12-2014

Omnidirectional Control of the Hexapod Robot TigerBug

Christopher B. Johnson

Omnidirectional Control of the Hexapod Robot TigerBug

By

Christopher B. Johnson

A Thesis Submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved by:

PROF_____
(Dr. Ferat Sahin, Thesis Advisor)

PROF_____
(Dr. Mathew, Thesis Committee Member)

PROF_____
(Dr. Monteiro, Thesis Committee Member)

PROF_____
(Dr. Sohail A. Dianat, Department Head)

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

December 2014

ABSTRACT

TigerBug is a six legged, hexapod robot built and designed by students in the Rochester Institute of Technology's (RIT) Multi Agent Bio-Robotics Laboratory (MABL). TigerBug is comprised of 18 servo motors, 3 degrees of freedom (DOF) per leg, supported by carbon fiber wrapped foam legs placed in a circular pattern around its hexagon shaped body. In order to control such a complex system, much research has been done in the field of kinematics. There exist two derivations of kinematic solutions, forward and inverse. The forward kinematic (FK) solution tends to be much simpler than its inverse kinematic (IK) counterpart. There has been many methods developed to quickly, and efficiently solve the IK in order to control the position and orientation of a robot. This thesis details the process of developing the IK solution and two gait algorithms for TigerBug. The IK solution was developed by first solving for the FK solution of TigerBug using Denavit-Hartenberg (DH) Parameters. After the FK solution was solved, differentials were applied to each equation in order to solve for the IK solution. Once the IK solution was tested, a fixed gait algorithm was developed in order to understand basic motion control of hexapod locomotion. Once the fixed gait was implemented successfully a rule-based free gait algorithm was developed. The rule-based free gait was accomplished using the rule set governed by restrictiveness to determine when leg state transitions were to occur, as described in the literature. Once implemented, the different combinations of gait parameters were tested for quickness of convergence and efficiency to determine the most optimal set of walking parameters for TigerBug.

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Ferat Sahin for his continued support and guidance through this very difficult process. I would also like to thank Ryan and Shitij for their willingness to help during my time in the lab. I also appreciate Construction Robotics for their patience and support while I finished this thesis and simultaneously helped them develop their SAM brick laying system. Finally I would like to thank my family for providing me with the opportunity to go to RIT, and supporting me during both my undergraduate and graduate careers.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I: Introduction

1.1 Applications and Motivations

As working conditions and environments become increasingly hazardous, it is extremely desirable to remove the human from these potentially dangerous situations and replace them with a robot designed specifically to perform a given task. Some of these potentially dangerous environments include natural disaster sights, battlefields, and mining operations. In general these situations have an above average potential for injury or even death for a human working in the area. Being that we cannot replace human being's life, it makes sense to replace them with a robot that can do the same job as well, if not better, in these environments.

In order for a robot to replace a human, it must meet certain requirements. The robot must be stable statically, and often times dynamically stable as well as mobile, and manipulative [9]. Most of the current solutions for introducing robots that perform hazardous tasks still require people to setup the infrastructure needed for the robot to perform its task and therefore people are still entering these potentially dangerous environments. Also, in most cases, a 6 DOF robotic arm is used to perform the desired task. Standard robotic arms are often physically large, and take quite a bit of power to operate. Therefore a system implementing one of these robots will not be as maneuverable as a smaller system, and will probably need special infrastructure to support itself.

Currently the best solution to remove humans from these dangerous working environments is to develop a custom legged robot to perform the desired task. A legged robot is, in general, a better candidate than a wheeled robot due to a higher degree of mobility over uneven and rough terrain, as well as a more adaptable body structure. A legged robot will often be able to modify its body position and orientation when encountering an unforeseen obstacle in its working environment, while a wheeled robot has a fixed body structure and often has no control over its body orientation.

1.2 Legged Robots

Legged robots come in many different packages consisting of one leg to as many as ten legs. Having few legs provides, in general, a simpler control scheme, but sacrifices stability for such. More legs offer a greater degree of stability, but gait generation and control become more difficult. In general, the research field is often concerned with one of three styles of legged robots; Hexapods, quadrupeds, and bipeds.

1.2.1 Hexapods

Of the many different forms legged robots come in, hexapods are arguably the most diverse. The ability to have anywhere from three to six legs in contact with the ground at any time provides this type of robot with the ability to trade stability for speed through the modification of gait styles and parameters. The sheer number of legs also allows for the development of multiple statically stable gaits.

There are many different ways in which to control a robot's gait. In 2004, Fielding and Dunlop produced a study on efficient gaits using restrictedness. The concept of restrictedness is a way of defining the workspace a certain leg can operate in based on

where its neighboring legs reside [1]. There are two main modes a leg of a robot can be in; these are either swing mode or stance mode. When in stance mode, the leg is on the ground, and providing a force in the desired direction of motion for the robot. While in swing mode, the leg is off of the ground and is in the process of resetting itself in order to be placed back into stance mode.

Therefore the concept of restrictedness can be used to say that if a leg is in stance mode, its neighbors are on the ground, and its restrictiveness is increasing, change to swing mode. Otherwise, if the leg is in swing mode and the leg is on the ground, switch to stance mode [1]. The research into restrictedness from this group showed that, while on a flat surface, the robot's gait tended to converge to a tripod gait, and while on uneven surfaces converged to a wave gait.

Within the field of hexapod research, two main leg configurations dominate, one being the circular configuration, and the other being a stick insect configuration. The circular leg configuration offers a more energy efficient approach to locomotion, which in the field can be very important when the robot is working in a remote location. The higher energy efficiency comes from the lower torque required to move the robot's body due to the placement of the legs around the body. However, this configuration is often considered less stable. The insect leg configuration, on the other hand, provides a far more stable platform for locomotion, but tends to be less energy efficient [2]. Billah, et. al produced a study on hexapod locomotion and were able to build their own hexapod with the circular leg configuration that was able to be outfitted with two different gait styles. The first gait that was implemented was a tripod gait for walking over even

terrain, while the second gait that was implemented was a wave gait that was used for walking over uneven terrain. Both of these gaits are considered dynamically and statically stable thus making them good candidates for locomotion over simple and difficult terrain [2].

