A MULTI-PURPOSE FINITE STATE-BASED STANDING CONTROLLER FOR A POWERED TRANSFEMORAL PROSTHESIS

By

Brian Edward Lawson

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Mechanical Engineering

December, 2011

Nashville, Tennessee

Approved:

Professor Michael Goldfarb

Professor Eric Barth

Professor Nilanjan Sarkar

To my loving and patient partner, Katie

ACKNOWLEDGEMENT

The single most important contributor to this work is my advisor, Dr. Michael Goldfarb. His unwavering enthusiasm and focus drives everything forward. I would like to thank him for trusting in my ability to contribute to his research in the field of prosthetics.

I owe a significant amount of gratitude to Drs. Frank Sup and Atakan Varol for their constant technical assistance and advice. They selflessly welcomed me into their project during the final months of their dissertation work and continued to serve as mentors throughout this work.

The current and past members of the Center for Intelligent Mechatronics have all contributed to both this work directly and also to my general academic development. I would specifically like to recognize Jason Mitchell and Don Truex for contributing their many combined years of engineering experience.

Finally I would like to acknowledge my family's unending support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

The most general aim of the work described in this manuscript is to improve the life of the transfemoral amputee through the improvement of current prosthetic technology. Since up to 85% of lower limb amputees utilize a prosthesis, the improvement of prosthetic technology has the potential to positively affect a significant portion of the amputee population [1]. Furthermore, although the focus of this work is on the development and control of prosthetic devices for *transfemoral* amputees, many of the results and control algorithms presented may be equally applicable to persons living with other types of lower limb amputations. This chapter attempts to provide a brief overview of the difficulties faced by people living with a lower limb amputation and the prosthetic technologies that are currently available to help alleviate those difficulties.

1. Life with a Lower Limb Amputation

A study published in 2005 estimated that the number of people living in the United States with the loss of a limb was just under 1.6 million [2]. Within this population an estimated 623,000 people were living with a major lower limb amputation (where a major lower limb amputation was defined as an amputation at the foot or any more proximal location on the limb). The primary cause of amputation was due to dysvascular disease (81%), while the remaining significant causes were traumatic injury (17%) and cancer (2%). 1,112 amputations have been reported due to Operation Iraqi Freedom [3], and, although this may not be a large portion of the overall amputee population, veteran amputees have traditionally been early adopters of new prosthetic technologies due to government subsidies and generally higher activity levels.

Members of the lower limb amputee population are typically assigned a Medicare Functional Classification Level (MFCL) in order to characterize the level of activity they undergo on a daily basis. This classification scheme contains five levels, commonly referred to as K-levels, which rank the amputee in an order of increasing activity. Table 1, reprinted from [4], lists the formal definitions of the five activity levels specified by the MFCL.

Current prosthetic devices can provide aid to all activity levels except for K0. The higher the activity level of an amputee, the more activities he or she is

able to perform, and typically this corresponds with greater utilization of a prosthetic device.

HCFA	MFCL Description
K0	MFCL-0—Does not have the ability or potential to ambulate or transfer safely with or without assistance and a prosthesis does not enhance quality of life or mobility.
K1	MFCL-1—Has the ability or potential to use a prosthesis for transfers or ambulation on level surfaces at fixed cadence. Typical of the limited and unlimited household ambulator.
K2	MFCL-2—Has the ability or potential for ambulation with the ability to traverse low-level environmental barriers such as curbs, stairs, or uneven surfaces. Typical of the limited community ambulator.
K3	MFCL-3—Has the ability or potential for ambulation with variable cadence. Typical of the community ambulatory who has the ability to traverse most environmental barriers and may have vocational, therapeutic, or exercise activity that demands prosthetic utilization beyond simple locomotion.
K4	MFCL-4—Has the ability or potential for prosthetic ambulation that exceeds the basic ambulation skills, exhibiting high impact, stress, or energy levels, typical of the prosthetic demands of the child, active adult, or athlete.
HCFA = Health Care Financing Administration.	

Table 1: Medicare Functional Classification Level (MFCL) descriptions

Across all activity levels, however, severe biomechanical deficiencies are experienced by amputees when compared to the healthy population. A significant number of studies have been conducted on the metabolic cost of transport for lower-limb amputees in gait. A review paper by Waters and Mulroy highlights this problem by providing the chart reprinted here as Figure I-1 [5]. These data are derived from a combination of two studies with comparable methodology but applied to subjects of different amputation levels. The abbreviations for amputation level in the figure are hemipelvectomy (HP), hip disarticulation (HD), transfemoral (TF), through knee (TK), and transtibial (TT).

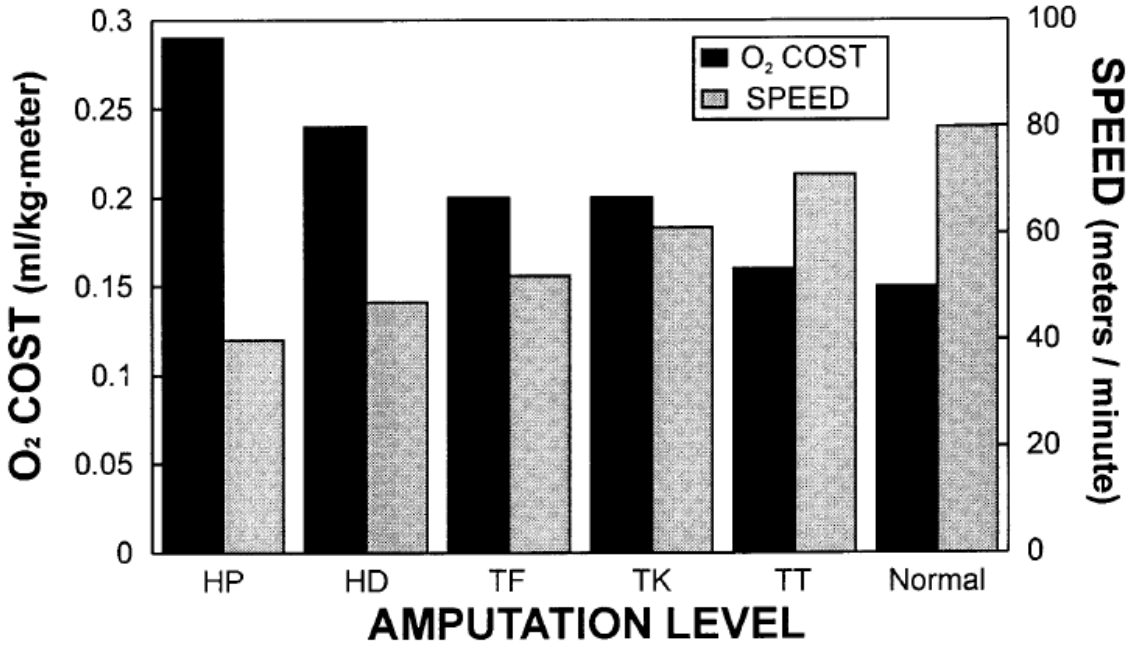

Figure I-1: Oxygen consumption and walking speed for unilateral amputees (reprinted from [5]).

There is a clear correlation between the level of amputation and the metabolic cost of transport. Specifically, transfemoral amputees exhibited a 30% increase in the metabolic cost of transport as compared to healthy subjects. In addition, the comfortable walking speed of the subjects (referred to as the self-selected walking speed, or SSWS) decreases with increasing levels of amputation. The reason for this inverse relationship, as reported in [5], is because the rate of metabolic energy consumption remains relatively constant across subjects. This naturally leads to the conclusion that humans adapt their walking speed to maintain a comfortable level of exertion. When walking becomes more difficult, for instance due to an amputation or other pathological gait deficiency, they choose to decrease their walking speed instead of increasing their exertion.

In addition to the mobility deficiency that arises from the tradeoff between exertion levels and SSWS, amputees also experience gait asymmetry even when using the current state of the art in prosthetic devices. This asymmetry has been characterized by the ratios of time spent in stance on the sound leg vs. the prosthetic leg either through the use of heel and toe contact indicators as in [6], or through center of pressure measurements as in [7]. Transfemoral amputees, even 10 years post amputation, spend approximately 9.2% longer in stance on their sound leg than their prosthesis [7].

In the same way that serious obstacles arise in terms of mobility, lower limb amputees also face severely compromised stability. In a survey of 435 community-living lower limb amputees, just over half reported falling in the last year, while a comparable number reported a fear of falling. More importantly, approximately 10% of those surveyed were forced to seek medical attention as a result of the fall [8, 9]. Additionally, 65% of the respondents received scores of 80 or below on the Activities-Specific Balance Confidence Scale [10], which indicates that this subset could experience significant benefits from treatment to improve their balance confidence [11]. The researchers that performed this survey concluded that "falling and fear of falling are pervasive among amputees" [9].

Biomechanical studies have also been conducted in order to characterize the deficiencies experienced by lower-limb amputees in balance. The predominant outcome measures utilized in these studies are postural sway [12, 13] and weight bearing distribution [14-16]. Postural sway increased for transtibial and transfemoral amputees in both static and dynamic standing conditions as compared to healthy subjects [13]. Both groups of amputees (transtibial and transfemoral) also favored their sound leg in terms of weight distribution by a ratio of approximately 60-40% during standing [14-16]. It is important to note that all of these studies were conducted for standing on level

surfaces only. (The dynamic condition in [13] consisted of a passive pivoting stabilometer on which the subjects attempted to balance in a level configuration.)

In summary, amputees that are well enough for active lifestyles are still biomechanically limited as compared with healthy subjects. This is especially true for the transfemoral amputee, who has lost a biomechanically intact knee and ankle joint. These limitations, as shown by the outcomes of the studies previously discussed, are inherently dependent on the type of prosthetic device used by the subjects. All devices in the above studies were passive devices. The following section will explore the most recent prosthetic devices specifically available to the *transfemoral* amputee, along with a discussion of the benefits and drawbacks of such devices.

2. Current Prosthetic Technology for Transfemoral Amputees

Today there is a host of commercially available prosthetic devices available to the transfemoral amputee. A complete prosthesis is typically assembled from independent components, allowing the amputee (or prosthetist) to choose amongst various brands and devices. The foot and ankle are generally integrated into a carbon fiber ankle-foot complex. These devices can be directly integrated to a prosthetic socket in order to create a transtibial prosthesis, or they can be coupled to a prosthetic knee in order to create a transfemoral prosthesis.

Major manufacturers of carbon fiber ankle-foot complexes currently include Otto Bock® (see Figure I-2), Freedom Innovations (see Figure I-3), and Össur®. The primary advantage of these ankle-foot complexes is that they exhibit an energy-storing behavior during gait that compensates for some of the lost efficiency as compared to healthy biomechanics. In addition, these devices provide a comfortable impedance for stability in stance, at least when standing on level ground.

Figure I-2: Lo Rider® carbon fiber ankle-foot complex by Otto Bock®

Figure I-3: Renegade® carbon fiber ankle-foot complex by Freedom Innovations

In order to construct a complete transfemoral prosthesis, the ankle-foot complex must be coupled with a suitable prosthetic knee. A wide variety of passive prosthetic knee designs have been developed and marketed over the years. Currently the most widely accepted passive knee consists of a microprocessor-controlled (MPC) damper that dynamically adjusts its resistance over the gait cycle based on input from mechanical sensors. Manifestations of this approach include the Otto Bock C-Leg® (see Figure I-4), the Össur Rheo Knee®, and Freedom Innovation's Plié Knee®.

Figure I-4: C-Leg® microprocessor controlled knee by Otto Bock

These knees are able to provide a relatively high amount of damping when the user is applying weight to the prosthesis, and then quickly change to a low value when the load is removed, allowing for a dynamic swing phase during gait. A significant amount of research attention has been focused on the performance of MPC knees in recent years. Performance enhancements include the ability to undergo stance knee flexion in gait [17], a reduced risk of falling

due to unexpected gait deviations [18, 19], balance enhancement as measured by the Sensory Organization Test (SOT) [17], improved performance on stair descent and both ascent and descent on hills [19], higher general activity levels and SSWS [20, 21], and reduced time for obstacle course completion [22]. Furthermore, many reports indicate that user preference highly favors MPC knees [19, 20, 23]. One study reported that the most metabolically efficient walking speed was the same for healthy subjects and amputees using MPC knees [24], although more general conclusions concerned the metabolic efficiency of MPC knees as compared to purely mechanical knees have been debated in the literature. A comprehensive overview of current prosthetic technology can be found in [25].

Although there are a wealth of advantages offered by carbon fiber ankle-foot complexes and MPC knees, there are still some fundamental inadequacies inherent in these devices that prevent them from being able to fully restore biomechanical function to the amputee. For the purposes of this thesis, the term *passive prosthesis* will refer specifically to a prosthetic leg consisting of a carbon fiber ankle-foot complex and an MPC knee. Obviously, many forms of ambulation (including level ground walking) require positive net power for healthy biomechanics. Since a passive prosthesis cannot supply this power, these activities will inherently be biomechanically deficient to some degree or another

(albeit to a lesser degree in the case of level ground walking relative to slope or stair ascent). An extensive description of these deficiencies is beyond the scope of this work, but is covered in the doctoral dissertation by Sup from 2009 [26].

Passive prostheses also suffer from performance degradation when they encounter unlevel terrain. This problem is not so much due to their inability to provide active power generation as it is a function of their inability to actively change their configuration. A passive ankle-foot complex, whether it is a high performance carbon fiber leaf spring or a solid ankle, cushioned heel (SACH) foot, maintains a constant equilibrium point at or near zero degrees in its unloaded state. This fact, in conjunction with the inability of a MPC knee to provide static torque to the user unless it is forced against its hyperextension hard stop, means that a passive prosthesis cannot fully support the user on unlevel terrain. Figure I-5 shows a selection of unstable standing configurations on different slopes with a passive prosthesis.

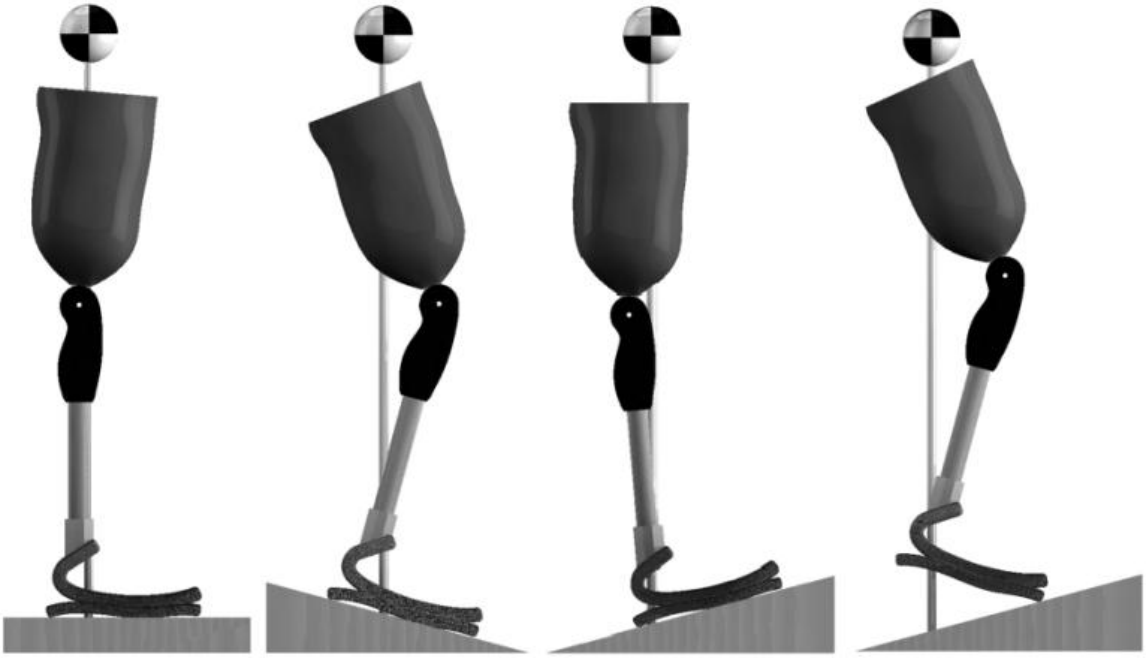

Figure I-5: Passive prosthesis configurations on slopes

Consider first the example of encountering a down slope. On a down slope, the ankle maintains its equilibrium position and forces the shank to rotate forward, bending the knee. If the user were to try to bear weight on the prosthesis in this condition, the torque supplied to the knee through the thigh would cause the leg to slowly buckle, since the knee acts only as a damper and cannot supply a static torque. The result is that the amputee is unable to bear any significant weight on the prosthesis when standing on a significant down slope. Also note that it would technically be feasible to force the knee against the hyperextension hard stop and try to load the heel of the foot significantly enough to cause the foot to conform to the ground. In this case the reaction torque at the

ankle would be de-stabilizing, and also severe. As a rough example of this condition, consider a nominal ankle stiffness of 4 N-m/deg, conforming to a 10 degree down slope. If the shank is to remain vertical, the ankle would react with 40 N-m of de-stabilizing torque. In order to match this torque for equilibrium, assume that the lever arm of the heel is approximately 4 cm. This would require a vertical load on the order of 1000 N. This example shows that even for a relatively flexible ankle the load the user would need to supply is possibly greater than the user's total body weight, and hence is an unfeasible static standing condition.

In the case of standing on an up slope, the loading condition just described is slightly easier to accomplish on account of the larger lever arm provided by the extension of the prosthetic foot. In this case the user may temporarily be able to load the ankle on mild slopes in order to receive support on the prosthetic side. There is still a lower threshold, however, in which the weight born on the prosthesis will not keep the ankle in contact with the slope and instead will revert to the final image in Figure I-5. The more extreme the ground angle, the more likely the user will opt to avoid loading the foot at all and instead bear all of his or her weight on the sound side.

As previously stated, the deficiencies that arise in standing stability for a passive prosthesis are due primarily to the fact that the ankle has an unchanging

equilibrium position and the knee cannot resist static torques when it is at an angle greater than zero degrees. A powered prosthesis, on the other hand, does not necessarily have a static equilibrium point at the ankle, nor does it necessarily act only as a damper at the knee. The remainder of this manuscript will outline a particular manifestation of a powered transfemoral prosthesis that can overcome these deficiencies, along with the design and implementation of a standing controller that has been demonstrated to provide full support to the user in a variety of ground slope conditions.

CHAPTER II

POWERED PROSTHESIS DESIGN AND PARAMETER TUNING

The controller presented in this thesis was implemented and tested on a prototype of a powered transfemoral prosthesis. An account of the detailed hardware design was published as a doctoral dissertation by Sup in 2009 [27]. The original control and intent recognition implemented in the device was published as a doctoral dissertation by Varol in 2009 [28]. An overview of the current hardware revision and control structure is now provided in order to highlight the changes in the system design since this previous work was published.

1. Mechanical Design

An image of the version of the prosthesis used for the experiments in this manuscript is provided in Figure II-1. The prosthesis contains two 200 W brushless DC motors for power generation. These motors supply torque to the knee and ankle joints via linear actuators formed from 2 mm pitch ball screw assemblies in slider-crank configurations. The resulting system is capable of supplying up to ±70 Nm of torque to either joint.

