \quad (4.24)$$

Solving for the sine and cosine pair yields two equations:

$$\cos\theta_2 = \frac{-(a_2 + a_3\cos\theta_3)({}_1p_{24})_x + (a_3\sin\theta_3)({}_1p_{24})_z}{-(2a_2a_3\cos\theta_3 + a_2^2 + a_3^2)}$$

$$\sin\theta_2 = \frac{(a_2 + a_3\cos\theta_3)({}_1p_{24})_z + (a_3\sin\theta_3)({}_1p_{24})_x}{-(2a_2a_3\cos\theta_3 + a_2^2 + a_3^2)}$$

$$\cos\theta_2 = \frac{-(6.828 + 12\cos\theta_3)({}_1p_{24})_x + (12\sin\theta_3)({}_1p_{24})_z}{-(163.87\cos\theta_3 + 190.62)} \qquad (4.25)$$

$$\sin\theta_2 = \frac{(6.828 + 12\cos\theta_3)({}_1p_{24})_z + (12\sin\theta_3)({}_1p_{24})_x}{-(163.87\cos\theta_3 + 190.62)}$$

The sine-cosine pair is again input into an ATAN2 MATLAB function to yield one value for theta$_2$ (Equation 4.26).

$$\theta_2 = ATAN2(\sin\theta_2, \cos\theta_2) \qquad (4.26)$$

4.2.3 Joint Space to Cylinder Space

Once joint angles are calculated, the information is converted to a directly controllable physical parameter, cylinder stroke length. With each individual joint angle command related to only one actuator, only one conversion per joint angle command is made each time-step.

A cosine law function is used for determining the required cylinder stroke length necessary to achieve the commanded joint angle (Figure 4.9), (Equation 4.27).

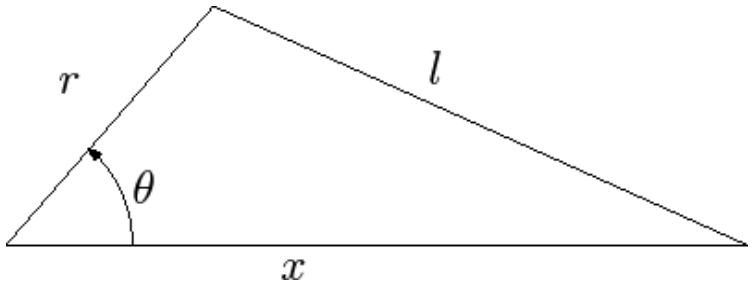

Figure 4.9: Law of Cosines Configuration

$$l^2 = r^2 + x^2 - 2rx\cos\theta \tag{4.27}$$

With this cosine caw configuration, *l* is the total length of the cylinder and stroke, *x* is the distance from the cylinder base to the joint pin, and *r* is the distance from rod end pin to the joint pin.

Cylinders L1 and R1 are described separately because as L1 retracts, the joint angle increases positively, and as R1 retracts, the joint angle grows negatively (Figure 4.10).

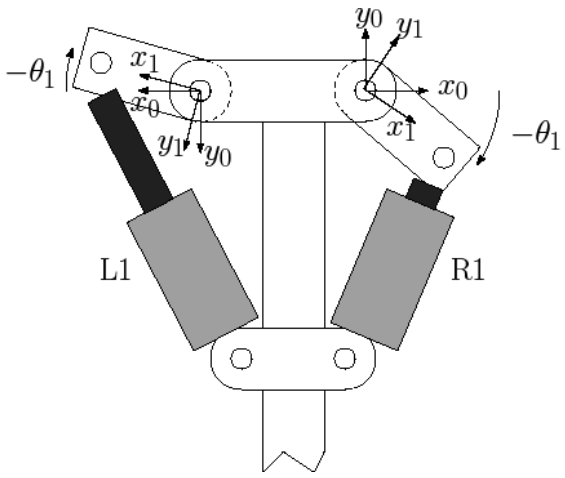

Figure 4.10: Top View of Joints R1 and L1 Coordinates and Angle Directions

The maximum stroke length of the Sentrinsic pneumatic cylinders made for the CRC is 1.4 inches.

4.2.3.1 Cylinder L1

Cylinder L1 moves Joint 1 from approximately -45 degrees to +45 degrees. To apply the Law of Cosines to the joint geometry, the static cylinder length is subtracted from the dimension *l* to isolate the exact stroke length (Figure 4.11).

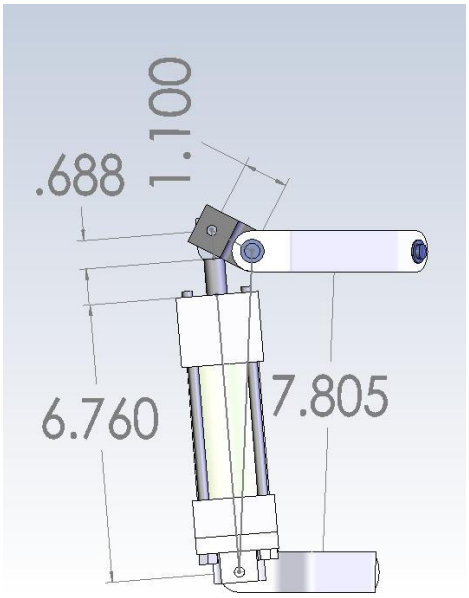

Figure 4.11: Joint L1 Link Geometry

The Cosine Law equation is used for evaluating the interior angle opposite the cylinder while the commanded joint angle is complementary (Figure 4.12).

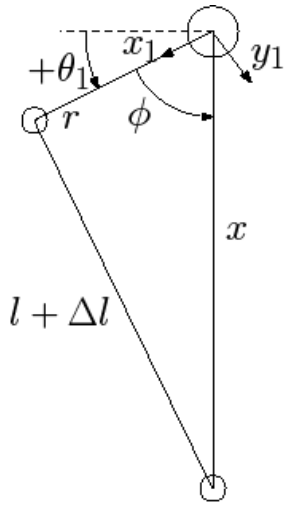

Figure 4.12: Joint L1 Cosine Law Geometry

Solving for the stroke length *Δl*, the cosine law equation for Joint 1 is written as:

$$\begin{aligned}
&\left(l+\Delta l_{L1}\right)^2 = r^2 + x^2 + rx\cos\phi \\
&\left(7.45in+\Delta l_{L1}\right)^2 = 1.21in^2 + 60.92in^2 + 8.59in^2 \cdot \cos\left(90°-\theta_{L1}\right) \\
&\Delta l_{L1} = \frac{\sqrt{62.13in^2 + 8.59in^2 \cdot \cos\left(90°-\theta_{L1}\right)}}{7.45in}
\end{aligned} \quad (4.28)$$

4.2.3.2 Cylinder R1

The stroke length for Cylinder R1 is calculated from the commanded joint angle θ_1 similarly to L1, but the joint angle geometry is opposite (Figure 4.13).

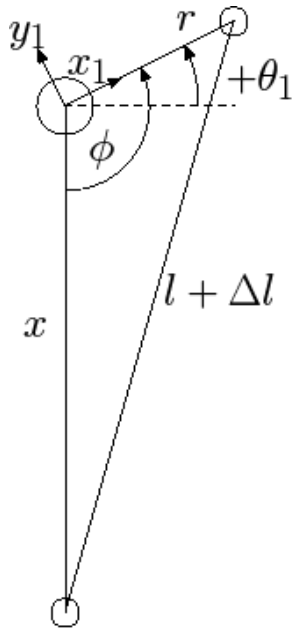

Figure 4.13: Joint R1 Cosine Law Geometry

Aside from the opposite coordinates relative to Joint L1, the physical dimensions of the parts are identical. θ_1 is calculated as a partial angle of the interior angle Φ (Equation 4.29).

$$\begin{aligned} \left(l + \Delta l_{R1}\right)^2 &= r^2 + x^2 + rx\cos\phi \\ \left(7.45in + \Delta l_{R1}\right)^2 &= 1.21in^2 + 60.92in^2 + 8.59in^2 \cdot \cos\left(90° + \theta_{R1}\right) \\ \Delta l_{R1} &= \frac{\sqrt{62.13in^2 + 8.59in^2 \cdot \cos\left(90° + \theta_{R1}\right)}}{7.45in} \end{aligned} \quad (4.29)$$

4.3.2.3 Cylinder 2

Joint 2 on each leg has the same coordinate system whereas when the actuator retracts, the joint angle grows negatively, and as it extends, the joint angle increases positively. The geometry of the joint-actuator triangles requires careful analysis of the mechanism to isolate the

interior angle Φ (needed for stroke length calculation) from the commanded joint angle θ_2 (Figure 4.14).

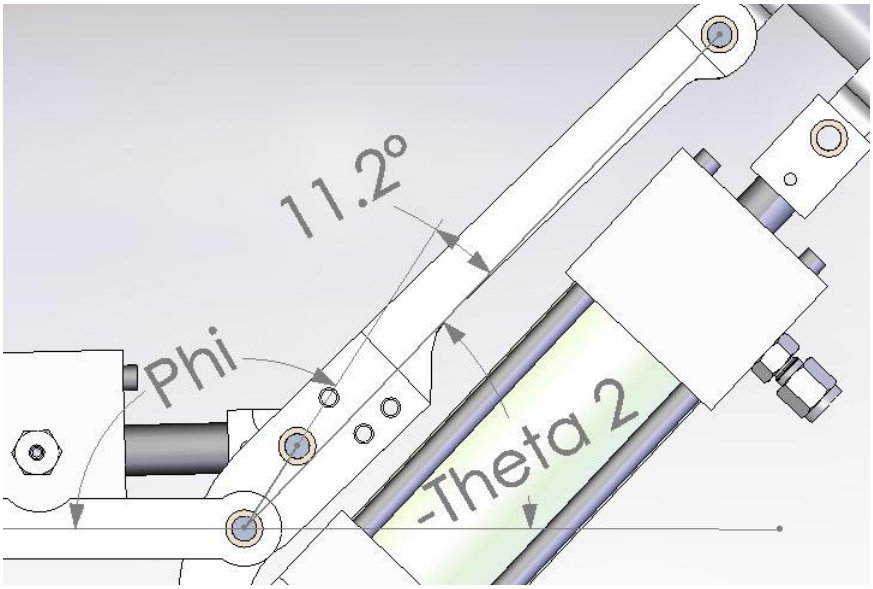

Figure 4.14: Joint 2 Angle Relationships

While Φ is the angle needed to calculate stroke length, only θ_2 is known. The 11.2 degree static offset, mentioned in 3.5.2 completes the three-angle supplement (Equation 4.30).

$$\phi = 180° - 11.2° + \theta_2 = 168.8° + \theta_2 \qquad (4.30)$$

The physical dimensions of the joint geometry are used in the same manner as for Joints L1 and R1 (Figure 4.15).

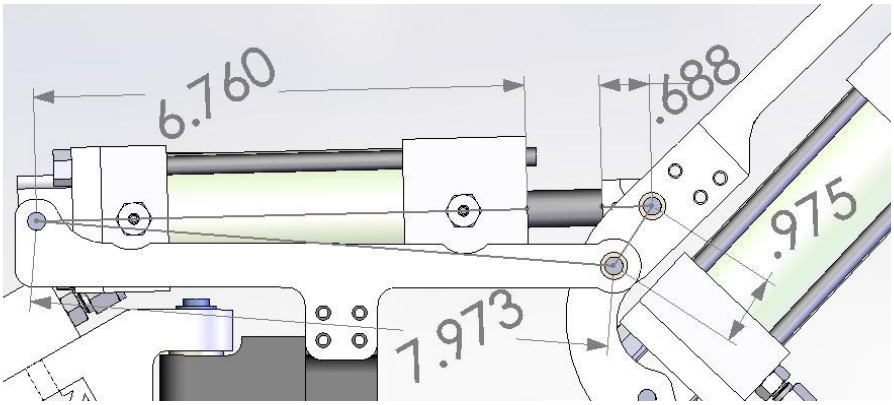

Figure 4.15: Joint 2 Link Geometry

With joint angle and dimension values known, the cosine law equation can be written and arranged to express Cylinder 2 stroke length as a function of commanded D-H joint angle.

$$\begin{aligned}
&\left(l+\Delta l_2\right)^2 = r^2 + x^2 + rx\cos\phi \\
&\left(7.45in+\Delta l_2\right)^2 = 0.95in^2 + 63.57in^2 + 7.77in^2\cdot\cos\left(168.8°+\theta_2\right) \\
&\Delta l_2 = \frac{\sqrt{64.52in^2 + 7.77in^2\cdot\cos\left(168.8°+\theta_2\right)}}{7.45in}
\end{aligned} \qquad (4.31)$$

4.2.3.4 Cylinder 3

Cylinder 3 stroke length on either leg uses an identical cosine law equation. Again, careful analysis of the joint angle geometry is required to properly isolate Φ and θ_3 (Figure 4.16).

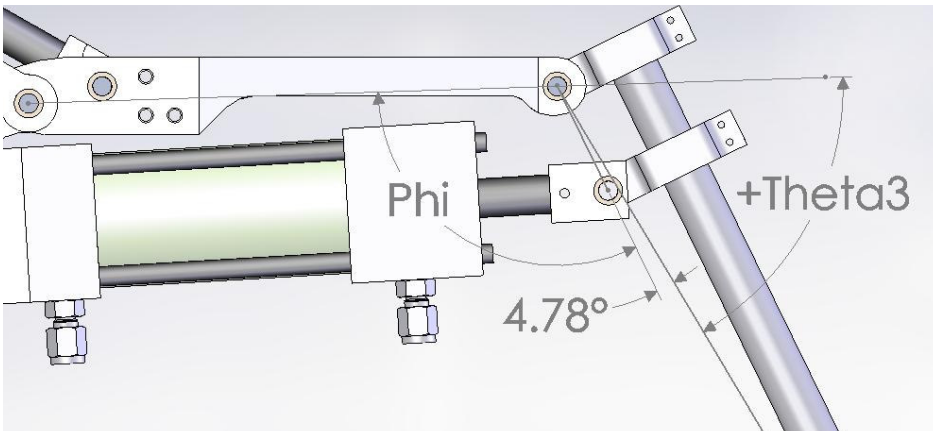

Figure 4.16: Joint 3 Angle Relationships

As with Joint 2, the Joint 3 angles contain a static angle offset due to the lateral distance between the physical joint pin and the foot. The relationship between the cosine law interior angle Φ and the commanded joint angle θ_3 is

$$\phi = 180° - 4.8° - \theta_3 = 175.2° - \theta_3 \qquad (4.32)$$

The physical dimensions of the joint geometry are used in the same manner as for Joints L1 and R1 (Figure 4.17).

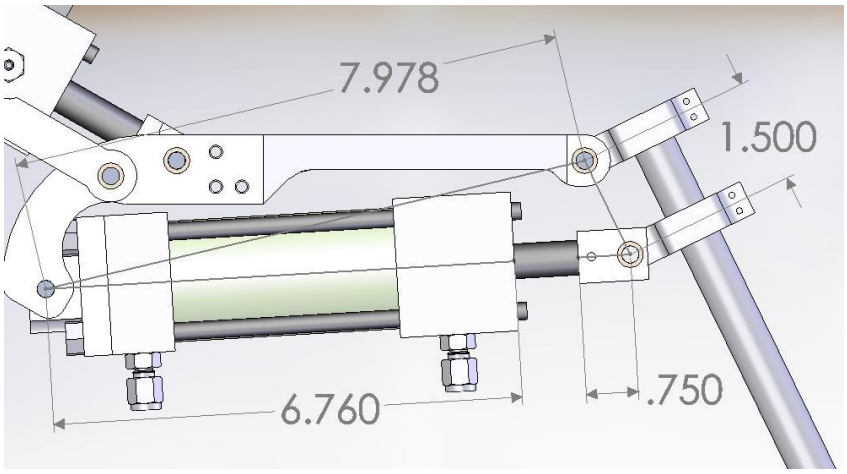

Figure 4.17: Joint 3 Link Geometry

With joint angle and dimension values known, the cosine law equation can be written and arranged to express Cylinder 3 stroke length as a function of commanded D-H joint angle.

$$\begin{aligned}
&\left(l+\Delta l_3\right)^2 = r^2 + x^2 + rx\cos\phi \\
&\left(7.51in+\Delta l_3\right)^2 = 2.25in^2 + 63.65in^2 + 11.97in^2 \cdot \cos\left(175.2° - \theta_3\right) \\
&\Delta l_3 = \frac{\sqrt{65.90in^2 + 11.97in^2 \cdot \cos\left(175.2° - \theta_3\right)}}{7.51in}
\end{aligned} \quad (4.33)$$

4.2.4 Cylinder Stroke Length Conversions

Each commanded stroke length must be converted from a value of 0-1.4 inches to a standard range for control. 0-10V was chosen for the conversion because the position sensor output is within the 0-10V range (Equation 4.34)

$$x_{command} = \frac{10V}{1.4in} x_{stroke} \quad (4.34)$$

4.3 POSITION OUTPUT TRANSFORMATION

During real-time operator control of the robot legs, the operator must be made aware of the environment through haptic feedback. Since the operator is using a three-dimensional position vector as an input to the system the system needs to responds back with a similar vector. This response vector indicates the current foot position, so that any position error is relayed to the operator via a directional haptic spring force.

The evaluation of the actual position vector p_{com} is completed by analyzing the stroke length output from the

pneumatic cylinders and processing the data through a forward displacement algorithm to determine the position vector based on joint angle input. The same four steps performed in 4.3 Control Input Transformation are performed inversely to produce a position vector in the operator input space from stroke length voltage data.

4.3.1 Cylinder Stroke Length Conversion

While reliable and extremely effective, the Sentrinsic cylinders equipped on the CRC are each electronically different. Each cylinder has a different range of output voltage for a full stroke. Occasionally the ranges drift and shift and the controller responsible for signal conversion must be recalibrated approximately every two weeks. Once each position sensor maximum and minimum voltage is known, the conversion from voltage signal to stroke length is performed (Equation 4.35).

$$x_{stroke} = 1.4in\frac{\left(x_{signal} - V_{\min}\right)}{\left(V_{\max} - V_{\min}\right)} \tag{4.35}$$

Each position sensor is measured with a voltmeter at full stroke and full retraction and the corresponding V_{max} and V_{min} values are entered into a MATLAB array read by the Simulink diagram for each sensor.

The output, then, x_{stroke} is the same range for each sensor, 0–1.4 inches.

4.3.2 Cylinder Space to Joint Space

With each actual cylinder stroke length known, the actual joint angles must be evaluated as an input into the forward displacement algorithm. Similar to the Joint Space to Cylinder Space method (4.2.3), Joints R1 and L1 are calculated separately, while Joints 2 and 3 on each leg are identical.

Each cylinder stroke length is used in a cosine law formula to evaluate the interior angle of the triangle made by the joint geometry (Equation 4.36). The joint angle is evaluated from the cosine law value.

$$l^2 = r^2 + x^2 + rx\cos(\phi)$$
$$\cos^{-1}\left(\frac{l^2 - r^2 - x^2}{rx}\right) = \phi \qquad (4.36)$$

4.3.2.1 Cylinder L1

Using Figures 4.11 and 4.12 for evaluation of the joint angle geometry, θ_{L1} is found by rewriting Equation 4.36:

$$\theta_{L1} = 90° - \phi = 90° - \cos^{-1}\left(\frac{(l + \Delta l_{L1})^2 - r^2 - x^2}{rx}\right)$$
$$\theta_{L1} = 90° - \cos^{-1}\left(\frac{(7.45 + \Delta l_{L1})^2 - 62.13in^2}{8.59in^2}\right) \qquad (4.37)$$

4.3.2.2 Cylinder R1

Using Figures 4.11 and 4.13 for evaluation of the joint angle geometry, θ_{R1} is found by rewriting Equation 4.36:

$$\theta_{R1} = \phi - 90° = \cos^{-1}\left(\frac{\left(l + \Delta l_{R1}\right)^2 - r^2 - x^2}{rx}\right) - 90°$$

$$\theta_{R1} = \cos^{-1}\left(\frac{\left(7.45 + \Delta l_{R1}\right)^2 - 62.13in^2}{8.59in^2}\right) - 90° \qquad (4.38)$$

4.3.2.3 Cylinder 2

Using Figures 4.14 and 4.15 for evaluation of the joint angle geometry, θ_2 for either leg is found by rewriting Equation 4.36:

$$\theta_2 = \phi - 168.8° = \cos^{-1}\left(\frac{\left(l + \Delta l_2\right)^2 - r^2 - x^2}{rx}\right) - 168.8°$$

$$\theta_2 = \cos^{-1}\left(\frac{\left(7.45 + \Delta l_2\right)^2 - 64.52in^2}{7.77in^2}\right) - 168.8° \qquad (4.39)$$

4.3.2.4 Cylinder 3

Using Figures 4.16 and 4.17 for evaluation of the joint angle geometry, θ_3 for either leg is found by rewriting Equation 4.36:

$$\theta_3 = 175.2 - \phi = 175.2° - \cos^{-1}\left(\frac{\left(l + \Delta l_3\right)^2 - r^2 - x^2}{rx}\right)$$

$$\theta_3 = 175.2° - \cos^{-1}\left(\frac{\left(7.51 + \Delta l_3\right)^2 - 65.90in^2}{11.97in^2}\right) \qquad (4.40)$$

4.3.3 Joint Space to Leg Space

Once each actual joint angle is known, the forward displacement algorithm computes the vector from the manipulator base to the foot from this data. The vector

describes the foot position relative to the spine before it is rotated, transformed, scaled and relayed to the PHANToMs for operator feedback.

First, the Denavit-Hartenberg homogeneous transform matrix $_{i-1,i}B$ is defined and used to transform the coordinates and positions of one link to another, starting from the base link to the next, serially (Equation 4.41).

$$ {}_{i-1,i}B = \begin{bmatrix} ({}_{i-1}x_i)_x & ({}_{i-1}y_i)_x & ({}_{i-1}z_i)_x & ({}_0p_{i-1,i})_x \\ ({}_{i-1}x_i)_y & ({}_{i-1}y_i)_y & ({}_{i-1}z_i)_y & ({}_0p_{i-1,i})_y \\ ({}_{i-1}x_i)_z & ({}_{i-1}y_i)_z & ({}_{i-1}z_i)_z & ({}_0p_{i-1,i})_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.41) $$

The D-H homogenous transform matrix is a partitioned matrix. The 3x3 section is the projection of coordinates O_i on coordinates O_{i-1}. The 3x1 matrix is the vector viewed from reference frame O_0 from O_{i-1} to O_i.

Once each homogeneous transform matrix has been calculated in terms of θ_1, θ_2, and θ_3, the final transform matrix $_{04}B$ is calculated (Equation 4.42).

$$ {}_{04}B = ({}_{01}B)({}_{12}B)({}_{23}B)({}_{34}B) \qquad (4.42) $$

The first transformation matrix $_{01}B$ is evaluated by analyzing the relationship between O_1 and O_0 (Figure 4.18).

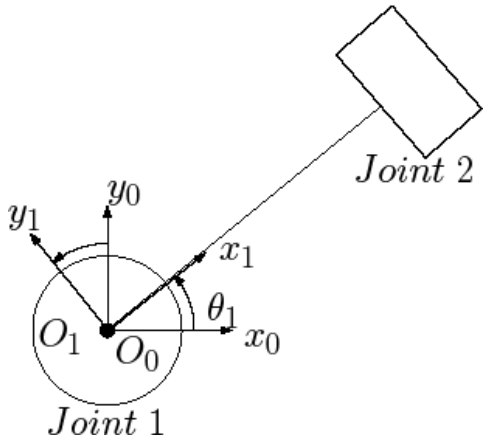

Figure 4.18: Projection of O_1 onto O_0

Using Figure 4.18, the homogeneous transform matrix ${}_{01}B$ can be evaluated (Equation 4.43).

$$ {}_{01}B = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.43) $$

Next, the transform matrix from O_1 to O_2 is evaluated using Figure 4.19 (Equation 4.44).

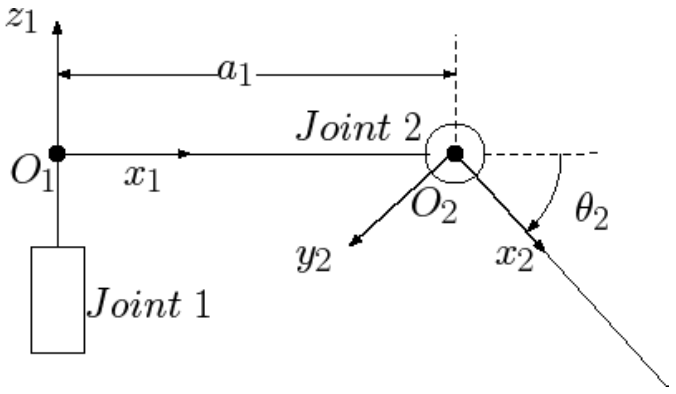

Figure 4.19: Transformation from O_1 to O_2

$$ {}_{12}B = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & a_1 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_2 & -\cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.44} $$

Next, the transform matrix from O_2 to O_3 is evaluated using Figure 4.20 (Equation 4.45).

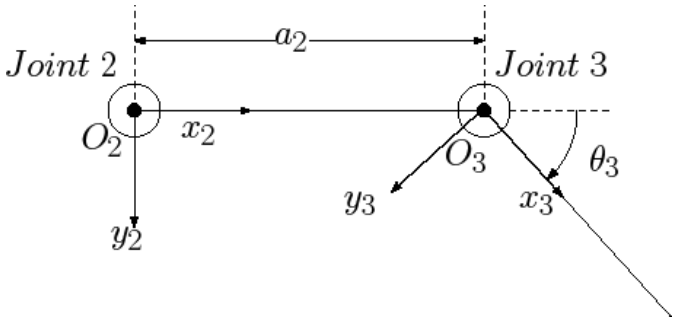

Figure 4.20: Transformation from O_2 to O_3

$$ {}_{23}B = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & a_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.45} $$

Finally, the homogenous transform matrix from O_3 to O_4 is evaluated using Figure 4.21 (Equation 4.46).

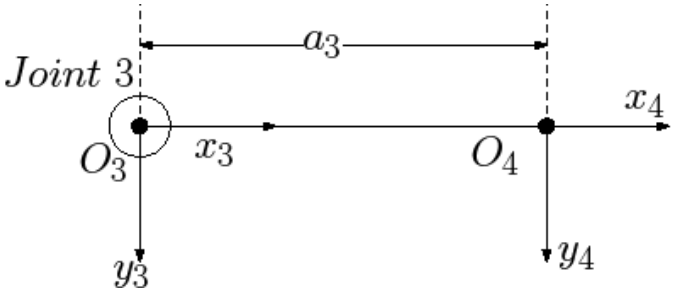

Figure 4.21: Transformation from O_3 to O_4

$$ {}_{34}B = \begin{bmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.46} $$

With each transformation matrix now in terms of the D-H parameters and standard joint angles, the total transformation matrix $_{04}B$ is calculated. Since only the vector $_{o}p_{04}$ is of consequence, its result is shown below (Equation 4.47).

$$ {}_{04}B = \left({}_{01}B\right)\left({}_{12}B\right)\left({}_{23}B\right)\left({}_{34}B\right) $$

$$ {}_{0}p_{04} = \begin{bmatrix} \left(\cos\theta_1\cos\theta_2\cos\theta_3 - \cos\theta_1\sin\theta_2\sin\theta_3\right)a_3 + \left(\cos\theta_1\cos\theta_2\right)a_2 + \left(\cos\theta_1\right)a_1 \\ \left(\sin\theta_1\cos\theta_2\cos\theta_3 - \sin\theta_1\sin\theta_2\sin\theta_3\right)a_3 + \left(\sin\theta_1\cos\theta_2\right)a_2 + \left(\sin\theta_1\right)a_1 \\ \left(-\sin\theta_2\cos\theta_3 - \cos\theta_2\sin\theta_3\right)a_3 - \left(\sin\theta_2\right)a_2 + d_1 \end{bmatrix} \tag{4.47} $$

The foot position vector $_{0}p_{04}$ is calculated each time-step from the stroke length data received by the analog card.