In any legged robot system, servo motors generally provide the means for leg control. In order to effectively move the legs comprising a robot, kinematic equations need to be generated for the specific robot. Forward kinematics take in servo, joint, angles and calculate the current foot position, while inverse kinematic solutions generally take in position and orientation data, and return the joint angles needed to achieve that position. Inverse kinematic solutions are often more difficult to obtain, and can be unstable as the leg approaches a singularity.

In general there are two ways to obtain an inverse kinematics solution. The first approach is to use trigonometric properties in order to develop systems of equations which can then be solved for individual joint angles [3]. While this approach may seem trivial, as the complexity of the leg design increases, the computational power to solve the system of equations becomes more costly.

Duan et. al performed a locomotion analysis on a hexapod with the insect style leg configuration. Calculating both forward and inverse kinematic solutions for their specific robot using trigonometric properties, as well as performing workspace analysis for each of the robot's legs, they were able to generate a tripod, and turn gait in order to allow the robot to navigate its environment [3]. Based on the workspace analysis of each leg, a

maximum turn angle for the robot was determined, which could then be iterated multiple times in order to allow the robot to avoid objects in front of it.

The second approach to solving the inverse kinematics problem uses rotation and position matrices to solve for the forward kinematics solution [4]. From here, a Jacobian matrix can be developed and the joint angles can be solved for by using the inverse of this Jacobian. The major problem with this approach is that it requires the use of small angle approximation to make the inverse of the Jacobian much simpler. Over time, this approximation develops compounded errors, and can become unstable. Another issue arises when certain leg positons approach a singularity, that is when the Jacobian becomes not full rank.

1.2.2 Quadrupeds

Quadrupeds are arguably the second most studied robot in the legged robot family. As their name implies, these robots are made up of four legs spaced either on the corners of a rectangular body, or forty-five degrees apart on a circular body. Quadrupeds are considered slower, and less flexible than their larger hexapod cousins. This is often due to a fewer number of stable gaits that can be developed for this particular system. Often a quadruped will either use a wave or ripple gait if stability is more important than speed, or a two legged or running gait, if speed is more important than stability. The latter gait style is not statically stable due to only two legs being on the ground at any given time. This means that the body of the robot never resides within the triangle of stability, and therefore is not statically stable.

Much like the hexapod, both forward and inverse kinematic solutions need to be developed in order to control robot's leg movements. In general two strategies are applied to a quadruped's inverse kinematics solution; the first strategy computes the three leg angles for one of the four legs while the position and orientation of the body is held constant. In this approach, a Jacobian matrix is developed from the forward kinematic solutions, and the inverse of the Jacobian is then computed to obtain the three joint angles [5]. The second approach computes a Jacobian based on the position and orientation of each leg in the system. This approach allows for the position and orientation of each leg to be known, but the body's position and orientation are not controlled, and therefore this solution can yield unexpected and impossible body positions [5].

Shkolnik and Tedrake (2007) implemented both of these strategies on a quadruped known as little dog. This robot uses a leg configuration and construction similar to canines, and is very apt at traversing difficult terrain. The individual leg inverse kinematic solution as well as full body inverse kinematic solution was developed for the robot and tested. Each control structure was tested to the edge of kinematic feasibility [6]. It was determined that a hybrid controller allowed the robot to move the quickest and most stable through its environment.

1.2.3 Bipeds

Bipeds have been gaining traction in the research field steadily over the past five to ten years. Building a successful bipedal robot is not a trivial task due to the support structure needed for a two legged system. Once major issue with bipedal robots is the vast amount of torque it takes to move each limb of the robot, specifically the motors that reside at the

hip and shoulder joints. This makes these robots very inefficient in terms of power consumption. Another major flaw in a bipedal system comes from only ever having one foot on the ground at a time during locomotion. Therefore controlling the balance of these robots is no simple task. Controlling position and orientation of the robot also becomes quite difficult due to the sheer number of degrees of freedom, DOF, of each limb a bipedal walking system can have.

A research group attempting to build a hip joint for a humanoid robot confirmed that large motors are needed in order to control the major joints of a biped robot. They discovered in order to allow the hip joint to either roll or yaw, a 20 Watt DC motor was needed, and to perform pitch, a 90 Watt DC motor was needed [7]. To control this joint, the team developed an inverse kinematics model using a trigonometric approach. Their goal was to develop a system of equations that yielded a single solution every time the inverse kinematics was calculated.

In all legged robotics implementing an inverse kinematics solution involving a least squares solution to avoid singularities, error can accumulate as run time elapses. If run for long enough without resetting the legs, a significant misstep can occur, and the robot can either cease to move correctly, or can physically break itself. To minimize the error that occurs from using a basic least squares method, a weighted least squares method can be implemented. Effectively, this weights the move based on relative importance to other tasks [8]. Therefore, if a certain move is going to cause a large error to occur, either a different move with less probability of error can be chosen, or this error can be accounted for in the following moves.

1.3 Gait Schemes

A gait is the way a robot, or any mobile legged animal, changes the speed and phase of its limbs in order to produce motion. Different gaits will have a variety of different limbs in contact with the ground at any given moment, as well as varying amount of time these limbs are in contact with the ground. A gait is usually selected based on the desire for speed versus the desire for stability. The fewer legs on the ground, in stance mode, the faster the robot will move, but, in general, the less stable it will be. Having more legs in stance mode will give the robot more stability, but will make it slower due to the increased number of phase delays which need to be used to move the robot the same distance as a gait with fewer legs in stance mode at one time.