The frame of the prosthesis is constructed from CNC-milled 7075 aluminum alloy and is black-anodized. The knee joint interfaces to a prosthetic socket via a standard pyramid connector. The foot of the prosthesis is also milled from aluminum and incorporates strain gage bridges at the heel and toe for load sensing.

The prosthesis weighs approximately 4.5 kg excluding the user's socket but including a standard foot shell and sneaker. The prosthesis was design for an 85 kg user and the minimum structural factor of safety used in the design of the prosthesis was 2.

Figure II-1: The powered transfemoral prosthesis

2. Electrical Design

A completely redesigned embedded system has been implemented to address the requirements of the standing controller presented herein, in addition to general improvements in the operation of the prosthesis. The embedded system consists of two separate custom printed circuit boards. The primary circuit board, known as the main board, is mounted on the shank of the prosthesis and is protected by a plastic cover. A secondary board is mounted inside the foot and is connected to the main board with a custom cable, which is shielded by a conduit. The boards were designed in Altium Designer Summer 09, fabricated by a custom board house and assembled and tested by hand. Figure II-2 and Figure II-3 show the artwork for the main circuit board and the foot board, respectively. The system is powered by a rechargeable 29.6 V lithium polymer battery rated at 3900 mA·h, which is fully contained within a cavity in the frame of the prosthesis.

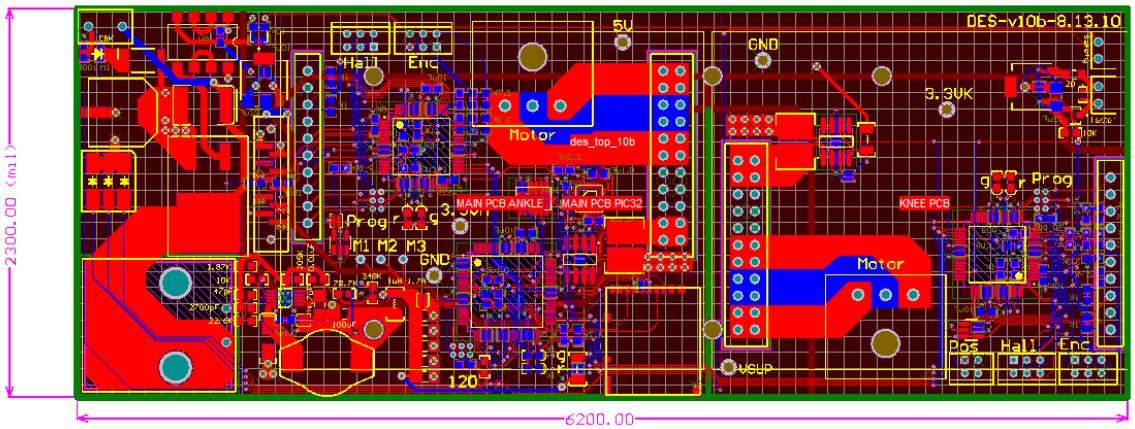

Figure II-2: Main board artwork for the embedded system

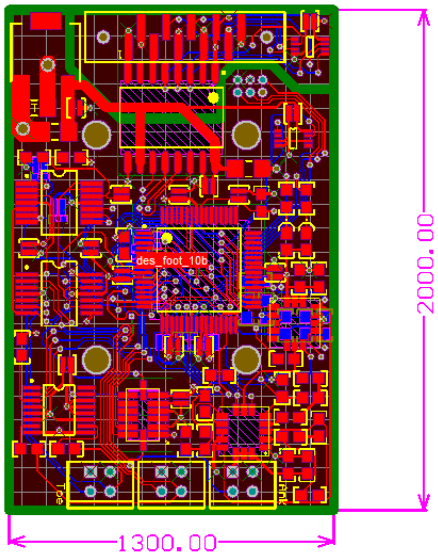

Figure II-3: Foot board artwork for the embedded system

The main microcontroller operates at a frequency of 80 MHz and executes its control routines at a rate of 500 Hz. For the purposes of tuning and development, the microcontroller logs 32 long integers (32-bit values) to an SD card at a rate of 250 Hz, or every other sampling instant. The SD card also contains a parameter file that is read by the microcontroller upon startup. The

parameters in the file are tuned specifically for each user of the prosthesis through the use of a custom graphical user interface implemented in MATLAB.

The knee and ankle actuators are each directly controlled by independent digital signal processors that communicate with the main microcontroller over an SPI bus at every sampling instant. These processors send current references to commercial servo-amplifiers that supply power to the brushless DC motors.

3. Control Framework

The behavior of the prosthesis is governed by a three-tiered control structure (see Figure II-4). At the lowest level, the embedded system implements torque control for each joint by modulating current to the motors. The middle level controller determines the commanded torque input for the low level controller. This controller is implemented as a finite state machine (FSM) that implements a passive impedance behavior for each state. There is a separate middle level controller for each activity that the prosthesis is able to perform (standing, walking, etc.). The states within each FSM correspond to distinct motions or actions within the activity. For instance, the states in the walking controller are roughly equivalent to phases of the gait cycle.

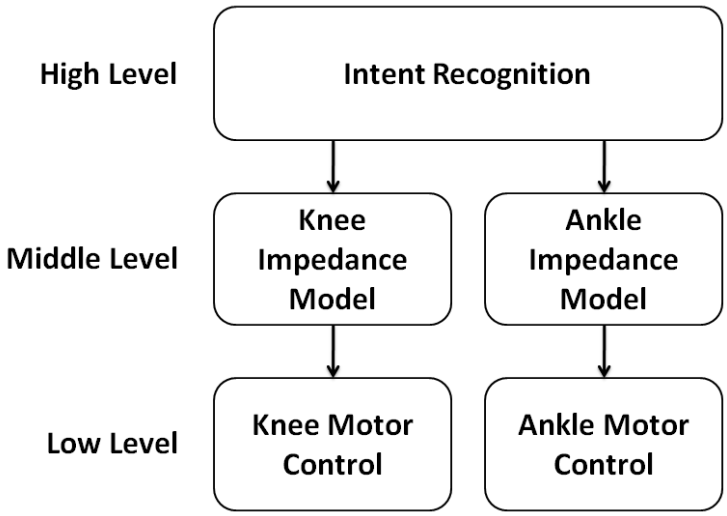

Figure II-4: Control hierarchy for the embedded system

The torque command for each state is selected according to an impedance model that simulates a spring and damper system. Separate torque commands are generated for each joint.

$$\begin{aligned} \tau_k &= k_k\left(\theta_k - \theta_{eq_k}\right) + b_k\dot{\theta}_k \\ \tau_a &= k_a\left(\theta_a - \theta_{eq_a}\right) + b_a\dot{\theta}_a \end{aligned} \tag{1}$$

Equation (1) describes the impedance model that is used to generate the torque commands for the knee and ankle. In this equation, τ denotes the torque command, θ denotes the joint's angular position, and $\dot{\theta}$ denotes the joint's angular velocity. The three parameters, k, b, and θ_{eq}, are referred to as the stiffness, damping, and equilibrium position of the joint for a given impedance. The subscripts k and a indicate whether the given parameter pertains to the knee

or the angle, respectively. These parameters completely determine the behavior of the joint for a given state in the middle level controller.

The advantage of such an approach for the middle level controller is that the behavior of the prosthesis within any given state of the finite state model is governed by passive dynamics. Since the user has control over the transitions between states (state transitions are instigated by user-supplied mechanical cues, such as heavily loading the toe for toe-off in the walking controller), the prosthesis will never unintentionally introduce power or behave erratically. The result is a powered prosthesis that is capable of simulating an infinite variety of different configurations of passive prostheses. Power is introduced by arbitrarily changing the passive configuration during a state transition.

4. Prosthesis Data Visualizer

Because every state in the state machine has three tunable parameters for the knee's behavior and three separate parameters for the ankle's behavior, the total number of parameters necessary to tune the prosthesis for a particular user is quite large. In order to make the process of tuning the leg tractable, a custom graphical user interface (GUI) was written in MATLAB that is capable of both analyzing data logged by the prosthesis and also adjusting the embedded system parameters. The GUI is referred to as the Prosthesis Data Visualizer, or PDV.

The main window of PDV, shown in Figure II-5, contains four panels that separate different tasks. The SD Card panel organizes the controls for reading log files from the SD card used in the prosthesis. When an SD card with a valid parameter file is inserted in the computer, it can be selected in this panel by navigating to the proper drive. Once the card has been found, the user can plot data from the card by selecting the appropriate log file. The configuration of the displayed data is controlled by the Tools panel, where the user can select the number of plots to display and which data is displayed on which plot. All data is plotted by default as a time series, but a group of custom plots can be generated for the analysis of various gait data.

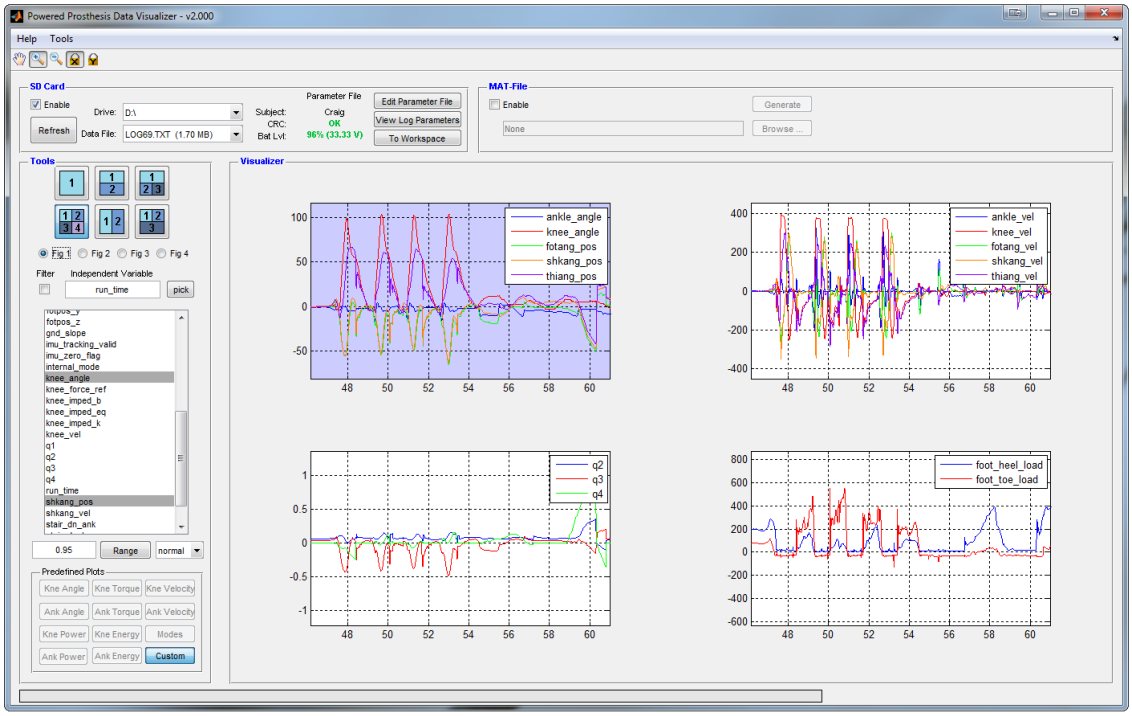

Figure II-5: Main window of the Prosthesis Data Visualizer

If the log file of interest contains walking or stairs data, the user can select the region of interest and automatically generate a series of plots that overlay variables as a function of stride percentage. For standard variables such as joint kinetics, an overlay of healthy subject data as published by Winter is provided as a reference [29]. An example of walking data is presented in Figure II-6, showing angles and velocities of the knee and ankle for level ground walking.

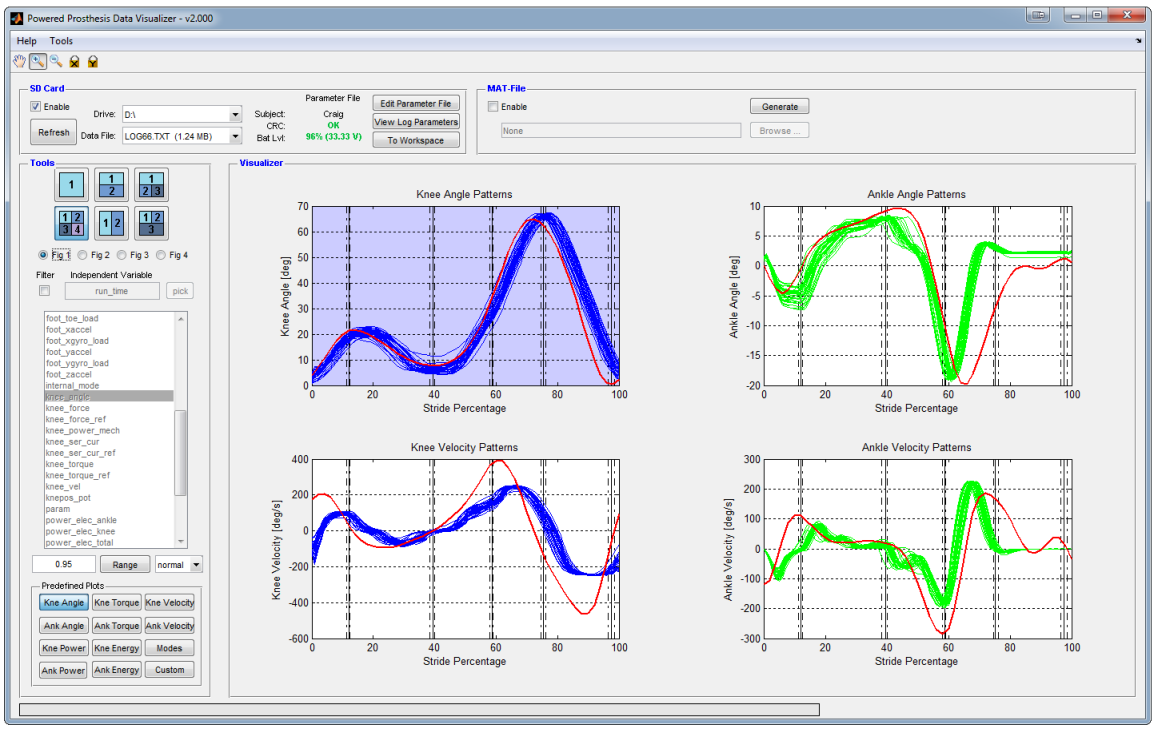

Figure II-6: Automatically generated gait analysis plots in PDV

In addition to data visualization, PDV provides a utility for adjusting the impedance parameters of the prosthesis in order to facilitate tuning the device for each individual user. The window shown in Figure II-7 can be accessed by selecting the "Edit Parameters" button in the main window. In this window the operator can adjust the impedance parameters for each middle level controller based upon the data collected in the log files. The operator can also selectively enable or disable various features and middle level controllers in the prosthesis from this window, should the user prefer it.

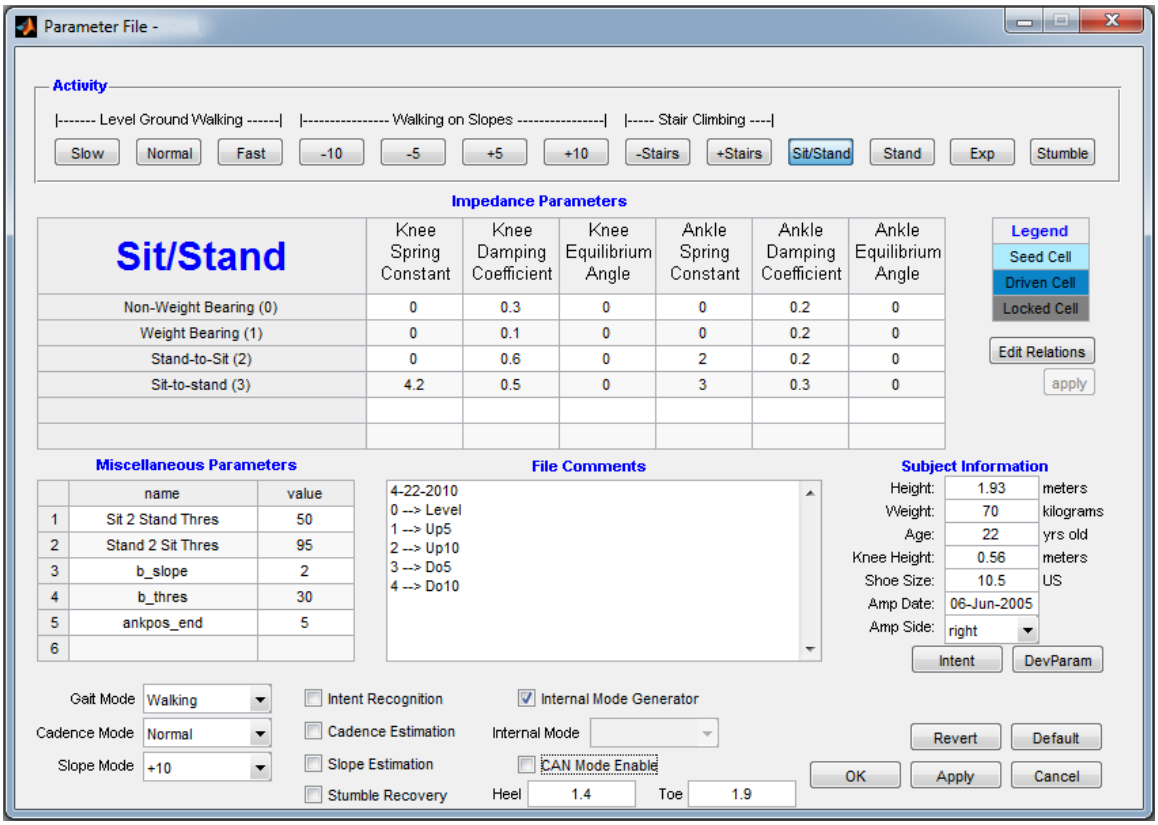

Figure II-7: Parameter window of the Prosthesis Data Visualizer

The operator can also selectively enable or disable various features and middle level controllers in the prosthesis from this window, should the user prefer it.

CHAPTER III

INERTIAL MEASUREMENT

If one would like to measure the translational kinematics of an object relative to an inertial reference frame, there is only one quantity that can be measured intrinsically: acceleration. With an accurate measurement of an object's acceleration in an inertial reference frame, the complete kinematics of the object can be recovered. When the measurement of an object's acceleration is used to recover its changes in position, the process is commonly referred to as dead-reckoning. The fundamental problem with this technique (which is the universal problem with practical integration) is that the accumulation of a measured quantity likewise accumulates the measurement error. Therefore, barring a perfect measurement, error will increase in the system in an unbounded fashion. Since recovering the position of an object requires a double integration of acceleration, this problem is magnified in the case of dead-reckoning. The conclusion that must be drawn, then, is that the accuracy of any navigation system that is based purely on inertial measurement will be bounded in time. In other words, after some finite period of time all dead-reckoning algorithms must be re-zeroed or updated with some absolute form of position measurement in

order to offset the accumulation of error. Most of the work in the field of inertial navigation is therefore focused on improving the precision of the measurement devices such that error is minimized and the period of time available for valid tracking is effectively increased.