4.3.4 Leg Space to Input Task Space

The foot position vector $_{0}p_{04act}$ is calculated in reference to the coordinates of the base joint of the robot, which are rotated downward at 30 degrees. To send a meaningful vector to the PHANToM devices, $_{0}p_{04act}$ must be rotated and transformed to match the coordinates and scale of the input vector $_{0}p_{04com}$.

The foot position vector $_{0}p_{04act}$ must first be moved from the serial manipulator base origin O_0 to the arbitrary origin O set in the leg task space in 4.2.2 (Figure 4.22).

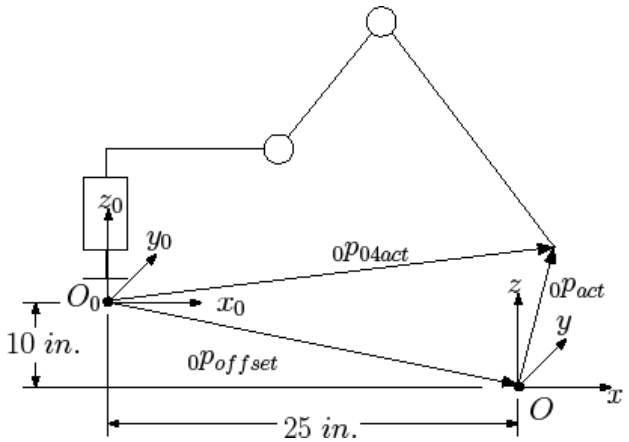

Figure 4.22: Leg Origin Placement

Again, simple vector addition yields the vector $_0p_{act}$ (Equation 4.48).

$$ {}_0p_{act} = {}_0p_{04act} - {}_0p_{offset} \qquad (4.48) $$

With $_0p_{act}$ known, the inverse procedure to the transformation described in Equation 4.5 of 4.2.1 is performed to transform $_0p_{act}$ into p_{phan_act}(Equation 4.49).

$$ p_{phan\_act} = A^{-1}_{phan-leg}\,{}_0p_{act} = \begin{bmatrix} -6.35 & -11.07 & 0 \\ 0 & 0 & 12.66 \\ -11.07 & 6.35 & 0 \end{bmatrix} \begin{bmatrix} ({}_0p_{act})_x \\ ({}_0p_{act})_y \\ ({}_0p_{act})_z \end{bmatrix} \qquad (4.49) $$

4.4 Conclusions

The PC104 CPU performing the lengthy real-time coordinate transforms and displacement analyses computes the results in less than 30% of each control time-step. The forward displacement algorithm yields accurate results for the foot position based on the stroke length inputs. Accuracy was verified by physically measuring the joint angles and foot position and comparing to the displayed

results of the algorithm. The inverse displacement algorithm, updated every 1 ms, outputs accurate stroke length commands. This accuracy and coordination can be verified visually by powering both the leg sensors and the PHANToM controllers. The legs are able to back-drive the PHANToMs due to the bilateral teleoperation condition of the system. When the foot is moved in a straight line, relative to the spine (multiple joints moving), the PHANToM controller follows in a straight line. When the entire system is powered, the PHANToM can be moved in a straight line, and the foot position will follow to the best ability of the controller.

CHAPTER 5

LEG CONTROL

Each leg controller uses the transformations discussed in Chapter 4 to apply a control effort to each individual pneumatic cylinder. Each actuator is position controlled independently via a PD controller with added force control from a differential pressure gain scheduler.

The control laws for each cylinder are essentially identical, save for different gain values. The control inputs are 0-10V signals to each valve. The 0-10V analog signal corresponds to the valve spool position. A 5V signal commands the spool to the center position, blocking all flow to either cylinder chamber (Figure 5.1).

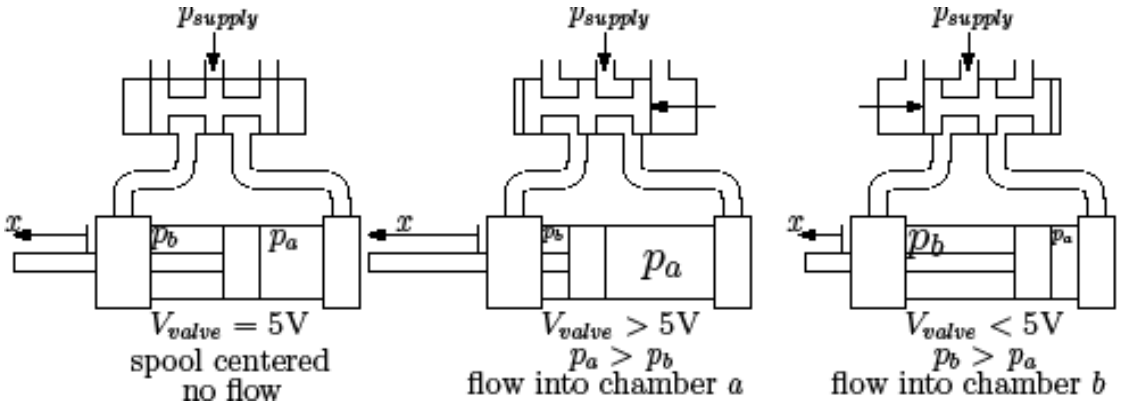

Figure 5.1: Spool Position and Cylinder Relationship

Each cylinder is, by itself, a fourth or fifth order nonlinear, discontinuous, time varying system. Coupled with the valve dynamics, the high order system is further

complicated by the coupling and discontinuities and nonlinearities imposed by cantilevered links, ground interaction forces, and the compressibility of air. Instead of modeling the entire leg system and optimizing a controller for different scenarios and conditions, a generally robust PD controller was chosen for servo control. The control variable, spool position, is directly proportional to the volumetric flow rate to and from each cylinder chamber (Figure 5.2).

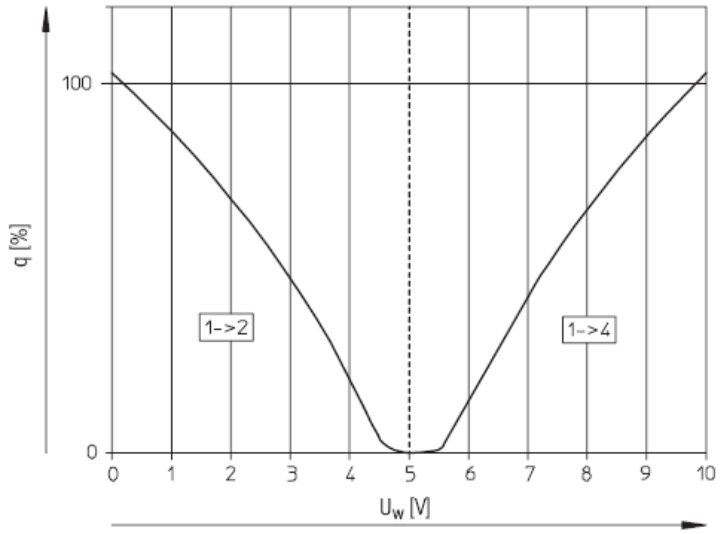

Figure 5.2: Flow Rate vs. Spool Position Command

While the direct control input is a voltage signal controlling spool position, the physical correlation is mass flow rate into and out of each cylinder chamber. Directly coupled to the flow rate is a pressure term. The overall control effort becomes a complex relationship between fluid flow rate, pressure and temperature.

In a simple analogy, the valve regulates input effort similar to the way a motor servo controller regulates voltage and current (force and flow) to achieve desired position. The system exhibits traits of internal integration behavior (Type I system).

5.1 Control Objective

The goal of establishing control over each cylinder is to maintain tracking control of the foot (endpoint) of each leg. Tracking control, rather than tuned responses to pre-generated inputs, is vital to this application because each leg will be driven by an operator giving direct inputs via two bilateral teleoperated PHANToM haptic devices. The force generated by the PHANToMs is proportional to the position error between the commanded and actual foot positions. If the foot were to strike an obstacle or become entangled, the operator will feel the sharp increase in position error. However, if the operator is constantly driving the feet while 'wading' through a constantly high position error, the haptic force increase generated by an obstacle will be less noticeable, and the operator will quickly tire from the constant forces.

Ideal tracking, i.e. zero error between commanded and actual position, requires very complex and accurate modeling techniques which were neither employable nor employed through the course of this project. Standard PD control methods are incompatible with ideal tracking

because an error signal is required to generate a control signal.

Since ideal tracking is, in the scope of this project, unattainable, a controller was designed to provide “good” tracking.

5.1.1 Controller Requirements

Several requirements were determined during controller design and tuning. These sets of limitations and expectations were to be met by the final version of the control scheme.

Stability – First and foremost, the controller designed for this system must produce a stable system response. Stability was required in response to a step input, and sinusoidal inputs up to 5 Hz at 80% stroke command. An 80% stroke command was chosen for the stability analysis so that the piston would have sufficient space in the chamber to overshoot its commanded position. Otherwise, the sinusoid would simply be dead-heading the cylinder fully back and forth like an on-off valve. 5 Hz was chosen as the stability limit because the operator should not be able to command the foot position to change that rapidly, and such a command would actually approach the maximum flow capacity of the Festo valves, introducing an entirely new dynamic into the leg systems as the actuators become starved for flow (3.2.1).

While an exponential rise in position instability will not damage the actuators, the wild oscillations caused by

certain conditions certainly pose safety risks both to the operator and to those who it would rescue.

In addition to control parameter tuning, saturations and filters are used to adequately harness the pneumatic control system.

Robustness – A robust controller is obviously necessary for such a remote, teleoperated system because the operator must rely solely on the control software if any hardware or sensor failures occur. Operating the system open-loop in an emergency (due to sensor failure) is a desired feature of the system.

The controller must maintain control of the system when a sensor fails, experiences noise, malfunctions, or when the system experiences a fluid leak. The robustness of the controller must also compensate for the sharp disparity between ground contact and free-space movements.

Tracking response – In order to provide effective operator control, the foot position response must be crisp and reactive to the operator's inputs. Through controller tuning and development, good tracking control was detectable by feel through the haptic controllers. Through testing, it was determined that a tracking error < 10% stroke length (< 0.14 inches) provides a satisfactory medium between tracking control and stability.

5.2 Position Control

Servo position control is accomplished by using a discrete proportional-derivative (PD) controller with a

velocity feed-forward command and velocity damping. Each cylinder is controlled by an individual PD controller, using the commanded stroke length x_{ref} as the input command. X_{ref} is generated for each cylinder by the input coordinate transformations. Each cylinder features a position sensor which feeds back actual stroke length x_{act} to the controller for comparison to the command position x_{ref}.

5.2.1 Control Law

The position control law assigned to each cylinder is

$$\begin{aligned} y_{PD} &= \left(k_p \left(x_{ref} - x_{act} \right) - k_d \dot{x}_{act} + k_{vff} \dot{x}_{ref} \right) \\ V_{valve} &= y_{PD} + 5 \end{aligned} \tag{5.1}$$

This control law assigns a valve spool position V_{valve} based on the position error, the position command rate, and the actual position rate. The gain values k_p, k_d, and k_{vff} are tuned for each cylinder pair (L1/R1, L2/R2, L3/R3) because each actuator encounters different loading conditions.

The proportional gain constants k_p were determined experimentally through testing. The derivative gain constants k_d were also determined experimentally, though the method of obtaining a signal derivative also required tuning. Since the signals coming from all the sensors included some electrical noise, a standard discrete derivative only increased the noise output of the controller. The solution to the noise problem is to increase the sampling time over which the value is differentiated. For this case, the derivative values are

calculated over a period of 40 sampling intervals, or 0.04 seconds. This lengthening of the differentiation span greatly reduces the amount of noise introduced by the controller derivative functions. A 10Hz low-pass filter completes the signal smoothing operation.

5.2.2 Position Control Stability

Operating each leg cylinder under only closed loop PD control, stability of each actuator can be demonstrated experimentally via step inputs from 20% to 80% stroke (0.28 – 1.12 inches). This stroke limitation is chosen to allow for any position overshoot. A full stroke length step input would simply dead-head the piston at the travel limits of the cylinder, rendering stability analysis impossible due to the lack of information as to whether the piston is held in place by a controlled pressure differential. During testing, the robot is vertically constrained to its cart, allowing Joint 2 of each leg to apply force to the ground as they normally would. The operating pressure is 130 psi.

Figures 5.3, 5.4, and 5.5 below show that stability of stroke length x_{act} for each cylinder is achievable through this position controller.

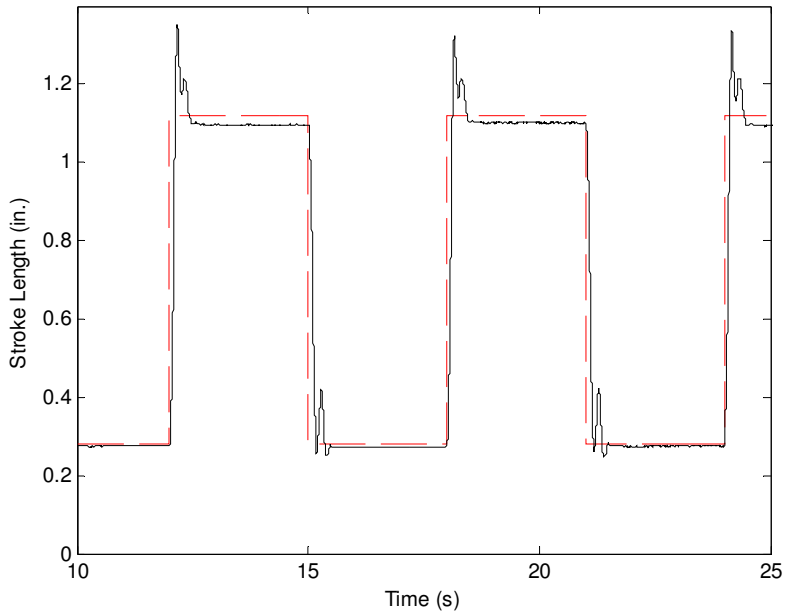

Figure 5.3: Cylinder L1 Step Response, PD Control, k_p = 0.5, k_d = 0.004, k_{vff} = 0.05, 130 psi

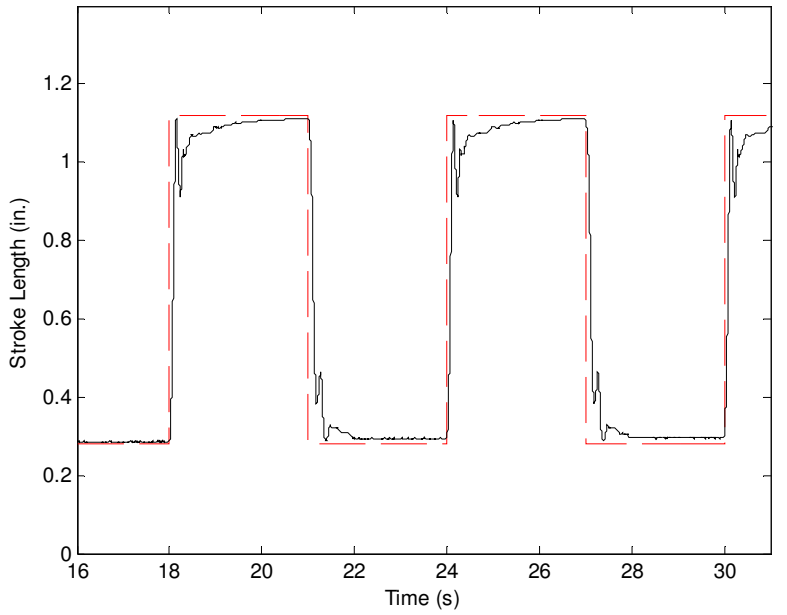

Figure 5.4: Cylinder L2 Step Response, PD Control, k_p = 0.5, k_d = 0.004, k_{vff} = 0.015, 130 psi

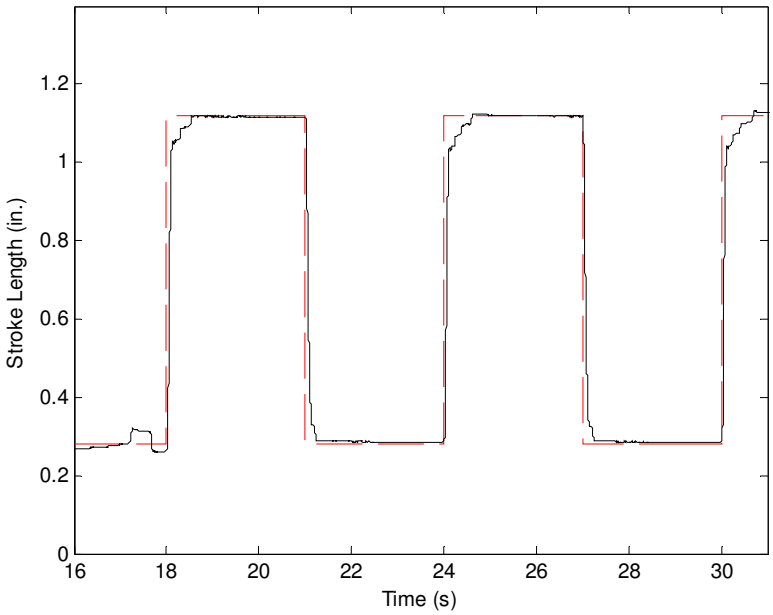

Figure 5.5: Cylinder L3 Step Response, PD Control, k_p = 0.55, k_d = 0.01, k_{vff} = 0.03, 130 psi

5.2.3 Tracking Response

The tracking response of the individual actuators is demonstrated below (Figures 5.6, 5.7, 5.8). A 3 rad/s sinusoid, 20%-80% stroke length is the position input. No other parameters were changed.

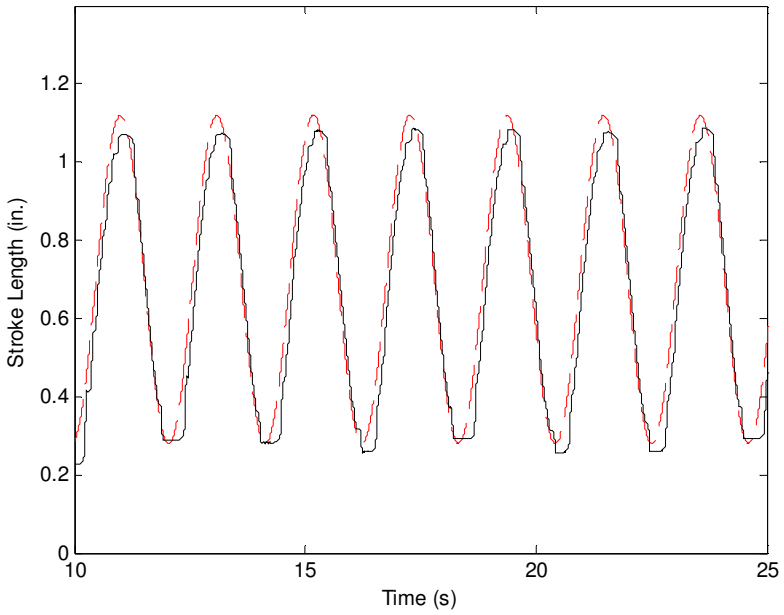

Figure 5.6: Cylinder L3 3 rad/s Tracking, PD Control, k_p = 0.55, k_d = 0.01, k_{vff} = 0.03, 130 psi

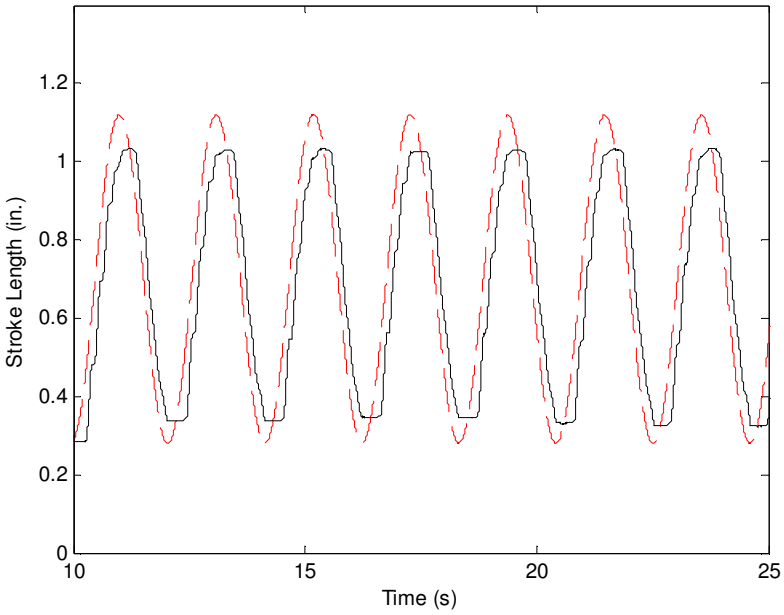

Figure 5.7: Cylinder L2, 3 rad/s Tracking, PD Control, k_p = 0.5, k_d = 0.004, k_{vff} = 0.015, 130 psi

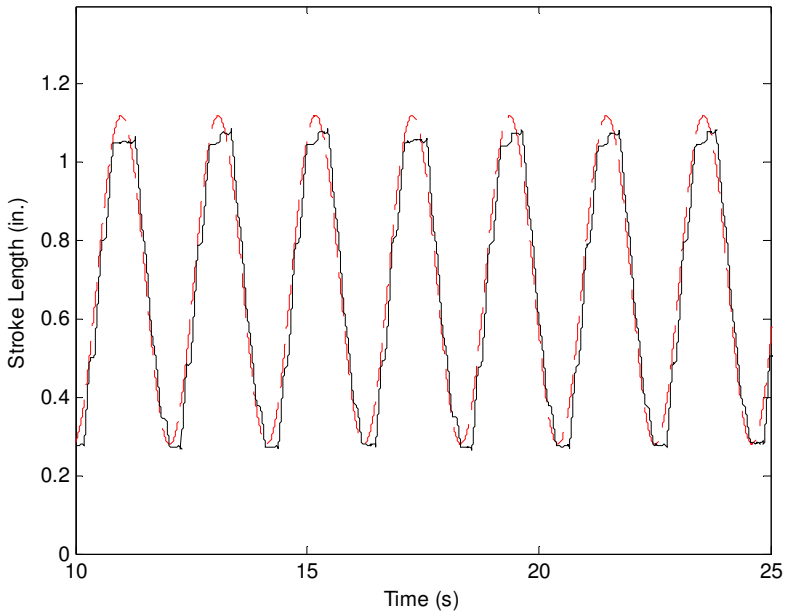

Figure 5.8: Cylinder L1, 3 rad/s Tracking, PD Control, k_p = 0.5, k_d = 0.004, k_{vff} = 0.05, 130 psi

The tracking results show good adherence to the input sinusoid. The only exception, though, was the results from cylinder 2. Since Links 2 and 3 are cantilevered around Joint 2, the position controller alone cannot regulate enough flow and pressure to maintain acceptable tracking.

The failure of the lone PD controller when ground forces are encountered is more evident when the foot position is guided through a stepping sequence, i.e. swing and stance phases (Figure 5.9).

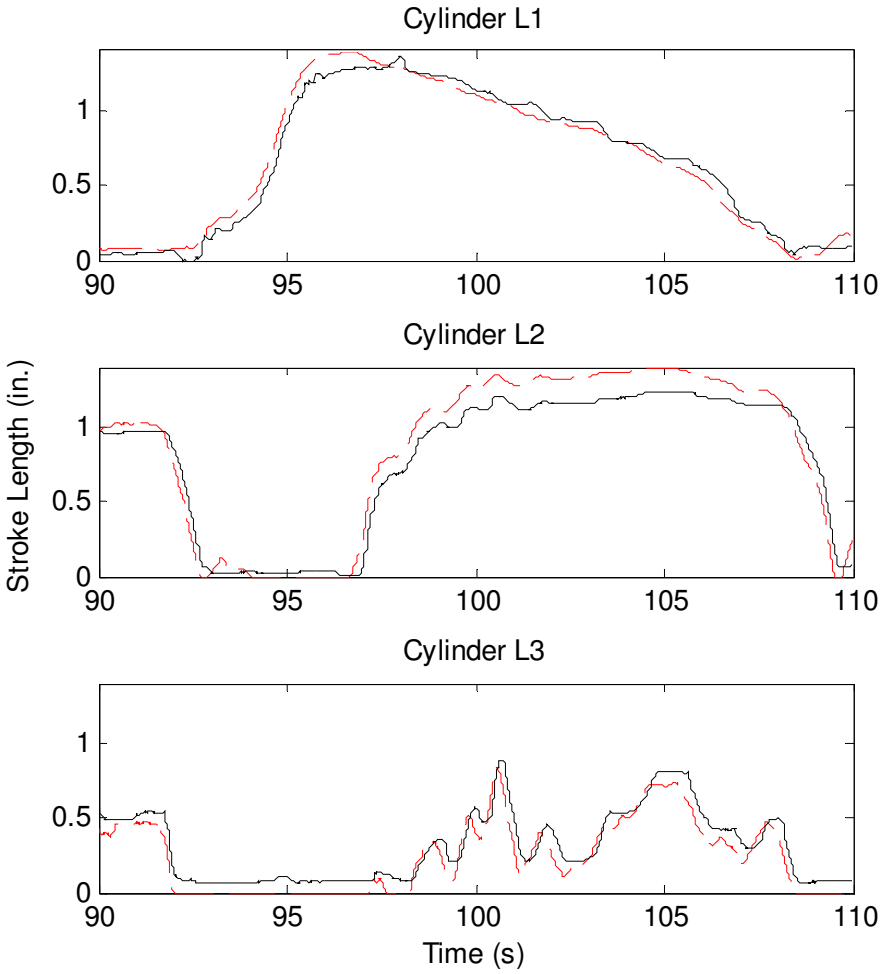

Figure 5.9: Cylinder Stroke Length Response During One Step Cycle, PD Control

The stroke length error of Cylinder 2 during the stance phase becomes rather significant as the actuator bears the weight of the robot. Without any additional control terms, the command to the valve is simply not sufficient to counter the static loading. The absolute position error through the gait cycle is shown below (Figure 5.10).

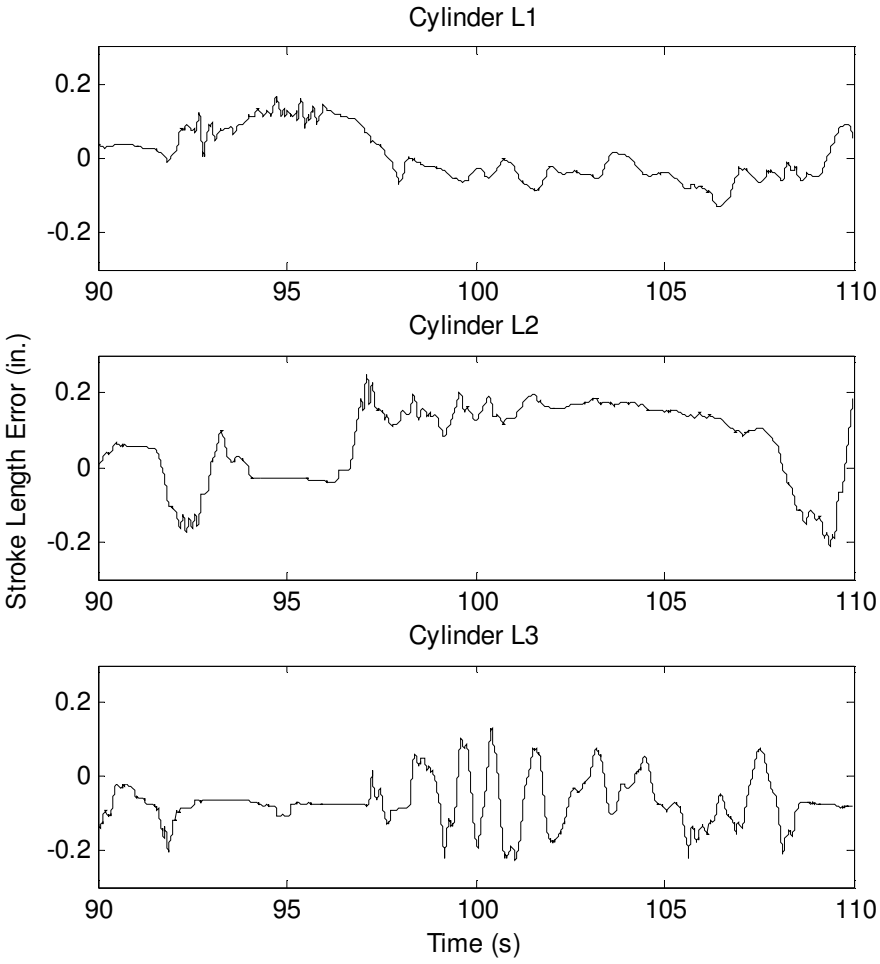

Figure 5.10: Cylinder Stroke Length Error During One Step Cycle, PD Control

The tracking errors in the three cylinders fall outside the 0.14 inch requirement set for this control system. Since this high error occurs only when large forces are applied to the endpoint, some additional force control is obviously needed to correct this position error. Elsewhere, though, the PD controller performs satisfactorily in controlling the leg through the swing phase, so the added force controller should be designed as

to not affect the position controller when the leg encounters minor forces such as its own inertia, or overhanging loads.