1.3.1 Fixed Gait

The fixed gait is often implemented on many robotic platforms for the simplicity it offers as well as the robustness and controllability the programmer has over the parameters that drive the gait. What this style lacks, however, is the ability to adapt to a changing environment on the fly. In a fixed gait controller, the leg fall position is determined beforehand, and the leg will always begin its transition from swing to stance mode when the leg reaches this position. Therefore the pattern in which the legs move will always be the same, regardless of any external forces.

Many fixed gaits are modeled off of gaits that are observed in insect locomotion. Researchers at the Beijing University in China (2009) designed and built a stick insect inspired hexapod to study a biologically inspired fixed tripod gait [3]. First the researchers determined the workspace of each leg, and realized that the workspaces of

each middle leg overlapped with the leg directly to the front, and the rear of it. To address this issue, the mechanical structure of the robot's body was modified to be more of an elliptical structure than a rectangle [3]. This significantly reduces the overlap of the workspaces, and allows for said overlap to be effectively ignored. Once the researchers developed their tripod gait, a fixed turn gait was also developed to allow the robot to avoid obstacles in its environment. The turn gait was developed based on the stability polygon. One set of legs would rotate a fixed amount in stance mode, until the stability polygon was about to be broken, then the legs would switch modes, and the other set of legs would rotate until the stability polygon was about to be violated.

A researcher at the Worcester Polytechnic Institute (2013) also used a fixed gait to develop a walking motion in a hexapod robot. In their research, both a faster tripod gait, and a more stable wave gait were implemented on a hexapod with a circular leg configuration. Legs were moved based on a trajectory planning algorithm which used the IK solution to determine where the each foot needed to go during a certain phase in the gait, and the velocity that each joint needed to move at in order to get the its foot there in the required amount of time [9]. A fixed gait calculated through the use of leg trajectory planning allows the robot to also switch gaits in the middle of walking without any delay or need for resetting the legs to a known position.

1.3.2 Rule-Based Free Gait

Free gaits controllers are referred to as aperiodic gaits due to the legs not acting in a predetermined periodic motion. Instead, legs change modes based on a set of

predetermined rules provided by the programmer. Properly applied, a free gait can provide a robot with a very effective terrain-adaptive locomotion scheme [11].

Fielding (2002) developed a rule based free gait based on the restrictedness of each leg in the walking system Hamlet. The driving factor behind this controller was that as a leg became more restricted, less able to move, the larger desire it had to switch modes and become less restricted [10]. Fielding identified leg workspace, joint angles, and Cartesian distances between feet, and ankles of adjacent legs to be driving factors in how "restricted" a leg had become. Using the concepts of restrictedness, Fielding was able to develop an omnidirectional walking scheme for Hamlet and was able to change vector directions at will without having to continuously pause, and reset the legs to known positions.

Estremera and Gonzalez de Santos (2005) developed an omnidirectional rule-based free gait for a quadruped robot based on the marginal stability of the robot. Three stability margins were considered in this study, the absolute stability margin, the fore longitudinal absolute stability margin, and the back longitudinal stability margin. Based on the current state of the robot's legs, foot holds were selected and compared with the stability margins. The best foot hold was then selected in order to keep the foot and body as far from one of these unstable positions as possible.

1.3.3 Optimized Free Gait

Optimized free gaits are the most efficient in terms of robot movement, but tend to consume a lot of computational power. This type of free gait controller looks for the most optimal next step for the robot's given state. Not only does this controller need to know

the robot's current state, but it also needs to know terrain information in order to plan ahead for the next move. This amount of forethought tends to require a lot of heuristics due to the quickly increasing number of possible steps that can be taken as time increases.

Pal and Jayarajan (1990) attempted to implement an optimized rule-based free gait on a quadruped walking machine. The rules used in this algorithm were mostly based on where the leg was in its workspace and if the leg were to continue, would the leg move out of its workspace. If yes, switch the mode of the leg, if not, keep moving in the same direction. The researchers also addressed one of the major issues with free gait controllers, their lack of foresight. Without looking ahead at the next possible motion, it is possible that the robot will put itself into an unstable state [11]. To combat this, Pal and Jayarajan implemented a heuristic graph search algorithm, A^*, in order to look at all of the possible next moves and determine their worthiness in terms of stability, and goal completion.

Buehler et al. implemented an optimized free gait on the Rhex platform at the University of Michigan. Due to the complexity and tediousness of hand tuning gait parameters, the researchers decided to have the robot tune its own gait based on a certain set of performance criterion the robot was trying to achieve. Using a function optimizer known as the Nelder-Mead algorithm, along with the aid of a cost function, the researchers were able to demonstrate the ability to shape gaits of a legged robot [13]. Experimental validation of the gait generation, and optimization proved a vast improvement in velocity of the Rhex system, as well as the benefits of moving in the leg coordination space of the robot, especially over difficult terrain [13].

1.4 Objectives

Efficient navigation over difficult terrain is a very valuable feature in legged robots. The ability to maneuver over a variety of terrain types without human interaction is a driving force behind many research efforts to remove humans from potentially hazardous environments. It is also imperative that the robot have omnidirectional capability. Being able to move in any direction at any time allows the robot to better adapt to the current terrain as well as change its heading based on environmental input. Therefore there are two features imperative for all legged locomotive systems:

1. The robot must be able to use efficient forward and backward walking gaits

2. The robot must have omnidirectional maneuverability

The objective of this study is to provide TigerBug with the aforementioned requirements in the form of a gait controller. First the forward and inverse kinematic solutions for TigerBug will be solved for and the solutions' accuracy will be tested. An omnidirectional fixed gait will then be implemented on TigerBug that will utilize the IK solution previously developed. Finally, an omnidirectional rule-based free gait will be developed and implemented on TigerBug using restrictiveness to decide when leg state transitions should occur.