Although the fundamental theory that drives inertial measurement of spatial motion has been well known for centuries, the practical techniques and algorithms necessary for its implementation were not developed until sufficiently precise measurement devices could be constructed. The practical history of inertial navigation begins around the middle of the 20th century, during and after World War II [30]. Most of this research was focused on intercontinental ballistics. Since that time the techniques for inertial navigation have coalesced into a well-defined field with a number of textbooks now available.

A more recent development in the field of inertial measurement is the introduction of low-cost, solid state inertial sensors. These sensors come as self-contained integrated circuits that can be directly integrated to a circuit board, essentially eliminating size and weight as a factor in most applications. The primary compromise made by these devices is a more limited resolution relative to their large-scale cousins used for missile guidance. Typical solid-state accelerometers effectively resolve approximately 1000 levels over their full scale

range (approximately 8 to 10 bit resolution once digitized). For tracking of human scale movement (say ±4 g), this produces a minimum resolution of 8 mg. In order to gain some intuition about how this resolution can affect position tracking, assume an ideal scenario where the accelerometer undergoes a one-dimensional acceleration perfectly aligned with the measurement axis. Now also assume that the true acceleration of the device is 0.992 g, while the measurement reported by the device is 1.000 g. Assuming that the device starts from rest, over the course of one second a tracking algorithm will report a translation of

$$1/2\, at^2 = 1/2\,(9.81)(1)^2\, m = 4.9050m \tag{2}$$

while the true translation of the object will be

$$1/2\, at^2 = 1/2\,(9.7315)(1)^2\, m = 4.8658m. \tag{3}$$

There difference in these two values is approximately 4 cm. After two seconds the difference has grown to 15.7 cm and after three seconds, 35.3 cm.

The implication of the previous example is that the period of tracking time for which dead-reckoning systems based upon currently available solid-state inertial sensors will be feasible is on the order of one to two seconds. When other sources of measurement error are considered (for instance errors in orientation

estimation and gravity compensation), this period of time can be significantly reduced.

1. Application in Lower-Limb Prosthetics

The powered prosthesis measures a number of signals that define its internal state. Signals such as joint angles, velocities and motor currents can be used to predict or determine what activity the user is trying to perform, along with providing cues for state transitions in the finite state machines that define each middle level controller. An inertial measurement unit, however, offers unique information as to the configuration of the prosthesis within the world coordinate frame. Specifically, it can inform the prosthesis of its orientation with respect to the gravity vector (or, in the case of a non-inertial reference frame, with respect to the net acceleration on the device). It can also inform the prosthesis of relative changes in position and velocity, within the period of time deemed valid based upon the resolution of the device. As such, a six axis IMU has been integrated into the embedded system of the powered prosthesis for the specific purpose of real time ground slope estimation, along with the general purpose of providing enhanced information as to the intent of the user.

The size and weight constraints of a fully self-contained powered prosthesis prohibit the use of the high resolution inertial measurement devices

found in missile and airplane guidance systems. Without such devices, dead-reckoning is necessarily limited to a time period of one to two seconds, as previously discussed. Luckily, for the application of a powered prosthesis, this limitation can be circumvented by frequent recalibration of the device. The IMU on the prosthesis is located on a circuit board embedded in the prosthetic foot. Consequently, the assumption can be made that the foot is resting in an inertial reference frame any time there is a sustained heel and toe load, which are measured via strain gage bridges. During this period of time, the IMU can be continually zeroed, since it is assumed that there is no net acceleration (separate from gravity). Any time the foot leaves the ground, the IMU can begin dead-reckoning to calculate the movement of the prosthesis in free space. For gait activities, the foot is typically off of the ground for no longer than a second, and so the device is limited in its ability to accumulate error.

2. Performance Considerations in IMU Algorithm Development

Because the IMU algorithm presented was to be implemented on an embedded system and run in real time, computational efficiency was a primary concern in its development. Due to the duration of other activities running on the microcontroller, a target time period of 500 microseconds (per sampling time) was allotted to the IMU algorithm. Since the fundamental sampling time for the

embedded system is 500 Hz, the IMU algorithm is given 25% of all the available computation.

The embedded system is capable of logging 32 parameters (stored as 32-bit integers) to an SD card at a rate of 250 Hz, or every other sample time. In order to store data at the sampling interval of 500 Hz, parameters from the previous sampling instant can be temporarily stored and subsequently written as a separate parameter, thereby effectively logging up to 16 parameters at 500 Hz. Although the microcontroller has a relatively large amount of memory available for its internal state, the limited ability to log data drove the decision to track orientation from the IMU signals with quaternions, which require four independent values, as opposed to the nine needed to store a complete rotation matrix.

3. A Review of Quaternion Representation of Orientation

A complete derivation of the properties of quaternions is beyond the scope of this thesis. However, a brief review of their application for representing orientation is included to facilitate the subsequent description of the IMU algorithm. Historically, quaternions were introduced by Hamilton between 1843 and 1853 [31]. For an intuitive interpretation of their use for 3D rotations, see [32].

Recall that a quaternion is a 4-dimensional extension of a complex number which can be expressed as

$$\boldsymbol{q} = q_o + q_1\hat{\imath} + q_2\hat{\jmath} + q_3\hat{k} = [q_o, \vec{q}_v]. \tag{4}$$

A quaternion has a real component, given by q_o, and an imaginary component, given by $\vec{q}_v$. The vector quantity $\vec{q}_v$ has three components denoted by the unit vectors $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$. Each of these unit vectors represents an orthogonal imaginary unit ($i^2 = j^2 = k^2 = ijk = -1$).

A quaternion can be normalized such that

$$\hat{\boldsymbol{q}} \Rightarrow {q_o}^2 + \vec{q}_v{}^T \vec{q}_v = 1. \tag{5}$$

In this case a unit quaternion can conveniently represent a 3D rotation of angle θ about an axis $\hat{n}$ by

$$\hat{\boldsymbol{q}} \equiv \left[\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\hat{n}\right]. \tag{6}$$

This rotation can be applied to a vector when the vector is cast in terms of a pure quaternion (i.e. a quaternion with no real component). The operation for applying such a rotation is given by

$$\boldsymbol{x'} = \hat{\boldsymbol{q}}\boldsymbol{x}\hat{\boldsymbol{q}}^{-\mathbf{1}}. \tag{7}$$

In this relation $\boldsymbol{x}$ is a pure quaternion representing the point $\vec{x}$. While quaternions themselves provide an efficient way to store a rotation (4 elements as opposed to 9 for a rotation matrix), equation (7) provides an efficient way to apply a rotation.

For prototyping purposes a custom Matlab class for a quaternion data type was implemented. This class overloads Matlab's native mathematical operators such that the syntax for performing quaternion operations is greatly simplified. This class facilitated proving the IMU algorithm before its final implementation in C. The Matlab class definition is included in APPENDIX C. For implementation in the microcontroller, a series of 64-bit fixed point quaternion functions were written. These functions are included in the file "`rleg_inertia.c`", which is found in APPENDIX A.

4. IMU Algorithm Overview

For implementation in the ground adaptive standing controller described in Chapter IV, all that is necessary from the IMU algorithm is an accurate estimation of the orientation of the prosthesis. Regardless, full 3 dimensional position and orientation tracking was implemented in order to provide enhanced

information for the purposes of intent recognition. The algorithm, presented subsequently, is capable of providing accurate position tracking for periods of up to 0.5 seconds. The conventions for the axes of the IMU signals are illustrated in Figure III-1.

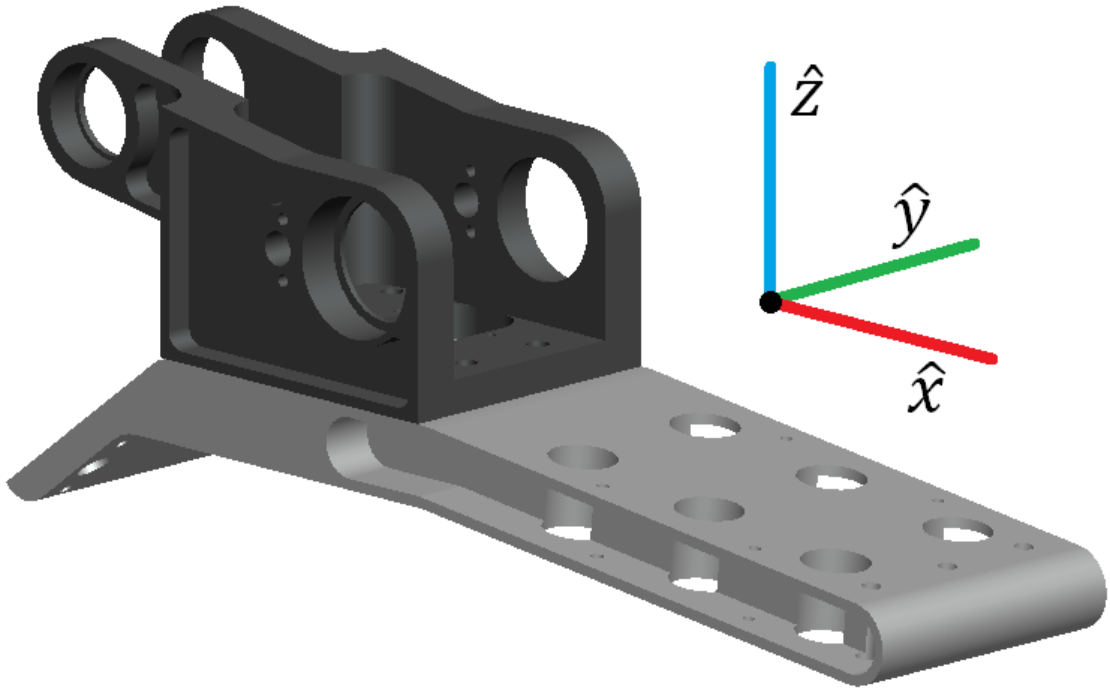

Figure III-1: Axis orientation for the IMU

The orientation and translation estimates are zeroed upon ground contact. (This process enables the transition from the ground searching mode as described in Chapter IV). Only the accelerometer signals are used (as an estimate of the gravity vector) during the zeroing process. In order to confirm that the signals are stable enough for use as a measure of the gravity vector, all three

accelerometers signals are high pass filtered (cutoff of 0.16 Hz), squared, low pass filtered (cutoff of 0.8 Hz) and then summed to give a crude measure of the power at high frequencies in the signals. When this quantity is sufficiently low (compared to a threshold), the gravity vector is calculated by the function "`reset_axes()`", the orientation quaternion is reset with respect to the gravity vector, and the absolute position of the foot in space is reset to zero in all axes. While this condition holds (i.e. the foot is on the ground and the accelerometers are providing a stable measure of the gravity vector), "`reset_axes()`" is continually called, which provides a progressively more accurate estimate as the accelerometer signals converge.

When the foot leaves the ground, as indicated by the absence of significant heel and toe loads in the prosthesis, a separate algorithm is called to provide the orientation and translation tracking. An overview of the algorithm is presented visually in Figure III-2.

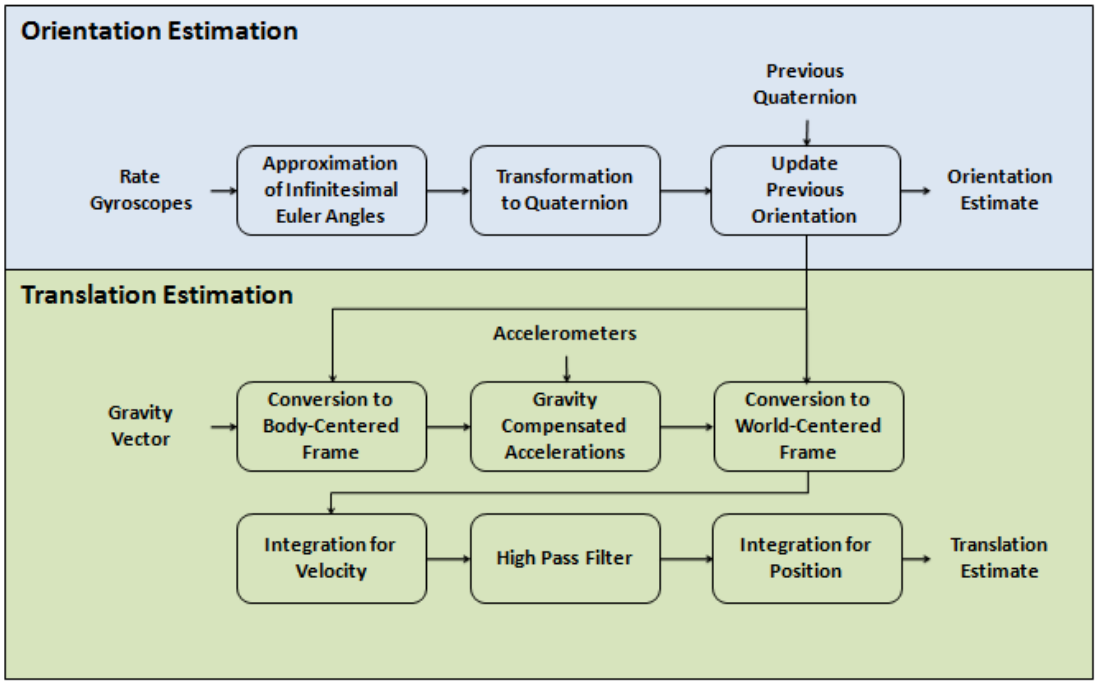

Figure III-2: Block diagram of IMU updating algorithm

The C code for this algorithm is contained in the function "`update_axes()`" found in APPENDIX A. This function is called at every sampling instant that the prosthesis detects that the foot is in the air (through monitoring of the heel and toe load sensors). The first task of this algorithm is to update the estimate of the orientation of the leg by performing a numerical integration of the rate gyros. The result is a small finite change in terms of Euler angles, which, when assumed to be small enough to approach infinitesimal quantities, are independent of the order in which they are applied. Since the

order of application does not matter, an arbitrary convention of *x*-*y*-*z* Euler rotations is applied. The rotations are represented by $d\phi$ for the *x* rotation, $d\theta$ for the y rotation, and $d\psi$ for the z rotation. These rotations are then transformed into an approximation of an infinitesimal quaternion, using the following relationship:

$$d\hat{\boldsymbol{q}} = \begin{bmatrix} q_o \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\frac{d\psi}{2}\cos\frac{d\theta}{2}\cos\frac{d\phi}{2} + \sin\frac{d\psi}{2}\sin\frac{d\theta}{2}\sin\frac{d\phi}{2} \\ \cos\frac{d\psi}{2}\cos\frac{d\theta}{2}\sin\frac{d\phi}{2} - \sin\frac{d\psi}{2}\sin\frac{d\theta}{2}\cos\frac{d\phi}{2} \\ \cos\frac{d\psi}{2}\sin\frac{d\theta}{2}\cos\frac{d\phi}{2} + \sin\frac{d\psi}{2}\cos\frac{d\theta}{2}\sin\frac{d\phi}{2} \\ \sin\frac{d\psi}{2}\cos\frac{d\theta}{2}\cos\frac{d\phi}{2} - \cos\frac{d\psi}{2}\sin\frac{d\theta}{2}\sin\frac{d\phi}{2} \end{bmatrix}. \quad (8)$$

The infinitesimal quaternion is then composed with the orientation quaternion from the previous sampling instant to create the current estimate.

$$\hat{\boldsymbol{q}}_{\boldsymbol{k}} = d\hat{\boldsymbol{q}} \cdot \hat{\boldsymbol{q}}_{\boldsymbol{k-1}}. \quad (9)$$

Once the orientation update is complete, the result is used to calculate the estimated translation of the foot. First, $\hat{\boldsymbol{q}}_{\boldsymbol{k}}$ is used to rotate the world frame gravity vector (simply $[0 \quad 0 \quad g]^T$, where g is the acceleration of gravity at the Earth's surface) into the body centered frame. The body centered gravity is then subtracted from the acceleration vector (as determined from the accelerometer

signals). The result of this operation is a gravity-compensated acceleration vector in the body centered frame, which can then be converted back to the world coordinate frame with the inverse rotation specified by $\hat{\boldsymbol{q}}_{\boldsymbol{k}}$. Once the gravity compensated world frame acceleration vector is computed, a numerical integration step updates the absolute translational velocity signal. In order to combat drift due to numerical inaccuracy and measurement error, this velocity signal is high pass filtered (cutoff of 0.08 Hz), with the rationale that the foot will never maintain a constant linear velocity for a significant amount of time. The velocity signal is then integrated again to provide the translational position estimate.

5. IMU Calibration

A great deal of effort can be spent in accurate calibration of a six-axis inertial measurement unit. However, because this application necessitates long term operation without tedious recalibration, a simple approach is employed to calibrate the linear scaling and offset of each accelerometer signal. Additionally, because all three axes are included on a single die in an integrated circuit, compensation for axis misalignment was not taken into consideration.

It is assumed that each un-calibrated accelerometer signal is linearly proportional to the true acceleration in its respective axis. As a result, it is

necessary to find a scaling factor and offset for each signal. It is assumed that during static conditions the accelerometer is only measuring the acceleration due to gravity, and consequently, the sum of the square of each signal should equal the square of 9.81 m/s^2. For a given orientation of the IMU, we have the following relation:

$$\left(s_x(a_x - o_x)\right)^2 + \left(s_y\left(a_y - o_y\right)\right)^2 + \left(s_z(a_z - o_z)\right)^2 = (9.81)^2 \qquad (10)$$

In this equation the a terms represent the measurements from each axis of the accelerometer, the s terms are the scaling values, and the o terms are the offsets. The equation is solved exactly for six independent orientations (assuming all a terms are non-zero). For calibration purposes all that is necessary is to take a large number of measurements at a variety of orientations and solve the non-linear least squares problem for this function.

A sample measurement set is shown prior to calibration in Figure III-3. The red data points correspond to estimates of the gravity vector that were overestimated by more than 0.25 m/s^2, the green data points correspond to estimates that were underestimated by more than 0.25 m/s^2, and the blue data points fell in between these two values. This data set consists of 348 independent configurations of the prosthetic foot. The measurements were averaged values

from 30 second periods in which the foot was clamped in a vice with a ball head attachment at various orientations (see Figure III-4). Calibration values were computed using the `lsqcurvefit` command in Matlab. The result of the calibration procedure can be seen in Figure III-5 while the calibration parameters are shown in Table 2.