5.3 Force Control

Force control is obtained for each cylinder through the two pressure sensors installed to measure absolute air pressure in chambers *a* and *b*. The pressure measurements are converted to force measurements simply by multiplying the value by the piston area, and the resultant forces are differenced, yielding a single force value, with direction and magnitude. Pressure sensors were used rather than actual force or torque sensors because of their compactness, simplicity, embedded design, and low cost.

5.3.1 Force Control Law

An added force term, as already discussed, is necessary to correct for high loading applications experienced during stance phases of gaits. Cylinders L2 and R2 experience the highest loading during stance, when most of the robot weight is supported by Joints L2 and R2. Since the standard PD control doesn't provide enough control effort signal to the valve, a gain scheduler is implemented to add the additional control effort needed to obtain low position error.

A simple differential pressure gain scheduler was discussed and tested in [9] and its results compared to standard PID control. The researchers showed that, in

their experiment, the position tracking results were better than those obtained with a simple PID controller.

Initially the simple differential pressure gain scheduler was implemented on Joint 2 (Equations 5.2-5.4).

$$\Delta p = p_a - p_b$$

$$y_{dp} = \begin{cases} \Delta p \cdot k_{dp1} : \Delta p > 0, e > 0 \\ -\Delta p \cdot k_{dp2} : \Delta p > 0, e < 0 \\ -\Delta p \cdot k_{dp3} : \Delta p < 0, e > 0 \\ \Delta p \cdot k_{dp4} : \Delta p < 0, e < 0 \end{cases} \qquad (5.2\text{-}5.4)$$

$$V_{valve} = y_{PD} + y_{dp} + 5$$

The results proved promising, whereas after some tuning, the tracking error was smaller while Joint 2 was supporting weight. Figure 5.11 shows a swing and stance phase tracking response for all three cylinders. Only L2 is controlled with the supplementary differential pressure gain.

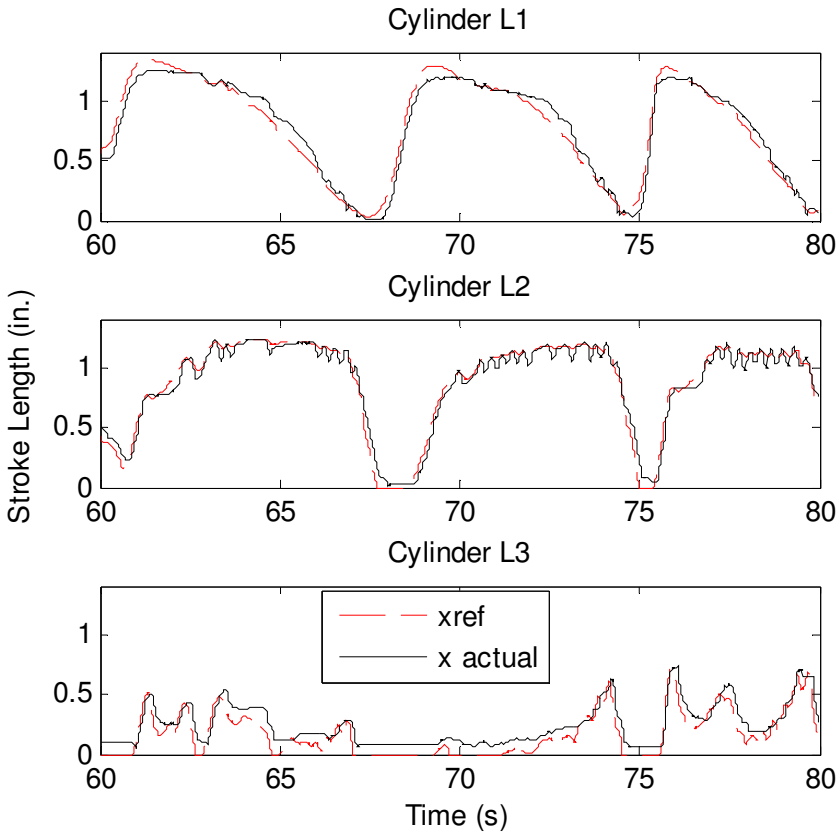

Figure 5.11: Cylinder Stroke Length Response During One Step Cycle, PD + dp Control

The position tracking response for Cylinder L2 has been markedly improved, but the gain scheduler has introduced an unwanted dynamic into the control effort signal. As the stroke length x_{act} crosses the reference value x_{ref}, the error changes sign and the gain scheduler instantaneously changes the gain value y_{dp} added to the control effort (Figure 5.12).

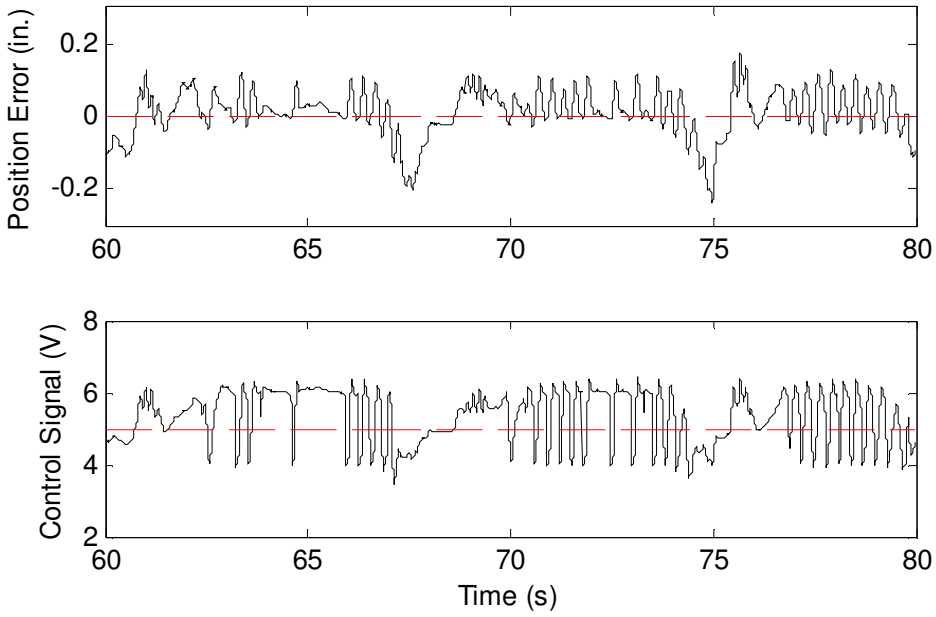

Figure 5.12: Cylinder L2 Position Error and Control Effort for Swing-Stance Phases in Figure 5.11

The rapid oscillations commanded to the valve spool position cause the robot to physically bounce as the control signal changes. This bouncing symptom would create severe instabilities if the gain scheduler output were not saturated at +/- 1 V.

Compared to Figure 5.9, the tracking error for L2 is much lower while the weight of the robot is supported by Joint 2 (Stroke length > 1 in.).

5.3.2 Improved Force Control Law

A new force control law was implemented in the gain scheduler to prevent the bouncing encountered through the use of the differential pressure controller. First, the control term was changed from differential pressure to differential force. This allows an equivalence to be made

on each side of the piston. When the stroke length is stationary, the differential pressure will never be zero due to the difference in areas of the piston sides. Using differential force as an input (Equation 5.5) means that when the stroke length is stationary, the force differential is zero.

$$\Delta F = \left(\left(p_a A_a \right) - \left(p_b A_b \right) \right) \tag{5.5}$$

The new gain scheduler was designed considering the bouncing symptoms caused by the error changing sign and causing the gain to instantaneously change. To combat the zero crossing problems, the structure of the gain scheduler was kept essentially the same, except the gain output is scaled by the actual error and an adjustable error gain k_e (Equations 5.6–5.7).

$$y_{dfe} = \begin{cases} \Delta F \cdot k_{dfe1} \cdot e \cdot k_e : \Delta F > 0, e > 0 \\ \Delta F \cdot k_{dfe2} \cdot e \cdot k_e : \Delta F > 0, e < 0 \\ \Delta F \cdot k_{dfe3} \cdot e \cdot k_e : \Delta F < 0, e > 0 \\ \Delta F \cdot k_{dfe4} \cdot e \cdot k_e : \Delta F < 0, e < 0 \end{cases} \tag{5.6–5.7}$$

$$V_{valve} = y_{PD} + y_{dfe} + 5$$

With this new differential force scheduler, the added control effort couples position and force feedback signals. As the position error decreases, i.e. the volume and pressure in the cylinder chambers is approaching the correct value to maintain the commanded setpoint, the added control effort from the gain scheduler decreases as well.

During the swing phase, when the force differential across the piston is low, the supplemental control effort y_{dfe} is also low. When the foot is in ground contact, and a large force differential causes a large position error, the supplemental control effort y_{dfe} grows to correct the error, and then tapers off once the error is zero. Figure 5.13 below shows the improved performance during a swing and stance cycle as Joint 2 supports the weight of the robot, avoids bouncing, and maintains a low tracking error.

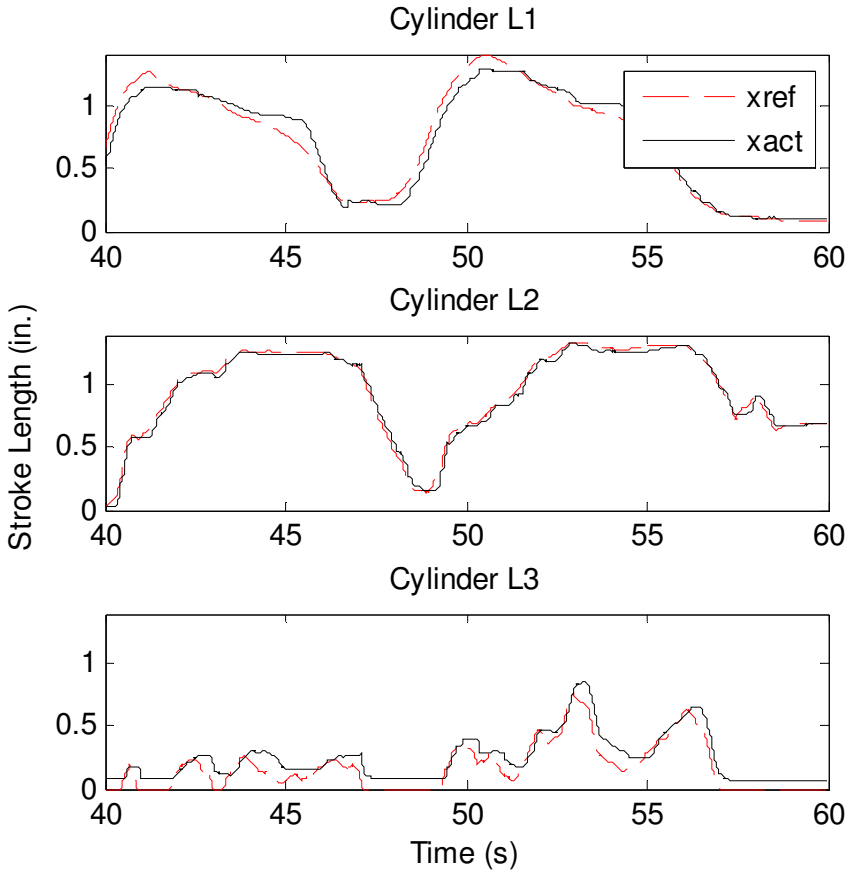

Figure 5.13: Cylinder Stroke Length Response During One Step Cycle, PD + dfe Control

The tracking response of Cylinder L2 has greatly improved as compared to the responses seen in Figures 5.9, and 5.11.

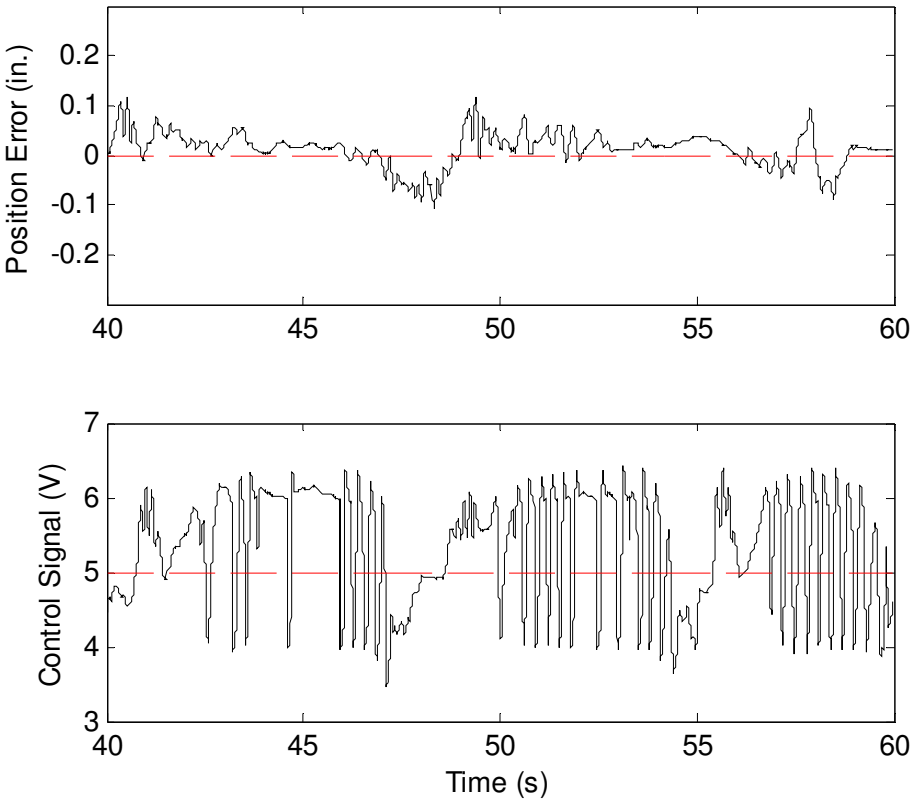

Figure 5.14: Cylinder L2 Position Error and Control Effort for Swing-Stance Phases in Figure 5.13

The improved force control term y_{dfe} improves the tracking error and eliminates the bouncing effect caused by the simple differential pressure control term y_{dp}. The position error is low enough to fit into the control requirements (error < 10% or 0.14 inches).

The error coefficient used k_e is 1.0, and the four differential force coefficients are k_{dfe1} = 0.03, k_{dfe2} = 0.03, k_{dfe3} = −0.06, and k_{dfe4} = 0.

5.3.3 Improved Force-based Position Controller on Three Joints

While the improved force-based position controller greatly improved the tracking responses of Cylinders L2 and R2, the other leg joints also benefit from this control structure. Joints L1 and R1 need this control to allow the operator to overcome the inertia of the robot as the command is given to pull forward during a stance phase. Joints L3 and R3 can greatly benefit from supplementary force control during the stance phase, providing enough lateral forces to maintain the commanded foot positions.

The supplementary force control, for L1 and R1 adds force in the most needed portion of the gait, the stance phase. During stance, the pressure and force differential is negative ($F_a < F_b$), and the position error is negative. To correct this error, the pressure in chamber b must be increased to further retract the cylinder, requiring a spool position command $V_{valve} < 5V$.

In Figure 5.13 above, tracking errors persist in Cylinders L1 and L3. The same supplementary force control applied to L1 yields better results as shown below in Figure 5.15.

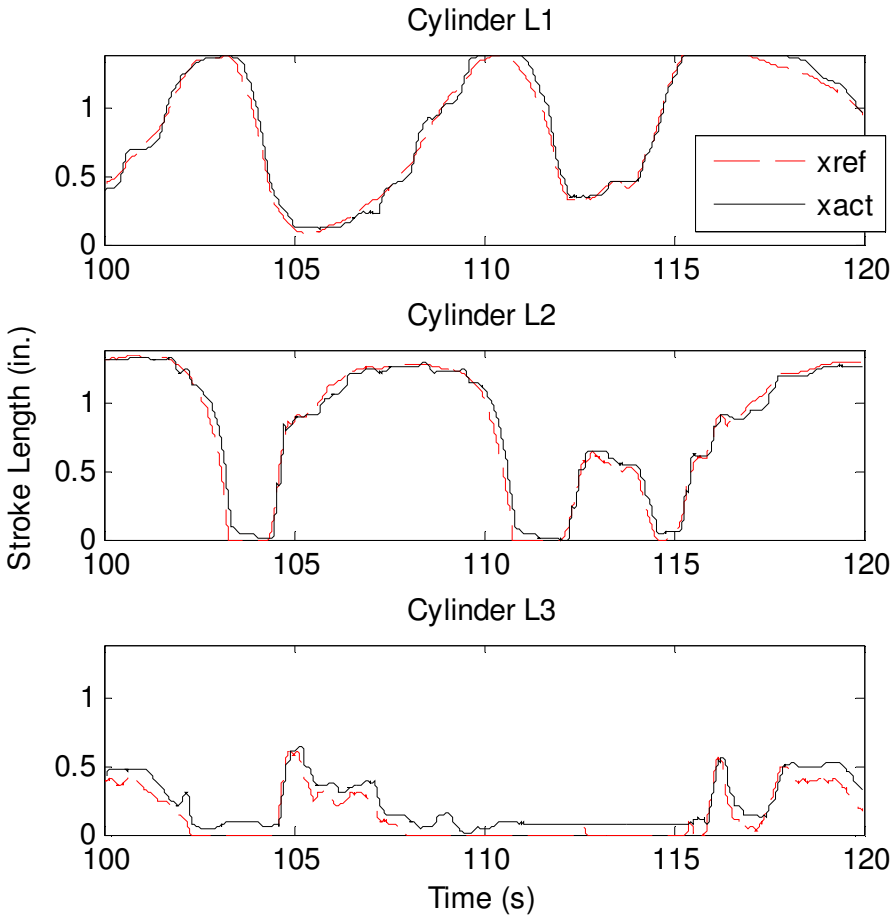

Figure 5.15: Full Controller Applied to L1 and L2 Through Multiple Swing-Stance Phases

The tracking error of L1, specifically, when the stroke length is decreasing, has greatly improved over the response in Figure 5.13. The position error (Figure 5.16) remains below the required 10% error stipulation. The error coefficient used k_e is 1.0, and the four differential force coefficients are k_{dfe1} = 0.01, k_{dfe2} = 0.005, k_{dfe3} = 0.02, and k_{dfe4} = -0.03.

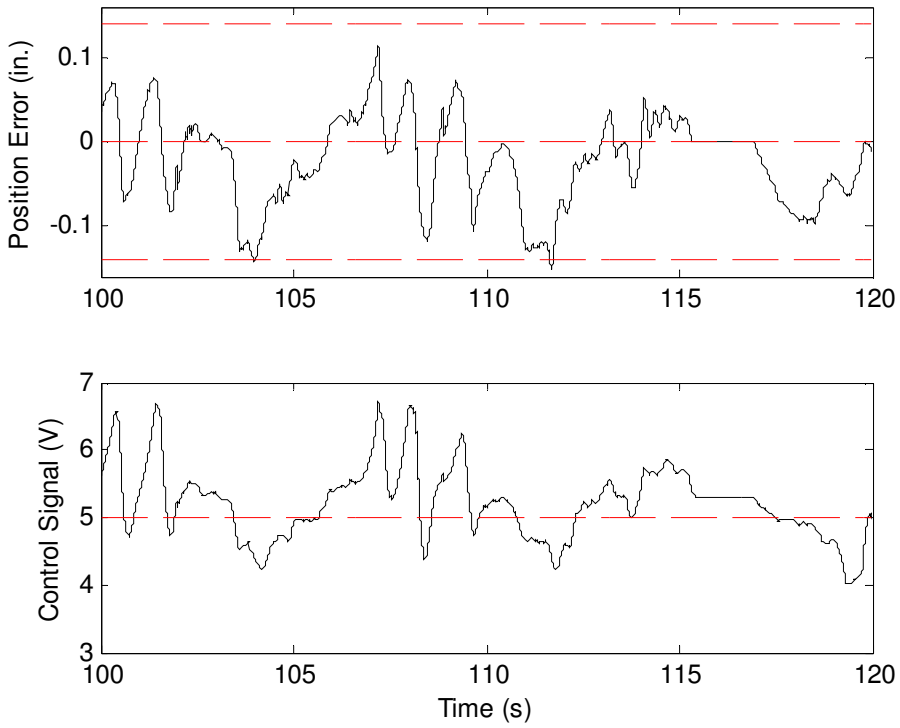

Figure 5.16: Cylinder L1 Position Error and Control Effort for Swing-Stance Phases in Figure 5.15

The tracking response of L3 and R3 benefits from the force-based position controller as well. During a stance phase, especially one in which the feet are set wide apart, supplementary control effort is needed to prevent the feet from spreading further apart under the weight of the robot. To accomplish this, extra force is focused on the same case as the L1/R1 controller where the pressure and force differential is negative ($F_a < F_b$), and the position error is negative. To correct this error, the pressure in chamber b must be increased to further retract the cylinder, requiring a spool position command $V_{valve} < 5V$.

Figure 5.13 above shows that as L3 stroke length increases, the magnitude of the position error increases as well. This is a symptom of the feet slipping outward as weight is applied, and the inability of the PD position to correct for the added load. Care must also be taken with Joint 3 to avoid too much supplementary force control. Only the case described above requires significant control effort due to the low mass and inertia of Link 3 relative to the power of its controlling actuator. Figure 5.17 shows the improved tracking response of L3 during a gait cycle.

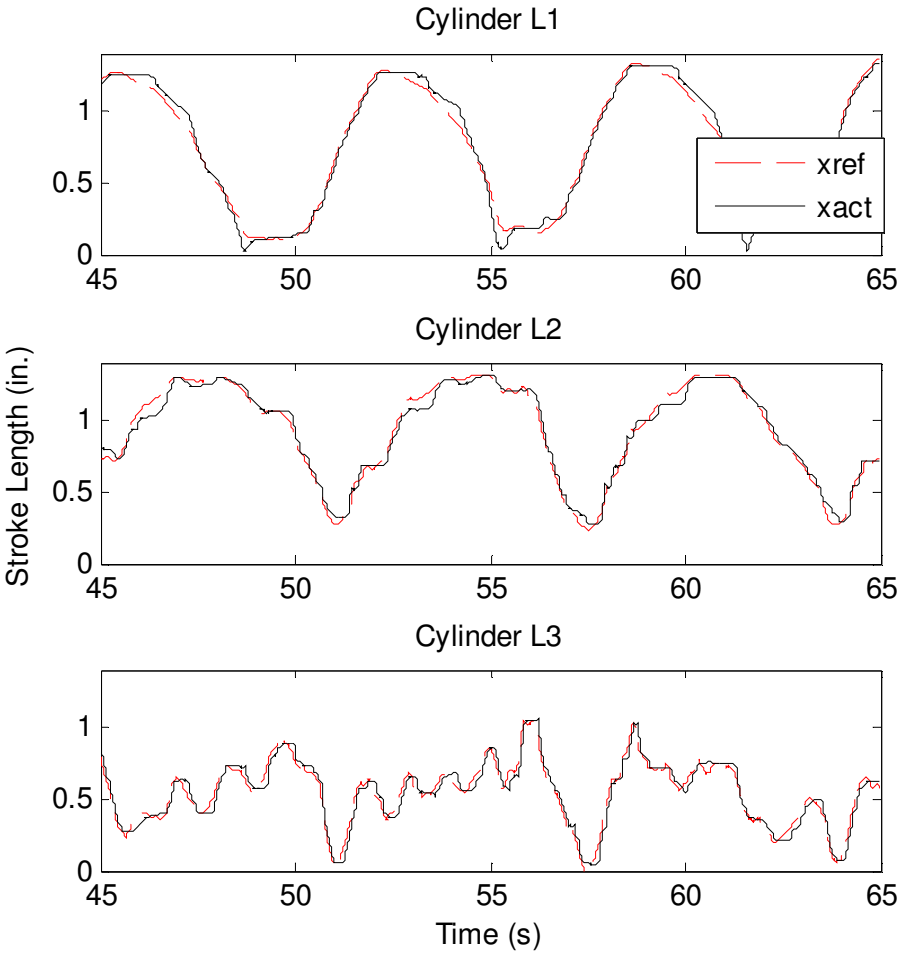

Figure 5.17: Full Controller Applied to L1, L2, and L3

The tracking error has been improved and the system remains stable. The supplementary force control keeps the stroke length near the reference value, and prevents the feet from slipping outward (Figure 5.18).

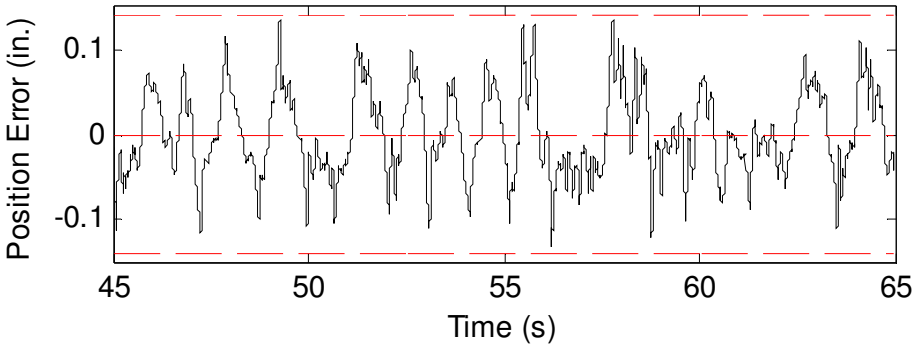

Figure 5.18: Cylinder L3 Position Error for Swing-Stance Phases in Figure 5.17

The position error (Figure 5.18) remains below the required 10% error stipulation. The error coefficient used k_e is 1.0, and the four differential force coefficients are k_{dfe1} = 0, k_{dfe2} = 0, k_{dfe3} = -0.01, and k_{dfe4} = -0.015.

5.4 Results and Conclusions

The force-based position controller implemented on each leg of the Compact Rescue Crawler allows the operator to directly control the stroke lengths of each cylinder simultaneously to achieve a desired endpoint position. The controller and system responses fulfill all requirements set forth at the inception of controller development. Each individual actuator system is stable, and the overall foot position is stable. The tracking error for each stroke length remains less than 10% during operator controlled walking. Since the operator-guided walking gait (swing and stance phases), requires tight tracking in both free-space and ground contact, the success of this one controller for both scenarios is all the more significant of a

contribution. The tracking error requirement is perhaps the most important to the application of haptic bilateral teleoperation discussed later in Chapter 6.

Overall, the applied real-time controller is successful in achieving the project control goals. It is not, however, the only control solution for this system. Other combinations of gains and force control techniques could control this system, but the controller presented herein was one which, through tuning, provided the desirable responses. The full Simulink diagram of the control structure is located in Appendix B.

The gains, low-pass filter (LPF) cutoff frequencies and constants used to produce the best control responses are located below in Table 5.1.