Being that TigerBug was not designed to be fitted with force sensors on any part of its body, the robot will have no real time environmental feedback and all work will be tested over a flat, smooth surface. The restrictiveness algorithm developed in this research will be able to be extended to another robot platform that, if built at a later time, and will be

able to use environmental feedback to adjust its gait in real time with limited code modifications.

Although simulations are useful, all solutions will need to be implemented and tested on a real system in order to prove successful. Being that a real world environment is extremely demanding on gait controllers, the robot should be able to continue a stable walking motion in the imperfect environment in which it was meant to operate without any human interaction.

1.5 Outline

This thesis presents four chapters in addition to the introduction and conclusion. Chapter 2 describes the physical construction of the robot used in this research, TigerBug. Chapter 3 will then describe the kinematic solutions that can be developed based on the physical configuration of TigerBug discussed in Chapter 2.

Chapter 4 discusses the concept of restrictedness and how it can be applied to the unique model that is TigerBug. The parameters chosen to calculate restrictedness are explained and shown to be useful in determining when a leg has become over restrained and a mode transition needs to occur. Chapter 5 will then curtail off of Chapter 4 and show how restrictedness can be used to obtain an omnidirectional, smooth, and efficient gait. The rule-based free gait using restrictedness will then be compared to the omnidirectional fixed tripod gait, which was also developed for TigerBug, in terms of smoothness and efficiency.

Chapter 6 will provide conclusions for the work presented from the research. Suggestions and improvements for future work will also be presented in Chapter 6.

Chapter 2: Equipment and Hardware

2.1Introduction

Although large strides have been made in simulated environments, in no way do they fully encapsulate all of the challenges that plaque robots operating in the real world. Therefore all experiments performed during this research project were done physically on TigerBug to ensure robustness of solutions. This section outlines the requirements placed on the construction of TigerBug, as well as the mechanical and electrical hardware used on TigerBug.

2.2 Requirements

TigerBug was designed by students in RIT's Advance Robotics course from the ground up to provide a better research platform than other older hexapods that were currently in the lab. TigerBug was required to have a larger degree of maneuverability, increased strength, and improved runtime when compared to the other available hexapods. TigerBug was also designed to be fully autonomous, and tether free.

2.2.1 Mechanical Hardware

TigerBug's body pieces were first designed using AutoCAD, and then each body piece was cut out using a precise laser cutter to ensure that any pieces that had a duplicate on the system were exactly the same. All supporting body and leg pieces were constructed from carbon fiber wrapped structural foam. This material provided the robot with an excellent strength to weight ratio, allowing the robot to be lighter than if the body was

constructed out of aluminum or polycarbonate, while still providing the structural support necessary for the larger motors implemented on the system.

The main body structure of TigerBug consists of two circles with supporting structures for each leg spaced out around the body every 60 degrees. The top and bottom body pieces are separated by the hip servos of the six legs, held in place by a servo horn and servo mounting bracket. The spacing between the two structural body pieces is 60.5 millimeters and the outer radius of the body piece is 101.6 millimeters.

Figure 1: Structural body piece for TigerBug

Each of the six legs is comprised of 3 RoBoard RS-1270 servo motors. Each motor is supported by either a servo motor bracket or custom carbon fiber structural leg pieces as shown in Figure 2. The length between the center of the hip servo and the knee servo, referred to as the femur, measures 42 millimeters. Connecting the knee and ankle servos were two custom carbon fiber pieces referred to as the shin, and measured 101 millimeters between servo centers. Each pair of shin pieces had internal angles of 135 degrees, and were used to provide less play between the knee and ankle joints, and

provided added stability. The final piece of the leg known as the foot was a piece of carbon fiber measuring 106 millimeters from ankle servo center to the tip of the foot.

Figure 2: Structural leg pieces for TigerBug

Figure 3: Completed TigerBug Leg

2.2.2 Electrical Hardware

Movement of the sex legs composing TigerBug was controlled by 3 servos, providing 3 unique degrees of freedom. This adds up to 18 servos in total on TigerBug, all of which were RoBoard RS-1270's. Each servo weighed in at 70 grams, and filled a volume of 2 cubic inches. For its relatively small size, these servos were capable of producing 486 oz-in of torque, and a maximum angular speed of 545.5 degrees per second [15]. It is important to note that these specifications can only be reached when the servos are operating under 7.4 volts. The RS-1270 provides an extremely powerful actuator inside of a very small foot print, providing TigerBug with the necessary power to adapt to a challenging environment, as well as allowing legs to have a compact and highly maneuverable design. Figure 4 shows the linearity between PWM entered into the servo,

and the angular output position of the servo, providing for simple conversions inside of the code. Table 1 shows the minimum, center, and maximum angles used for each joint servo.