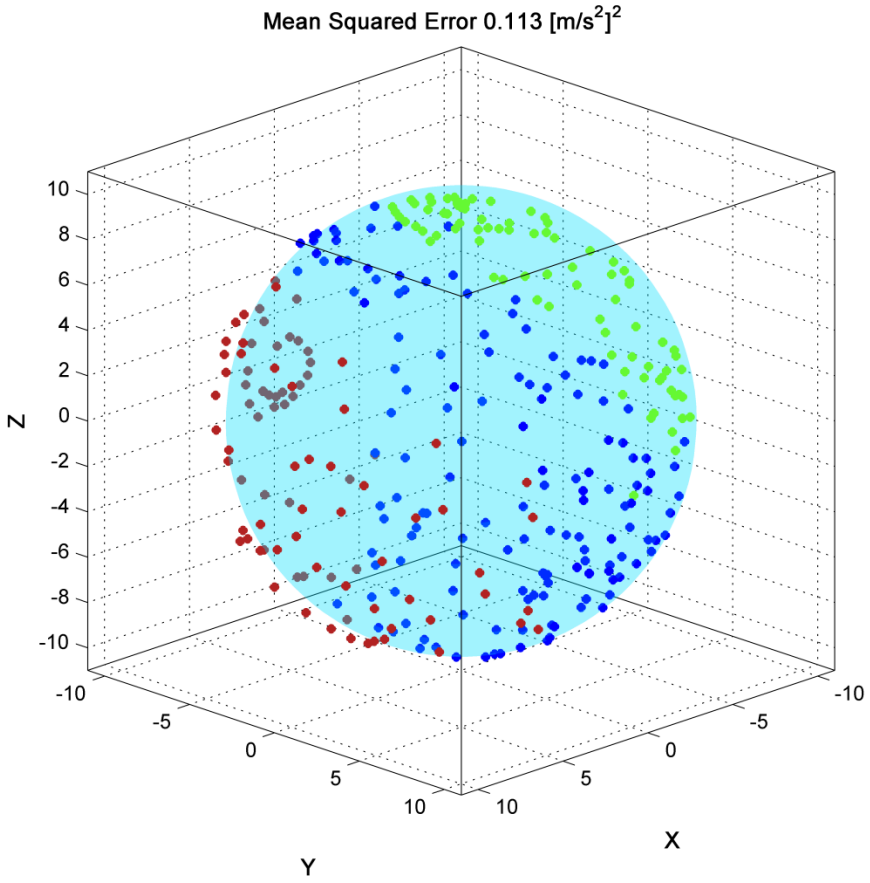

Figure III-3: Uncalibrated accelerometer measurement of the gravity vector

Figure III-4: Prosthetic foot clamped in vice for IMU calibration

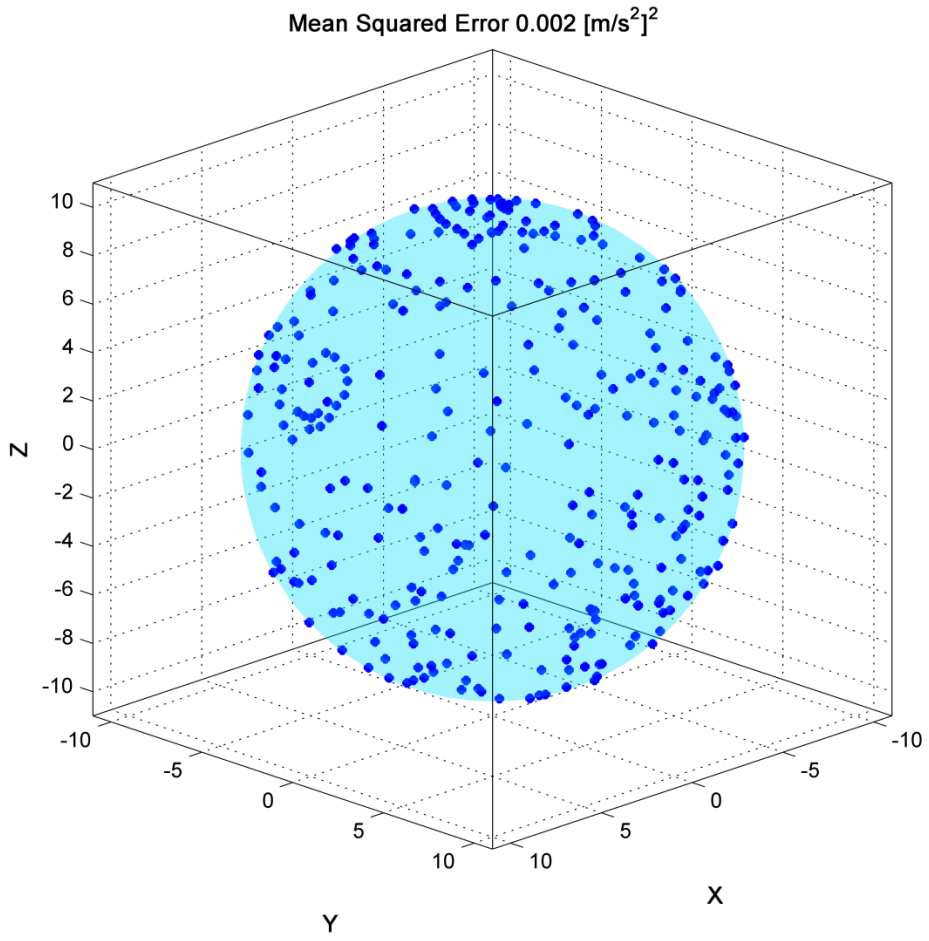

Figure III-5: Calibrated accelerometer measurement of the gravity vector

Axis	Scale	Offset
x	0.980136	0.490684
y	0.975858	-0.195463
z	1.021482	-0.235835

Table 2: Calculated parameters for IMU calibration

The mean squared error in the measurement of the gravity vector was reduced by almost two orders of magnitude through the calibration procedure. The cause of the skewedness in the accelerometers signals is most likely due to mechanical stress induced on the integrated circuit during both soldering and the tightening of the circuit board to the foot. Testing has revealed that re-calibration is necessary every time the board is removed from the prosthetic foot.

CHAPTER IV

STANDING CONTROLLER DESIGN

1. Original Standing Controller

The original standing controller implemented in the prosthesis described in [26, 27] consisted of a simple state machine with two states (see Figure IV-1). The purpose of this controller was to provide the high impedances necessary at the ankle and the knee for weight bearing, yet also allow movement of the knee when the user removes weight from the prosthesis.

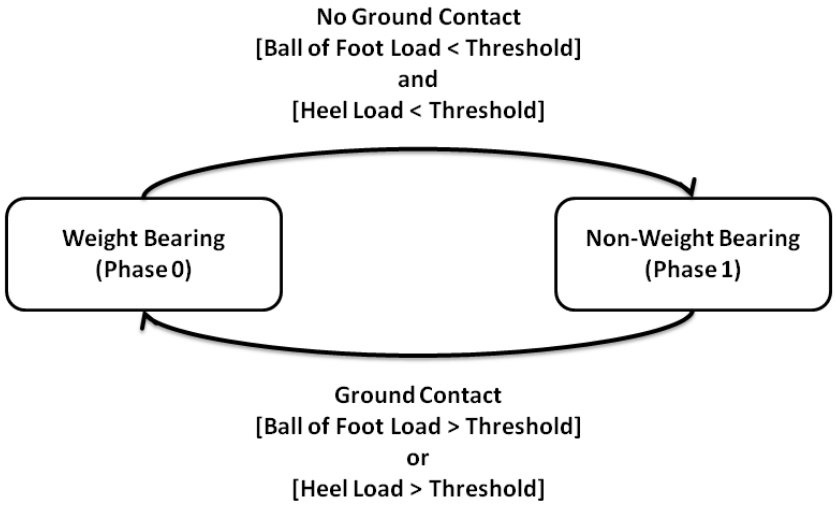

Figure IV-1: Finite state diagram for the previous standing controller

The primary advantage of such a simple standing controller was that its behavior was very predictable and intuitive for the user. For both states in this controller, a stiffness, damping coefficient, and an equilibrium position would be set for both the knee and ankle joints. As a result, the controller allowed full support for the user when standing on level ground. In addition, the user received greater stability at the knee due to the fact that the stiffness value of the knee in the weight bearing state would be tuned such that it could support the weight of the user. This is a significant advantage over a passive device that can only provide damping at the knee, forcing the user to hyperextend the knee and utilize the hard stop for static support. In addition, the low impedance, non-weight bearing state of the controller facilitated repositioning of the prosthesis as the user shifted his or her weight forward and back. The user could therefore shuffle about with a slow, meandering gait without necessarily switching into the more active walking mode of the prosthesis.

Despite these advantages, the original controller was still limited in that it assumed a static equilibrium position for the ankle during weight bearing. This means that, similar to a passive carbon fiber ankle-foot complex, the weight bearing state was only optimal for standing on level ground. Additionally, the state model contained no provisions for sitting, or the transitions from standing to sitting and vice versa. In order to perform these transitions, separate state

models were developed and the system's intent recognizer had to choose between them appropriately [33].

2. Multi-Purpose Standing Controller

The primary contribution of this thesis is the design and implementation of the following middle level standing controller. This controller addresses both the deficiencies found in the state of the art passive prostheses, along with those just highlighted in the previous standing controller for a powered prosthesis. The controller actively changes the equilibrium position of the ankle in its weight bearing phase through the use of the orientation estimation component of the IMU algorithm described in Chapter 3. It also incorporates provisions for making stand-to-sit and sit-to-stand transitions, allowing for a single middle level controller to govern all non-gait based activities. The result is significantly reduced complexity for the high level control of the prosthesis, along with improved stability for the user.

Figure IV-2 shows the state model for the complete multi-purpose standing controller. This controller contains 4 states, each of which corresponds to a different set of impedance parameters as dictated by Equation (1).

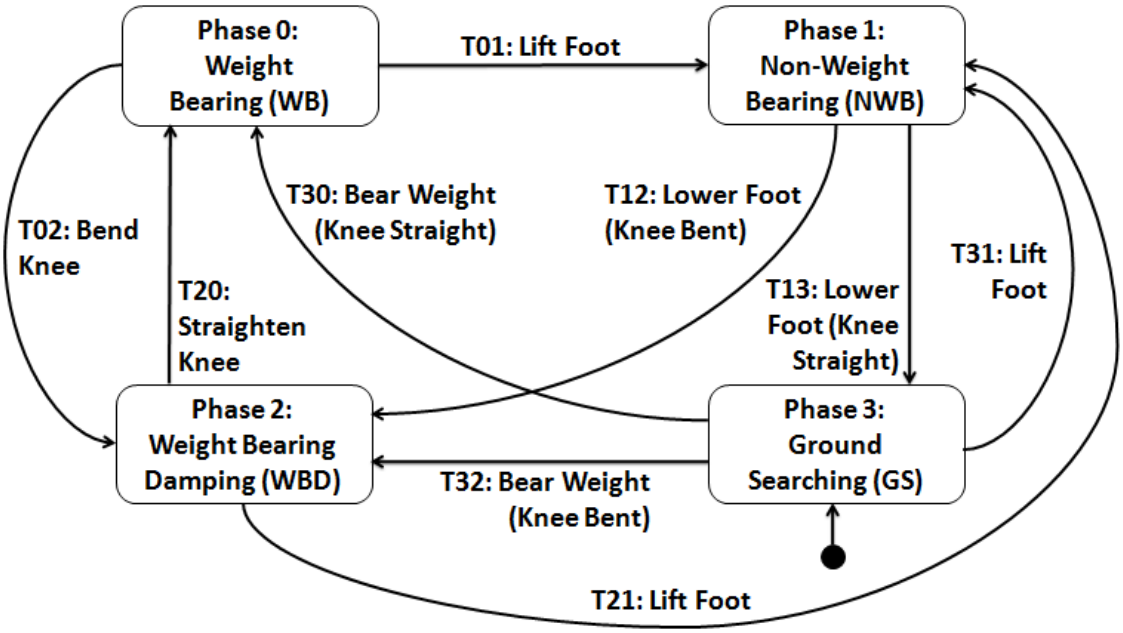

Figure IV-2: Finite state diagram for the ground adaptive standing controller

The first two phases (Phase 0 and Phase 1) are the weight bearing and non-weight bearing phases implemented in the original standing controller. The transition from weight bearing to non-weight bearing is triggered in the same way as before; the prosthesis detects a reduction in the heel and toe loads. In order to enter back into the weight bearing phase, however, the state machine must first move through Phase 3, the ground searching phase. This phase is characterized by a relatively stiff knee, but an ankle that acts as a moderate damper. This behavior allows the user to use his or her weight to cause the ankle to conform to the ground. In order to enter this phase from non-weight bearing, the user must bring either the heel or the toe of the foot into contact with the

ground (as indicated by the presence of a heel or toe load). The ground searching phase is completed once the IMU algorithm has successfully recalibrated itself by re-measuring the gravity vector. With an updated estimate of the gravity vector, the prosthesis can then produce an estimate of the ground slope and use this estimate to offset the ankle equilibrium point in the torque equation. Equation 11 shows the modified impedance equation used in the weight bearing phase for the generation of the ankle torque reference.

$$\tau_a = k_a\left(\theta_a - \theta_{eq_a} - \hat{\theta}_{gnd}\right) + b_a\dot{\theta}_a \tag{11}$$

In this equation, $\hat{\theta}_{gnd}$ is the estimate of the ground slope. It is important to note that θ_{eq_a} is still included as a tunable parameter for this phase. This is because the optimal ankle angle for standing on level ground is not necessarily zero degrees. θ_{eq_a} is a parameter that would ideally be tuned by a prosthetist, while $\hat{\theta}_{gnd}$ is dynamically selected by the controller. Regardless, once θ_{eq_a} is properly selected, the ankle equilibrium position will always be set correctly so long as the prosthesis is able to make a sufficient ground slope estimate.

In addition to the ground adaptation provided by the ground searching phase, the multi-purpose standing controller adds a weight bearing damping phase (Phase 2) in order to facilitate stand-to-sit and sit-to-stand transitions. The

weight bearing damping phase contains a uni-directional damping coefficient for the knee. The torque reference is generated by the following modification to the general impedance relation:

$$\tau_k = \begin{cases} k_k\left(\theta_k - \theta_{eq_k}\right) + b_k\dot{\theta}_k & \forall\ \dot{\theta}_k > 0 \\ k_k\left(\theta_k - \theta_{eq_k}\right) & \forall\ \dot{\theta}_k \leq 0 \end{cases} \tag{12}$$

In this way the knee provides damping as the knee angle increases (i.e. the user is trying to sit down), but does not provide viscous damping when the user is trying to stand up. This resistance allows the prosthesis to bear weight when sitting down, effectively slowing the user's descent.

A complete list of phase transitions is provided in Table 3 along with the conditions that are tested in the prosthesis in order to initiate the transitions. Phase transitions are labeled by the letter *T*, and suffixed first by the origin phase and then by the destination phase. For example, transition *T01* refers to a transition from the weight bearing phase (phase 0) to the non-weight bearing phase (phase 1). These transitions can also be examined in the source code provided in the Appendix.

Transition	Description	Conditions
T01	User shifts weight off of the prosthesis	Toe load is low Heel load is low
T02	User sits down	Knee is bent Toe load is high
T12	User puts weight on prosthesis while sitting	Toe load or heel load is high Knee is bent
T13	User puts weight on prosthesis while standing	Toe load or heel load is high Knee is straight
T20	User stands up	Toe load or heel load is high Knee is straight
T21	User removes weight from prosthesis while sitting	Toe load is low Heel load is low Knee is bent
T30	User maintains weight on prosthesis and ground angle has been determined	Toe load is high Heel load is high IMU has zeroed Knee is straight
T31	User shifts weight off prosthesis before ground angle has been determined	Toe load is low Heel load is low
T32	User maintains weight on prosthesis while sitting and ground angle has been	Toe load is high Heel load is high IMU has zeroed Knee is bent

Table 3: Finite state transitions for the multi-purpose standing controller

With the 4 impedance phases defined in this controller, an amputee should be able to perform the majority of non-gait based lower limb activities. When the thresholds and impedance parameters have been empirically tuned to account for the weight and behavior of the user, the movements and torques produced by the prosthesis should be smooth, predictable and seamless. In order to verify that this framework can achieve such behavior, the following chapter describes several experiments that were conducted with a transfemoral amputee using the powered prosthesis with the multi-purpose standing controller.

CHAPTER V

STANDING BEHAVIOR VERIFICATION

The multi-purpose standing controller was implemented on the powered prosthesis previously described in order to verify that it is capable of providing a comprehensive standing behavior. A series of tests were performed by a transfemoral amputee subject 22 years of age and 4 years post amputation. The subject was male and his amputation was the result of a traumatic injury. His daily use prosthesis consists of an Otto Bock C-Leg® and a Renegade® carbon fiber ankle-foot complex by Freedom Innovations.

Three experiments were performed in order to characterize the behavior of the prosthesis. In these experiments the subject was asked to stand on a series of wedges constructed from wood and covered with anti-slip tread. The wedges ranged from -15 degrees to +15 degrees relative to gravity. Before any data was collected the subject was trained on the prosthesis and the impedance parameters were empirically tuned in order to provide the most comfortable behavior for the subject. The structure of the controller was also explained in detail to the subject such that he had reasonable intuition as to how the device would behave. The

impedance parameters used for the subject in these experiments are given in Table 4.

Phase	**Knee Stiffness**	**Knee Damping**	**Knee Equilibrium** $\boldsymbol{\theta_{eq_k}}$	**Ankle Stiffness**	**Ankle Damping**	**Ankle Equilibrium** $\boldsymbol{\theta_{eq_a}}$
0	2.00	0.10	0.00	4.00	0.20	0.00
1	0.80	0.04	5.00	1.00	0.20	-3.00
2	0.00	0.04	0.00	0.00	0.40	0.00
3	1.40	0.30	0.00	0.00	0.10	0.00
Units	$\frac{Nm}{deg}$	$\frac{Nm \cdot s}{deg}$	deg	$\frac{Nm}{deg}$	$\frac{Nm \cdot s}{deg}$	deg

Table 4: Impedance parameters used in the experiments

1. Experiment 1: Ground Slope Estimation and Adaptation

In the first experiment the subject was asked to step sequentially from wedge to wedge, causing the prosthesis to adapt to ground slopes varying from negative 15 degrees to positive 15 degrees in 5 degree increments. Before moving to the next wedge, the subject was asked to increase his postural sway on purpose in order to verify the proper stiffness and equilibrium of the ankle. Figure V-1 shows the ground slope estimate for a representative trial as

measured internally by the embedded system and logged on the SD card. The subject started the experiment on level ground, and then he proceeded to the next wedge approximately every 10 to 12 seconds. The blue bands in the figure indicate the range of ±1 degree around the ground angle for each wedge. The absolute angles of the wedges relative to gravity were verified before the experiment with a digital protractor and found to be within 0.3 degrees of their nominal values.

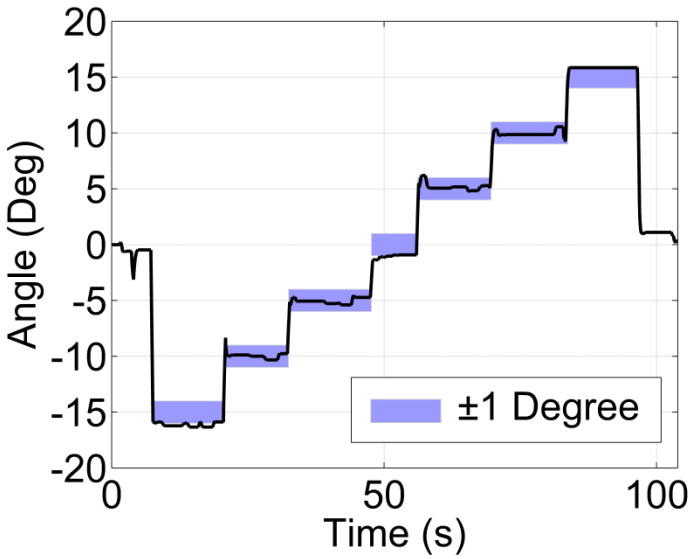

Figure V-1: Ground slope estimate during a series of transitions from -15° to +15°

Figure V-1 shows that the prosthesis is capable of measuring ground slopes within ±1 degree over the entire range of angles that would typically be seen in daily life. More extreme ground angles could not be tested without

compromising the safety of the amputee, but ground angles greater than 15 degrees are typically avoided by healthy subjects for the same reason. Regardless, there is no inherent reason that the device should perform differently at different angles.