Table 5.1: Control Gains and Settings

		L1/R1	L2/R2	L3/R3
PD Controller	k_p	0.5	0.5	0.7
	k_d	0.004	0.005	0.01
	k_{vff}	0.05	0.015	0.05
	PD Output Saturation	±3V	±3V	±3V
	Derivative Time Constant	0.04s	0.03s	0.04s
Gain Scheduler	k_e	1	1	1
	k_{dfe1}	0.01	0.03	0
	k_{dfe2}	0.005	0.03	0
	k_{dfe3}	0.02	−0.06	−0.01
	k_{dfe4}	−0.03	0	−0.015
	Gain Scheduler LPF cutoff	4Hz	4Hz	4Hz
	Valve signal LPF cutoff	100Hz	100Hz	100Hz
	Position input LPF Cutoff	50Hz	50Hz	50Hz
	Pressure input LPF Cutoff	20Hz	20Hz	20Hz

CHAPTER 6

OPERATOR INTERFACE

The Compact Rescue Crawler operator interface, physically, is a bilateral teleoperation workstation from which the operator is immersed into the controller and remotely pilots the vehicle (Figure 6.1).

Figure 6.1: Operator Remotely Pilots the Crawler

Virtually, the operator workstation is designed to immerse the operator into an augmented reality with visual, aural, and haptic cues. Haptic force is generated by the

two PHANToM controllers which generate position inputs to the robot. A head-mounted display feeds live video from the robot to the operator. The head-mounted display is equipped with a motion tracker which controls the orientation of the pan-tilt-zoom (PTZ) camera on the front of the robot. The head mounted display is also equipped with earphones which will relay audio to the operator from the sounds “heard” by microphones on the robot.

This operator workstation attempts to immerse the operator in the remote environment of the robot by virtually placing the operator on the front of the robot. Searching for survivors will, through future research, be aided by information gathering software looking for signs of life in the work environment.

6.1 Workstation Design

The workstation was designed and constructed with focus on configurability. The base and uprights are fabricated from 80/20 aluminum extrusion. A comfortable task chair is mounted to the base to seat the operator. Two uprights arise from the sides of the base to hold the PHANToM controllers. One upright behind the operator chair holds the motion tracking hardware and the head mounted display when not in use (Figure 6.2).

Figure 6.2: Operator Workstation

The PHANToMs are supported by two planar positioning arms, allowing the devices to be moved to a position comfortable to the operator. The planar positioning arms also permit the PHANToMs to be moved while the operator is entering or exiting the workstation. PHANToMs are placed in an inverted position, with the endpoints facing each other rather than away. The inverted positioning of the PHANToMs allows the hand position of the operator to remain comfortable while the armrests support the elbows. Vertical positioning is adjusted by loosening the positioning arm bracket and sliding it up or down on the upright. The uprights are stabilized by a cross-member to damp vibrations and prevent unintended inputs.

6.2 Haptic Interface

The PHANToM master devices are active haptic units, as opposed to passive. The directional haptic force is

generated by three motors mounted on each axis of the device. When the operator guides the slave legs into an obstacle and the position error from the commanded foot position p_{com} to the actual foot position p_{act} increases, the haptic force generated by the PHANToM master increases as well. The haptic force vector is generated by scaling the position error vector by the constant k_{spring}.

The haptic display is a "spring" force generated on the endpoint. As the absolute position error of the foot increases away from the endpoint, the PHANToM force guides the operator's hand back toward the current leg position. Equation 6.1 evaluates the haptic force to be displayed each controller time-step. When the operator guides the foot into an obstacle and the actuators cannot physically converge to zero position error, the sudden increase in haptic resistance immediately alerts the operator that a collision has occurred.

$$\vec{F}_{haptic} = -k_{spring}\left(\vec{p}_{com} - \vec{p}_{act}\right) \qquad (6.1)$$

Achieving reliable and precise tracking control is important for the implementation of this type of haptic feedback. Low tracking error is crucial for the repetitive motions which the operator will be expected to perform while navigating through debris. A large tracking error in this scenario would apply a constantly high spring force resisting the operator's motion inputs. This "muddy" feeling would quickly tire the operator's arms, effectively

reducing the amount of time for which the operator is capable of effectively piloting the robot [30].

Haptic spring-force feedback can be provided in lieu of tight tracking control if the spring constant, k_{spring} set in software, is kept low. Initial experiments with this system used a standard PD controller with no velocity feed-forward and no pressure feedback. When the foot position was commanded downwards to lift the robot, the controller could not issue enough control effort voltage to the valves, therefore the position error from cylinders R2 and L2 was always high. The swing cylinders L1 and R1, when commanded to pull the body forward, also suffered from this large position error, as seen in Chapter 5.

One method, with the simple PD controller, to provide any kind of useful haptic feedback was to set the spring constant between 0.02 and 0.04 N/mm. This low spring constant alleviated the "muddy" feeling caused by poor tracking. The drawback though, was that when an obstacle was encountered in the foot trajectory, the constantly present tracking error was not greatly increased, therefore, the operator would not perceive that the leg was guided into an obstacle.

The improved force-based position controller described above allows for higher spring constants to be applied to the PHANToM endpoints due to its improved tracking characteristics in both free-space and ground contact scenarios. The higher spring constants, 0.06 – 0.10N/mm,

provide a crisper feel to the operator. Constants higher than 0.10N/mm tend to cause the PHANToM motors to overheat quickly and the internal controller shuts them off to prevent damage. As seen in Figs. 6.3 and 6.4, the improved error tracking decreases the ambient and false forces displayed to the operator due to tracking error. Data taken for these figures was collected from the operator guiding the crawler through walking cycles.

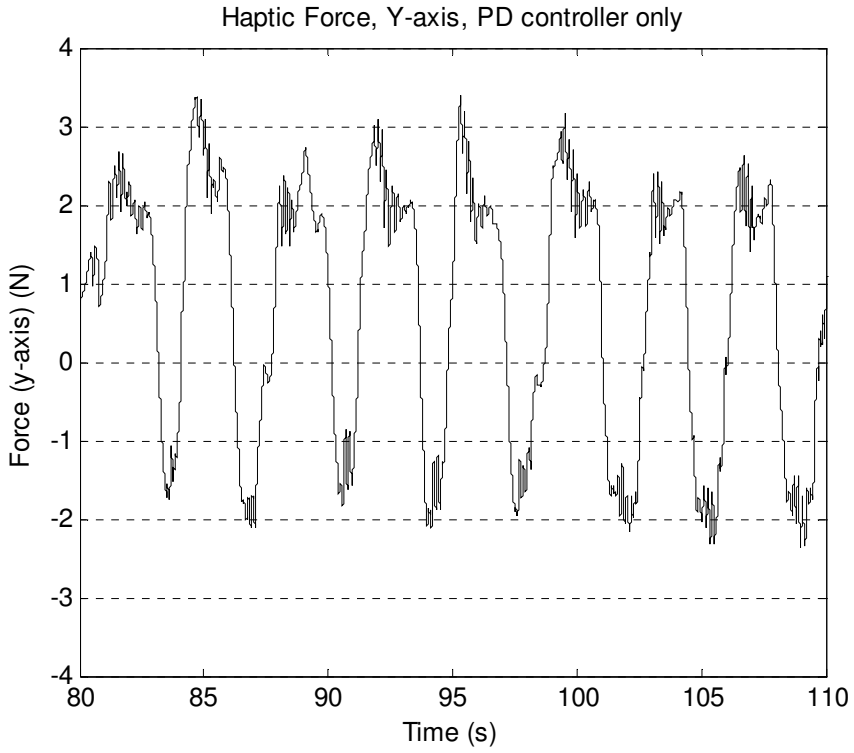

Figure 6.3: Vertical Haptic Force during Walking, PD Controller Only

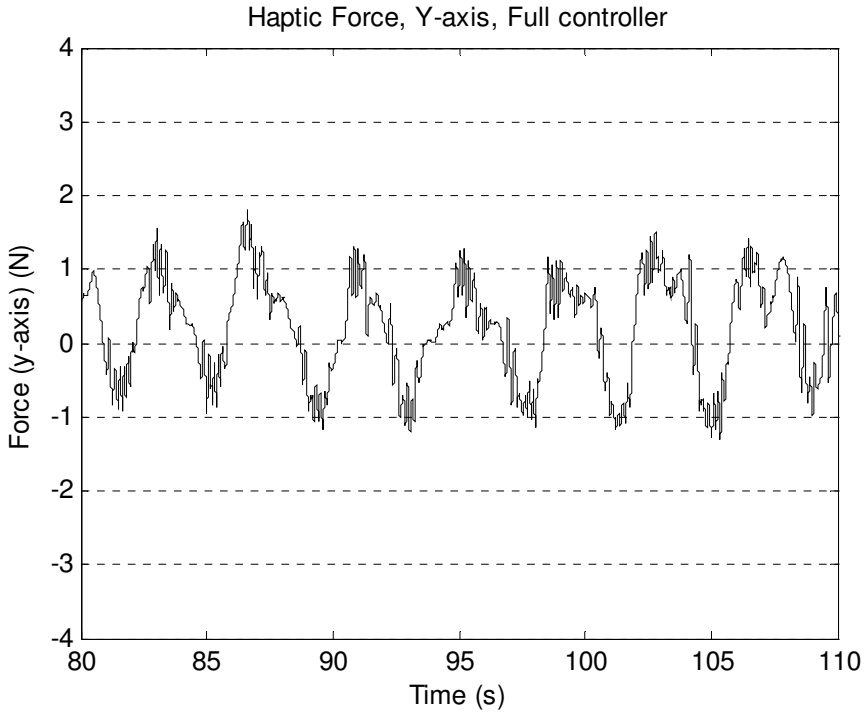

Figure 6.4: Vertical Haptic Force during Walking, Full Controller

One factor of the improved tracking control and low ambient haptic force is that if the leg strikes a light or mobile obstacle such as a small rock, the position controller will most likely power the leg through it without creating a position error. This may be beneficial, though, because the operator will not receive constant haptic signals when the leg strikes small mobile objects and debris. Testing in a controlled environment must be conducted before definite results are drawn, though. Another benefit of the improved tracking is that the operator does not feel the weight of the robot through the PHANToMs. Since only position error creates the haptic spring force, a collision with a massive object such as an

immobile beam or wall will create a significant force feedback.

The C++ code that interfaces the PHANToMs to the robot through UDP can be found in Appendix C.

6.3 AUGMENTED REALITY INTERFACE

Two computers control the augmented reality visual interface for the operator. One computer, onboard the robot is a mini-ITX form factor, fanless PC. It receives the raw video feed from the onboard PTZ camera through a frame-grabber PCI card. The video packets are then sent via a TCP/IP routine to the video host PC. The host PC unpacks the video images and displays them to the operator via a head-mounted display. The video feed, seen remotely by the operator is interactive whereas the operator's head controls the position of the camera.

A Polhemus Minuteman 3-axis motion tracker is mounted to the top of the head-mounted display (Figure 6.5). The tracker measures the angle of the operator's head side-to-side and up and down. The angle measurement is then calculated into a signal for the PTZ camera and sent via TCP/IP back to the onboard mini-ITX PC on the robot. The onboard computer then sends the position commands to the camera via an RS-232 communication interface, and internal motors move the camera to its commanded orientation.

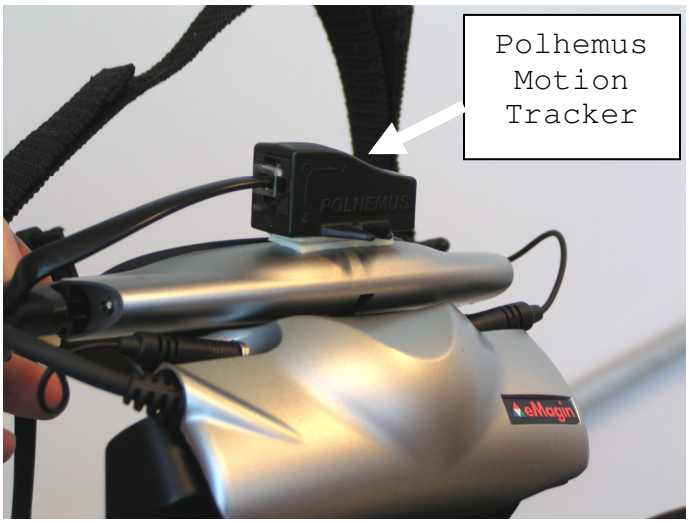

Figure 6.5: Head-Mounted Display with Motion Tracker

6.3.1 Display

A temporary display was designed as a place-holder for the augmented reality techniques in development at NCAT (Figure 6.6).

Figure 6.6: Prototype Operator Display

The video interface overlay is designed with mission-specific tools to provide the operator with information as quickly and efficiently as possible. The current prototype employs mock gauges and a compass. Data from the robot will, through research at NCAT, be presented in such an intuitive manner that the operator will know every important detail, yet not be overwhelmed with a flood of too much information.

CHAPTER 7

GUIDED GAIT COORDINATION

While human-machine interfaces have developed rapidly over recent years, the evolution of man has not. Humans possess only a limited number of degrees of freedom to physically interact with machines. In piloting a highly maneuverable hexapod, the ideal scenario would place the operator in direct simultaneous control of all six legs, or 18 degrees of freedom. This scenario, unfortunately, is infeasible due to the fact that direct, simultaneous control of 18 degrees of freedom would overwhelm the operator, rendering the vehicle ineffective.

A fully autonomous gait, at the other end of the control spectrum, is also undesirable for the entire search and rescue mission. The operator will most likely need direct control of the robot to inspect areas of interest where survivors will likely be found. Therefore, a hybrid guided-gait coordinator has been analyzed and designed for use on the CRC and possibly other operator-guided legged platforms.

A guided-gait coordination routine was designed for walking on flat straight terrain as an initial point for further research. The methods developed herein should be expandable multi-legged robots and to three and even six-axis high level coordination.

A few basic assumptions were made during the design of the basic guided-gait coordination routine:

1. Terrain is flat, level, and travel is in a straight line

2. With one foot at its PEP (Posterior Extreme Position), the foot posterior to it can touch it at its AEP (Anterior Extreme Point),

3. Feet are not slipping on the ground.

4. The leg pairs are identical, evenly spaced along the length of the robot, able to reach the same angle whether reaching anterior or posterior.

5. An experienced operator is piloting the robot, keen to situational changes and able to take direct control of a posterior leg should the environment change and ensnare it.

CRC legs are annotated in a similar fashion to joints and actuators. Left legs are named L1, L2, L3 with L1 as the foremost leg. Right legs are named R1, R2, R3 respectively (Figure 7.1).

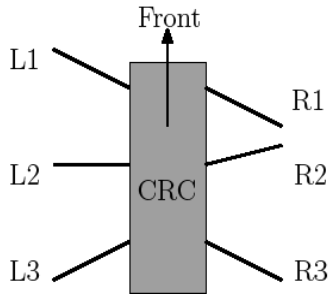

Figure 7.1: Leg Notation of CRC

7.1 OVERVIEW

The ultimate goal of the guided-gait coordination routine is to allow the operator to map a series of “stepping stones” using the front two legs, and force the rear legs to follow the same steps. The safely mapped trajectories propagate through successive leg pairs as the robot progresses through the environment. To maneuver the CRC through rugged terrain via guided-gait coordination, the operator must perform a repetitive series of commands and tasks, interacting with the gait controller to move to the next phase. A general overview of the gait sequence is outlined below in Figure 7.2. The guided-gait flowchart also serves as an outline of the structure of Chapter 7.

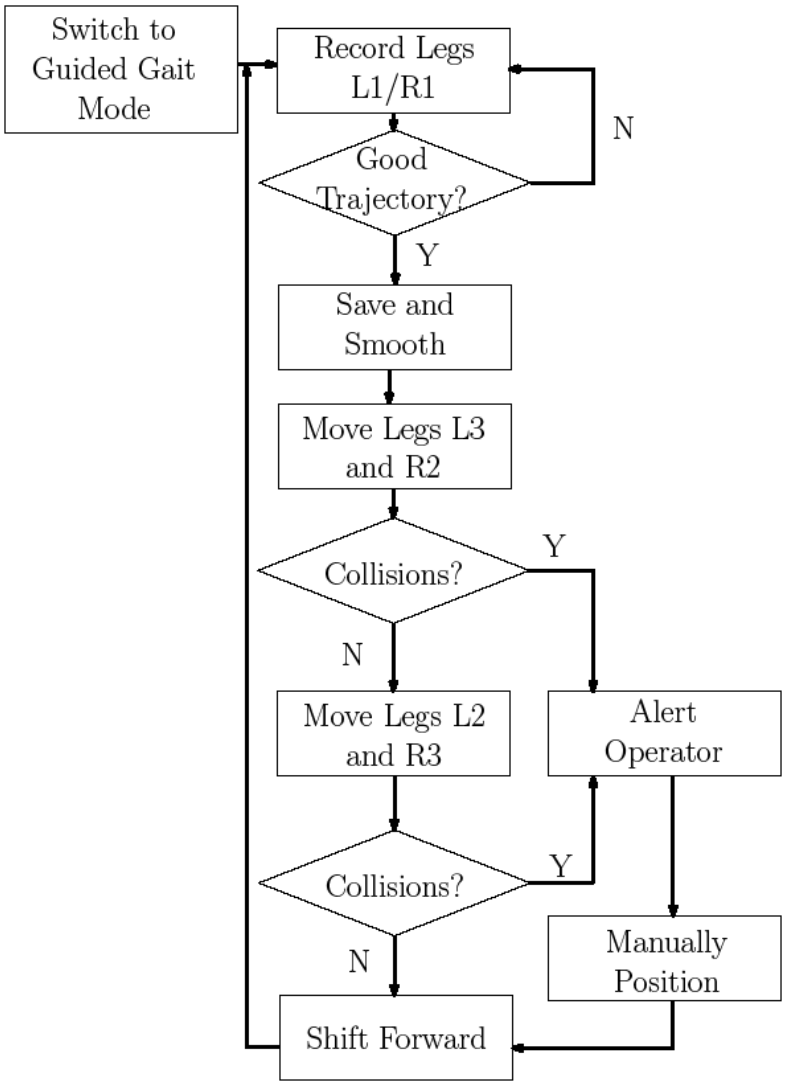

Figure 7.2: Generalized Guided-Gait Coordination Flowchart

This guided-gait coordination allows the operator to avoid an obstacle with the front legs while propagating safe, obstacle-avoiding trajectories automatically to subsequent leg pairs. Therefore, once the operator steps over a fallen beam, for example, and moves forward, the middle leg pair will step over it when the body reaches that position in the environment. Then, the rear-most leg pair will step over it when it reaches the obstacle.

Details of the gait coordination are laid out below in the order of operation shown above in Figure 7.2.

7.2 Trajectory Recording

The CRC operator, when in guided-gait mode, will need to provide input to the coordinator by recording the trajectories of the front two legs. Wielding haptic control over the front legs gives the operator the advantage of sensory feedback about the environment.

Each PHANToM is equipped with a small button to receive the record command. When the operator presses the record button, the PHANToM controller code begins storing data points from the respective PHANToM each time-step. Once a successful leg trajectory has been made, the operator simply presses the record button again to stop the data saving process.

This button-pressing routine could be replaced by verbal commands in the future. When voice commands are integrated into the high-level architecture, the operator will also be able to command the controller to delete a failed trajectory so another, smoother attempt can be performed.

Ideally, the operator begins recording the swing phase leg trajectory from its posterior extreme position (PEP), and ends recording when the leg is at its anterior extreme position (AEP). This maximization of workspace will optimize the overall speed at which the CRC moves through the environment. If the foot will not reach over an

obstacle, or the operator is unsure of the foothold conditions, recording should stop when the foot is on the ground and supporting the weight of the robot once again. Visual and haptic cues can alert the operator to touchdown and weight support by monitoring the pressure sensors in each actuator.

7.2.1 Recording Detail

When the operator triggers the record command, a 5V signal is sent directly to the serial port of the PHANToM control computer. The C++ software running the PHANToMs polls the Data Send Ready (DSR) and Clear To Send (CTS) pins of the serial port. DSR is pin 6, and CTS is pin 8 on the standard RS-232 9-pin plug. Since the CTS and DSR pins are able to be polled directly from the Windows C++ programming environment, the RS-232 serial port doubles as a rudimentary digital input card. Two other pins could be used in this manner, the Data Terminal Ready (DTR pin) and the Ring Indicator (RI pin). With a total of four digital input bits, up to 15 switches or inputs could technically be implemented through this method, if read in binary.

The record switches are two Cherry limit switches (Figure 7.3). The 5V supply is taken directly from the PC power supply and fed to the switches. When the switch is activated, its respective pin on the RS-232 serial port is pulled high to 5V, indicating a digital value of 1, or ON. Since the routine only polls the DSR and CTS pins once

every 1ms time-step, high frequency electro-mechanical switch bouncing is not a significant issue.

Figure 7.3: Record Start/Stop Switch Operation

The switches are positioned in an ergonomic location where the operator need only roll the index finger forward to lightly depress the switch lever while grasping the ball on the PHANToM endpoint. The switches are wired in such a way that the stiffness of the wire adds no additional resistance to the joint operation of the PHANToM controllers.

The points recorded from the PHANToMs are saved each time-step as raw points in a space delimited text file. When recording stops, the file is saved to disk and named with the time at which it was created. A UNIX timestamp is used to provide a consistent time naming convention.

7.3 Trajectory Manipulation

Once trajectories have been recorded, a smoothing operation is performed on the raw trajectories. Raw trajectories recorded directly from PHANToM motion are jagged from operator-induced (Figure 7.4).

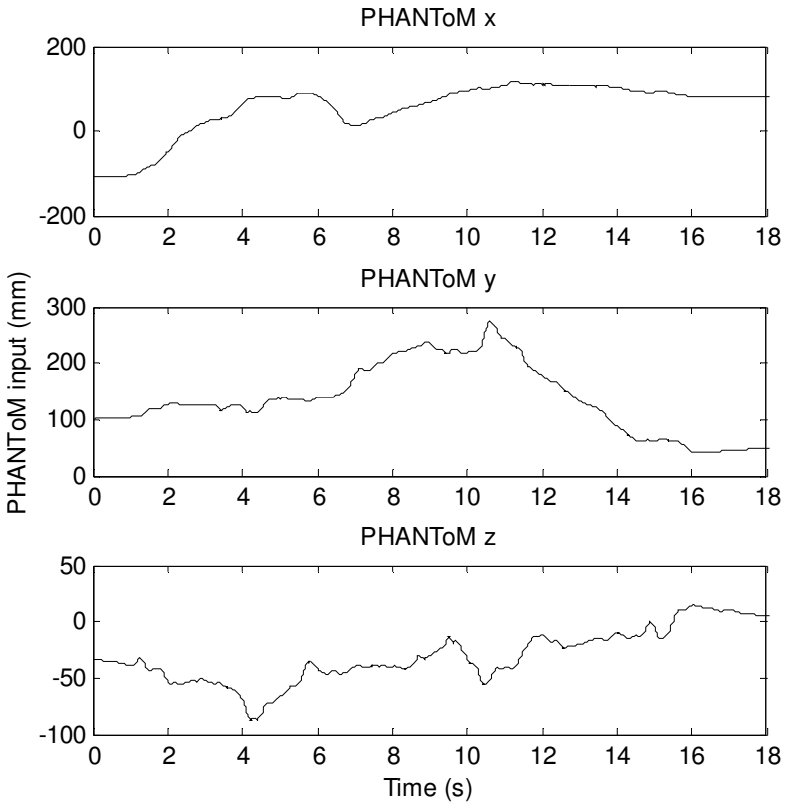

Figure 7.4: Raw PHANToM Points Captured During Left Leg Swing Phase

In 3D view, the captured trajectory shows that the operator guided the leg into an obstacle (along x-axis) from which the operator retreated and rerouted the foot (Figure 7.5).

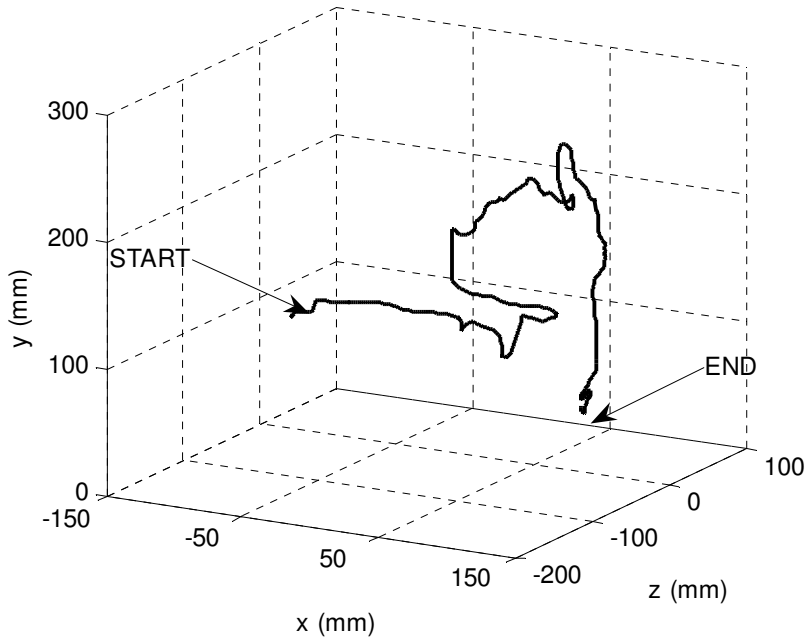

Figure 7.5: Three Dimensional View of Same Trajectory

Playing the same raw trajectory through the leg would yield an acceptable system response, but a smoother profile will play more smoothly through the controller, creating an overall smoother motion. The raw trajectory is smoothed by wrapping a spline, or piecewise polynomial curves, through evenly spaced points along the trajectory.

Each successive splined trajectory is made from the same number of spline points *n* for simplicity. An appropriate number of spline points was determined by analyzing sample trajectories in a controlled setting, then comparing the effects of 5, 20, and 30 spline points. The match of the spline to the original trajectory is desired to be close, but not too exact or too general. Over-generalization will cause the foot to collide with

previously avoided obstacles, and too tight of a fit may either take too much computing time and have little advantage over a looser fit. The selected number of spline points is used for every recorded trajectory.

First, the raw data is split into its x, y, and z component vectors. Then, *n* points are selected at regular intervals along each path. The number of spline points *n* is determined visually by viewing the different spline results and comparing to the original path. Once *n* evenly spaced points are selected, a piecewise polynomial curve is fitted for each segment and sampled at the 1ms time-step. The final splined trajectory is exactly the same length as the original, with the same start and end points, but the sharp edges and jitteriness of the inputs have been significantly reduced.

A five-point spline was wrapped over a sample swing phase trajectory (Figures 7.6, 7.7).