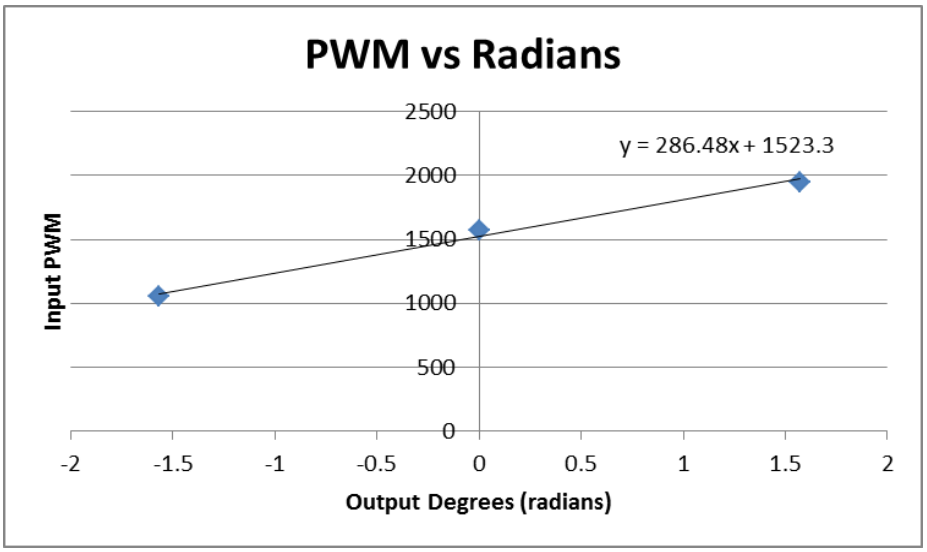

Figure 4: Input PWM vs. output angle of RS-1270

Table 1: Servo Angle Limits

Joint	Min. Angle (rad)	Center Angle (rad)	Max. Angle (rad)
Hip	-1.05	0	1.05
Knee	-0.785	0	0.785
Ankle	-0.785	0	0.785

In order to harness the power of the 18 servos, an SSC-32 servo controller was implemented on the underside TigerBug's upper body piece. This servo controller was ideal for the project due to a ready to use RS-232 serial port provided right on the board, and its 1 microsecond positioning resolution [16]. The RS-232 port provides for extremely simple integration and fast communication with either a computer of a microcontroller. The controller also boasts a sequencer for gait generation, although this

feature was not used for this research, and gives the programmer control over length of time for a move, speed of the motor, and position of the motor, with very simple commands.

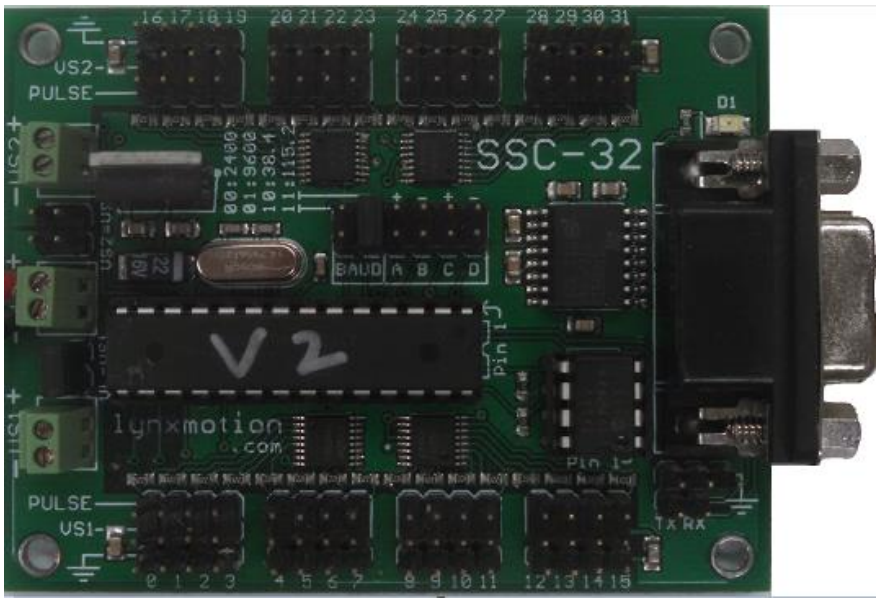

Figure 5: SSC-32 Servo Controller by Lynxmotion

In order to power the entire system, two lithium polymer (LiPo) battery packs were placed on top of TigerBug's body piece. These batteries were capable of producing 7.4 volts with an operating power of 4000 milliamp hours. Note that the aforementioned servos obtained peak performance at 7.4 volts, therefore one battery was used to power half of the servos, and the other was used to the other half of the servos, and the SSC-32. Due to the superior energy density of LiPo batteries, they were the obvious choice over nickel-cadmium or nickel metal hydride batteries. LiPo batteries are also, in general, lighter than other battery styles of similar output voltages. Being that these batteries are light, provide sufficient power, and do not require regular use to keep up battery life, they provide an excellent power source for TigerBug in its demanding environment.