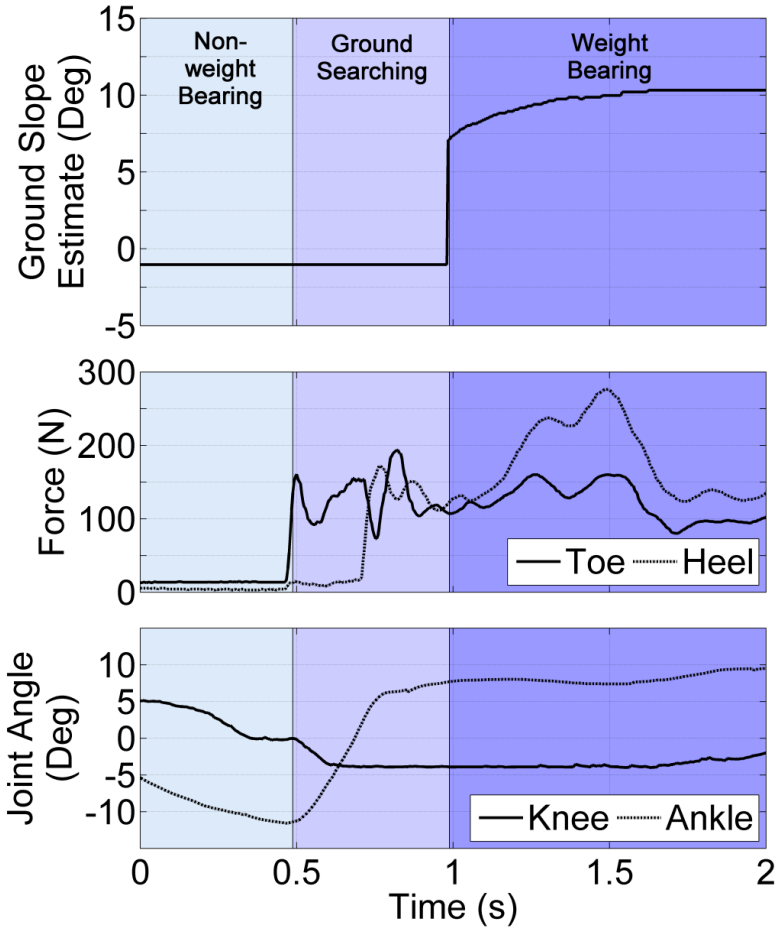

Figure V-2: Transition from non-weight bearing to weight bearing with ground adaptation

Figure V-2 highlights a particular transition from the non-weight bearing phase through the ground searching phase and into the weight bearing phase as the subject lands on a 10 degree up slope. Initially, the prosthesis is in the non-weight bearing phase (phase 1), indicating that the foot is currently off of the ground. This can be confirmed by the fact that the toe and heel loads are near zero during this time. At approximately t = 0.5 seconds a sudden toe load causes the prosthesis to enter the ground searching phase (phase 3). This corresponds to the instant that the user brings the prosthesis into contact with the up slope.

Once in the ground searching phase, the ankle acts purely as a damper and the knee stiffens to transmit the user's body weight (see Table 4). As the user continues to put weight on the prosthesis the ankle conforms to the ground slope, eventually registering a heel load at t = 0.75 seconds. During this time the ankle angle moves from its plantar-flexed position up to approximately 8 degrees (Note that the ankle angle does not correspond to the ground angle unless the shank is entirely vertical which, in general, it is not).

At this point the IMU algorithm begins monitoring the variation in the gravity vector signal from the accelerometers. Once this variation becomes sufficiently small, the prosthesis zeroes the inertial measurements and assumes an initial estimate of the ground angle. This event allows the transition into the weight bearing phase. The knee remains stiff and the ankle assumes an

appropriate stiffness around its tuned set point, but offset by the measured ground angle. Once in the weight bearing phase, the IMU algorithm continues to improve its estimate of the gravity vector as the foot becomes more solidly planted on the slope. By approximately t = 1.5 seconds (or 1 second after ground contact) the ground slope is at a steady-state condition and the prosthesis is fully adapted to the ground slope.

When the prosthesis is fully adapted in the weight bearing phase, the ankle is able to exhibit a passive stiffness behavior equivalent to that of a traditional carbon fiber ankle-foot complex, though at any equilibrium point chosen by the ground slope estimate. Stiffness plots for all the conditions in the first experiment are shown in Figure V-3. Also shown in the figure are the corresponding linear least squares fits for each standing condition. In the plot the effective equilibrium point of the ankle in each condition can be seen by inspecting the angle at which zero torque is commanded. Since this is the same data as Figure V-1, the equilibrium angles remain within ±1 degree of the measured ground slope. Additionally, the slopes of the linear least squares fits give an approximate measure of the ankle stiffness, which is set at 4.0 Nm/deg for the weight bearing phase. Of course, these fits are not an exact measure of the stiffness because they do not account for the velocity dependence of the torque reference due to the damping coefficient (0.20 Nm·s/deg). Regardless, the

subject's postural sway was relatively slow in this trial and the slopes of the linear fits (which can be seen in the caption of Figure V-3) all show approximately 4.0 Nm/deg.

This experiment has characterized three important performance metrics regarding ground adaptation in the multi-purpose standing controller. Firstly, it has shown that the controller is capable of measuring ground slopes with an accuracy of ±1 degree. Secondly, it can make these measurements and adjust the ankle's equilibrium position within approximately 0.25 seconds of complete ground contact. Finally, it can provide an appropriate stiffness for postural sway at the ground angle it has measured, allowing for full standing support on ground slopes within a range of ±15 degrees relative to gravity.

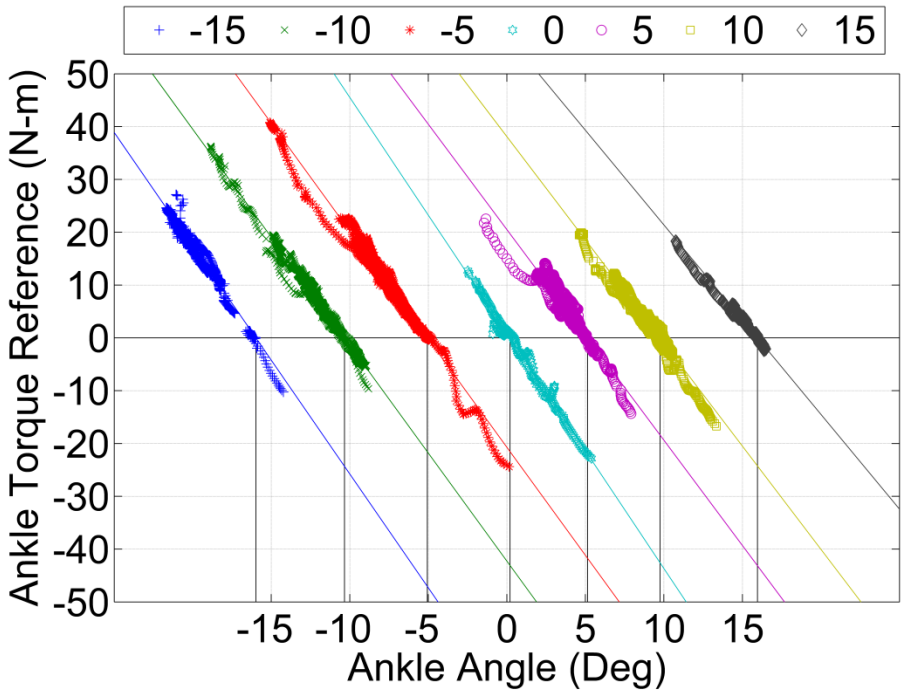

Figure V-3: Stiffness plots for a range of ground angles

$$\tau_0 = -4.5(\theta + 0)$$
$$\tau_{-15} = -4.3(\theta + 16) \quad \tau_{15} = -3.6(\theta - 16)$$
$$\tau_{-10} = -4.1(\theta + 10) \quad \tau_{10} = -3.9(\theta - 10)$$
$$\tau_{-5} = -4.1(\theta + 5) \quad \tau_5 = -4.0(\theta - 5)$$

2. Experiment 2: General Behavior Testing

The second experiment aimed to assess the general performance of the controller by simultaneously testing all the impedance states and their transitions. This test was subjective in nature and was performed in order to receive feedback from the subject regarding the overall functionality of the controller. In order to assess functionality, the subject was asked to navigate three terrain scenarios constructed from the wedges used in the first experiment.

All scenarios were recorded on video for visual assessment, and they were performed with both the subject's daily use prosthesis and the powered prosthesis.

In the first scenario (referred to as the static scenario), the subject was asked to step from wedge to wedge in a manner similar to Experiment 1. At each wedge the subject was asked to stop and forcibly increase his postural sway before moving to the next wedge.

In the second scenario (referred to as the semi-dynamic scenario), the subject was asked to navigate a short walkway of varying and alternative up and down slopes. The goal of this scenario was to navigate in as safe and slowly a manner as possible to demonstrate the stability of the prosthesis. A more deliberate and faster gait would cause the intent recognizer in the prosthesis to change the middle layer controller to the walking mode, which is outside the scope of this work.

The third scenario consisted of having the subject navigate several wedges and then make a stand-to-sit transition and sit-to-stand transition, demonstrating the complete functionality of the controller in a single video.

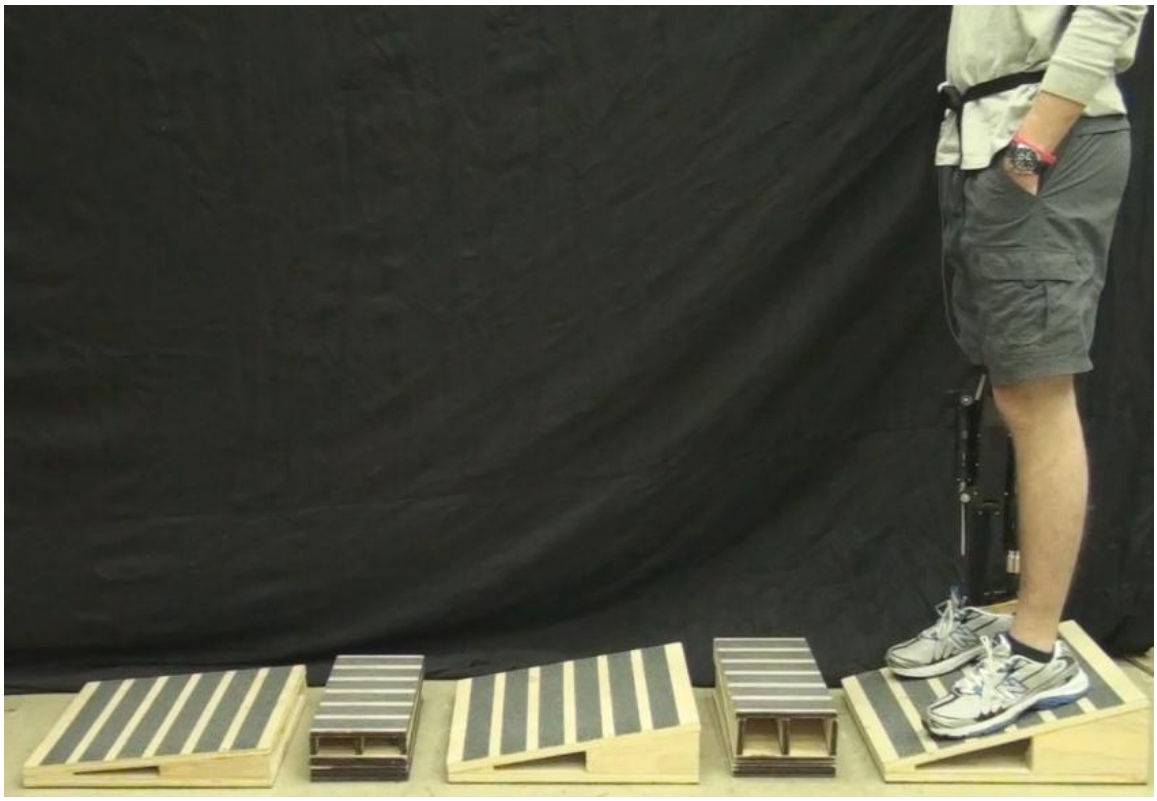

Figure V-4: Terrain setup for the first (static) scenario

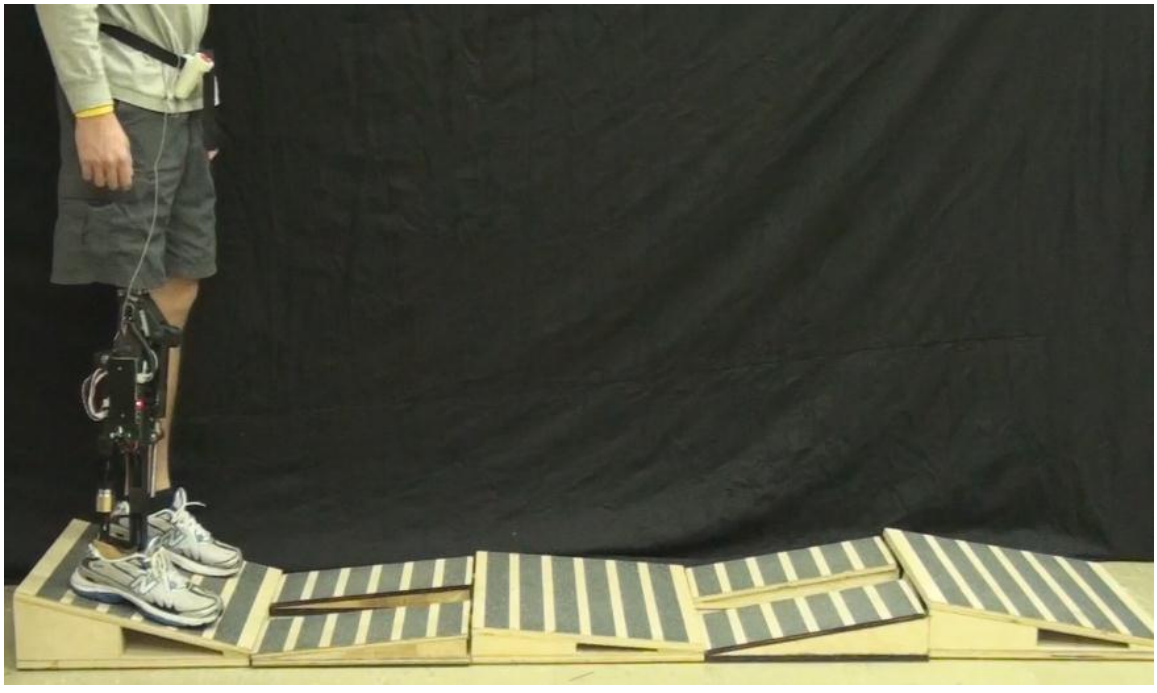

Figure V-5: Terrain setup for the second (semi-dynamic) scenario

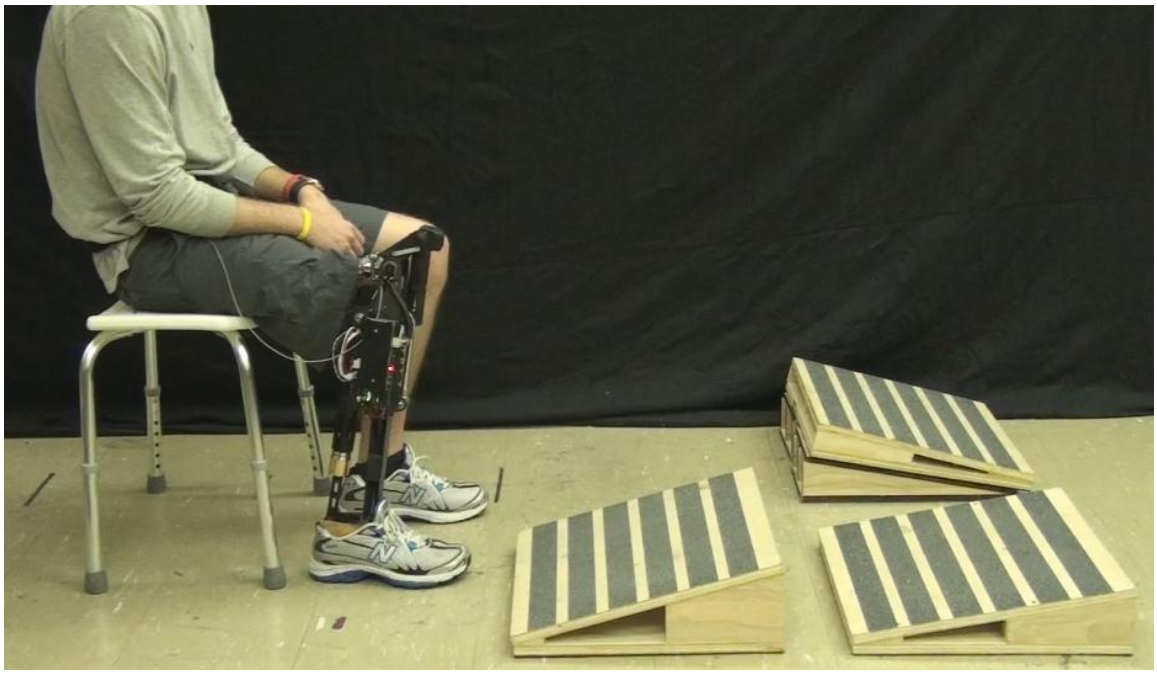

Figure V-6: Terrain setup for the third (sitting and slope) scenario

In all tests the operation of the prosthesis was smooth and predictable. As can be seen in the videos, the subject was able to traverse the walkways in a slow and deliberate fashion with the powered prosthesis, while the periods of single support with his passive prosthesis were necessarily short and unstable. The fixed equilibrium position of the carbon fiber ankle-foot complex is also visually evident when he stands on the more extreme up and down slopes.

3. Experiment 3: Weight Bearing Distribution

The goal of the third experiment was to measure to degree to which the subject was able to receive support from the prosthesis once adapted to the ground slope. As previously stated, lower limb amputees tend to favor their sound leg in terms of weight bearing distribution by a ratio of approximately 60-40% when they stand on level ground [14-16]. In this experiment the subject was asked to stand on all the slope conditions tested previously, though in this case the wedges were split and load cells were placed under each segment to measure the relative load on each leg.