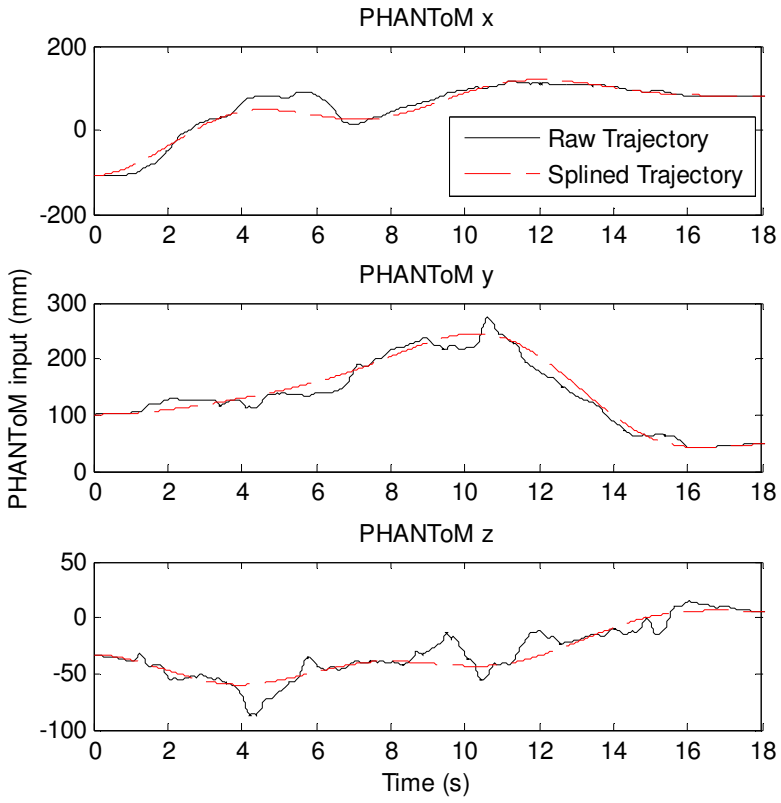

Figure 7.6: 5 Point Spline Over Trajectory

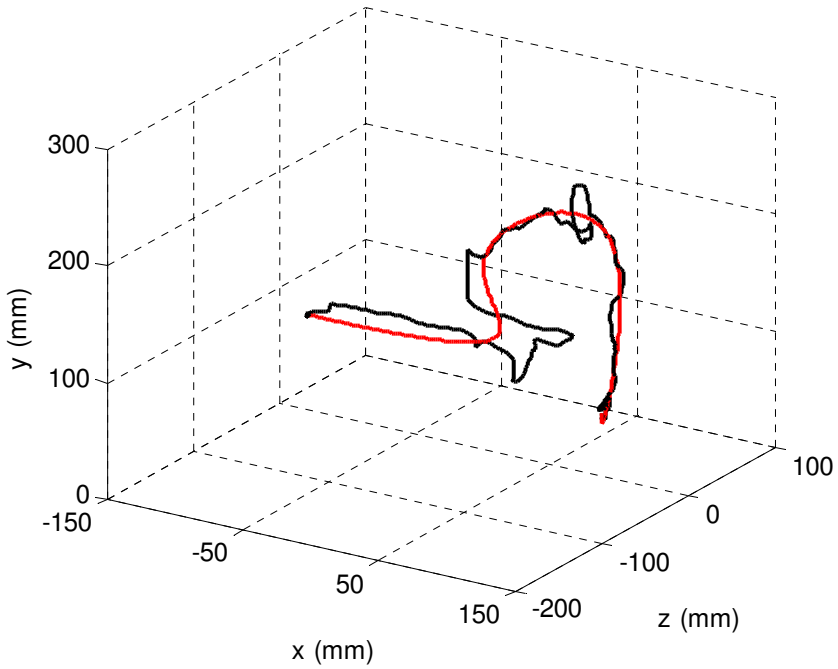

Figure 7.7: 5 Point Spline Over Foot Trajectory, 3D

Clearly, the 5 point spline is insufficient to cover the necessary areas of the trajectory, especially along the z-axis. Subsequently, a 20 point spline was tested to analyze its fit to the original trajectory (Figures 7.8, 7.9).

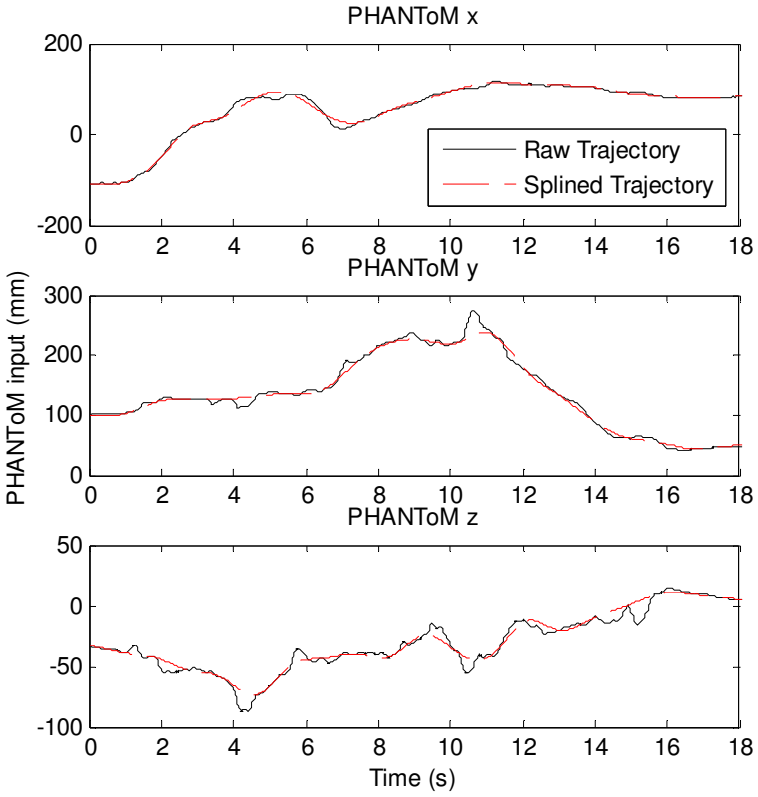

Figure 7.8: 20 Point Spline Over Trajectory

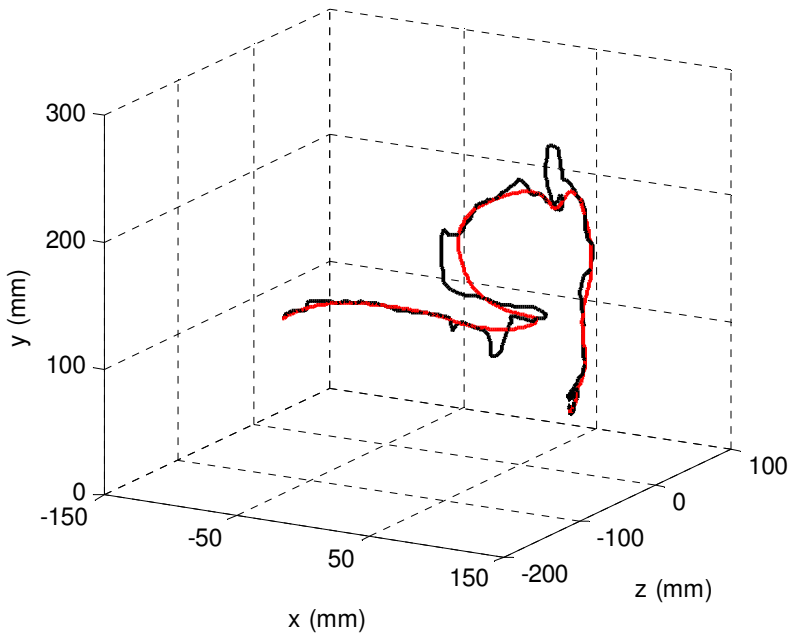

Figure 7.9: 20 Point Spline Over Foot Trajectory, 3D

The 20 point spline fits the original trajectory well and does not over-generalize the original trajectory. A 30 point spline was then applied to the trajectory for analysis (Figures 7.10, 7.11).

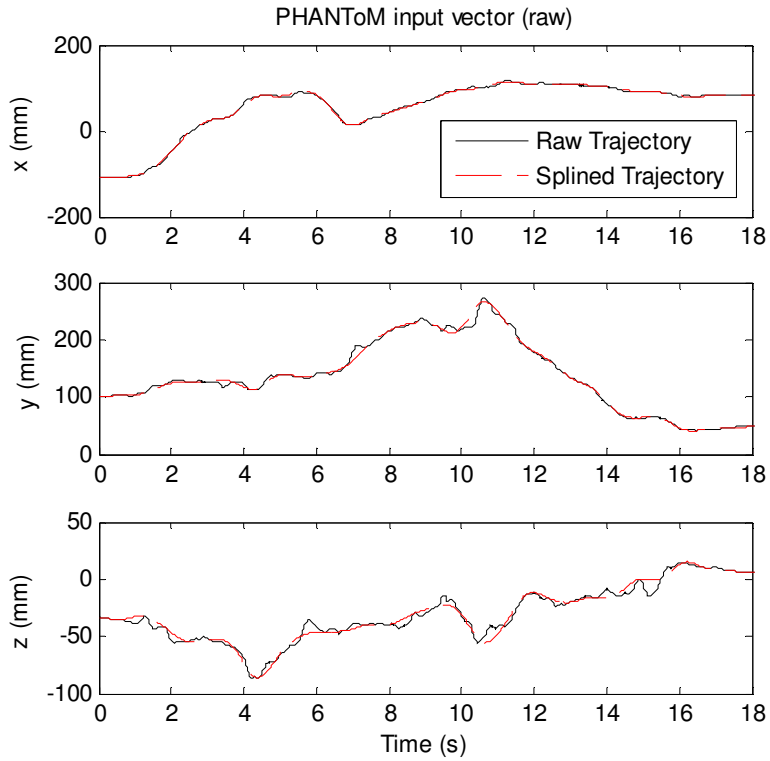

Figure 7.10: 30 Point Spline Over Trajectory

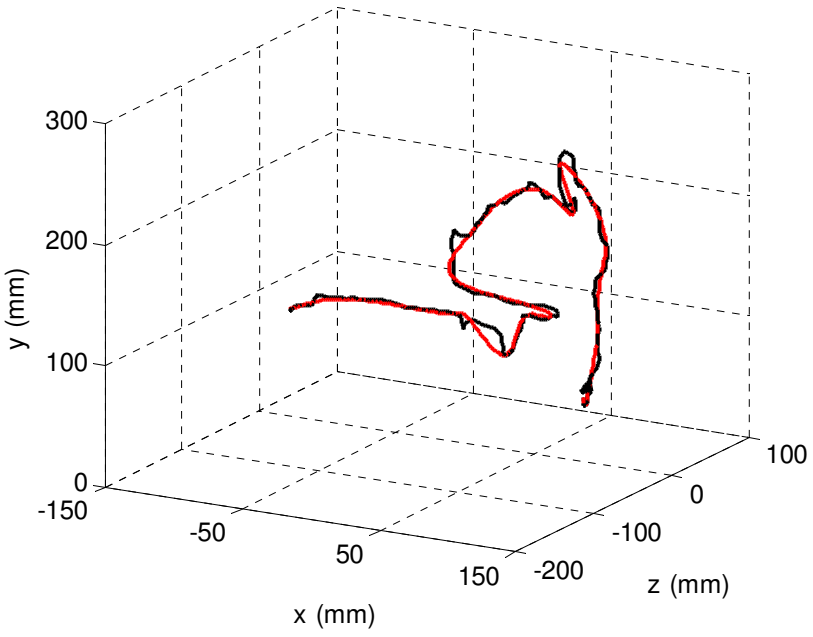

Figure 7.11: 30 Point Spline Over Foot Trajectory, 3D

The 30 point spline fits the original trajectory well, but shows very little improvement over the 20 point spline in Figures 7.8 and 7.9. Any additional spline points will also show the same diminishing returns. A 20 point spline is sufficient for fitting recorded PHANToM trajectories.

After the smoothed trajectory has been resaved over the original file, the initial and final record points are saved into a master file. A master file for each side of the robot contains records of the particular trajectory name (timestamp) and its beginning and endpoints stored in a large array. The beginning and endpoints can be thought of as "stepping stones" because they are the specific points in the environment where a known, good, foothold is known to exist.

The MATLAB script written to read, smooth, and resave leg trajectories is included in Appendix D.

7.4 Trajectory Selection for Playback

Once a successful trajectory has been found and recorded through the motion of a front leg, the rear leg pairs will make their moves based on known, safe trajectories and the "stepping stones" mapped by the operator.

7.4.1 Global Coordinate System

The global coordinates for the robot are split by side. For simplicity, the left side legs and right side legs use inverse coordinates which match the coordinates of

the PHANToM controllers (Figure 7.12). Robot origins O_L and O_R are centered at the shoulders of legs L1 and R1 respectively.

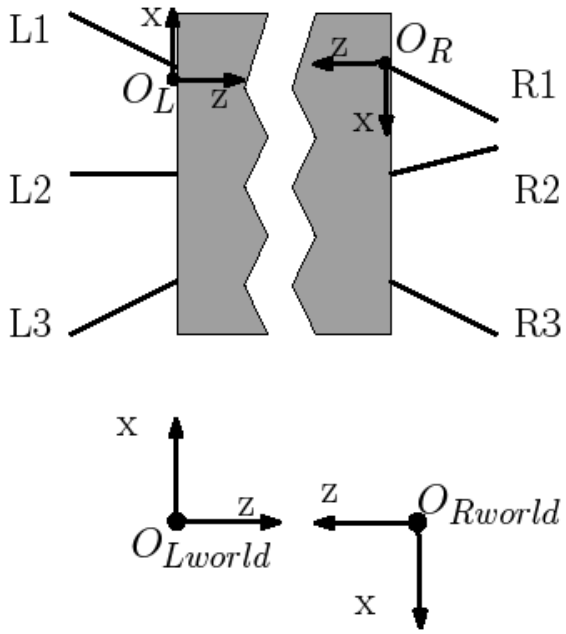

Figure 7.12: Global Coordinates and Leg Naming Convention

The global origins $O_{L,world}$ and $O_{R,world}$ are stationary relative to the inertial reference frame, and are set when the operator switches into the guided-gait mode. Once set, the distance traveled by the robot is recorded each time the body shifts forward, creating vector ${}_w p_r$. Vector ${}_w p_r$ is the vector from the global origin to the respective robot origin, in the world reference frame of O_{world}.

Each trajectory T_i is saved as a list of vectors ${}_r t_i$ from the robot origin O to the respective foot, and as a list of vectors ${}_w p_t$ from the global origin O_{world} to the respective foot (Figure. 7.13).

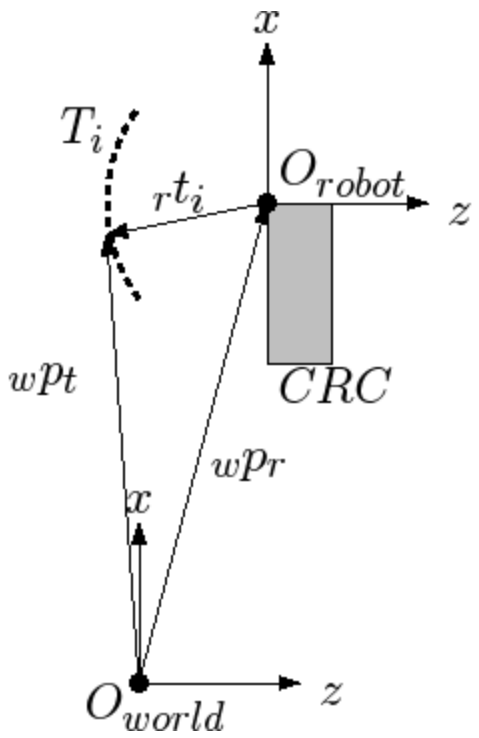

Figure 7.13: Global Gait Vector Relationship

With $_w p_r$ known, and $_r t_i$ the operator-recorded trajectory, the vector $_w p_t$ is simply calculated through vector addition (Equation 7.1).

$$_w p_t = {}_w p_r + {}_w t_i \qquad (7.1)$$

Vector $_w t_i$ is an orthogonal coordinate rotation from the global reference frame *w* to the robot reference frame *r*. In future versions of this guided-gait procedure in which the straight-line walking assumption is not held and the robot body rotates with respect to the inertial reference frame, $_w t_i$ will be required for evaluation of $_w p_t$. Currently, though, $_w t_i$ is equivalent to $_r t_i$ because the robot body is assumed to always be in the same coordinate

reference frame as the global origin because of Assumption 1.

From an overhead view, the master list of stepping stones and trajectories can be visualized as in Figure 7.14.

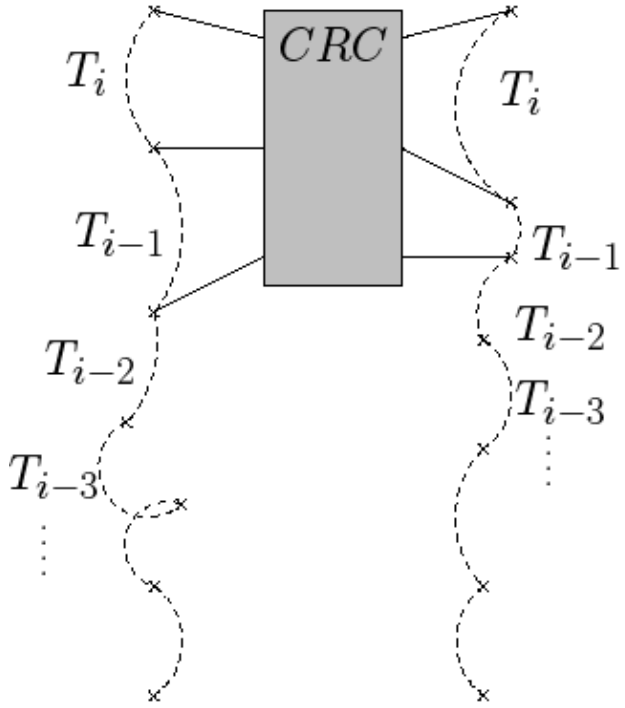

Figure 7.14: "Stepping Stone" Trajectories

Since, through Assumptions 2 and 3, the front feet do not slip or move once placed, and can reach the AEPs of the middle legs, no gap is left between the safe points without a corresponding trajectory. This enables an assured movement of the rear legs.

7.4.2 Leg Selection and Requirements

Once the leg movement sequence has begun, only two legs can move simultaneously. This requirement will

maintain the stability of the robot by maintaining four points of contact with the ground. While only three are necessary for stability, in the actual mission scenario, a fourth point of contact will make the entire platform more robust against slipping on loose footholds.

The gait controller's overall goal while moving rear legs is "Advance each foot as far as possible while stepping only on mapped-out 'stepping stones'". This rule will ensure that each leg reaches forward as far as possible, maintaining the highest feasible forward speed through the search and rescue mission. This goal also maximizes the possible forward body advancement during stance phases, explained below in 7.5.

Legs L3 and R2 will move through their appropriate trajectories first. Then, legs L2 and R3 will move through their trajectories, completing the sequence. No preference is held over moving the L3/R2 set or the L2/R3 set first, except that the two legs moving must not be from the same pair. This method is aligned with Cruse's original WALKNET rules, and the modified version by Wait et al. [3].

Before the first leg moves, the coordinator must first ascertain the foot position relative to the robot shoulder and calculate whether the target stepping stone is within its workspace or beyond the AEP. If the target stepping stone is reachable, the coordinator will play the next trajectory on queue for the leg. If the target stepping stone is beyond the AEP of the leg, the coordinator will

withhold movement until the next gait cycle after the body has shifted forward bringing the target closer to the shoulder joint and within reach. Since the trajectory was first explored and completed by the front leg, Assumption 4 holds that each successive leg pair will be able to traverse the same trajectory.

If more than one successive stepping stone is reachable by a single leg, the coordinator will move the leg to the first one through the proper trajectory, then to the next. This double stepping routine will maintain overall speed by not requiring a one gait cycle delay between two small steps.

7.4.3 Playback Detail

Leg trajectory playback can be physically accomplished simply by using the onboard real-time controller to translate trajectory points into physical foot positions. The onboard controller presently operates two real-time inverse and forward displacement analysis algorithms along with six simultaneous pneumatic displacement controllers. The expansion of the controller to coordinate six legs is a simple Simulink expansion of existing code.

The high-level gait coordinator, running alongside the PHANToM control software, must send trajectory points to the CRC via wireless UDP in the same manner as the PHANToM controller. Through this method, the interface of the gait coordinator to the leg controllers will be a simple software connection.

7.4.4 Conflict Resolution

A situation may arise where debris or other unknown obstacles may fall into the path of the CRC while en route via the guided-gait mode. The conflicting obstacle may upset or impede the intended path of the leg in motion. Direct operator intervention will resolve the conflict.

The gait coordinator will monitor position and pressure sensors from the leg controllers and determine whether the leg has touched down or is ensnared on the environment. If the position error grows too large while the leg is commanded to be on a trajectory, the leg controller will send an error flag to the gait coordinator, signaling it to pause. The trajectory playback must stop and alert the operator that a collision has occurred.

The operator, at this point, will take direct control of the ensnared leg through a PHANToM haptic device. The operator can then 'feel' the environment and work to free the leg while the other five legs hold the robot stationary. When the leg is no longer in conflict with the environment, the operator will then manually guide it to the target stepping stone, recording the trajectory to replace the one which ensnared the leg.

7.4.5 Motion Completion

Once the legs have successfully finished their respective trajectories, the operator must be made aware that the gait cycle can continue. The proposed method is

to provide the operator with a video overlay during guided-gait coordination mode. The overlay will depict the six legs of the robot in either green or red. Before and during the automated swing phase of each leg, the corresponding depiction will appear red. Once the leg has completed its trajectory, the corresponding depiction will change to green (Figure 7.15).

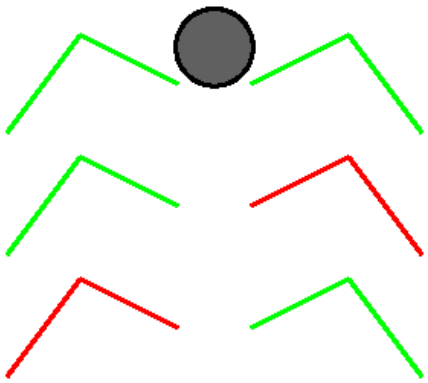

Figure 7.15: Six Legged Status Overlay Example

In the example shown above, the status message inferred by the operator is, “L1/R1 have moved, L2 and R3 have completed their trajectories, and L3 and R2 are not done moving.” Once all four rear legs have successfully moved through their trajectories, all depicted legs become green, and the operator may move on to the next sequence in the gait cycle.

7.5 Body Advancement

Once the four rear legs have moved through the operator guided trajectories (or withheld movement for the cycle), the body must shift forward to maintain overall

forward progress through the search and rescue mission. First, the gait coordination routine will calculate the shoulder joint angles of each leg, and the distance of each foot from the shoulder using cylinder position feedback and a forward displacement algorithm. Since the CRC is moving only straight forward, the gait coordinator can easily analyze how far each foot can move within its workspace in a straight line parallel to the spine before it reaches its PEP.

The operator will give either a verbal or manual command to begin the body advancement procedure. The leg with the shortest amount of travel distance, predetermined by the coordinator, sets the actual distance through which the body can advance. Therefore, if the trajectory playback routine is always moving legs as far forward as possible, the body advancement routine will always start with six legs positioned as far from their PEPs as possible, based on the available stepping stones.

The body then advances by commanding all six feet to move parallel toward the rear of the robot. The length of the forward shift trajectory must be equal to the distance of the leg with the shortest amount of travel. If commanded to go further, one leg will stop when it reaches the end of its workspace while the other 5 legs will continue advancing. Since, through Assumption 2, the feet do not slip, three legs providing thrust on one side of the

robot versus two legs on the other side will induce an undesired body rotation.

7.6 Conclusions

Figure 7.16 below depicts five demo gait cycles through the guided-gait coordinator. During Cycles 2 and 3, the operator only records a half step on the left leg, not able to completely step over an obstacle. The trajectories propagate through successive leg pairs until, in Cycle 5, the double-stepping playback allows Legs L3 and R2 to move through two stepping stones in a single gait cycle, maximizing overall forward advancement speed of the CRC.

While not physically present on the robot testbed, the guided-gait coordinator designed through this research project will be a powerful semi-autonomous tool between direct operator inputs and a central gait coordinator. The method describe herein is applicable not only to the CRC, but to any multi-legged vehicle which traverses unknown terrain via operator input.

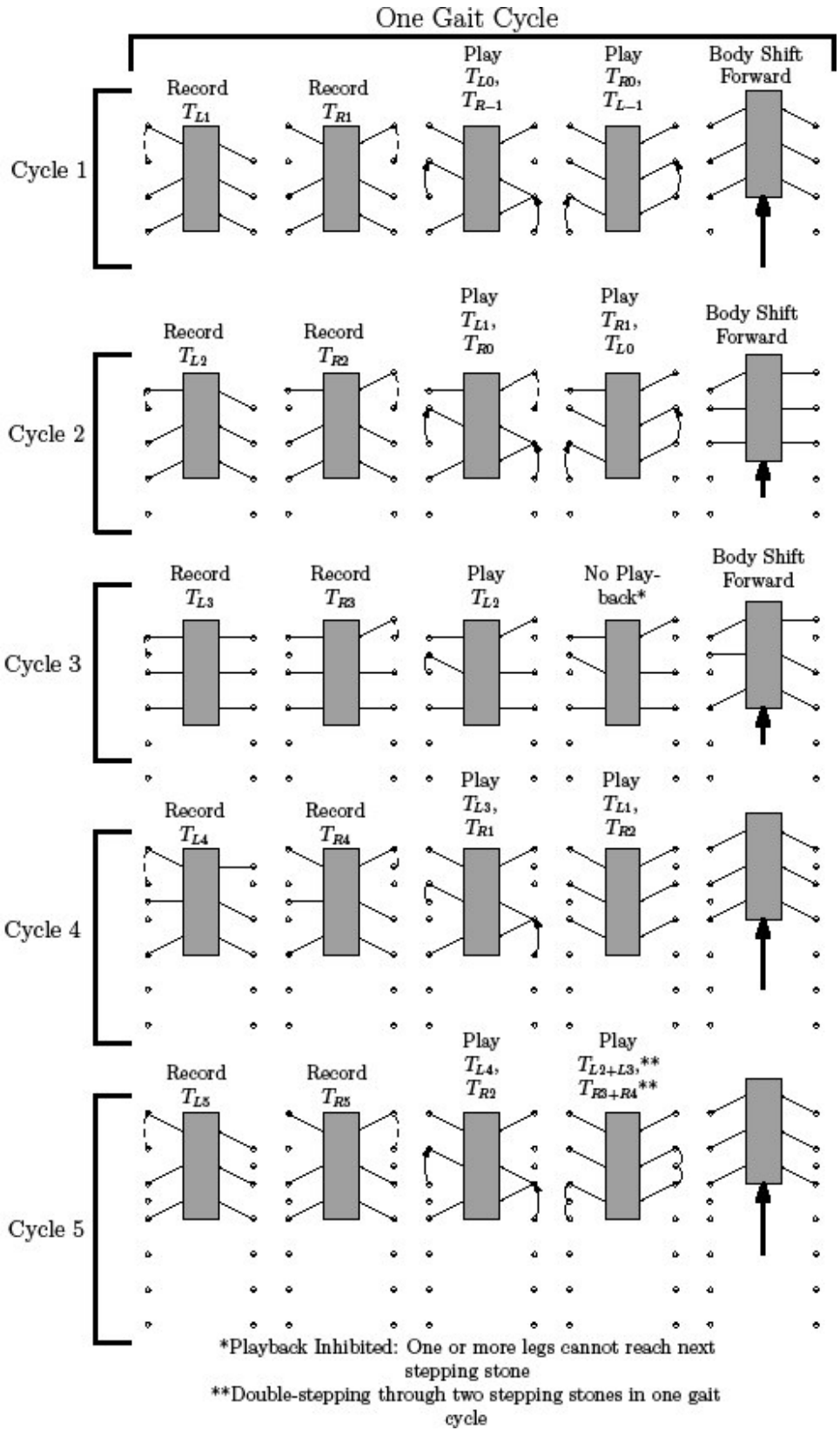

Figure 7.16: Sample Gait Cycles

CHAPTER 8

RESULTS AND CONCLUSIONS

The Compact Rescue Crawler Testbed developed through this research project has met several goals. A two-legged version of the hexapedal concept was designed, fabricated, and tested. New pneumatic actuators featuring embedded position and pressure sensing were developed and installed. Real-time control of the pneumatic cylinders to coordinate foot movement with user inputs has been designed, tested and implemented. Haptic feedback is provided to the operator through the two PHANToMs which provide position input commands. A guided-gait coordination strategy has also been developed and presented.

8.1 Robot Design and Fabrication

The robot design described in Chapter 2 has proved to be a robust and reliable design. Placing the valves as close as possible to the cylinders and using small air lines (0.125 inch) allows little volume for undesirable compression to occur. The joints operate smoothly and very little mechanical interference constricts leg movement.

8.1.1 Recommendations for Future Work

The next major redesign of the CRC should include a larger cross-section spine. The 1 inch 80/20 beam has a

very small torsional cross-section and suffers from high twist angles when large forces are applied by one leg.

When newer, smaller cylinders are available from Sentrinsic, they should replace the current prototype generation cylinders. A cylinder with shorter top and bottom endcaps will improve the overall range of motion of the leg. Both front and rear cylinder mounting shoulders for Cylinders L1 and R1 should be redesigned to eliminate the slight mechanical interference exhibited on the current version.

A new Signal Routing PCB should be designed using smaller components and an onboard DC power supply. Surface mount op-amps and resistors should be used in conjunction with test loops to allow for rapid system diagnoses.