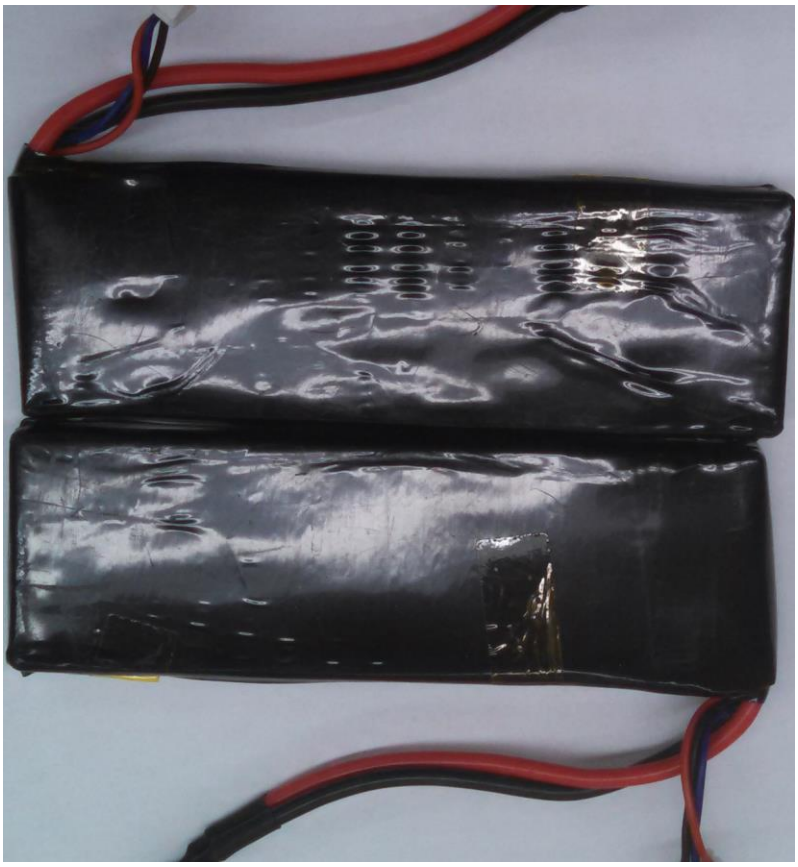

Figure 6: Powerizer 7.4 volt, 4000 mAH, LiPo Batteries

Figure 7: Completed TigerBug Build

2.3 Recommendations for Improvement

Although TigerBug was quite well built for a first revision, there still exists some room for improvement. First, shortening the length of all servo cables would largely improve the ease of access to components that lie between the two body pieces like the SSC-32, and would also allow for easier maintenance of these parts. Multiple times during the course of this research, nuts would come loose from the legs and body support structures of TigerBug. This was extremely annoying as many of these areas were very difficult to access for service. Therefore a design utilizing better maintenance spots would be ideal. Also adding an adhesive, such as blue Loctite, to better hold the connecting hardware in place while under stress, while still allowing the robot to be taken apart when desired,

would greatly improve the structural integrity of the robot. Another improvement would be to add force sensors to each foot in order to allow for a more reliable, albeit more complex, control system. This will also allow for extended future research potential as a retrofit is not feasible on the current design. The final recommendation for a future hardware revision would be to use servos with less gear play as, at times, this caused control issues with the legs.

Chapter 3: Kinematic Solutions

3.1 Introduction

Even before considering how a robot will walk, it is imperative to understand how the body and legs of the robot are to be controlled. There are multiple ways to control body and leg movements, but the most common solutions are either through a forward/inverse kinematic solution or through a trigonometric solution.

A trigonometric solution is found through examination of leg joints and links and applying various trigonometric properties in order to associate joint angle movements to differential Cartesian movements of the foot in the foot frame. For a 3 DOF leg, like the ones that exist on TigerBug, 3 solutions will be found that will associate the movement of one of the joints with its effect on the x, y, and z position of the foot. These solutions can then be inverted in order to take in an x, y, and z desired foot location, and produce the 3 necessary joint angles to place the foot at that location from its current location.

A forward/inverse kinematic solution follows in the same vain as a trigonometric solution in that a solution will be derived to first take in joint angles and determine the foot location in Cartesian space, the forward kinematic solution, and then the solutions will be inverted in order to take in the desired position in Cartesian space and produce the necessary joint angles to achieve the position, the inverse kinematic solution.

3.2 The Forward and Inverse Kinematic Solutions

In this research, both body and leg movements were controlled through a forward and inverse kinematic solution. This solution was chosen over the trigonometric solution for its adaptability to many different leg configurations as well as the practicality this solution brings to real world applications.

3.2.1 Devinant-Hartenberg Model

Before any kinematic solutions were developed, the joint frames of a leg were laid out according to the Denavit-Hartenberg (DH) model. Denivant and Hartenberg developed a simple, robust way of representing link and joint configuration of any robot, regardless of complexity. Applying this model also provides the added luxury of being able to directly obtain other desired calculations including the Jacobian matrix of the robot that was used in this research.

The DH model states that all joints are represented by the motion either along or about the z-axis. Therefore if a joint is revolute, the z-axis will point in the direction of positive rotary motion of the joint determined by the right-hand rule. If the joint is prismatic, however, the z-axis will lie in the direction that is parallel to the linear motion of the joint. All of the TigerBug's joints are revolute.

Another defining parameter of this model is that a common normal must be placed between any two z-axes. Since joints may not be parallel or intersecting in practice, they will be represented as skew lines. The common normal is therefore the single, shortest, mutually perpendicular line that can be drawn between the two skew lines [14]. The x-axis of the joint's local reference frame is always placed in the direction along the

common normal. An example of this can be observed between joints two and three in Figure 8. If two z-axes happen to be perfectly parallel to each other, there exist an infinite number of common normals that can be achieved [14]. For simplicity, the common normal that is in line with the previous joint's common normal is chosen.

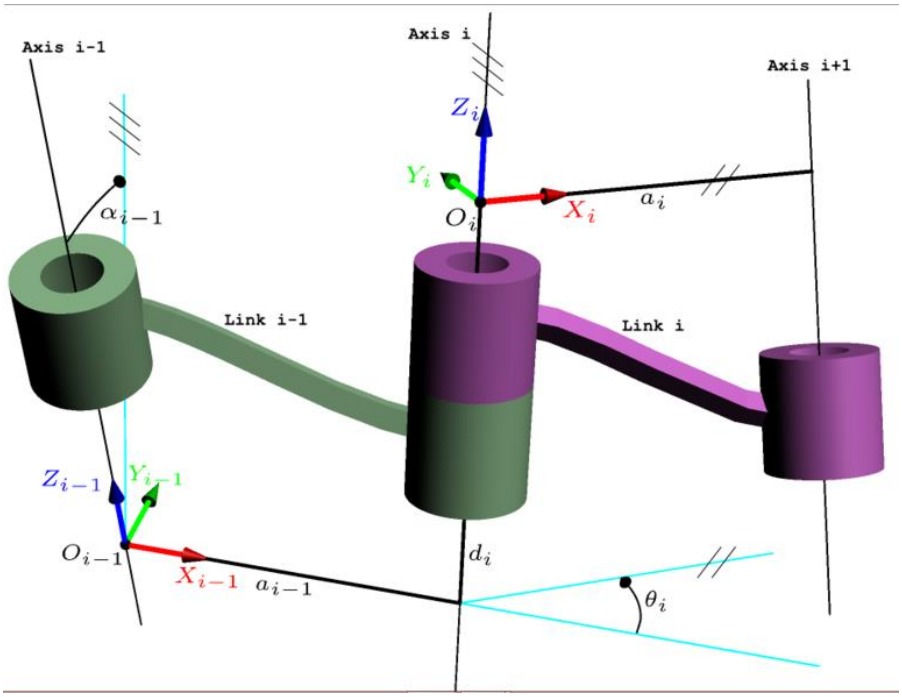

Figure 8: Common diagram depicting DH model and parameter uses, [17]