The results of this experiment are displayed as bar graphs in Figure V-7. For the passive prosthesis, the subject chose a weight bearing distribution approximately equivalent to that found in the literature for level ground and small slopes. As the ground slope increased, however, this ratio increased dramatically. In the case of a 15 degree down slope the subject bore virtually all his weight on his sound leg. For the powered prosthesis with the multi-purpose standing controller, the subject was able to maintain a consistent weight bearing distribution across all slopes. Furthermore, this ratio improved relative to his passive prosthesis on level ground, presumably because the knee could provide static support at a comfortable angle instead of resting against the hyperextension hard stop.

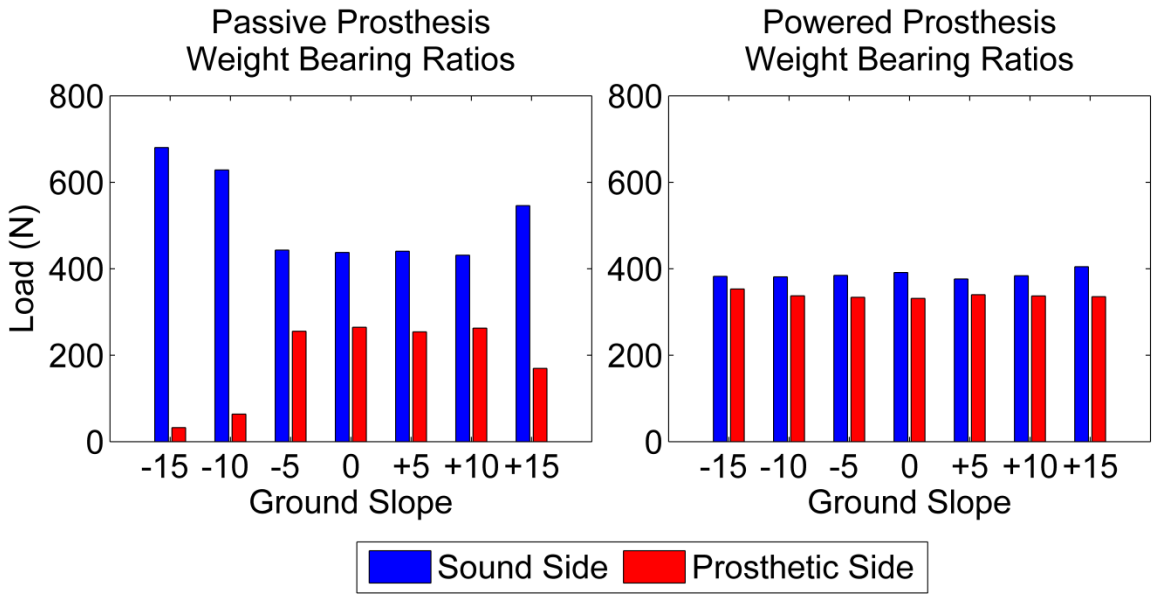

Figure V-7: Comparison of weight bearing ratios

4. Conclusion

Although the multi-purpose standing controller described in this work applies to a certain manifestation of a powered prosthesis, a major implication of its success is that intelligence and power can offer significantly enhanced behaviors in lower limb prosthetic devices. Specifically, the combination of inertial measurement and the ability to supply virtual (i.e. reconfigurable) stiffnesses allows a prosthesis to quickly, accurately and safely adapt to a variety of terrain conditions. This adaptation allows the user to receive full support from their affect side in these conditions. Presumably, this increase in support will lead to enhanced stability as measured by a reduction in the rate of falls and injuries seen in the amputee population.

REFERENCES

[1] C. Gauthier-Gagnon, M. C. Grise, and D. Potvin, Enabling factors related to prosthetic use by people with transtibial and transfemoral amputation, *Arch Phys Med Rehabil,* vol. 80, pp. 706-13, Jun 1999.

[2] K. Ziegler-Graham, E. J. MacKenzie, P. L. Ephraim, T. G. Travison, and R. Brookmeyer, Estimating the prevalence of limb loss in the United States: 2005 to 2050, *Arch Phys Med Rehabil,* vol. 89, pp. 422-9, Mar 2008.

[3] A. Leland and M. J. Oboroceanu, American war and military operations casualties: Lists and statistics: Congressional Research Service, 2010.

[4] B. J. Hafner and D. G. Smith, Differences in function and safety between Medicare Functional Classification Level-2 and -3 transfemoral amputees and influence of prosthetic knee joint control, *J Rehabil Res Dev,* vol. 46, pp. 417-33, 2009.

[5] R. L. Waters and S. Mulroy, The energy expenditure of normal and pathologic gait, *Gait Posture,* vol. 9, pp. 207-31, Jul 1999.

[6] S. M. Jaegers, J. H. Arendzen, and H. J. de Jongh, Prosthetic gait of unilateral transfemoral amputees: a kinematic study, *Arch Phys Med Rehabil,* vol. 76, pp. 736-43, Aug 1995.

[7] M. Schmid, G. Beltrami, D. Zambarbieri, and G. Verni, Centre of pressure displacements in trans-femoral amputees during gait, *Gait Posture,* vol. 21, pp. 255-62, Apr 2005.

[8] W. C. Miller, M. Speechley, and B. Deathe, The prevalence and risk factors of falling and fear of falling among lower extremity amputees, *Arch Phys Med Rehabil,* vol. 82, pp. 1031-7, Aug 2001.

[9] W. C. Miller, A. B. Deathe, M. Speechley, and J. Koval, The influence of falling, fear of falling, and balance confidence on prosthetic mobility and social activity among individuals with a lower extremity amputation, *Arch Phys Med Rehabil,* vol. 82, pp. 1238-44, Sep 2001.

[10] L. E. Powell and A. M. Myers, The Activities-specific Balance Confidence (ABC) Scale, *J Gerontol A Biol Sci Med Sci,* vol. 50A, pp. M28-34, Jan 1995.

[11] W. C. Miller, M. Speechley, and A. B. Deathe, Balance confidence among people with lower-limb amputations, *Phys Ther,* vol. 82, pp. 856-65, Sep 2002.

[12] E. Isakov, J. Mizrahi, H. Ring, Z. Susak, and N. Hakim, Standing sway and weight-bearing distribution in people with below-knee amputations, *Arch Phys Med Rehabil,* vol. 73, pp. 174-8, Feb 1992.

[13] J. G. Buckley, D. O'Driscoll, and S. J. Bennett, Postural sway and active balance performance in highly active lower-limb amputees, *Am J Phys Med Rehabil,* vol. 81, pp. 13-20, Jan 2002.

[14] M. Lord and D. M. Smith, Foot loading in amputee stance, *Prosthet Orthot Int,* vol. 8, pp. 159-64, Dec 1984.

[15] A. H. Vrieling, H. G. van Keeken, T. Schoppen, E. Otten, A. L. Hof, J. P. Halbertsma, and K. Postema, Balance control on a moving platform in unilateral lower limb amputees, *Gait Posture,* vol. 28, pp. 222-8, Aug 2008.

[16] P. R. Rougier and J. Bergeau, Biomechanical analysis of postural control of persons with transtibial or transfemoral amputation, *Am J Phys Med Rehabil,* vol. 88, pp. 896-903, Nov 2009.

[17] K. R. Kaufman, J. A. Levine, R. H. Brey, B. K. Iverson, S. K. McCrady, D. J. Padgett, and M. J. Joyner, Gait and balance of transfemoral amputees using passive mechanical and microprocessor-controlled prosthetic knees, *Gait Posture,* vol. 26, pp. 489-93, Oct 2007.

[18] S. T. Blumentritt, T. Schmals, and R. Jarasch, Safety of C-Leg: Biomechanical Tests, *Journal of Prosthetics and Orthotics,* vol. 21, pp. 2-17, 2009.

[19] B. J. Hafner, L. L. Willingham, N. C. Buell, K. J. Allyn, and D. G. Smith, Evaluation of function, performance, and preference as transfemoral amputees transition from mechanical to microprocessor control of the prosthetic knee, *Arch Phys Med Rehabil,* vol. 88, pp. 207-17, Feb 2007.

[20] K. R. Kaufman, J. A. Levine, R. H. Brey, S. K. McCrady, D. J. Padgett, and M. J. Joyner, Energy expenditure and activity of transfemoral amputees using mechanical and microprocessor-controlled prosthetic knees, *Arch Phys Med Rehabil,* vol. 89, pp. 1380-5, Jul 2008.

[21] M. S. Orendurff, A. D. Segal, G. K. Klute, M. L. McDowell, J. A. Pecoraro, and J. M. Czerniecki, Gait efficiency using the C-Leg, *J Rehabil Res Dev,* vol. 43, pp. 239-46, Mar-Apr 2006.

[22] R. Seymour, B. Engbretson, K. Kott, N. Ordway, G. Brooks, J. Crannell, E. Hickernell, and K. Wheeler, Comparison between the C-leg microprocessor-controlled prosthetic knee and non-microprocessor control prosthetic knees: a preliminary study of energy expenditure, obstacle course performance, and quality of life survey, *Prosthet Orthot Int,* vol. 31, pp. 51-61, Mar 2007.

[23] D. Datta and J. Howitt, Conventional versus microchip controlled pneumatic swing phase control for trans-femoral amputees: user's verdict, *Prosthet Orthot Int,* vol. 22, pp. 129-35, Aug 1998.

[24] T. Chin, S. Sawamura, R. Shiba, H. Oyabu, Y. Nagakura, I. Takase, K. Machida, and A. Nakagawa, Effect of an Intelligent Prosthesis (IP) on the walking ability of young transfemoral amputees: comparison of IP users with able-bodied people, *Am J Phys Med Rehabil,* vol. 82, pp. 447-51, Jun 2003.

[25] Z. L. Justin and G. Robert, Advances in lower-limb prosthetic technology, *Physical medicine and rehabilitation clinics of North America,* vol. 21, pp. 87-110, 2010.

[26] F. Sup, H. A. Varol, J. Mitchell, T. J. Withrow, and M. Goldfarb, Preliminary Evaluations of a Self-Contained Anthropomorphic Transfemoral Prosthesis, *IEEE ASME Trans Mechatron,* vol. 14, pp. 667-676, 2009.

[27] F. Sup, A powered self-contained knee and ankle prosthesis for near normal gait in transfemoral amputees, Ph.D., Mechanical Engineering, Vanderbilt University, Nashville, TN, 2009.

[28] H. A. Varol, Progress towards the intelligent control of a powered transfemoral prosthesis, Ph.D., Mechanical Engineering, Vanderbilt University, Nashville, TN, 2009.

[29] D. A. Winter, The biomechanics and motor control of human gait: Normal, elderly and pathological. Waterloo, Ontario, Canada: University of Waterloo Press, 1991.

[30] W. Wrigley, The History of Inertial Navigation, *The Journal of Navigation,* vol. 30, pp. 61-68, 1977.

[31] W. R. Hamilton, Lectures on quaternions; containing a systematic statement of a new mathematical method; of which the principles were communicated in 1843 to the Royal Irish academy; and which has since formed the subject of successive courses of lectures, delivered in 1848 and subsequent years, in the halls of Trinity college, Dublin: with numerous illustrative diagrams, and with some geometrical and physical applications. Dublin,: Hodges and Smith; etc., 1853.

[32] J. C. Hart, G. K. Francis, and L. H. Kauffman, Visualizing quaternion rotation, *ACM Trans. Graph.*, vol. 13, pp. 256-276, 1994.

[33] H. A. Varol, F. Sup, and M. Goldfarb, Powered Sit-to-Stand and Assistive Stand-to-Sit Framework for a Powered Transfemoral Prosthesis, *IEEE Int Conf Rehabil Robot,* vol. 5209582, pp. 645-651, 2009.

APPENDIX A.