8.2 Leg Control

The force based position controller developed through this project accurately positions the robot feet while tracking the input commands from the operator. Control requirements were < 10% position tracking error through both the swing and stance phases of the gait, robustness to external loading and mechanical failures, and stability under all loading scenarios.

The initial PD controller performed unsatisfactorily with a large tracking error during stance phase. Velocity feed-forward input greatly improved tracking during swing phase, and a stroke velocity damping term improved transient responses.

A gain scheduler was added to the PD controller to improve system response under the extreme loading variations encountered during a walking gait. Initially, a gain was applied to the pressure differential. One of four gains was selected based on the pressure differential sign and the position error sign. This proved a great improvement in tracking, but added an undesired “hopping” effect as the position error changed sign, instantaneously changing the additive control effort.

A revised gain scheduler was developed to continue adding supplementary control effort based on the differential pressure and position error. The new scheduler scaled the additive effort by the magnitude of the position error and the differential force on the actuator. The “hopping” effect disappeared because the supplementary control effort decreased with position error, so once the actuator reached its commanded stroke length, the correct pressure differential was in place across the piston.

The same controller was applied to all six cylinders and the gains were tuned for stability and response. The results of the PD + v_{ff} + dfe controller fit within the control requirements, yielding a position tracking error < 10% through the entire gait cycle, stable system responses, and robust to the varied loads experienced by each different cylinder. Figure 8.1 below, again, demonstrates the satisfactory position responses through a gait cycle.

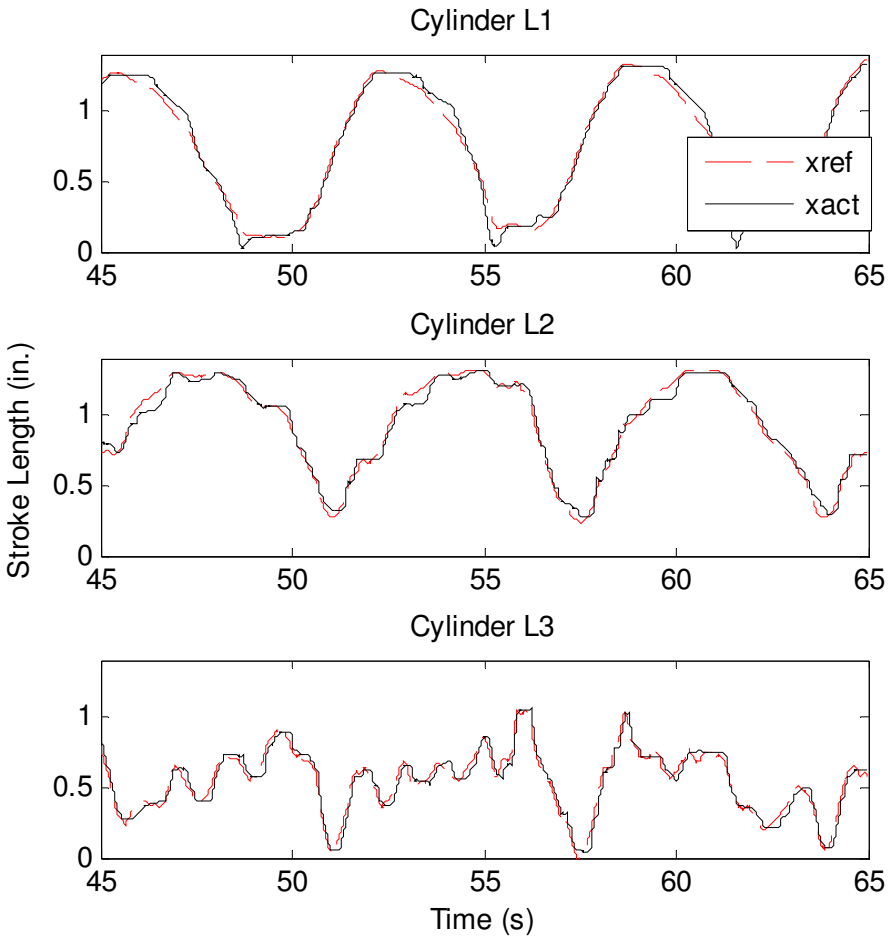

Figure 8.1: Full Controller on Left Leg Through Numerous Gait Cycles

8.2.1 Recommendations for Future Work

While the current controller works well for each individual leg and for general robot movement, a new controller should be developed based on more modern control theories to achieve a tracking error magnitude < 5%. This smaller tracking error will improve the haptic "feel" of the robot and yield more desirable system responses. A

model-reference controller may provide a good control effort, but will be difficult due to the modeling inaccuracies bound to occur when working with pneumatic systems.

The dynamics of the entire robot body should also be modeled for control development. Since, when standing, the robot body is essentially the platform of a parallel manipulator, a higher level body controller could assist in leg coordination and subdue some of the conflicting lateral forces produced by the operator while standing, and more effectively coordinate the pulling forces during stance phase body movement.

8.3 Operator Interface

A prototype operator workstation was built to seat the operator with two PHANToMs and a head-mounted display. The PHANToMs are mounted on movable arms, adjustable to fit any operator comfortably. The motion tracker mounted to the head-mounted display provides commands to the PTZ camera on the robot. This setup moves the camera to match the head orientation of the operator, placing the viewpoint of the operator on the front of the robot.

Haptic feedback is produced through the two PHANToM controllers. The operator is provided with a directional force relative to the position error. The scaled position error yields a “spring” sensation to the operator as if the endpoints of the PHANToMs were attached to the robot feet by springs. Due to the spring sensation, improved tracking

control will maintain a low position error, therefore a lower ambient force at the PHANToM endpoint. Tighter tracking will produce a crisper rise in force when the leg encounters an obstacle.

The current state of the haptic feedback allows the operator to feel large obstacles, but not the ground itself due to the smoothness of the position controller. Large obstacles must also be much more massive than the robot. Smaller obstacles are simply pushed out of the way by the robot's powerful legs.

8.3.1 Recommendations for Future Work

The operator workstation should be optimized through research on human factors and workstation layouts. The positions of the PHANToMs, size and shape of the chair, and posture of the operator should be optimized for long-term continuous usage.

The video overlay viewed through the head-mounted display must also be optimized for ease of access and information flow. Researchers at NCAT are currently exploring this, but final implementation must be made with the overall robot workstation controller.

Specifically, a study should be made evaluating the effectiveness of voice commands over strategically placed buttons for repetitive operational commands. The display style of mission-critical data should be evaluated as well. Optimal camera position must also be determined based on mission parameters and practicality.

Future work focused on the CRC haptic interface should be coupled to the overall control research. Since the robot is a bilateral teleoperated device, the effectiveness of the robot controller can affect the performance of the haptic feedback.

The haptic control should be improved to the point where the operator can bring the foot on a collision trajectory and exert only a small amount of force on the obstacle before stopping. A current metric for this experiment is to exert less than 1/6 of the robot weight into the obstacle during swing phase movement.

8.4 Guided-Gait Coordination

The guided-gait coordination routine designed through this research shows one method by which the operator is capable of creating specifically guided foot trajectories which propagate to subsequent leg pairs. The network of "stepping stones" mapped out through the front legs are the known safe, stable footholds in the unknown environment. Each following leg is constrained to start and finish its foot trajectory on such a point and follow the recorded trajectory.

The guided gait trajectory designed herein allows for straight-line motion on level terrain. The basic order of operations of the gait routine combine simple operator commands in harmony with cues from the gait coordinator. Once the gait routine has begun, the operator moves and records the swing phases of the front two legs. Then, a

simple command, verbal or manual, commands the coordinator to move the rear four legs. A simple visual cue is displayed to the operator indicating the status of each leg being automatically positioned. The operator then gives another simple command to begin body advancement, a six-legged coordinated stance phase. Once complete, the operator will begin the sequence again.

8.4.1 Recommendations for Future Work

A global control architecture must be developed to execute the guided-gait coordination routine. Recorded trajectories should be more thoroughly analyzed for signs of obstacle avoidance so the spline points can be more efficiently placed to reduce the overall time of the swing trajectory when played through trailing legs. Experiments should be done to validate the straight-line effectiveness of the gait routine.

Advanced work could combine the onboard camera with the guided-gait coordination routine. To enable body rotation, the mapped “stepping stone” points can be enlarged to areas where the camera detects no obstacles and stable ground. The mapped trajectories must also be manipulated to avoid known obstacles after the robot body has rotated in the global coordinate frame.

The finale of the guided-gait development should allow the robot to traverse chaotic 3D terrain while changing body orientation and elevation. Effective synergy of the man-machine interface, coupling haptic, visual, and aural

feedback and sensations will greatly expand the reach of search and rescue operations in times of dire need.

8.5 Academic Contributions

Several areas of this research project have yielded contributions to the engineering and academic communities. This robot leg design and construction yielded two rugged, powerful, and maneuverable leg structures. The design and construction of the operator interface workstation yielded a configurable basis for future work on this high degree of freedom fluid power testbed.

The force-based position control algorithm controlling the six pneumatic cylinders is simple, robust, and stable. The ability to track direct user inputs while operating under a wide variety of loading conditions is a significant contribution.

The outline and development of the Guided-Gait Coordination routine contributes to the engineering community by allowing hybrid control of six or more legs by allowing the operator to directly control the two leading legs. The application of this routine to a rescue robot will allow an operator to guide the vehicle through unknown terrain using haptic and visual feedback as guides.

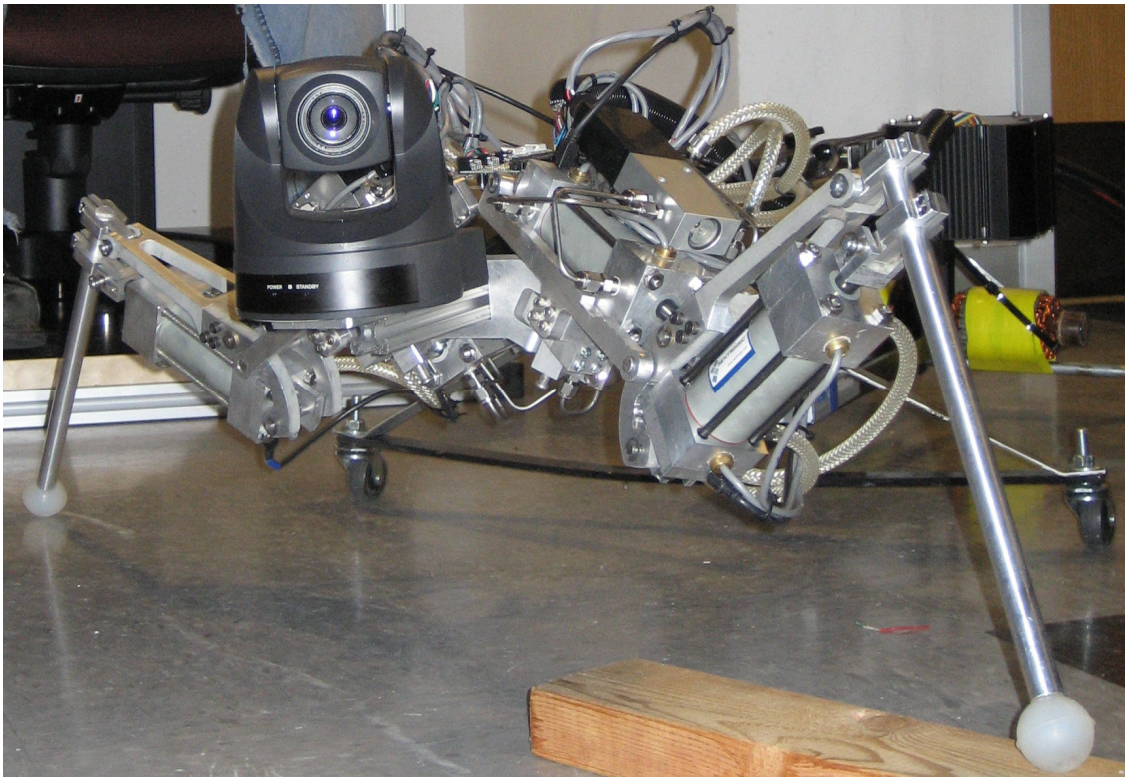

Figure 8.2: Compact Rescue Crawler

APPENDIX A:

SOLUTION OF LINEAR TRIGONOMETRIC EQUATION

3.1 Linear trigonometric equation

In the generalized Puma reverse displacement analysis it is necessary to solve a linear trigonometric equation for angle θ,

$$a\cos\theta + b\sin\theta - d = 0$$

where a, b, and d are known coefficients. The basic identities required are

$$\begin{aligned} 1 &= \cos^2\alpha + \sin^2\alpha \\ \cos(\alpha \pm \beta) &= \cos\alpha\cos\beta \mp \sin\alpha\sin\beta \\ \sin(\alpha \pm \beta) &= \sin\alpha\cos\beta \pm \cos\alpha\sin\beta \end{aligned}$$

Dividing the linear expression by $\sqrt{a^2+b^2}$ and defining angle γ by

$$\cos\gamma \equiv \frac{a}{\sqrt{a^2+b^2}}, \quad \sin\gamma \equiv \frac{b}{\sqrt{a^2+b^2}}$$

yields two angles,

$$\begin{aligned} \cos(\theta-\gamma) &= \frac{d}{\sqrt{a^2+b^2}} \\ \sin(\theta-\gamma) &= \pm\frac{\sqrt{a^2+b^2-d^2}}{\sqrt{a^2+b^2}} \end{aligned}$$

Denote the two angles as $\psi^{(\pm)} = \theta^{(\pm)} - \gamma$ so that there are two solutions $\theta^{(\pm)}$ with $(\pm)$ denoting the upper or lower signs in the sin term, see Figure 3. Expanding out $\cos\theta^{(\pm)} = \cos\left(\gamma + \psi^{(\pm)}\right)$ and $\sin\theta^{(\pm)} = \sin\left(\gamma + \psi^{(\pm)}\right)$ yields the desired forms,

$$\begin{aligned} \cos\theta^{(\pm)} &= \frac{ad \mp b\sqrt{a^2+b^2-d^2}}{a^2+b^2} \\ \sin\theta^{(\pm)} &= \frac{bd \pm a\sqrt{a^2+b^2-d^2}}{a^2+b^2} \end{aligned} \tag{1}$$

The two solutions correspond to whether the upper signs or the lower signs are selected,

$$\theta^{(\pm)} = ATAN2(\sin\theta^{(\pm)}, \cos\theta^{(\pm)})$$

where $ATAN2(\sin\theta, \cos\theta)$ is an arctangent function that returns a unique angle within the range of $-\pi < \theta \leq \pi$. In contrast, the $\arctan\theta$ function returns an angle in the range of $-\frac{\pi}{2} < \theta \leq \frac{\pi}{2}$ and generally will not be used. Note that the two angles can be real and distinct, repeated, or complex according whether $a^2+b^2-d^2$ is positive, zero, or negative.

A *singularity* in the solution occurs when $a = b = 0$. For the linear equation to be consistent then $d = 0$ and the equation is satisfied for all values of θ yielding an infinite number of solutions. The expressions for $\cos\theta$ and $\sin\theta$ become indeterminate, i.e. $\frac{0}{0}$. When this occurs for the solution of any joint displacement then the robot is said to be in a *displacement singularity*, a situation that causes difficulty in the control of robots.

APPENDIX B

SIMULINK CONTROL DIAGRAMS

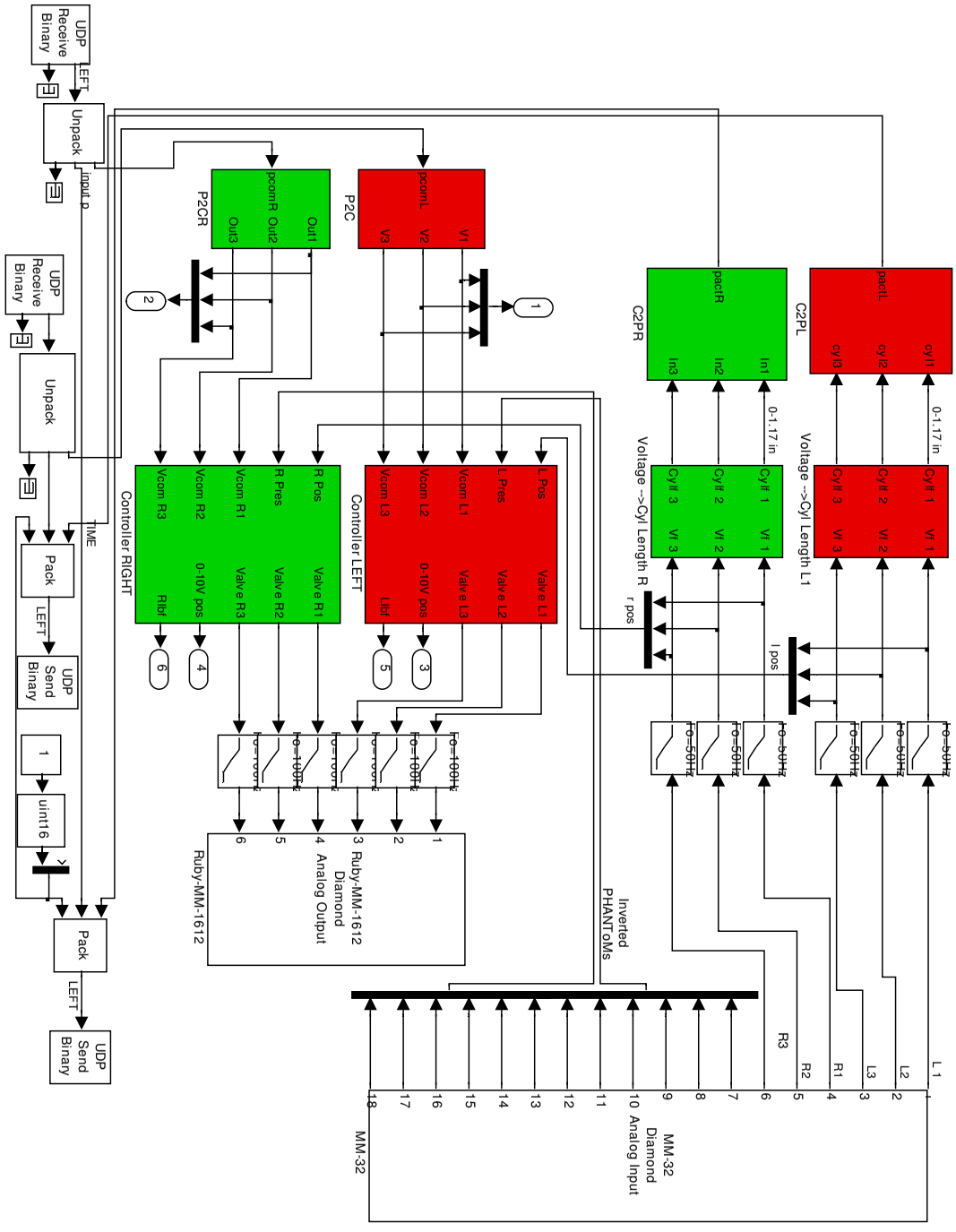

Figure B.1: Main Simulink Control Diagram

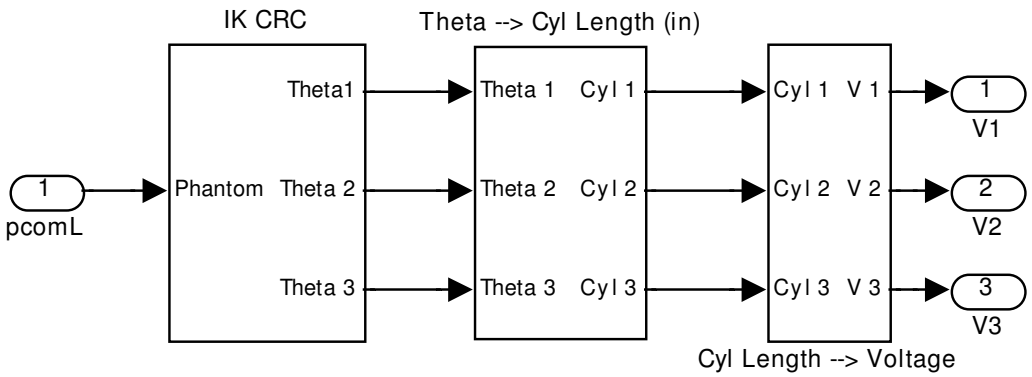

Figure B.2: PHANToM Input to Stroke Length Voltage Transformation (Simulink)

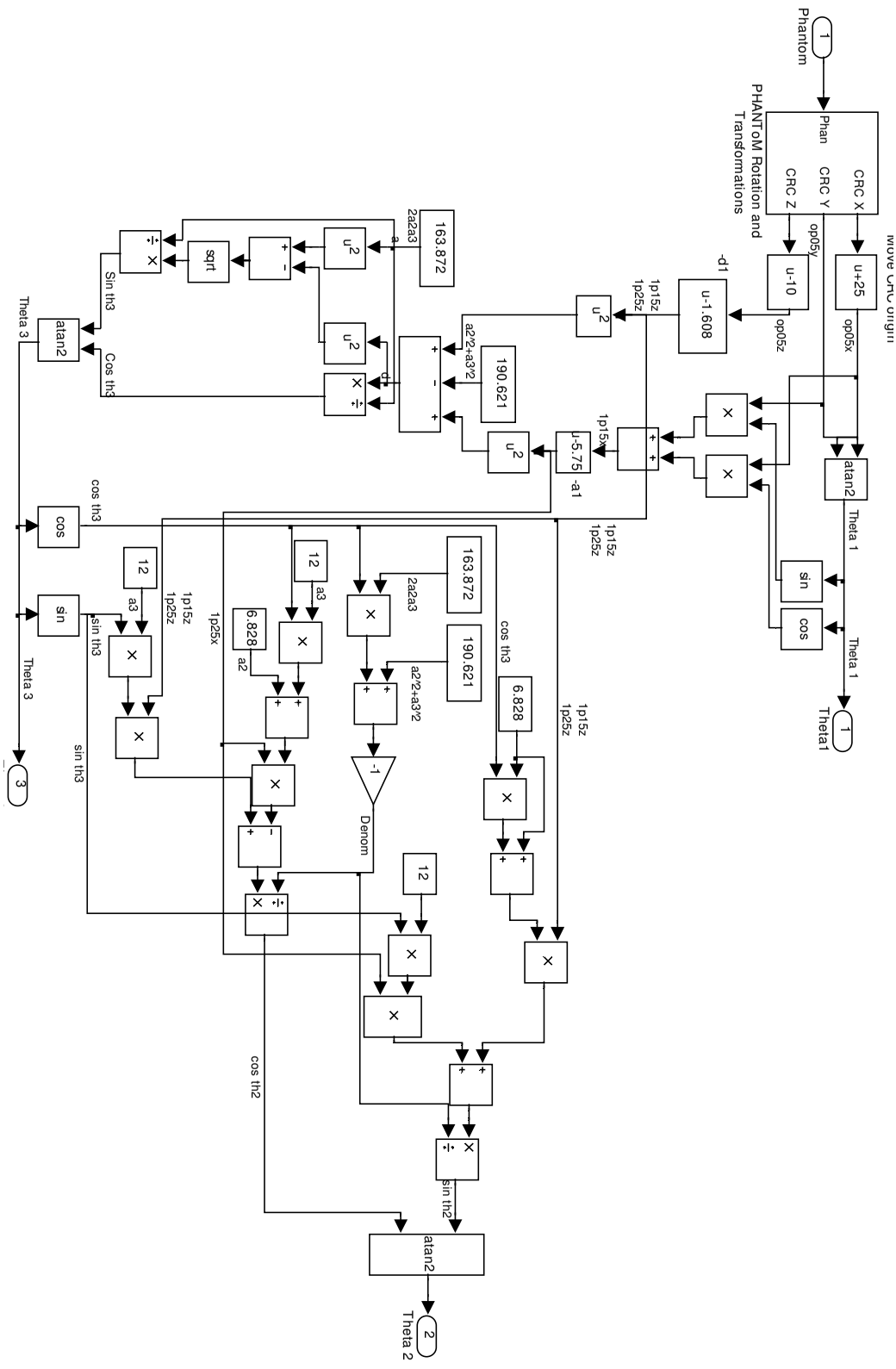

Figure B.3: Inverse Displacement Algorithm from PHANToM Vector Input to Joint Angles (Simulink)

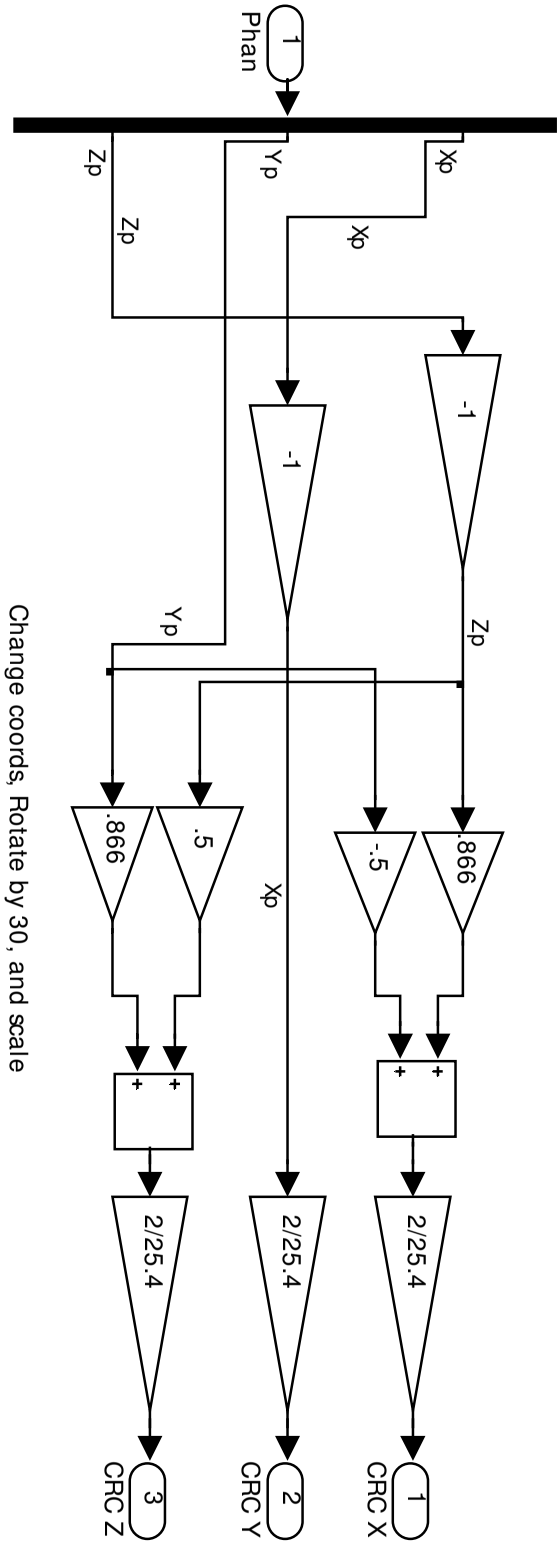

Figure B.4: PHANToM Vector Input Transformation and Rotation (Simulink)