In the case of two z-axes that intersect at a given point, a common normal does not exist. Technically the common normal is there, but has zero length, so therefore it is considered to not exist. In such an instance, the x-axis is placed along a line, perpendicular to the plane formed by the two axes [14]. This solution also simplifies the model.

There is an inherent problem with this model however. Since the frame transformations between joints are processed using only motions in the x and z-axes, any motion around or in the direction of the y-axis cannot be interpreted by the model. This is not to say that the robot cannot move in the y Cartesian direction, but to say that the sequential axes

cannot be offset or rotated in or about the y direction of motion. Many researchers have attempted to solve this flaw, but none have proved fully successful as of yet [14].

3.2.2 Devinant-Hartenberg Parameters

Once the local joint frames are established using the DH model, a set of DH parameters can be used to describe relative motion between the joint frames. These parameters come in the form of θ, d, a, and α. The parameter θ represents a rotation about the z-axis following the right-hand rule. The d parameter represents the linear distance in the direction of the z-axis between two common normals. The parameter a, signifies the length of the common normal, i.e. the distance between two parallel or skew z-axes. Finally the α parameter represents the rotation between two sequential z-axes. This rotation occurs around the common normal or, as previously described, the x-axis.

Niku (2001) outlines the necessary steps needed to transform one joint reference frame into another. Niku starts by stating that the current reference frame is represented by n, and the n is attempting to be moved onto frame n+1 using the following four standard motions:

1. Rotate about the z_n-axis an angle of θ_{n+1} in order to make x_n and x_{n+1} parallel to one another.

2. Translate along z_n-axis a distance of d_{n+1}, making x_n and x_{n+1} colinear.

3. Translate along the x_n axis a distance of a_{n+1}, aligning the z_n and z_{n+1} axes. After performing this step, the origin of the two reference frames will be at the same location.

4. Rotate the z_n-axis about the x_{n+1}-axis an angle of α_{n+1}, thus aligning the z_n and z_{n+1} axes. Now reference frame n has been completely transformed into reference frame z_{n+1}.

The same set of transformations can then be applied to each successive sequential pairs of joints. In order to holistically capture the set of motions moving one reference frame onto another, an *A* matrix can be created as shown in the equations below. These equations mathematically represent the steps outlined above.

$$^{n}T_{n+1} = A_{n+1} = Rot(z,\theta_{n+1}) * Trans(0,0,d_{n+1}) * Trans(a_{n+1},0,0) * Rot(x,\alpha_{n+1}) \qquad [1]$$

$$A_{n+1} = \begin{bmatrix} \cos(\theta_{n+1}) & -\sin(\theta_{n+1}) * \cos(\alpha_{n+1}) & \sin(\theta_{n+1}) * \sin(\alpha_{n+1}) & a_{n+1} * \cos(\theta_{n+1}) \\ \sin(\theta_{n+1}) & \cos(\theta_{n+1}) * \cos(\alpha_{n+1}) & -\cos(\theta_{n+1}) * \sin(\alpha_{n+1}) & a_{n+1} * \sin(\theta_{n+1}) \\ 0 & \sin(\alpha_{n+1}) & \cos(\alpha_{n+1}) & d_{n+1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2]$$

3.2.3 Solving the Forward Kinematic Problems

Utilizing the DH model and parameters outlined above, the forward kinematic solution was produced for one TigerBug's legs. Since all six of the legs are replicas of each other, the derived solution can be applied to all legs identically. The first step in the solution is to assign the local reference frames for each of the three joints that make up the leg. Following the practices of the DH model, the z-axis of each joint was placed such that the positive direction of motion was around this axis. This also allowed the common normals to be placed along the links comprising the legs. Figure 9 shows the placement of each of the local reference frames for the hip, knee, and ankle joints of TigerBug.

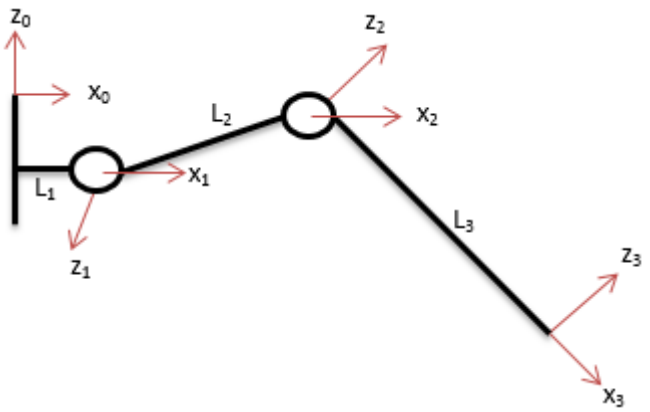

Figure 9: DH model of TigerBug Leg. All joints are revolute

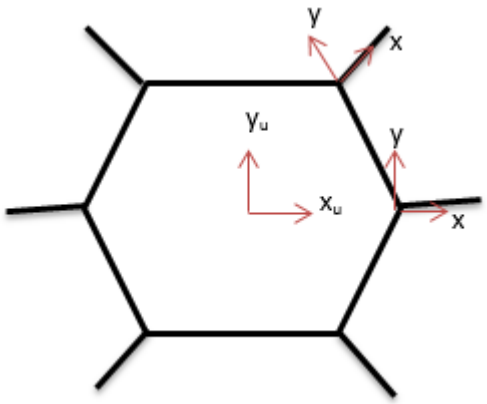

Figure 10: TigerBug's body coordinate system and leg positioning around body

Utilizing the DH parameters explained above, a DH table was created in order to organize all of the DH parameters needed to represent the frame motions from TigerBug's body frame as shown in Figure 10, to TigerBug's foot frame, as shown as reference frame 3 in Figure 9.