Source Code for IMU Routines

The following source code was written for the PIC32MX575F512L (Microchip Technology Inc.) microcontroller. It was written and tested in Microchip's integrated development environment (MPLAB IDE v8.50) and was compiled by the MPLAB C32 compiler, v1.10. The first code segment is an excerpt from the main body of the code. This segment governs the logic behind the IMU routines and calls the updating algorithm. The second code segment contains the contents of the file "`rleg_inertia.c`", which contains all the IMU-specific routines. The third code segment contains the contents of the file "`rleg_inertia.h`", which is the header file for the IMU routines. The code provided here is for reference only and has significant outside dependencies, including references to global variables and other functions. It is not intended to be independently compiled.

```
// IMU EXCERPT FROM MAIN SOURCE FILE FOR PIC32
MT1_HI;                                 // Flag for benchmarking

sort_foot_signals(&imu_foot);           //  Organizing raw data

// Measure variation in the accelerometers
settledness = rleg_settle(imu_foot.accel, imu_foot.acclp);

// Ground Contact Condition
if (((anktor_ref > 5000) || (anktor_ref < -5000)) && ( am != 4 ))
{
  if ( ((settledness_old+(run_time-settle_time_old)*30)>settledness) )
    {
      imu_tracking_start_time = run_time;
      imu_tracking_valid = 1;
      imu_zero_flag = 1;
      imu_zero_once = 1;

      imu_zero_gyros(&imu_foot);
      mG = reset_axes(imu_foot, &axe_foot, &axe_foot_old);

      settledness_old = settledness;
      settle_time_old = run_time;
    }
  else
    {
      axe_foot = update_axes(imu_foot,imu_foot_old,axe_foot_old,mG);
    }
}
else
{
  settledness_old = 1000000000;
  settle_time_old = 0;
  imu_zero_flag = 0;
  axe_foot = update_axes(imu_foot,imu_foot_old,axe_foot_old,mG);
}

// Rotation Matrix out of a Quaternion
//-------------------------------------------------------------------
//    q1^2+q2^2-q3^2-q4^2     2*q2*q3 - 2*q1*q4     2*q2*q4 + 2*q1*q3
//R = 2*q2*q3 + 2*q1*q4       q1^2-q2^2+q3^2-q4^2   2*q3*q4 - 2*q1*q2
//    2*q2*q4 - 2*q1*q3       2*q3*q4 + 2*q1*q2     q1^2-q2^2-q3^2+q4^2
//-------------------------------------------------------------------
// Z-X-Y Euler Angles make the most sense for our application, as a
// "barrel roll" is the least likely motion for an amputee to make with
// the prosthesis, so we will put the singularity there.
//
//    c(1)*c(3)-s(1)*s(2)*s(3)    -s(1)*c(2)   c(1)*s(3)+s(1)*s(2)*c(3)
//R = c(1)*s(2)*s(3)+s(1)*c(3)     c(1)*c(2)   s(1)*s(3)-c(1)*s(2)*c(3)
//   -c(2)*s(3)                    s(2)        c(2)*c(3)
```

```
if (imu_zero_once == 1)
{
  fotang_pos = lkarctan2(
                  -(2*axe_foot.q_w2b[1]*axe_foot.q_w2b[3]/TRIG_RADIX
                  - 2*axe_foot.q_w2b[0]*axe_foot.q_w2b[2]/TRIG_RADIX),
                  axe_foot.q_w2b[0]*axe_foot.q_w2b[0]/TRIG_RADIX
                  - axe_foot.q_w2b[1]*axe_foot.q_w2b[1]/TRIG_RADIX
                  - axe_foot.q_w2b[2]*axe_foot.q_w2b[2]/TRIG_RADIX
                  + axe_foot.q_w2b[3]*axe_foot.q_w2b[3]/TRIG_RADIX )
                  - 5500;
  if ((fotang_pos - fotang_pos_old) > 2500)
  {
    fotang_pos = fotang_pos_old + 2500;
  }
  else if ((fotang_pos - fotang_pos_old) < -2500)
  {
    fotang_pos = fotang_pos_old - 2500;
  }
  shkang_pos = fotang_pos-ankpos_calib;
  thiang_pos = shkang_pos+knepos_calib;

  fotang_vel = (fotang_vel*9 + (-imu_foot.omega[1]))/10;
  shkang_vel = fotang_vel-ankvel_calib;
  thiang_vel = shkang_vel+knevel_calib;
}
else
{
  fotyaw_calib = 0;
  fotrol_calib = 0;

  fotang_pos = 0;
  shkang_pos = 0;
  thiang_pos = 0;

  fotang_vel = 0;
  shkang_vel = 0;
  thiang_vel = 0;
}

imu_foot_old = imu_foot;
axe_foot_old = axe_foot;
fotang_pos_old = fotang_pos;

if ( imu_tracking_valid == 1 )
{
  if ((toload_calib < 15000)&&(heload_calib < 15000))
  {
    ground_contact_lost = 1;
  }
  if ( (ground_contact_lost==1)
      &&
     ( (toload_calib>20000) || (heload_calib > 20000) ) )
  {
    ground_contact_lost = 0;
    imu_tracking_valid = 0;
  }
  if (run_time > (imu_tracking_start_time + 1500/SAMPLE_TIME) )
```

```
        {
        ground_contact_lost = 0;
        imu_tracking_valid = 0;
        }
}
////////////////////////////////////////////////////////////
MT1_LOW;
```

```
/************************************************************************
Center for Intelligent Mechatronics
Copyright (c) 2010,2011 Vanderbilt University
All Rights Reserved.

FileName:        rleg_inertia.c
Processor:       PIC32MX575F512L
Compiler:        MPLAB® C32 v1.10 or higher
IDE:             MPLAB® IDE v8.50 or later

 ADDITIONAL NOTES:


************************************************************************/
#include "rleg_inertia.h"

// System Include Files
#include <plib.h>
#include <math.h>

// User Include Files
#include "Pic32Types.h"
#include "Pic32Main.h"
#include "P2PSpi.h"

// Symbolic Constants
#define IMU_PI       3.14159265358979323846
#define IMU_3DEG     0.052359877559830           // 3 degrees in radians
#define IMU_1DEG     0.017453292519943           // 1 degree in radians
#define IMU_NLKUP    1024   // elements in the lookup tables for trig functions
#define N_HP         3

// Global Variable Declarations
// Shared - These variables are declared 'extern' in header.
extern P2P_DATA      gP2P;
extern INT32         gnd_slope;

extern INT32 accx_no_grav;
extern INT32 accy_no_grav;
extern INT32 accz_no_grav;

extern INT32 rawvelx;
extern INT32 rawvely;
extern INT32 rawvelz;
extern INT32 rawvelx_old;
extern INT32 rawvely_old;
extern INT32 rawvelz_old;
extern INT32 velxhp;
extern INT32 velyhp;
extern INT32 velzhp;

// Private - The scope of these variables is this file.
INT64 _dstable[IMU_NLKUP] = {0};
INT64 _dctable[IMU_NLKUP] = {0};
INT32 arcsintable[IMU_NLKUP] = {0};

// Function Prototypes
// Shared - These functions are declared 'extern' in header.
```

```
void   sort_foot_signals(Imu *foot);
Axes   update_axes(Imu imu, Imu imu_old, Axes axes_old, INT32 mG);
INT32  reset_axes(Imu imu_foot, Axes *axe_foot, Axes *axe_foot_old);
void   imu_zero_gyros(Imu *foot);
void   init_IMUlookup(void);
void   quatmultiply64(Qtrn64 q1, Qtrn64 q2, Qtrn64 qout, INT64 radix);
void   quatnormalize64(Qtrn64 q, INT64 radix);
void   quatinverse64(Qtrn64 q, Qtrn64 qout, INT64 radix);
void   quatrotate64(Qtrn64 q, Qtrn64 qinv, Vec3 v, Vec3 n, INT64 radix);
INT32  dot32(Vec3 v1, Vec3 v2, INT32 radix);
INT32  rleg_settle(Vec3 acc, Vec3 alp);
INT32  lkarctan2(INT64 y, INT64 x);
INT32  lkarcsin(INT64 oh);

// Private

// Sin Lookup (small angle) In: IMU_RADIX*1000 of deg | Out: ratios of TRIG_RADIX
INT64 _ds(INT32 angle);

// Cos Lookup (small angle) In: IMU_RADIX*1000 of deg | Out: ratios of TRIG_RADIX
INT64 _dc(INT32 angle);
INT32 imu_quat_estimator(Vec3 acc, Qtrn64 q, Qtrn64 qinv);
INT32 fxsqrt_32(INT32 x);
INT64 fxsqrt_64(INT64 x);

// Functions
void sort_foot_signals(Imu *foot)
{
       static INT32  hpx = 0;
       static INT32  hpy = 0;
       static INT32  hpz = 0;

       foot->accel[0]=calimuf.sx*(gP2P.footIn.p.accX*ACC_FACTOR-calimuf.ox)/1000;
       foot->accel[1]=calimuf.sy*(gP2P.footIn.p.accY*ACC_FACTOR-calimuf.oy)/1000;
       foot->accel[2]=calimuf.sz*(gP2P.footIn.p.accZ*ACC_FACTOR-calimuf.oz)/1000;
       // LYPR540 Y axis is true X axis
       foot->omega[0]=gP2P.footIn.p.gyroX*(calimuf.gx)-foot->gyrof[0];
       // LYPR540 -X axis is true Y axis
       foot->omega[1] = -gP2P.footIn.p.gyroX*(calimuf.gy)-foot->gyrof[1];
       // LYPR540 Z axis is true Z axis
       foot->omega[2] =  gP2P.footIn.p.gyroZ*(calimuf.gz)-foot->gyrof[2];
       foot->acclp[0] =  (99*foot->acclp[0] + foot->accel[0])/100;
       foot->acclp[1] =  (99*foot->acclp[1] + foot->accel[1])/100;
       foot->acclp[2] =  (99*foot->acclp[2] + foot->accel[2])/100;

       hpx =  (499*hpx + foot->accel[0])/500;
       hpy =  (499*hpy + foot->accel[1])/500;
       hpz =  (499*hpz + foot->accel[2])/500;

       foot->acchp[0] =  foot->accel[0] - hpx;
       foot->acchp[1] =  foot->accel[1] - hpy;
       foot->acchp[2] =  foot->accel[2] - hpz;

       foot->omflt[0]=(99*foot->omflt[0] + gP2P.footIn.p.gyroX*(calimuf.gx))/100;
       foot->omflt[1]=(99*foot->omflt[1] - gP2P.footIn.p.gyroX*(calimuf.gy))/100;
       foot->omflt[2]=(99*foot->omflt[2] + gP2P.footIn.p.gyroZ*(calimuf.gz))/100;
}
```

```
Axes update_axes(Imu imu, Imu imu_old, Axes axes_old, INT32 mG)
{
	static Axes   axes = {0};
	static Qtrn64 dq = {0};
	static INT32  dpsi = 0;
	static INT32  dtheta = 0;
	static INT32  dphi = 0;
	static Vec3   bfG = {0};
	static Vec3   G = {0};
	static Vec3   acc_ng = {0};

	// STEP 1: ORIENTATION ================================================
	// Approximation of Infinitesimal Euler Rotations
	// omega[mdeg/s]*2[ms] = dpsi [udeg/s]
	dpsi   = imu.omega[2]*SAMPLE_TIME/2;
	dtheta = imu.omega[1]*SAMPLE_TIME/2;
	dphi   = imu.omega[0]*SAMPLE_TIME/2;

	// Transform Approx. Inf. Euler Angles into an Approx Inf. Quaternion
	dq[0].f64 = _dc(dpsi)*_dc(dtheta)/TRIG_RADIX*_dc(dphi)/TRIG_RADIX +
	_ds(dpsi)*_ds(dtheta)/TRIG_RADIX*_ds(dphi)/TRIG_RADIX;
	dq[1].f64 = _dc(dpsi)*_dc(dtheta)/TRIG_RADIX*_ds(dphi)/TRIG_RADIX -
	_ds(dpsi)*_ds(dtheta)/TRIG_RADIX*_dc(dphi)/TRIG_RADIX;
	dq[2].f64 = _dc(dpsi)*_ds(dtheta)/TRIG_RADIX*_dc(dphi)/TRIG_RADIX +
	_ds(dpsi)*_dc(dtheta)/TRIG_RADIX*_ds(dphi)/TRIG_RADIX;
	dq[3].f64 = _ds(dpsi)*_dc(dtheta)/TRIG_RADIX*_dc(dphi)/TRIG_RADIX -
	_dc(dpsi)*_ds(dtheta)/TRIG_RADIX*_ds(dphi)/TRIG_RADIX;

	quatmultiply64(dq, axes_old.q_b2w, axes.q_b2w, TRIG_RADIX);
	quatnormalize64(axes.q_b2w, TRIG_RADIX);
	quatinverse64(axes.q_b2w, axes.q_w2b, TRIG_RADIX);
	quatnormalize64(axes.q_w2b, TRIG_RADIX);

	// STEP 2: COORDINATE TRANSFORM =======================================
	// Converting Body-Fixed Accelerations to the World Coordinate Frame
	// A = q_b2w*(a-gravity)*(q_b2w)^-1 => A = q_b2w*a*(q_b2w)^-1 - [0; 0; mG]

	G[0] = 0;
	G[1] = 0;
	G[2] = mG;
	quatrotate64(axes.q_w2b, axes.q_b2w, G, bfG, TRIG_RADIX);

	acc_ng[0] = imu.accel[0]+bfG[0];
	acc_ng[1] = imu.accel[1]+bfG[1];
	acc_ng[2] = imu.accel[2]-bfG[2];
	accx_no_grav = acc_ng[0];
	accy_no_grav = acc_ng[1];
	accz_no_grav = acc_ng[2];
	quatrotate64(axes.q_b2w, axes.q_w2b, acc_ng, axes.accel, TRIG_RADIX);

	// STEP 3: COMPUTE STATES =============================================
	// Velocity
	rawvelx = rawvelx_old + axes.accel[0]*SAMPLE_TIME;
	rawvely = rawvely_old + axes.accel[1]*SAMPLE_TIME;
	rawvelz = rawvelz_old + axes.accel[2]*SAMPLE_TIME;

	rawvelx_old = rawvelx;
	rawvely_old = rawvely;
```

```
        rawvelz_old = rawvelz;

        velxhp =  (999*velxhp + rawvelx)/1000;
        velyhp =  (999*velyhp + rawvely)/1000;
        velzhp =  (999*velzhp + rawvelz)/1000;

        axes.veloc[0] = rawvelx - velxhp;
        axes.veloc[1] = rawvely - velyhp;
        axes.veloc[2] = rawvelz - velzhp;

        // Position
        axes.posit[0] = axes_old.posit[0]+(axes.veloc[0]*SAMPLE_TIME)/IMU_RADIX/10;
        axes.posit[1] = axes_old.posit[1]+(axes.veloc[1]*SAMPLE_TIME)/IMU_RADIX/10;
        axes.posit[2] = axes_old.posit[2]+(axes.veloc[2]*SAMPLE_TIME)/IMU_RADIX/10;

        return axes;
}

INT32 reset_axes(Imu imu_foot, Axes *axe_foot, Axes *axe_foot_old)
{
        Qtrn64 q = {0};
        Qtrn64 qinv = {0};
        INT32  mG = 0;

        axe_foot_old->posit[0] = 0;
        axe_foot_old->posit[1] = 0;
        axe_foot_old->posit[2] = 0;
        axe_foot_old->veloc[0] = 0;
        axe_foot_old->veloc[1] = 0;
        axe_foot_old->veloc[2] = 0;
        axe_foot_old->accel[0] = 0;
        axe_foot_old->accel[1] = 0;
        axe_foot_old->accel[2] = 0;

        axe_foot->posit[0] = 0;
        axe_foot->posit[1] = 0;
        axe_foot->posit[2] = 0;
        axe_foot->veloc[0] = 0;
        axe_foot->veloc[1] = 0;
        axe_foot->veloc[2] = 0;
        axe_foot->accel[0] = 0;
        axe_foot->accel[1] = 0;
        axe_foot->accel[2] = 0;

        rawvelx_old = 0;
        rawvely_old = 0;
        rawvelz_old = 0;
        velxhp = 0;
        velyhp = 0;
        velzhp = 0;

        mG = imu_quat_estimator(imu_foot.acclp, q, qinv);

        axe_foot->q_b2w[0] = q[0];
        axe_foot->q_b2w[1] = q[1];
        axe_foot->q_b2w[2] = q[2];
        axe_foot->q_b2w[3] = q[3];
```

```
	axe_foot_old->q_b2w[0] = q[0];
	axe_foot_old->q_b2w[1] = q[1];
	axe_foot_old->q_b2w[2] = q[2];
	axe_foot_old->q_b2w[3] = q[3];

	gnd_slope=-(lkarcsin(2*(q[0].f64*q[2].f64/TRIG_RADIX-
q[3].f64*q[1].f64/TRIG_RADIX))+4500);

	axe_foot->q_w2b[0] = qinv[0];
	axe_foot->q_w2b[1] = qinv[1];
	axe_foot->q_w2b[2] = qinv[2];
	axe_foot->q_w2b[3] = qinv[3];

	axe_foot_old->q_w2b[0] = qinv[0];
	axe_foot_old->q_w2b[1]= qinv[1];
	axe_foot_old->q_w2b[2] = qinv[2];
	axe_foot_old->q_w2b[3] = qinv[3];

	return mG;
}

INT32 imu_quat_estimator(Vec3 acc, Qtrn64 q_out, Qtrn64 qinv_out)
{
	// Calculates the rotation quaternion from g to G (BFF to WCF)
	Vec3_64		g = {0};
	Vec3_64		ng = {0};
	Vec3_64		nG = {0, 0, TRIG_RADIX};
	INT64		mg = 0;
	INT64		qmod = 0;
	INT64		qnrm = 0;
	Qtrn64		q = {0};
	Qtrn64		qinv = {0};

	// This routine uses all 64-bit values to avoid overflow on ^2 functions
	g[0] = ((INT64) acc[0]);
	g[1] = ((INT64) acc[1]);
	g[2] = ((INT64) acc[2]);

	mg = fxsqrt_64(g[0]*g[0] + g[1]*g[1] + g[2]*g[2]);
	if (mg == 0)
	{
		mg = 1;
	}
	ng[0] = g[0]*TRIG_RADIX/mg;
	ng[1] = g[1]*TRIG_RADIX/mg;
	ng[2] = g[2]*TRIG_RADIX/mg;

	q[0].f64 =
1*TRIG_RADIX+(ng[0]*nG[0])/TRIG_RADIX+(ng[1]*nG[1])/TRIG_RADIX+(ng[2]*nG[2])/TRIG_
RADIX;
	// |  i   j   k  |   | ng1nG2-ng2nG1 |
	q[1].f64 = (ng[1]*nG[2])/TRIG_RADIX-(ng[2]*nG[1])/TRIG_RADIX;
	// | ng0 ng1 ng2 | = | ng2nG0-ng0nG2 |
	q[2].f64 = (ng[2]*nG[0])/TRIG_RADIX-(ng[0]*nG[2])/TRIG_RADIX;
	// | nG0 nG1 nG2 |   | ng0nG1-ng1nG0 |
	q[3].f64 = (ng[0]*nG[1])/TRIG_RADIX-(ng[1]*nG[0])/TRIG_RADIX;

	quatnormalize64(q, TRIG_RADIX);
```

```
	quatinverse64(q, qinv, TRIG_RADIX);
	quatnormalize64(qinv, TRIG_RADIX);

	q_out[0] = q[0];
	q_out[1] = q[1];
	q_out[2] = q[2];
	q_out[3] = q[3];

	qinv_out[0] = qinv[0];
	qinv_out[1] = qinv[1];
	qinv_out[2] = qinv[2];
	qinv_out[3] = qinv[3];

	return ((INT32) mg);
}

void imu_zero_gyros(Imu *foot)
{
	foot->gyrof[0] = foot->omflt[0];
	foot->gyrof[1] = foot->omflt[1];
	foot->gyrof[2] = foot->omflt[2];
}

INT32 fxsqrt_32(INT32 x)
{
	UINT32 root;
	UINT32 remHi;
	UINT32 remLo;
	UINT32 testDiv;
	UINT32 count;

	root = 0;
	remHi = 0;
	remLo = x;
	count = 15;

	do
	{
		remHi = (remHi<<2) | (remLo>>30); remLo <<= 2;
		root <<= 1;
		testDiv = (root << 1) + 1;
		if (remHi >= testDiv)
		{
			remHi -= testDiv;
			root++;
		}
	} while (count-- != 0);

	return(root);
}

INT64 fxsqrt_64(INT64 x)
{
	UINT64 root;
	UINT64 remHi;
	UINT64 remLo;
	UINT64 testDiv;
	UINT64 count;
```

```
        root = 0;
        remHi = 0;
        remLo = x;
        count = 31;

        do
        {
                remHi = (remHi<<2) | (remLo>>62); remLo <<= 2;
                root <<= 1;
                testDiv = (root << 1) + 1;
                if (remHi >= testDiv)
                {
                        remHi -= testDiv;
                        root++;
                }
        } while (count-- != 0);

        return(root);
}

void init_IMUlookup(void)
{
        INT32 ndx = 0;

        // The fastest angular velocity that can be measured by the gyros is 1200
        deg/s, or w = 1200000.
        // For a SAMPLE_TIME of 2ms, the largest change in angle would be
        1200000*2/1000 = 2400 mdeg.
        // Let's select a range for the small lookup tables of +/- 3 degrees.
        // Small Angle Sine Lookup Table
        for(ndx=(-IMU_NLKUP/2);ndx<=IMU_NLKUP/2;ndx++)
        {
                _dstable[ndx+IMU_NLKUP/2] = ((INT64) (sin(((double)
                ndx)*2/IMU_NLKUP*(IMU_3DEG))*TRIG_RADIX));
        }
        // Small Angle Cosine Lookup Table
        for(ndx=(-IMU_NLKUP/2);ndx<=IMU_NLKUP/2;ndx++)
        {
                _dctable[ndx+IMU_NLKUP/2] = ((INT64) (cos(((double)
                ndx)*2/IMU_NLKUP*(IMU_3DEG))*TRIG_RADIX));
        }
        for(ndx=(-IMU_NLKUP/2);ndx<=IMU_NLKUP/2;ndx++)
        {
                arcsintable[ndx+IMU_NLKUP/2] = ((INT32) (asin(((double)
                ndx)*2/IMU_NLKUP)*IMU_RADIX*180/IMU_PI));
        }
}

INT32 lkarcsin(INT64 oh)
{
        INT32 ndx = 0;
        INT32 ans = 0;

        ndx = oh*IMU_NLKUP/2/TRIG_RADIX + IMU_NLKUP/2;
        ans = arcsintable[ndx];

        return ans;
```

```
}

INT32 lkarctan2(INT64 y, INT64 x)
{
	INT64 h = 0;
	INT32 ans = 0;

	h = fxsqrt_64(x*x+y*y);
	if (h == 0)
	{
		h = 1;
	}

	if ((x == 0)&&(y > 0))
	{
		ans = 90000;
	}
	else if ((x == 0)&&(y < 0))
	{
		ans = -90000;
	}
	else
	{
		if (y >= 0)
		{
			if (x > 0)		// QUADRANT I (0 to 90 degrees)
			{
				ans = lkarcsin(y*TRIG_RADIX/h);
			}
			else if (x <= 0)	// QUADRANT II (90 to 180 degrees)
			{
				ans = (90000-lkarcsin(y*TRIG_RADIX/h))+90000;
			}
		}
		else if (y < 0)
		{
			if (x < 0)		// QUADRANT III (-90 to -180 degrees)
			{
				ans = -(90000+lkarcsin(y*TRIG_RADIX/h))-90000;
			}
			else if (x > 0)		// QUADRANT IV (0 to -90 degrees)
			{
				ans = lkarcsin(y*TRIG_RADIX/h);
			}
		}
	}

	return ans;
}

// Sin Lookup Function (small angle) In: micro-deg | Out: ratios of TRIG_RADIX
INT64 _ds(INT32 angle)
{
	INT64 ndx = 0;
	INT64 rmd = 0;
	INT64 radang = 0;
	INT64 ans = 0;
```

```
	// Avoid invalid angles
	if (angle > 3000000) angle = 3000000;
	else if (angle < -3000000) angle = -3000000;

	radang = ((INT64) angle)*314159265359LL/18000000000LL; // nano-radians
	// 3*pi/180 = 0.052359877559830
	ndx = (radang+52359877)*IMU_NLKUP/104719754;
	rmd = ((radang+52359877)*IMU_NLKUP)%104719754;

	if ((rmd == 0)||(ndx == IMU_NLKUP))
	{
		ans = _dstable[ndx];
	}
	else if (ndx > IMU_NLKUP/2)
	{
		ans = _dstable[ndx] + (_dstable[ndx+1]-_dstable[ndx])*rmd/104719754;
	}
	else
	{
		ans = _dstable[ndx] - (_dstable[ndx]-_dstable[ndx+1])*rmd/104719754;
	}

	return ans;
}

// Cos Lookup Function (small angle) In: micro-deg | Out: ratios of TRIG_RADIX
INT64 _dc(INT32 angle)
{
	INT64 ndx = 0;
	INT64 rmd = 0;
	INT64 radang = 0;
	INT64 ans = 0;

	// Avoid invalid angles
	if (angle > 3000000) angle = 3000000;
	else if (angle < -3000000) angle = -3000000;

	radang = ((INT64) angle)*314159265359LL/18000000000LL; // nano-radians
	// 3*pi/180 = 0.052359877559830
	ndx = (radang+52359877)*IMU_NLKUP/104719754;
	rmd = ((radang+52359877)*IMU_NLKUP)%104719754;

	if ((rmd == 0)||(ndx == IMU_NLKUP))
	{
		ans = _dctable[ndx];
	}
	else if (ndx > IMU_NLKUP/2)
	{
		ans = _dctable[ndx] + (_dctable[ndx+1]-_dctable[ndx])*rmd/104719754;
	}
	else
	{
		ans = _dctable[ndx] - (_dctable[ndx]-_dctable[ndx+1])*rmd/104719754;
	}

	return ans;
}
```

```
INT32 imu_highpass(INT32 *pz_1, INT32 *pz_2)
{
	INT32  z_0, z_1, z_2;

	z_2 = *pz_2;
	z_1 = *pz_1;
	z_0 = (991 - 1982*z_1/IMU_RADIX + 991*z_2/IMU_RADIX)/
	     (1000 - 1982*z_1/IMU_RADIX + 982*z_2/IMU_RADIX);
	*pz_2 = z_1;
	*pz_1 = z_0;
	return z_0;
}



void quatmultiply64(Qtrn64 q1, Qtrn64 q2, Qtrn64 qout, INT64 radix)
{
	qout[0].f64 = q1[0].f64*q2[0].f64/radix-q1[1].f64*q2[1].f64/radix-
q1[2].f64*q2[2].f64/radix-q1[3].f64*q2[3].f64/radix;
	qout[1].f64 =
q1[0].f64*q2[1].f64/radix+q1[1].f64*q2[0].f64/radix+q1[2].f64*q2[3].f64/radix-
q1[3].f64*q2[2].f64/radix;
	qout[2].f64 = q1[0].f64*q2[2].f64/radix-
q1[1].f64*q2[3].f64/radix+q1[2].f64*q2[0].f64/radix+q1[3].f64*q2[1].f64/radix;
	qout[3].f64 = q1[0].f64*q2[3].f64/radix+q1[1].f64*q2[2].f64/radix-
q1[2].f64*q2[1].f64/radix+q1[3].f64*q2[0].f64/radix;
}

void quatnormalize64(Qtrn64 q, INT64 radix)
{
	INT64 qmod;

	qmod = fxsqrt_64(q[0].f64*q[0].f64 + q[1].f64*q[1].f64 + q[2].f64*q[2].f64
+ q[3].f64*q[3].f64);
	if (qmod == 0)
	{
		qmod = 1;
	}
	q[0].f64 = (q[0].f64*radix)/qmod;
	q[1].f64 = (q[1].f64*radix)/qmod;
	q[2].f64 = (q[2].f64*radix)/qmod;
	q[3].f64 = (q[3].f64*radix)/qmod;
}

void quatinverse64(Qtrn64 q, Qtrn64 qout, INT64 radix)
{
	INT64 qnrm;

	qnrm = (q[0].f64*q[0].f64/radix
	  + q[1].f64*q[1].f64/radix
	   + q[2].f64*q[2].f64/radix
	   + q[3].f64*q[3].f64/radix);
	if (qnrm == 0)
	{
		qnrm = 1;
	}
	qout[0].f64 =  (q[0].f64*radix)/qnrm;
	qout[1].f64 = -(q[1].f64*radix)/qnrm;
```

```
	qout[2].f64 = -(q[2].f64*radix)/qnrm;
	qout[3].f64 = -(q[3].f64*radix)/qnrm;
}

void quatrotate64(Qtrn64 q, Qtrn64 qinv, Vec3 v, Vec3 n, INT64 radix)
{
	Qtrn64 qv;
	Qtrn64 qs;
	Qtrn64  qn;

	qv[0].f64 = 0;
	qv[1].f64 = v[0];
	qv[2].f64 = v[1];
	qv[3].f64 = v[2];
	quatmultiply64(qinv, qv, qs, radix);
	quatmultiply64(qs, q, qn, radix);
	n[0] = qn[1].f64;
	n[1] = qn[2].f64;
	n[2] = qn[3].f64;
}

INT32 dot32(Vec3 v1, Vec3 v2, INT32 radix)
{
	INT32 ans;

	ans = (v1[0]*v2[0])/radix
		+ (v1[1]*v2[1])/radix
		+ (v1[2]*v2[2])/radix;

	return ans;
}

INT32 rleg_settle(Vec3 acc, Vec3 alp)
{
	static Vec3		pwr = {0};
	static Vec3		flt = {0};
	static Vec3		dif = {0};

	INT32  ndx = 0;

	for (ndx=0;ndx<3;ndx++)
	{
		dif[ndx] = acc[ndx] - alp[ndx];
		if (dif[ndx]>4000)
		{
			dif[ndx] = 4000;
		}
		else if (dif[ndx]<-4000)
		{
			dif[ndx] = -4000;
		}
		pwr[ndx] = dif[ndx]*dif[ndx];
		flt[ndx] = (99*flt[ndx] + pwr[ndx])/100;
	}

	return (flt[0]+flt[1]+flt[2]);
}
```

```
/*****************************************************************************
* © 2010 Center for Intelligent Mechatronics            Vanderbilt University
*
* FileName:        rleg_inertia.h
* Processor:       PIC32MX575F512L
* Compiler:        MPLAB® C32 v1.10 or higher
* IDE:             MPLAB® IDE v8.50 or later
*
* ADDITIONAL NOTES:
*
*
*****************************************************************************/
#ifndef __RLEG_INERTIA__
#define __RLEG_INERTIA__

// User Include Files
#include "Pic32Types.h"
//#include "rleg_pic32.h"
#include "Pic32Structs.h"

// Shared Symbolic Constant Definitions
#define IMU_RADIX 1000L        // Radix point multiplier for IMU calculations
#define TRIG_RADIX 1000000000L// Radix point multiplier for trig fcns
#define BIG_TRIG_RADIX 10000L // Radix point multiplier for full-scale trig fcns

#define ACC_FACTOR   380    // Units are [(mm/s^2)/LSB/10]

// Shared Global Variable Declarations

// Shared Function Prototypes
extern void   sort_foot_signals(Imu *foot);
extern Imu          sort_shank_signals(void);
extern Axes   update_axes(Imu imu, Imu imu_old, Axes axes_old, INT32 mG);
extern INT32  reset_axes(Imu imu_foot, Axes *axe_foot, Axes *axe_foot_old);
extern void   imu_zero_gyros(Imu *foot);
extern void   init_IMUlookup(void);
// Sin Lookup (every angle) In: IMU_RADIX of deg | Out: ratios of TRIG_RADIX
extern INT64  _s(INT32 angle);
// Cos Lookup (every angle) In: IMU_RADIX of deg | Out: ratios of TRIG_RADIX
extern INT64  _c(INT32 angle);
// Sin Lookup (small angle) In: IMU_RADIX of deg | Out: ratios of TRIG_RADIX
extern INT64  _ds(INT32 angle);
// Cos Lookup (small angle) In: IMU_RADIX of deg | Out: ratios of TRIG_RADIX
extern INT64  _dc(INT32 angle);

extern INT32  imu_quat_estimator(Vec3 acc, Qtrn64 q, Qtrn64 qinv);
extern void   quatmultiply64(Qtrn64 q1, Qtrn64 q2, Qtrn64 qout, INT64 radix);
extern void   quatnormalize64(Qtrn64 q, INT64 radix);
extern void   quatinverse64(Qtrn64 q, Qtrn64 qout, INT64 radix);
extern void   quatrotate64(Qtrn64 q, Qtrn64 qinv, Vec3 v, Vec3 n, INT64 radix);
extern INT32  dot32(Vec3 v1, Vec3 v2, INT32 radix);
extern INT32  rleg_settle(Vec3 acc, Vec3 alp);

#endif
```