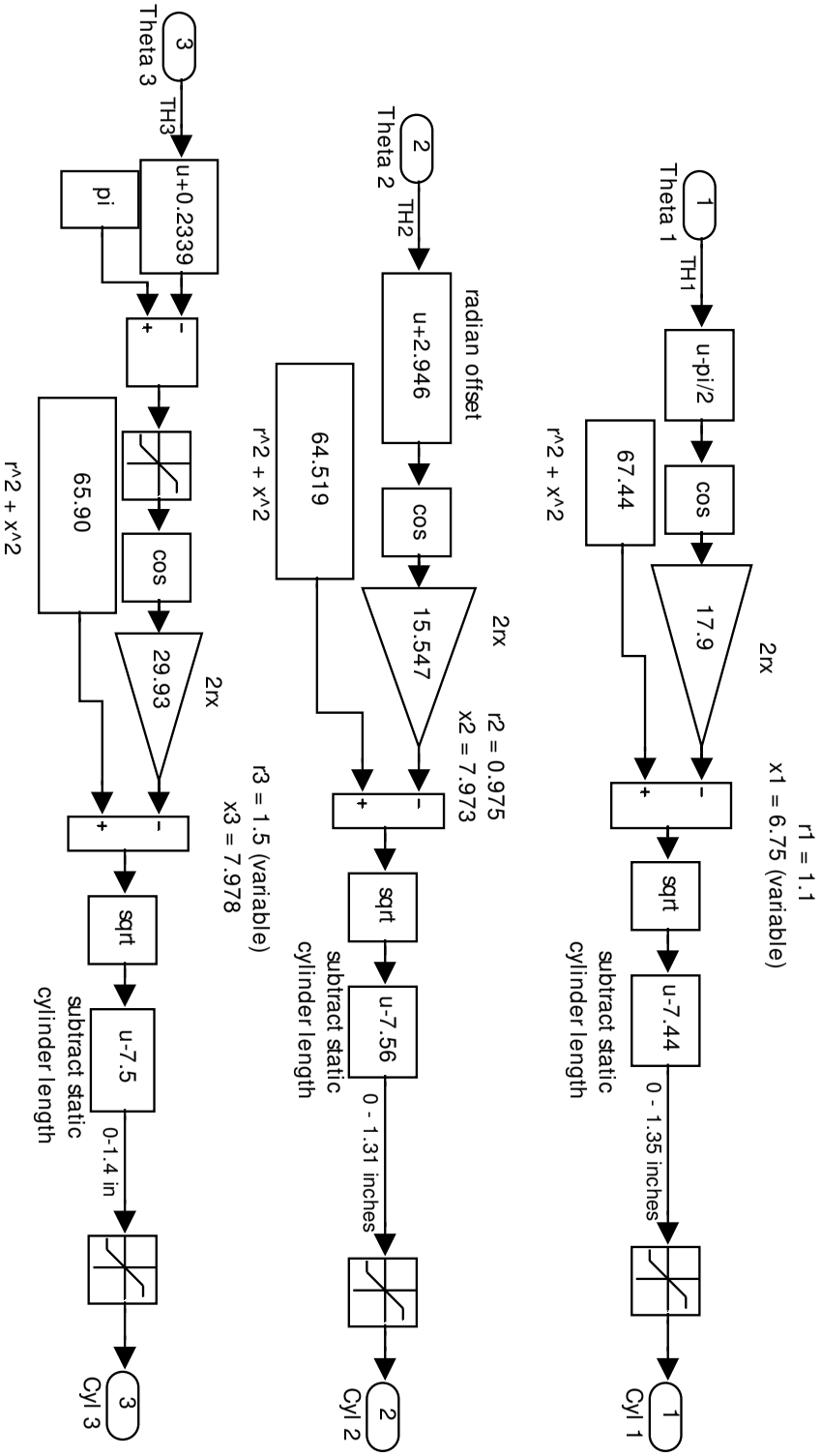

Figure B.5: Joint Angle to Stroke Length Conversion (Simulink)

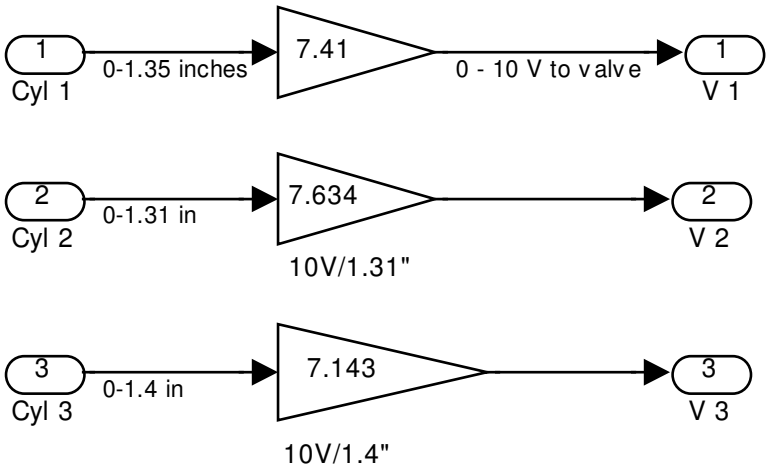

Figure B.6: Stroke Length to 0-10V Conversion

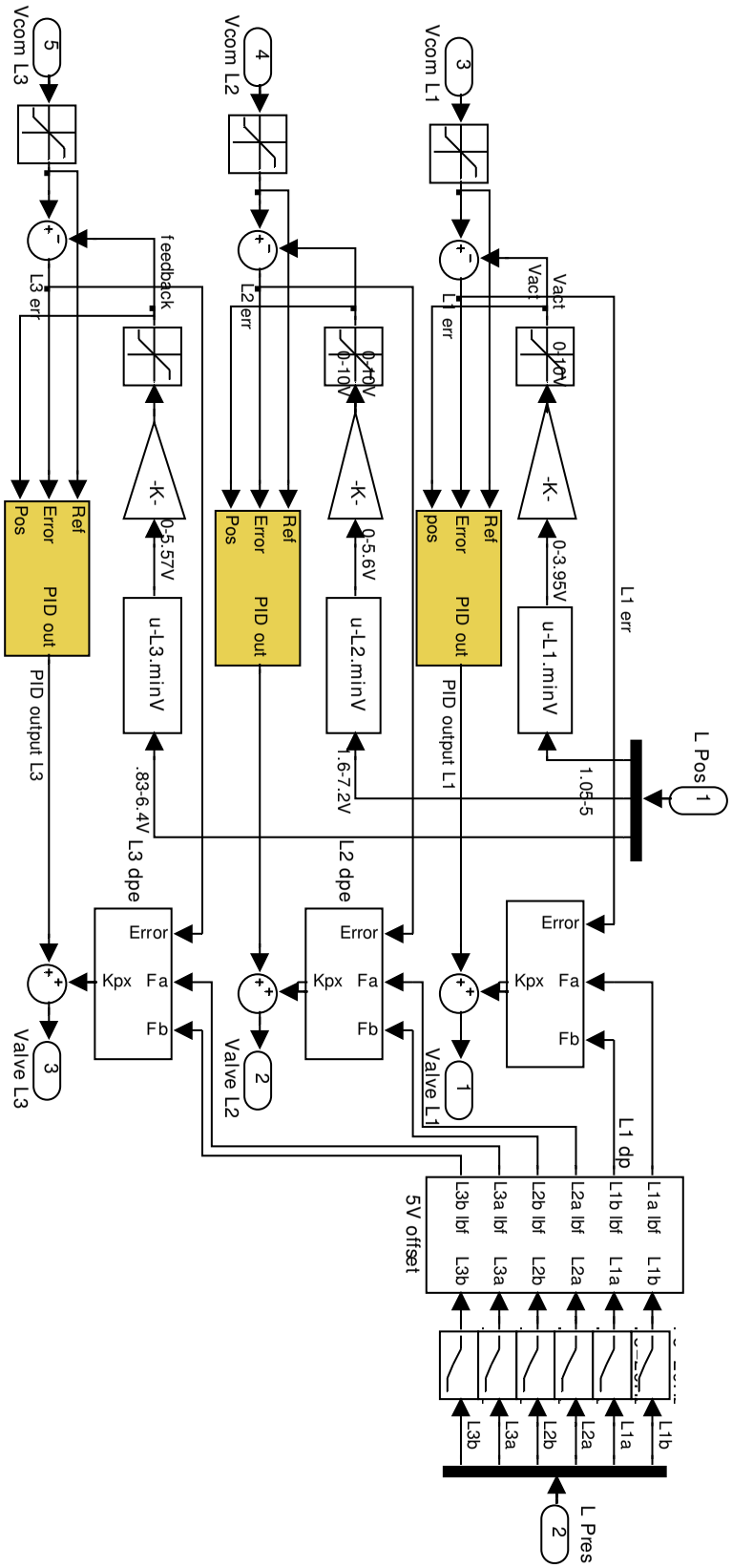

Figure B.7: Controller Layout, Left Leg (Simulink)

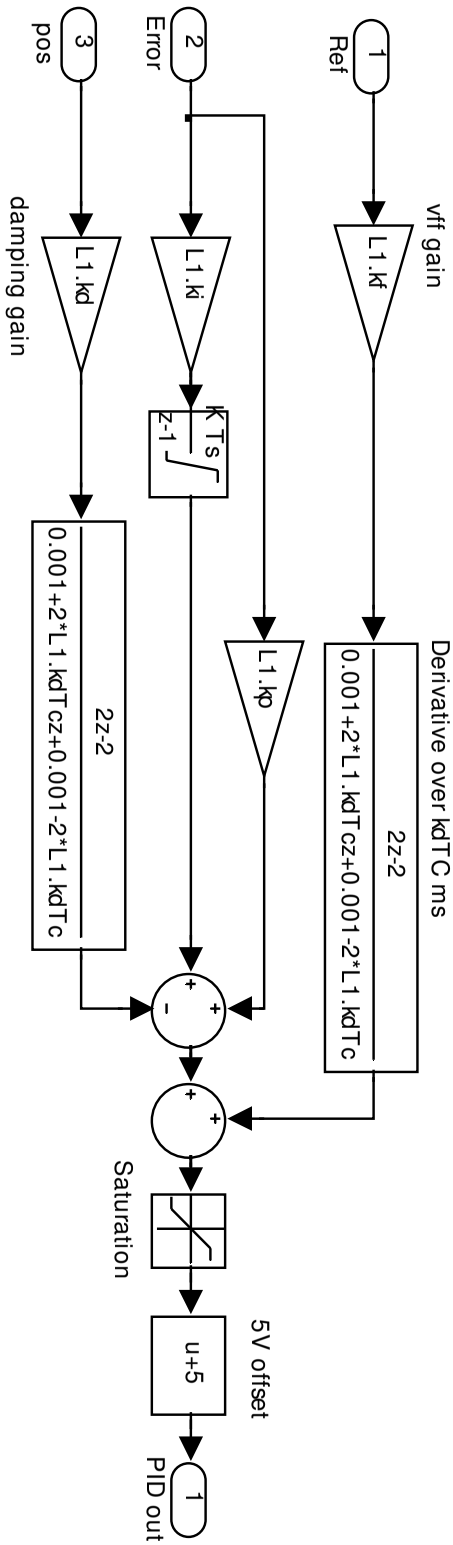

Figure B.8: PID Controller (Simulink)

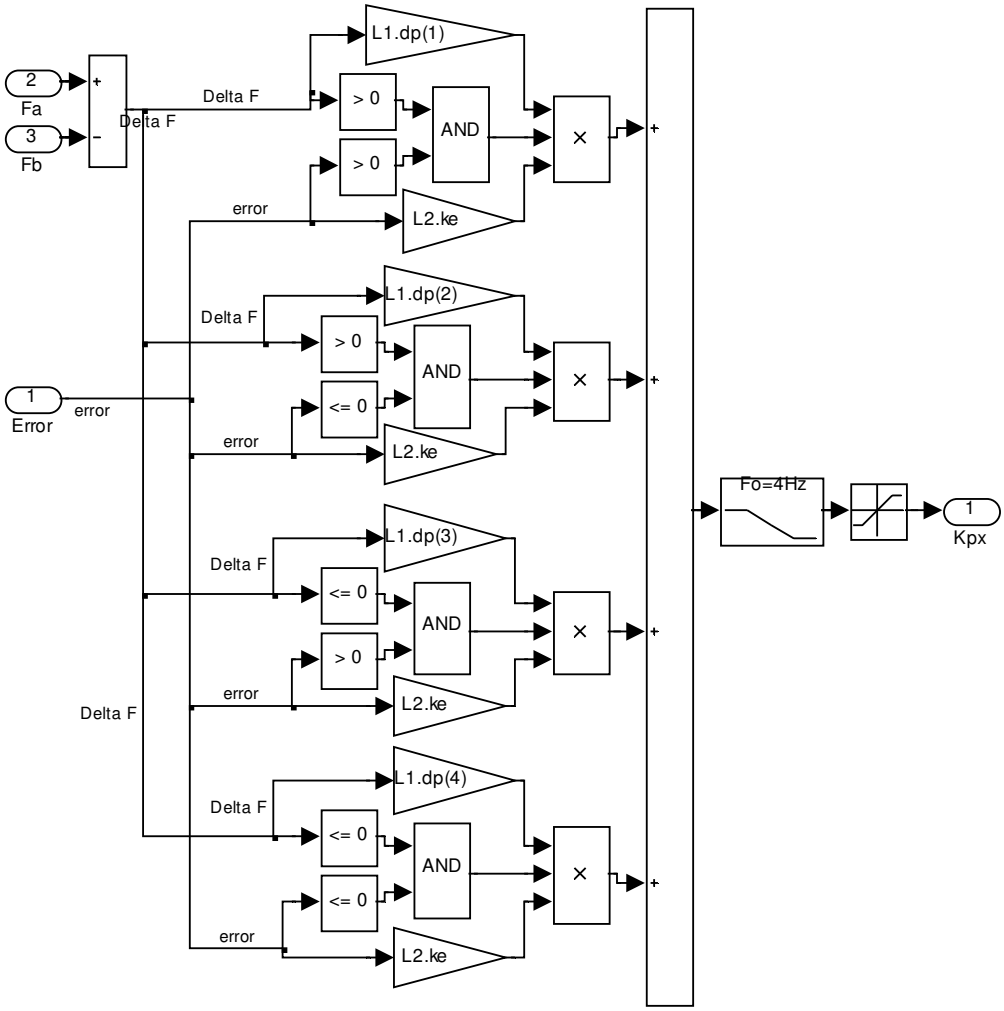

Figure B.9: Differential Force Gain Scheduler (Simulink)

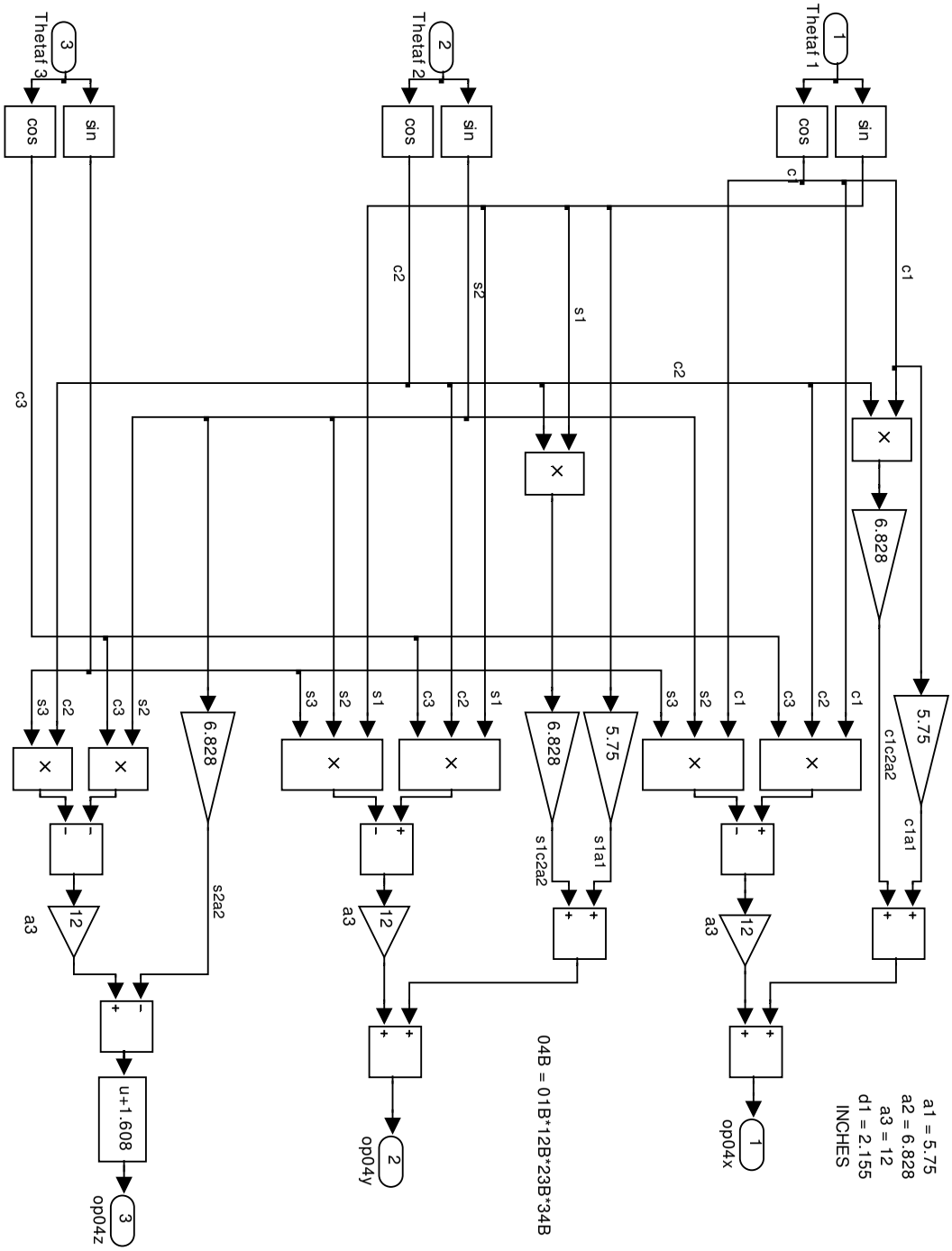

Figure B.10: Forward Displacement Algorithm (Simulink)