APPENDIX B.

Source Code for the Multi-Purpose Standing Controller

The following source code was written for the PIC32MX575F512L (Microchip Technology Inc.) microcontroller. It was written and tested in Microchip's integrated development environment (MPLAB IDE v8.50) and was compiled by the MPLAB C32 compiler, v1.10. The code provided here is for reference only and has significant outside dependencies, including references to global variables and other functions. It is not intended to be independently compiled.

```
Int32 rleg_stdmod_gen(void)
{
  Int32 im_new;

  im_new = im_old;
  if (am_old != 0)
  {
    im_new = 3;          // If new in standing start in GROUND SEARCHING
mode
  }
  else
  {
    switch (im_old)
    {
        case 0:   // Weight Bearing Standing
        im_new = rleg_stdmod_wb_state();
        break;
      case 1: // Non-Weight Bearing Standing
        im_new = rleg_stdmod_nwb_state();
        break;
      case 2: // Weight Bearing Damping
        im_new = rleg_stdmod_wbd_state();
        break;
      case 3: // Ground Searching
        im_new = rleg_stdmod_gs_state();
        break;
    }
  }
  return im_new;
}

// WEIGHT BEARING
Int32 rleg_stdmod_wb_state(void)
{
  static Int32 wb_timer = 0;
  static Int32 start_wb_timer = 0;
  static Int32 start_wb_timer_on = 0;

  Int32 im_new = 0;

  // Timer to count wbmode from another activity mode
  if (am_old != 0)
  {
    start_wb_timer_on = 1;
  }
  else if (start_wb_timer > (3000/SAMPLE_TIME) )
  {
    start_wb_timer = 0;
    start_wb_timer_on = 0;
  }

  if ( start_wb_timer_on == 1)
  {
    start_wb_timer =  start_wb_timer + 1;
  }
```

```
  if ((am_old == 0)&&(im_old == 0))
  {
    wb_timer = wb_timer + 1;
  }
  else
  {
    wb_timer = 0;
  }

  if
  (
    ( knepos_calib > 10000 )
    &&
    ( (toload_calib >= intmod.sta_wb_turnon_toload_thres)
      ||
      (heload_calib >= intmod.sta_wb_turnon_heload_thres) )
    &&
    ( wb_timer > (intmod.sta_im0_tmrlen/SAMPLE_TIME) )
    &&
    ( start_wb_timer_on == 0 )
    &&
    ( imu_tracking_valid == 1 )
    &&
    ( gnd_slope < 4000 )
    &&
    ( gnd_slope > -4000 )
  )
  {
    im_new = 2;
    wb_timer = 0;
  }
  else if
  (
    ( toload_calib <= intmod.sta_nwb_turnon_toload_thres )
    &&
    ( heload_calib <= intmod.sta_nwb_turnon_heload_thres )
    &&
    ( wb_timer > (intmod.sta_im0_tmrlen/SAMPLE_TIME) )
    &&
    (start_wb_timer_on == 0)
  )
  {
    im_new = 1;
    wb_timer = 0;
  }

  return im_new;
}

// NON-WEIGHT BEARING
Int32 rleg_stdmod_nwb_state(void)
{
  static Int32 nwb_timer = 0;

  Int32 im_new = 1;

  if ((am_old == 0)&&(im_old == 1))
```

```
  {
    nwb_timer = nwb_timer + 1;
  }
  else
  {
    nwb_timer = 0;
  }

  glob_timer_nwb = nwb_timer;

  if
  (
    ( nwb_timer > 500/SAMPLE_TIME )
    &&
    (
      ( toload_calib >= intmod.sta_wb_turnon_toload_thres )
      ||
      ( heload_calib >= intmod.sta_wb_turnon_heload_thres )
    )
  )
  {
    if ( knepos_calib > 60000 )
    {
      im_new = 2;
      nwb_timer = 0;
    }
    else
    {
      im_new = 3;
      nwb_timer = 0;
    }
  }

  return im_new;
}

// WEIGHT BEARING DAMPING
Int32  rleg_stdmod_wbd_state(void)
{
  static Int32 wbd_timer = 0;

  Int32 im_new = 2;

  if ((am_old == 0)&&(im_old == 2))
  {
    wbd_timer = wbd_timer + 1;
  }
  else
  {
    wbd_timer = 0;
  }

  if
  (
    ( knepos_calib < 0 )
    &&
    ( (toload_calib >= intmod.sta_wb_turnon_toload_thres)
```

```
      ||
      (heload_calib >= intmod.sta_wb_turnon_heload_thres) )
    &&
    ( wbd_timer > (intmod.sta_im2_tmrlen/SAMPLE_TIME) )
  )
  {
    im_new = 0;
    wbd_timer = 0;
  }
  else if
  (
    ( toload_calib <= intmod.sta_nwb_turnon_toload_thres )
    &&
    ( heload_calib <= intmod.sta_nwb_turnon_heload_thres )
    &&
    ( knepos_calib < 60000 )
    &&
    ( wbd_timer > (intmod.sta_im2_tmrlen/SAMPLE_TIME) )
  )
  {
    im_new = 1;
    wbd_timer = 0;
  }

  return im_new;
}

// GROUND SEARCHING
Int32 rleg_stdmod_gs_state(void)
{
  static Int32 gs_timer = 0;

  Int32 im_new = 3;

  if ((am_old == 0)&&(im_old == 3))
  {
    gs_timer = gs_timer + 1;
  }
  else
  {
    gs_timer = 0;
  }

  glob_timer_gs = gs_timer;

  if
  (
    ( toload_calib <= intmod.sta_nwb_turnon_toload_thres )
    &&
    ( heload_calib <= intmod.sta_nwb_turnon_heload_thres )
    &&
    ( gs_timer > (intmod.sta_im3_tmrlen/SAMPLE_TIME) )
  )
  {
    im_new = 1;
    gs_timer = 0;
  }
```

```
  else if
  (
    ( toload_calib >= intmod.sta_wb_turnon_toload_thres )
    &&
    ( heload_calib >= intmod.sta_wb_turnon_heload_thres )
    &&
    ( gs_timer > (intmod.sta_im3_tmrlen/SAMPLE_TIME) )
    &&
    ( imu_tracking_valid == 1)
  )
  {
    if
    (
      (knepos_calib > 20000)
      &&
      ( imu_tracking_valid == 1 )
      &&
      ( gnd_slope < 4000 )
      &&
      ( gnd_slope > -4000 )
    )
    {
      im_new = 2;
      gs_timer = 0;
    }
    else
    {
      im_new = 0;
      gs_timer = 0;
    }
  }

  return im_new;
}
```

APPENDIX C.

Matlab Class Definition for a Quaternion Data Type

The following Matlab class definition creates a native ‘quat’ data type in Matlab with overloaded function definitions for quaternion algebra. The purpose of this class is to simplify the syntax for a quaternion representation of rotation.

```
classdef quat
    %QUAT Construct a quaternion data type
    %   The quaternion data type allows overloaded methods for
    %  quaternion algebra. The constructor for a quat can be
    %  called in several ways. With no arguments, the constructor
    %  simply returns a quaternion.

  properties (SetAccess = private, GetAccess = private)
      q0;
      qv;
  end

  methods
    function obj = quat(varargin)
      switch nargin
        case 0
          obj.q0 = 1;
          obj.qv = [0, 0, 0];
        otherwise
          switch varargin{1}
            case {'a' 'axis-angle'}
              switch (nargin-1)
                case 2
                  obj.q0 = cos(varargin{2}/2);
                  obj.qv = sin(varargin{2}/2).*...
                           reshape(varargin{3}./...
                           norm(varargin{3}),3,1);
                otherwise
                  error('Bad input');
              end
            case {'R' 'matrix'}
              switch (nargin-1)
                case 1
                  R = varargin{2};
                  if (det(R)~=1)
                    error('Matrix is not orthogonal');
                  end
                  obj.q0 = sqrt(1+R(1,1)+R(2,2)+R(3,3))/2;
                  obj.qv(1) = (R(3,2)-R(2,3))/(4*obj.q0);
                  obj.qv(2) = (R(1,3)-R(3,1))/(4*obj.q0);
                  obj.qv(3) = (R(2,1)-R(1,2))/(4*obj.q0);
                otherwise
                  error('Bad input');
              end
            case {'q' 'quat'}
              switch (nargin-1)
                case 2
                  obj.q0 = varargin{2};
                  obj.qv =  reshape(varargin{3},3,1);
                otherwise
                  error('Bad input');
              end
            otherwise
              error('Bad input');
          end
      end
```

```
end    % Quaternion Constructor
function disp(q)
  fprintf('     %f, [%f; %f; %f]\n\n',q.q0,q.qv);
end                 % Quaternion Display
function q12 = plus(q1,q2)
  q0 = q1.q0+q2.q0;
  qv = q1.qv+q2.qv;
  q12 = quat('q',q0,qv);
end       % Quaternion Addition
function q12 = minus(q1,q2)
  q0 = q1.q0-q2.q0;
  qv = q1.qv-q2.qv;
  q12 = quat('q',q0,qv);
end      % Quaternion Subtraction
function q2 = uminus(q1)
  q0 = -q1.q0;
  qv = -q1.qv;
  q2 = quat('q',q0,qv);
end         % Quaternion Unary Minus
function q2 = uplus(q1)
  q0 = q1.q0;
  qv = q1.qv;
  q2 = quat('q',q0.qv);
end          % Quaternion Unary Plus
function pq = mtimes(p,q)
  q0 = q.q0*p.q0 - q.qv'*p.qv;
  qv = q.q0*p.qv + p.q0*q.qv + cross(p.qv,q.qv);
  pq = quat('q',q0,qv);
end        % Quaternion Multiplication
function dq = times(d,q)% Quaternion Scalar Multiplication
  if strcmp(class(d),'quat')
    if strcmp(class(q),'quat')
      q0 = d.q0*q.q0;
      qv = [d.qv(1)*q.qv(1);...
            d.qv(2)*q.qv(2);...
            d.qv(3)*q.qv(3)];
    else
      q0 = d.q0*q;
      qv = [d.qv(1)*q; d.qv(2)*q; d.qv(3)*q];
    end
  else
    q0 = q.q0*d;
    qv = [q.qv(1)*d; q.qv(2)*d; q.qv(3)*d];
  end
  dq = quat('q',q0,qv);
end
function q_bar = conj(q)
  q_bar = quat('q',q.q0,-q.qv);
end         % Quaternion Conjugate
function n = norm(q)
  qn = q*conj(q);
  n = qn.q0;
end             % Quaternion Norm
function e = exp(q)
  e0 = exp(q.q0)*cos(norm(q.qv));
  ev = exp(q.q0)*q.qv/norm(q.qv)*sin(norm(q.qv));
  e = quat('q',e0,ev);
```

```
    end              % Quaternion Exponential
    function l = log(q)
      l0 = log(norm(q));
      lv = q.qv/norm(q.qv)*acos(q.q0/norm(q));
      l = quat('q',l0,lv);
    end              % Quaternion Natural Logarithm
    function p = mpower(q,a)
      lq = log(q);
      if strcmp(class(a),'quat')
        lqa = lq*a;
      else
        lqa = quat('q',lq.q0*a,lq.qv.*a);
      end
      p = exp(lqa);
    end         % Quaternion Power
    function R = rmatrix(q)
      R(1,1) = 1-2*q.qv(2)^2-2*q.qv(3)^2;
      R(1,2) = 2*q.qv(1)*q.qv(2)-2*q.qv(3)*q.q0;
      R(1,3) = 2*q.qv(1)*q.qv(3)+2*q.qv(2)*q.q0;
      R(2,1) = 2*q.qv(1)*q.qv(2)+2*q.qv(3)*q.q0;
      R(2,2) = 1-2*q.qv(1)^2-2*q.qv(3)^2;
      R(2,3) = 2*q.qv(2)*q.qv(3)-2*q.qv(1)*q.q0;
      R(3,1) = 2*q.qv(1)*q.qv(3)-2*q.qv(2)*q.q0;
      R(3,2) = 2*q.qv(2)*q.qv(3)+2*q.qv(1)*q.q0;
      R(3,3) = 1-2*q.qv(1)^2-2*q.qv(2)^2;
    end          % Quaternion to Rotation Matrix
    function p = q2p(q)
      if (q.q0~=0)
        warning('Not a pure quaternion');
      end
      p = q.qv;
    end              % Quaternion to Point Conversion
    function d = dot(q1,q2)
      d = q1.q0*q2.q0+q1.qv(1)*q2.qv(1)+...
          q1.qv(2)*q2.qv(2)+q1.qv(3)*q2.qv(3);
    end          % Quaternion Dot Product
  end
end
```