APPENDIX C

C++ CODE FOR PHANTOM HAPTIC INTERFACE

```
/*******************************************************
     Author:        Matt Kontz <mkontz@mail.com>
     Lab:      IMDL ME nGaTech
     Created:  February 7, 2005
/*******************************************************
     PHANToM/Omni coordinates
          x-axis -> to the right
          y-axis -> up
          z-axis -> towards user
/*******************************************************/
#include <stdio.h>
#include <conio.h>
#include <assert.h>
#include <iostream.h>         // for cout, cerr
#include <iomanip.h>          // for setw, setpreci sion
#include <fstream.h>          // for writing to files
#include <windows.h>          // WIN32 Threads
#include <time.h>
#include <HD/hd.h>
#include <HDU/hduVector.h>
#include <windows.h>          // WIN32 Threads
#include "Callback.h"         //   local   header   file   /w
callback
#include "DataStorage.h" // local storage
```

```
#include "DataStruct.h"       //  Phantom,  and  Hal  data
structures
#include "ArgStruct.h"        // ThrArgs & CallbackArgs
#include "Sock.h"
void sendP2CThread(void*);
void recvC2PThread(void*);
typedef struct
{
     //internal device handle
     HHD handle;
     const char* name;
     //phantom data
     DataStorage* data;
     HANDLE mutex;
     HANDLE sendThread;
     HANDLE;
     DWORD sendThreadID;
     DWORD recvThreadID;
     udpSocket* sendP2C; //P2C
     udpSocket* recvC2P; //C2P
     FILE* logFile;
     bool writeLog;
} Phantom;
typedef struct
{
     Phantom phanL;
     Phantom phanR;
```

```
} UserData;
double pi = 3.1415926535897932;
void handleDev(Phantom phan)//HHD handle, DataStorage*
data, udpSocket* recv, udpSocket* send)
{
     unsigned short int mode = 1;
     hduVector3Dd Ph_Pos;
     hduVector3Dd Ph_Vel;
     HDdouble Ph_Phi;
     hduVector3Dd Ph_Theta;
     hdMakeCurrentDevice(phan.handle);
     hdEnable(HD_FORCE_OUTPUT);
     hdBeginFrame(hdGetCurrentDevice());
     hdGetDoublev(HD_CURRENT_POSITION, Ph_Pos);
     hdGetDoublev(HD_CURRENT_VELOCITY, Ph_Vel);
     hdGetDoublev(HD_CURRENT_GIMBAL_ANGLES, Ph_Theta);
     Ph_Phi = 3*pi/2 + Ph_Theta[2];
     static const HDdouble kspring = 0.06;   // N/mm
     hduVector3Dd CRC_Pos;
     hduVector3Dd CRC_For;
     HDdouble CRC_phi;
     hduVector3Dd Delta_Pos;
     hduVector3Dd Force;
     CRC_Pos = phan.data->getCRCPos();
     CRC_For = phan.data->getCRCFor();
     CRC_phi = phan.data->getCRCPhi();
     Delta_Pos = Ph_Pos - CRC_Pos;
```

```
     hduVecScale(Force, Delta_Pos, -kspring);
    hdSetDoublev(HD_CURRENT_FORCE, Force);
     phan.data->setPhanData(Ph_Pos,   Ph_Vel,   Ph_Phi,   0,
mode);
     hdEndFrame(hdGetCurrentDevice());

     ::ResumeThread(phan.sendThread);
}
HDCallbackCode Trigger(void *pUserData)
{
     UserData args = *((UserData*)pUserData);
     handleDev(args.phanL);
     handleDev(args.phanR);
     return HD_CALLBACK_CONTINUE;
}
//foreign IP address
char *forIP = "192.168.1.111";
//char *forIP = "192.168.1.1";
void  initPhantom(Phantom*  phan,  const  char*  name,  int
sendP, int recvP)
{
     //setup log file
     //logFile name convention
     phan->writeLog = false;
     phan->mutex = CreateMutex(0, false, 0);
     phan->name = name;
     //init the internal device handle
```

```
	phan->handle = hdInitDevice(name);
	//initialize all of the sockets
	phan->sendP2C = new udpSocket(forIP, sendP, sendP+1);
	phan->recvC2P = new udpSocket(forIP, recvP+1, recvP);
	phan->data = new DataStorage();
	phan->recvThread = ::CreateThread(
		NULL,0, (LPTHREAD_START_ROUTINE) recvC2PThread,
		(LPVOID) phan,0,(LPDWORD) &phan->recvThreadID);
	::SetThreadPriority(phan->recvThread ,15);
	phan->sendThread = ::CreateThread(
		NULL,0,(LPTHREAD_START_ROUTINE) sendP2CThread,
		(LPVOID) phan,0,(LPDWORD) &phan->sendThreadID);
	::SetThreadPriority(phan->sendThread,15);
}
int main(int argc, char* argv[])
{
	//two phantoms, left and right
	UserData userDat;
	//phantom schedule handler
	HDSchedulerHandle hServoCallback;

	initPhantom(&userDat.phanL, "Lefty", 26401, 23201);
	initPhantom(&userDat.phanR, "Righty", 26501, 23301);
	//Get initialization data from CRC
	PhanStruct Get_Pos;
	memset(&Get_Pos, 0, sizeof(PhanStruct));          //
empty structure
```

```
     Get_Pos.mode = 3;                                    //  3
ques for CRC pos
     Get_Pos.x = 1;
     Get_Pos.y = 2.2;
     Get_Pos.z = 3.33;
     Get_Pos.flags = 5;
     CRCStruct Reply;
     memset(&Reply, 0, sizeof(CRCStruct));        //   empty
structure
     cout << "TEST MESSAGE." << endl;
     cout << "x = " << Reply.x  << endl;
     cout << "y = " << Reply.y  << endl;
     cout << "z = " << Reply.z  << endl;
     //clear the phantom data
     userDat.phanL.data->setCRCData(Reply);
     userDat.phanR.data->setCRCData(Reply);

     cout << "Next incoming message from CRC" << endl;
     cout << "x = " << Reply.x  << endl;
     cout << "y = " << Reply.y  << endl;
     cout << "z = " << Reply.z  << endl;
     //set the phantom callback
     hServoCallback = hdScheduleAsynchronous(Trigger, (void
*) &userDat, HD_MAX_SCHEDULER_PRIORITY);
    hdStartScheduler();
    // create a com port file for testing CTS and DSR
     char ch;
```

```
     short bRet, bCont, bNoChange;
     Phantom * p = 0;
     DWORD dwStatus, dwNextSwitch;
     HANDLE   fIn   =   CreateFile("com1",GENERIC_READ   |
GENERIC_WRITE, 0, NULL,
                                OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (fIn == INVALID_HANDLE_VALUE) {
          MessageBox(NULL, "Open of com1 failed", "Error",
MB_OK); }
    dwNextSwitch = 0;
     bCont = TRUE;

     printf("Press   'z'   or   'x'   to   record   data   from
phantom\n");
    printf("Press any other key to quit.\n\n");
     while (bCont)
     {
          bNoChange = TRUE;
          while (bNoChange) {
               if (kbhit()) {
                 bNoChange = FALSE;
                 ch = getch();
               if (ch == 'x') {
                 p = &userDat.phanR; }
              else if (ch == 'z') {
                 p = &userDat.phanL; }
```

```
                else {
                      bCont = FALSE; } }
                 if (!bNoChange) break;
              if     (fIn     !=     INVALID_HANDLE_VALUE     &&
GetTickCount() > dwNextSwitch) {
                   bRet = GetCommModemStatus(fIn, &dwStatus);
                   if (dwStatus & MS_CTS_ON) {
                      p = &userDat.phanL;
                      dwNextSwitch = GetTickCount() + 1000;
                      bNoChange = FALSE; }
                   else if (dwStatus & MS_DSR_ON) {
                      p = &userDat.phanR;
                      dwNextSwitch = GetTickCount() + 1000;
                      bNoChange = FALSE; } }
             Sleep(1); }
             WaitForSingleObject(p->mutex, INFINITE);
             if (p->writeLog)
             {
                  //stop rec
                  printf("Stopped recording: %s\n", p->name);
                  p->writeLog = false;
                  fclose(p->logFile);
                  p->logFile = 0;
             }
             else
             {
                  time_t seconds = time(0);
```

```
                char fname[255];
                sprintf(fname,    "%s_%d_v1.txt",    p->name,
seconds);
                printf("Started recording: %s into %s\n", p-
>name, fname);

                p->writeLog = true;
                p->logFile = fopen(fname, "w");
            }
            ReleaseMutex(p->mutex);
        }
        printf("test\n");
       // close the com port test handle
        if (fIn != INVALID_HANDLE_VALUE) CloseHandle(fIn);
        //stop schedule and distable phantoms
       hdStopScheduler();
        hdDisableDevice(userDat.phanL.handle);
        hdDisableDevice(userDat.phanR.handle);
        /// Clean up ///
        // Destroy Thread
        ::TerminateThread(userDat.phanL.recvThread,userDat.pha
nL.recvThreadID);
        ::TerminateThread(userDat.phanL.sendThread,userDat.pha
nL.sendThreadID);
        ::TerminateThread(userDat.phanR.recvThread,userDat.pha
nR.recvThreadID);
```

```
	::TerminateThread(userDat.phanR.sendThread,userDat.pha
nR.sendThreadID);
	//close open files
	if (userDat.phanL.logFile != 0)
	{
		fclose(userDat.phanL.logFile);
	}
	if (userDat.phanR.logFile != 0)
	{
		fclose(userDat.phanR.logFile);
	}
	return 0;
}
void sendP2CThread(void* args)
{
	Phantom* p = (Phantom*)args;
	PhanStruct msg;
	char line[255];
	while (true)
	{
		::SuspendThread(p->sendThread);
		//populate msg with new Phantom data
		msg = p->data->getPhanData();
		p->sendP2C->send((char *) &msg, sizeof(msg));

		WaitForSingleObject(p->mutex, INFINITE);
		//write out file data
```

```
          if (p->writeLog && p->logFile != 0)
          {
               //PhanStruct ps = p->data->getPhanData();
               sprintf(line,  "%.2f  %.2f  %.2f\n",  msg.x,
msg.y, msg.z);
               fputs(line, p->logFile);
               fflush(p->logFile);
          }
          ReleaseMutex(p->mutex);
     }
}
void recvC2PThread(void* args)
{
     Phantom p = *(Phantom*)args;
     CRCStruct msg;
     while (true)
     {
          p.recvC2P->recv((char*) &msg, sizeof(msg));
          if (msg.time % 1000 == 100)
          {
               cout << "CRC POS" << endl;
               cout << "x = " << msg.x;
               cout << ", y = " << msg.y;
               cout << ", z = " << msg.z;
               cout << ", Time = " << msg.time <<endl;;
          }
          p.data->setCRCData(msg);
```

```
            }
        }
```

```
//*********************************************************
//        Filename : DataStruct.h
//   This file declare two different data structures.  One
is made to store data
//   from the Phantom and the second is to store data from
Backhoe.
//   Author:        Matt Kontz <mkontz@mail.com>
//   Lab:      IMDL ME GaTech
//   Created:  February 8, 2005
//*********************************************************
#ifndef _ARG_STRUCTURE_INCLUDE__        //  if  not  defined
'......'
#define _ARG_STRUCTURE_INCLUDE__        // defines '......'
so only happens once.
#include "DataStorage.h"
#include "Sock.h"
#include <windows.h>
struct CallbackArgs {
     HANDLE hThread;
     DataStorage *data;
};
struct ThrArgs {
     udpSocket *sock;
     DataStorage *data;
};
#endif
```

```
/*******************************************************
          Filename : Callback.h
/*******************************************************
     This file handles the call back that the Omni will
excute each servo-loop;
     Author:        Matt Kontz <mkontz@mail.com>
     Lab:      IMDL ME GaTech
     Created:  Februery 7, 2005
/*******************************************************/
#ifndef __Callback_OMNI_INCLUDED__
#define __Callback_OMNI_INCLUDED__
#include <HD/hd.h>
#include <HDU/hduVector.h>
#include "DataStorage.h" // local storage
#include "DataStruct.h"       //   Phantom,   and   Hal   data
structures
#include "ArgStruct.h"        // ThrArgs & CallbackArgs

#include <iostream.h>
//#include "..\..\FlagModelDef.h"
HDCallbackCode OmniCallback(void *pUserData)
{
     CallbackArgs CbArg = * (CallbackArgs *) pUserData;
          DataStorage *data = CbArg.data;
          HANDLE hThreadP2H = CbArg.hThread;
     int time;
     // Device State Declartions
```

```
//   HDint ButtonStates[1];            //   first   bit   ->
blue, second bit -> white button
    hduVector3Dd Ph_Pos;           //    PHANToM    Position
(translational)
     hduVector3Dd Ph_Center;           //  PHANToM  Position
(translational)
     hduVector3Dd Ph_Vel;              //  PHANToM  Velocity
(translational)
     hduVector3Dd Ph_Theta;            //   PHANToM   Gimbal
angle
     HDdouble Ph_Phi;
     hduVector3Dd Delta;               //Delta
     Delta.set(10,10,10);
     double pi = 3.1415926535897932;
     //bool BlueButton;                //   true   if   Blue
button is depressed
//   bool GreyButton;                  //   true   if   Grey
button is depressed
//   bool OnOff;
     hduVector3Dd Force;               // Force to display
     // Control variables - Postion Mode
     static const HDdouble kspring = 0.1;    // N/mm
     hduVector3Dd CRC_Pos;
     hduVector3Dd CRC_For;
     HDdouble CRC_phi;
     hduVector3Dd Delta_Pos;
     //hduVector3Dd Delta_pos2;
```

```
     unsigned short int mode;
/*******************************************************
     Get latest States from PHANToM
*******************************************************/
    hdBeginFrame(hdGetCurrentDevice());
     // Update device states
    hdGetDoublev(HD_CURRENT_POSITION, Ph_Pos);
     hdGetDoublev(HD_CURRENT_VELOCITY, Ph_Vel);
     hdGetDoublev(HD_CURRENT_GIMBAL_ANGLES, Ph_Theta);
     printf("cb   %f   %f   %f\n",   Ph_Pos[0],   Ph_Pos[1],
Ph_Pos[2]);
     //hdGetIntegerv(HD_CURRENT_BUTTONS, ButtonStates);
//   if (ButtonStates[0] % 2 == 1)
//   {
//        BlueButton = true;            // 1 or 3 -> blue
button is depressed
//   }
//   else
//   {
//        BlueButton = false;
//   }
//   if (ButtonStates[0] > 1)
//   {
//        GreyButton = true;            // 2 or 3 -> grey
button is depressed
//   }
//   else
```

```
//   {
//        GreyButton = false;
//   }
/*********************************************************
     Get latest data from CRC
*********************************************************/
     CRC_Pos = data->getCRCPos();
     CRC_For = data->getCRCFor();
     CRC_phi = data->getCRCPhi();
/*********************************************************
     Send PHANToM data to CRC
*********************************************************/
     time = data->getTime();                              //
get time (ms ~= # callbacks)
     data->incTime();                                     //
increment time
//   if ( (BlueButton == true) || (GreyButton == true) )
//        OnOff = data->OnOffState(true);
//   else
//        OnOff = data->OnOffState(false);
//   if (OnOff == true)
          mode = 1;
//   else
//        mode = 4;
     Delta_Pos = Ph_Pos - CRC_Pos;
     //Delta_Pos2 = Delta_Pos*DeltaPos;
     //Delta_Pos = Ph_Pos ;
```

```
     Ph_Phi = 3*pi/2 + Ph_Theta[2];
     // Ph_Phi = -pi/2 - Ph_Theta[2];
     data->setPhanData(Ph_Pos, Ph_Vel, Ph_Phi, 0, mode);
          // set PHANToM states
     ::ResumeThread(hThreadP2H);                          //
trigger P2H thread to start
     Send New Haptic Force
    hduVecScale(Force, Delta_Pos, -kspring);
    hdSetDoublev(HD_CURRENT_FORCE, Force);
    hdEndFrame(hdGetCurrentDevice());
    return HD_CALLBACK_CONTINUE;
}
#endif         // #ifndef __Callback_OMNI_BOOM_INCLUDED__
```

```
//        Filename : DataStorage.h
//   This file is creates a Data Storage object used to
store, change and
//  retrieve data associated with the Phantom.
//   Author:        Matt Kontz <mkontz@mail.com>
//   Lab:      IMDL ME GaTech
//   Created:  March 6, 2002
//   Edited:        April 16, 2005
#ifndef _DATA_STORAGE_INCLUDE__         //  if  not  defined
'......'
#define _DATA_STORAGE_INCLUDE__         // defines '......'
so only happens once.
#include <HD/hd.h>
#include <HDU/hduVector.h>
#include <string.h>
#include <fstream.h>
#include <iostream.h>
#include <math.h>
#include "DataStruct.h"
//#include "..\..\FlagModelDef.h"
//#include <FlagModelDef.h>
const int RECENT = 1;              //   Number   of   current
points being stored
class  DataStorage
{
private:
// Class variables
```

```
    unsigned int Time;
    PhanStruct PhData;            //  Recent  Phantom  data
for control use
    CRCStruct CRCData;            //  Recent  CRC  data  for
control use
    hduVector3Dd Ph_Vel;
    hduVector3Dd Ph_Pos;
    hduVector3Dd CRC_Pos;
    hduVector3Dd CRC_For;
    hduVector3Dd CRC_Origin;
    HDdouble CRC_phi;
    unsigned short int flags;
    bool flagVector[16];
    int k;
    //bool OnOff;
    //bool LastButtonState;
public:
//   Constructors
    DataStorage()
    {
        Time = 0;
        memset(&PhData, 0, sizeof(PhanStruct));
        memset(&CRCData, 0, sizeof(CRCStruct));
        memset(&Ph_Vel, 0, sizeof(hduVector3Dd));
        memset(&Ph_Pos, 0, sizeof(hduVector3Dd));
        memset(&CRC_Pos, 0, sizeof(hduVector3Dd));
        memset(&CRC_For, 0, sizeof(hduVector3Dd));
```

```
          memset(&CRC_Origin, 0, sizeof(hduVector3Dd));
          memset(&flagVector, 0, sizeof(flagVector));
          // control/mode flags
//        flagVector[RATE_MODE_FLAG] = 0;         //   0   =
position mode, 1 = rate mode
//        flagVector[HENRE_V_HENRE_FLAG] = 0;          //  0
= HEnRE, 1 = V-HEnRE
          flags = 0;
          for( k = 0 ; k < 16 ; k++ )
          {
               flags  =  flags  +  flagVector[k]  *  (unsigned
short int) pow(2, k);
          }
//        OnOff = 0;
//        LastButtonState = 0;
     }
// Time functions
     void incTime() { Time++; }
     int getTime() { return Time; }
// Time functions
     void setFlags(unsigned short int num, bool f) {
          flagVector[num] = f;
          flags = 0;
          for( k = 0 ; k < 16 ; k++ )
          {
               flags  =  flags  +  flagVector[k]  *  (unsigned
short int) pow(2, k);
```

```
          }
     }
     unsigned short int getFlags() { return flags; }
// On-Off function
/*   bool OnOffState(bool ButtonState) {
          if ( ButtonState == 1 )  {
               if (LastButtonState != 1 ) {
                    if ( OnOff == 1)
                         OnOff = 0;
                    else
                         OnOff = 1;
               }
          }
          LastButtonState = ButtonState;
          return OnOff;
     }*/
//   Function to retrieve data structure
     PhanStruct getPhanData() { return PhData; }
     hduVector3Dd getCRCPos() { return CRC_Pos; }
     hduVector3Dd getCRCFor() { return CRC_For; }
     HDdouble getCRCPhi() { return CRC_phi; }
//   Functions to store CRC data
     void setCRCOrigin(CRCStruct CRC)
     {
          CRCData = CRC;
          CRC_Origin.set(CRC.x,CRC.y,CRC.z);
     }
```

```
//   Functions to store CRC data
     void setCRCData(CRCStruct CRC)
     {
          CRCData = CRC;
          CRC_Pos.set(CRC.x,CRC.y,CRC.z);
//        Bh_For.set(Bh.fx,Bh.fy,Bh.fz);
//        Bh_phi = Bh.phi;
     }

//   Functions to store Phan data
     void   setPhanData(hduVector3Dd   p,   hduVector3Dd   v,
HDdouble Ph_Phi, HDdouble Ph_vPhi, unsigned short int mode)
     {
          Ph_Pos = p;
          Ph_Vel = v;
          PhData.mode = mode;
          PhData.flags = flags;
          PhData.time = Time;
          PhData.x = p[0];
          PhData.y = p[1];
          PhData.z = p[2];
//        PhData.phi = Ph_Phi;
//        PhData.vx = v[0];
//        PhData.vy = v[1];
//        PhData.vz = v[2];
//        PhData.vphi = Ph_vPhi;
```

```
    }
};
#endif    //    _DATA_STORAGE_INCLUDE__    (ascociated    with
"#ifndef")
```

```
//*********************************************************
*******************
//        Filename : DataStruct.h
//  ---------------------------------------------------------
--------------------
//   This file declare two different data structures.  One
is made to store data
//   from the Phantom and the second is to store data from
the Backhoe
//   Author:        Matt Kontz <mkontz@mail.com>
//   Lab:      IMDL ME GaTech
//   Created:  October 19, 2003
//   Edited:        November 11, 2005
//*********************************************************
*******************
#ifndef DATA_STRUCTURE_INCLUDE          //  if  not  defined
'......'
#define DATA_STRUCTURE_INCLUDE          // defines '......'
so only happens once.
// #include "DataStorage.h"
// #include "Sock.h"
#include <windows.h>
struct PhanStruct
{
     double x;                                  //  8  bytes  =
64bits
     double y;
```

```
     double z;
     unsigned int time;                    //  4  bytes  =
32bits
     unsigned short int mode;          // 2 bytes = 16bits
     unsigned short int flags;             //  2  bytes  =
16bits
};
// Stores all relevant data from Backhoe for each sampling
peroid
struct CRCStruct
{
     double x;
     double y;
     double z;
     unsigned int time;                    //  4  bytes  =
32bits*/
     unsigned short int mode;          // 2 bytes = 16bits
     unsigned short int flags;             //  2  bytes  =
16bits
};
#endif
```

```
//        Filename : Sock.h
//  ------------------------------------------------------------
--------------------
//   This file is creates the object udpSocket.  This class
has four associated
//   functions: a constructor, send, recv and close.  Being
a class object these
//   classes are stand alone and can be used by function
using pointers.
//   If you are using Visual C++ you must include the
wsock32.lib library under
//   "Link" -> "Input".
//   Author:        Matt Kontz <mkontz@mail.com>
//   Lab:      IMDL ME GaTech
//   Created:  July 10, 2002
//   Edited:        na
#ifndef __SOCK_INCLUDED__
#define __SOCK_INCLUDED__
#include <iostream.h>              // For cout and cerr
#include <string.h>                // for memset()
#include <stdlib.h>                // for atoi() and exit()
#include <stdio.h>                 //   for   printf()   and
fprintf()
#include <errno.h>
#ifdef WIN32
     #include <winsock.h>          //     for     socket(),
connect(), send(), and recv()
```

```
     typedef int socklen_t;
#else
     #include <sys/types.h>        //      for      socket(),
connect(), send(), and recv()
     #include <sys/socket.h>       //      for      socket(),
connect(), send(), and recv()
     #include <netdb.h>            // for gethostbyname()
     #include <arpa/inet.h>        //  for  sockaddr_in  and
inet_addr()
     #include <unistd.h>           // for close()
#endif
class udpSocket
{
private:
     int sock;                                  // Socket
     unsigned short localPort;                  // Local port
     unsigned short forPort;                    //      Foreign
port
    struct sockaddr_in localAddr;       // Local address
     struct sockaddr_in forAddr;                //      Foreign
address
     struct hostent *host;                      //  pointer  to
server information
     char *forIP;                               //  Foreign  IP
address
     unsigned int addrLen;
public:
```

```
     udpSocket(char *fip, unsigned short fp,unsigned short
lp)
     {
          forIP = fip;
          forPort = fp;
          localPort = lp;
          sock = -1;                    // Less than 0 mean
not connected
          #ifdef WIN32
               WORD wVersionRequested;
               WSADATA wsaData;
               wVersionRequested    =    MAKEWORD(2,    0);
// Request Winsock v2.0
               if  (WSAStartup(wVersionRequested,  &wsaData)
!= 0)   // Load Winsock DLL
               {
                    cerr << "WSAStartup() failed" << endl;
                    exit(1);
               }
          #endif
          // Create a datagram/UDP socket
          if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
          {
               cerr   <<   strerror(errno)   <<   "socket()
failed!" << endl;
               exit(1);
          }
```

```
          // Construct local address structure
          memset(&localAddr, 0, sizeof(localAddr));
     // Zero out structure
          localAddr.sin_family = AF_INET;
     // Internet address family
          localAddr.sin_addr.s_addr   =   htonl(INADDR_ANY);
     // Any incoming interface
          localAddr.sin_port = htons(localPort);
     // Local port
          // Bind to the local address
          if  (bind(sock,  (struct  sockaddr  *)  &localAddr,
sizeof(localAddr)) < 0)
          {
               cerr  <<  strerror(errno)  <<  "bind()  failed"
<< endl;
               exit(1);
          }
          // find foreign address
          memset((char *) &forAddr, 0, sizeof(forAddr));
          int addr = inet_addr(forIP);
          forAddr.sin_addr.s_addr = addr;
          if(addr != -1)
          {
               forAddr.sin_family = AF_INET;
          }
          else
          {
```

```
                host = gethostbyname(forIP);
                if (host)
                {
                forAddr.sin_family = host->h_addrtype;
                forAddr.sin_addr.s_addr  =  *((unsigned  long
*)host->h_addr_list[0]);
                }
                else
                {
                     cerr  <<  strerror(errno)  <<  "Cannot  get
host information for server." << endl;
                     exit(1);
                }
           }
           forAddr.sin_port = htons(forPort);
           addrLen = sizeof(forAddr);
     }
     void send(char *msg, const int msgLen)
     {
           // Send the string to the server
           if (sendto(sock, msg, msgLen, 0, (struct sockaddr
*) &forAddr, addrLen) != msgLen)
           {
                cerr << strerror(errno) << "sendto() sent an
incorrent number of bytes" << endl;
                exit(1);
           }
```

```
    }
    void recv(char *buffer, const int msgLen)
    {
        struct sockaddr_in fromAddr;       //       Source
address of echo
        int recvLen;                              //  Length
of received response */
        // Recv a response
        recvLen  =  recvfrom(sock,  buffer,  msgLen,  0,
(struct sockaddr *) &fromAddr, (socklen_t *) &addrLen);
        if (recvLen != msgLen)
        {
            cerr  <<  strerror(errno)  <<  "recvfrom()
failed: incorrent number of bytes" << endl;
            //exit(1);
        }
        // Check sender of message
        if          (fromAddr.sin_addr.s_addr          !=
forAddr.sin_addr.s_addr)
        {
            cerr  <<  strerror(errno)  <<  "recvfrom()
failed: unknown host" << endl;
            exit(1);
        }
    }
    void close()
    {
```

```
            // If the socket is open, close it.
            if (sock > -1)
            {
                 #ifdef WIN32
                      ::closesocket(sock);
                 #else
                      ::close(sock);
                 #endif
                 sock = -1;
            }
            #ifdef WIN32
                 if (WSACleanup() != 0)
                 {
                      cerr << "WSACleanup() failed" << endl;
                      exit(1);
                 }
            #endif
       }
};
#endif
```

APPENDIX D

MATLAB SCRIPT FOR READING, SMOOTHING, AND SAVING RECORDED TRAJECTORIES

```
function trajcon(time,side,n)
clear MASTER pinit pfinal
if side == 2;
    prefix='Righty_';
else prefix='Lefty_';
end
sidechar = num2str(side);
timechar = num2str(time);
fname = strcat(prefix, timechar, '_v1.txt');
lmastername = strcat('LMASTER.txt');
rmastername = strcat('RMASTER.txt');
%load in the recorded trajectory and it shall be called
'vec'
vec = load (fname);
%load in the master trajectory matrix for the appropriate
side
if side == 2
    MASTER = load('RMASTER.txt');
else MASTER = load('LMASTER.txt');
end
sm=size(MASTER);
%make a time vector
t=ones(length(vec(:,1)),1);
L=length(t);
for k=1:L;
    t(k)=k.*.001;
end
%split vec into PHANToM xyz vectors
x = vec(:,1);
y = vec(:,2);
z = vec(:,3);
%plot the vertical plane trajectory
figure;
plot(x,y);
if side == 2
    title('<----- Forward; View from body looking outward
to RIGHT');
```

```
    xlabel('(body axis) End Swing Phase <--------------
Start Swing Phase');
else
    title('View from body looking outward to LEFT; Forward
----->');
    xlabel('(body axis) Start Swing Phase -------------->
End Swing Phase');
end
ylabel('Down ------------------------> Up');
%take n points from raw vectors
tc=fix(L/(n));
for k=1:n+1;
    tt(k)=((k-1)*tc)+1;
    xx(k)=x(tt(k));
    yy(k)=y(tt(k));
    zz(k)=z(tt(k));
end
tt=tt.*.001;
%make spline equations from those points (xx yy zz)
%initial slope = final slope = 0
spx=spline(tt, [0 xx 0]);
spy=spline(tt, [0 yy 0]);
spz=spline(tt, [0 zz 0]);
%make curves from spline equations
xxx=ppval(spx,t);
yyy=ppval(spy,t);
zzz=ppval(spz,t);
% plot splines xyz subplotted
figure;
subplot(3,1,1); plot(t,x,'k',t,xxx,'r--');
title('PHANToM input vector (raw)');
%xlabel('Time (s)');
ylabel('x (mm)');
subplot(3,1,2); plot(t,y,'k',t,yyy,'r--');
%xlabel('Time (s)');
ylabel('y (mm)');
subplot(3,1,3); plot(t,z,'k',t,zzz,'r--');
xlabel('Time (s)');
ylabel('z (mm)');
%plot the 3d steppin action'
figure;
plot3(x,z,y,'k',xxx,zzz,yyy,'r--');
axis([-250 250 -250 250 -250 250]);
xlabel('x (mm)');
ylabel('z (mm)');
zlabel('y (mm)');
```

```
%put smoothed trajectories into new vector, and it shall be
called 'nuvec'
for k=1:L;
    nuvec(k,1)=xxx(k);
    nuvec(k,2)=yyy(k);
    nuvec(k,3)=zzz(k);
end
%pull out first and last points
pinit=ones(3,1);
pfinal=ones(3,1);
for k=1:3;
    pinit(k,1)=nuvec(1,k);
    pfinal(k,1)=nuvec(L,k);
end
poffset=[0;-127;-762]; %mm; offset from foot origin to
shoulder in PHANTom reference frame
pinit=pinit+poffset;    %evaluate distance from shoulder
base to foot command point
pfinal=pfinal+poffset;
pinit1 = (pinit(1));
pinit2 = (pinit(2));
pinit3 = (pinit(3));
pfinal1 = (pfinal(1));
pfinal2 = (pfinal(2));
pfinal3 = (pfinal(3));
%APPEND DATA TO MASTER
%MASTER IS 7 items WIDE (7 columns)
% TIME  PINIT(1)  PINIT(2)  PINIT(3)  PFINAL(1)  PFINAL(2)
PFINAL(3)
MASTER((sm(1)+1),1)=time;
MASTER((sm(1)+1),2)=pinit1;
MASTER((sm(1)+1),3)=pinit2;
MASTER((sm(1)+1),4)=pinit3;
MASTER((sm(1)+1),5)=pfinal1;
MASTER((sm(1)+1),6)=pfinal2;
MASTER((sm(1)+1),7)=pfinal3;
%Resave MASTER.txt
if side == 2;
    save(rmastername, 'MASTER', '-ascii');
else save(lmastername, 'MASTER', '-ascii');
end
%save nuvec as r/ltimestamp.txt
savefile = strcat(prefix, timechar, 'smoothed.txt');
%savefile=timechar;
save(savefile, 'nuvec', '-ascii')
```

REFERENCES

[1] Guerriero, B. and Book, W., "Haptic Feedback Applied to Pneumatic Walking," submitted to ASME Dynamic Systems Control Conference, Ann Arbor, MI, Oct 20-22, 2008.

[2] Shields, B., Goldfarb, M. 2005. "Design and Energetic Characterization of a Solenoid Injected Liquid Monopropellant Powered Actuator for Self-Powered Robots". *IEEE International Conference on Robotics and Automation*. Vol. 2005. pp. 241-6.

[3] Wait, K., Goldfarb, M. 2007. "A Biologically Inspired Approach to the Coordination of Hexapedal Gait". *IEEE International Conference on Robotics and Automation*. April. pp. 275-280.

[4] Noritsugu, T. 1987. "Development of PWM Mode Electro-Pneumatic Servomechanism. II. Position Control of a Pneumatic Cylinder". *Journal of Fluid Control*. Vol. 17. Issue 2. pp. 7-31.

[5] van Varseveld, R. B., Bone, G. M. 1997. "Accurate Position Control of a Pneumatic Actuator Using On/Off Solenoid Valves". *Proceedings from IEEE International Conference on Robotics and Automation*. Vol. 2. pp. 1196-1201.

[6] Kunt, C., Singh, R. 1990. "A Linear Time Varying Model for On/Off Valve Controlled Pneumatic Actuators". *ASME Journal of Dynamic Systems, Measurement, and Control*. Vol. 112. Issue 4. pp. 740-7.

[7] Shen, X., Zhang, J., Barth, E., Goldfarb, M. 2006. "Nonlinear Model-Based Control of Pulse Width Modulated Pneumatic Servo Control". *ASME Journal of Dynamic Systems, Measurement, and Control*. Vol. 128. pp. 663-9.

[8] Wang, J., Pu, J., Moore, P. 1999. "A Practical Control Strategy for Servo-Pneumatic Actuator Systems". *Control Engineering Practice*. Vol. 2. Issue 7. pp. 1483-8.

[9] Chillari, S. Guccione, S., Muscato, G. 2001. "An Experimental Comparison Between Several Pneumatic Position Control Methods". *Proceedings of the 40th IEEE Conference of Decision and Control*. Dec. pp. 1168-1173.

[10] Tanaka, K., Yamada, Y., Shimizu, A., Shibata, S. 1996. "Multi-Rate Adaptive Pole-Placement Control for Pneumatic Servo System with Additive External Forces". *IEEE Advanced Motion Control Proceedings*. Pp. 213-8.

[11] Korondi, P., Gyeviki, J. 2006. "Robust Position Control for a Pneumatic Cylinder". *IEEE International Power Electronics and Motion Control Conference*. Aug. pp. 513-8.

[12] Guvenc, L. 1999. "Closed Loop Pneumatic Position Control Using Discrete Time Model Regulation". *Proceedings of the American Control Conference*. June. pp. 4273-7.

[13] Al-Dakkan, K., Barth, E., Goldfarb, M. 2006. "Dynamic Constraint-Based Energy-Saving Control of Pneumatic Servo Systems". *ASME Journal of Dynamic Systems, Measurement, and Control*. Vol. 128. pp. 655-662.

[14] Shields, B., Fite, K., Goldfarb, M. 2006. "Design, Control, and Energetic Characterization of a Solenoid-Injected Monopropellant-Powered Actuator". *IEEE/ASME Transactions on Mechatronics*. Vol. 11. Issue 4. pp. 477-486.

[15] Barth, E., Gogola, M., Goldfarb, M. 2003. "Modeling and Control of a Monopropellant-Based Pneumatic Actuation System". *IEEE International Conference on Robotics and Automation*. pp. 628-633.

[16] Fite, K., Mitchell, J., Barth, E., Goldfarb, M. 2006. "A Unified Force Controller for a Proportional-Injector Direct-Injection Monopropellant-Powered Actuator". *ASME Journal of Dynamic System, Measurement and Control*. Vol. 128. Issue 1. pp. 159-164.

[17] H. Zhu, W. Book, "Position Sensing for Every Pneumatic Cylinder", *National Fluid Power Association Fall Conference*, Pittsburg, 2005, *National Fluid Power Association's Reporter*, Vol.53, No.2, 2005.

[18] Muscato, G., Spampinato, G. 2005. "A Multi Level Control Architecture for a Pneumatic Robotic Leg". *IEEE Symposium on Emerging Technologies and Factory Automation*. Vol. 2. pp. 773-9.

[19] Guihard, M., Gorce, P., Fontaine, J. 1995. "SAPPHYR: Legs to Pull a Wheel Structure". *IEEE International Conference on Systems, Man and Cybernetics*. Vol. 2. pp. 1303-8.

[20] Gorce, P., Vanel, O. 1996. "High Level Strategy to Control the Dynamic Evolutions of Bipedal Postures". *IEEE International Conference on Systems, Man, and Cybernetics*. Vol. 2. pp. 1459-1464.

[21] Gorce, P., Guihard, M. 2001. "Dynamic Controller of BIPMAN". *8th International Conference on Emerging Technologies and Factory Automation*. pp. 641-4.

[22] Guihard, M., Gorce, P. 2001. "BIPMAN Dynamic Impedance Controller Based on a Biomechanical Approach". *IEEE Conference on Systems, Man and Cybernetics*. Vol. 5. pp. 2997-3002.

[23] Denavit, J., Hartenberg, R. 1954 "Kinematic Notation for Lower-Pair Mechanisms Based on Matrices". *American Society of Mechanical Engineers Meeting A-34*. Paper 54 A-34.

[24] Pieper, D. 1968. "The Kinematics of Manipulators Under Computer Control". PhD Dissertation. Stanford University CA Department of Computer Science.

[25] Frantsevich, L., Cruse, H. 1996. "The Stick Insect, *Obrimus asperrimus* (Phasmida, Bacillidae) Walking on Different Surfaces". *Journal of Insect Physiology*. Vol. 43. No. 5. pp. 447-455.

[26] Cruse, H., Durr, V., Schmitz, J. 2006. "Insect Walking is Based on a Decentralized Architecture Revealing a Simple and Robust Controller".

Philosophical Transactions of the Royal Society A. Vol. 365. pp. 221-250.

[27] Torige, A., Noguchi, M., Ishizawa, N. 1993. "Centipede Type Multi-Legged Walking Robot". *IEEE/RSJ International Conference on Intelligent Robots and Systems*. July. pp. 567-571.

[28] FESTO. 2003. "Proportional Directional Control Valves". *2004/2005 Product Catalog*. 5/1.5-1 - 5/1.5-11.

[29] Lipkin, H. 2006. "Displacement Analysis for the Generalized Puma Robot". *ME 6407 Class Notes*. pp. 1-15.

[30] Ott, R., Gutierrez, M., Thalmann, D., Vexo, F. 2005 "Improving User Comfort in Haptic Virtual Environments through Gravity Compensation". *IEEE First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. pp. 401-9.