Table 2: TigerBug DH Table

Joint	**θ**	**d**	**a**	**α**
0	i*(π/3)	0	D	0
1	θ_1	0	L_1	π/2
2	θ_2	0	L_2	π
3	θ_3	0	L_3	0

Using Table 2, and Equation 2, four transformation, or *A*, matrices were derived. A_0 represents the transformation of the body frame onto the first joint's reference frame. A_1

represents the transformation of the first joint's reference frame onto the second joint's reference frame, and so on and so forth for A_2 and A_3. Once all of the transformation matrices have been obtained, post multiplying all of the transformation matrices together provides a complete transformation from the body's reference frame all the way to the foot's reference frame. This solution can be found in Appendix A.

The resulting transformation matrix can be divided into two subsections. The upper left 3x3 matrix describes all of the rotations the body frame goes through, while the first three elements of the fourth column describes the translation the body frame goes to on its way to the foot frame. Now that the complete transformation matrix has been obtained, it is possible to derive the Jacobian matrix that is needed to fully solve the forward kinematics problem. A Jacobian matrix is a matrix of the first order partial derivatives of a specific function. In this case, a solution is needed to translate differential joint space changes into differential Cartesian space changes, based on the definition of the forward kinematics.

Being that the fourth column of the final transformation matrix, ${}^{0}T_{3}$, is the positional vector of the foot with respect to the body reference frame in terms of the joint angles, θ_1, θ_2, and θ_3, and the link lengths, it can be used to solve the FK problem. It is important to note that the first element is representative of the foot's x position, the second, the y position, and the third, the z position. Three partial derivatives will be performed on each of these equations, one partial based on each joint angle. This transforms three equations into differential positions based on differential angles as shown in Equations 3, 4, and 5.

$$\partial x = i\partial\theta_1 + j\partial\theta_2 + k\partial\theta_3 \qquad [3]$$

$$\partial y = i\partial\theta_1 + j\partial\theta_2 + k\partial\theta_3 \quad [4]$$

$$\partial z = i\partial\theta_1 + j\partial\theta_2 + k\partial\theta_3 \quad [5]$$

Equations 3-5 can now be rewritten into matrix form, as shown in Equation 6, and the forward kinematics problem has been solved. Putting differential changes in joint angles into Equation 6, D_θ, will produce the resulting differential changes in position, D_p.

$$D_p = JD_\theta \quad [6]$$

3.2.4 Solving the Inverse Kinematics Problem

The last section detailed how to produce the forward kinematics solution through the use of the first order partial derivatives of each of the joint angle variables that make up a robot's leg. Building off of that previous derivation, the inverse kinematics can also be derived.

By definition, the inverse kinematics problem of an arm, or legged robot, is to be able to take in a differential position vector, and compute the differential joint angles needed to obtain this differential change in position. To do this all that has to be done is a simple rework of Equation 6, from the previous section.

Since Equation 6 provides a solution which produces a differential change in the foot's position based on the Jacobian multiplied by the given differential joint angle changes, it can be reworked to produce the needed differential joint angles based on a given differential position change. To do this, both sides of the equation will be pre-multiplied by the inverse of the Jacobian matrix, as shown in Equation 7.

$$J^{-1}D_p = J^{-1}JD_\theta \qquad [7]$$

Since any matrix multiplied by its inverse is the identity matrix, *I*, the right side of the Equation 7 simply becomes D_θ, and the inverse kinematic solution has been solved, as represented by Equation 8.

$$J^{-1}D_p = D_\theta \qquad [8]$$

3.3 Shortcomings of the Inverse Kinematics Solution

This particular inverse kinematics solution is subject to a few shortcomings. The first, and most apparent when put into practice, is that too large of a differential position change, will cause wildly inaccurate differential joint angles to be calculated and can cause the robot to become unstable, or if proper precautions are not taken, the robot to break itself apart or strip motor gearing. This particular issue is due to an approximation made previously, during the DH modeling procedure. In Equations 1 and 2, the small angle theorem is often applied in order to significantly simply the general transformation matrix equation. The small angle theorem states that given a small enough angle, the sine of that angle can be considered zero, and the cosine of that angle can be considered to be one as shown in Equation 9.

$$\begin{aligned} &\theta << \frac{\pi}{180}, \\ &\sin(\theta) \cong 0 \\ &\cos(\theta) \cong 1 \end{aligned} \qquad [9]$$

The small angle theorem also causes another slight problem. Due to the slight inaccuracy in rounding these trigonometric values to either 0 or 1, causes a slight rounding error in

the final joint angles of the leg. This is specifically noticeable during repetitive motions, such as a fixed walking gait. Given a small enough differential input, the error tends to be insignificant, unless the robot is repeating the same motion for a significant amount of time. A practical solution to this issue, is to reset the legs to a known “home” position based on absolute joint angles every once in a while. This can practically be done when the robot has either stopped receiving directional input, or if the robot stops for a moment to either make a path planning decision or to sample the surrounding environment.

The final issue that plagues this solution, and many other inverse kinematic solutions, is that of singularity points. These are joint configurations that cause the inverse of the Jacobian matrix to become unsolvable, due to the Jacobian becoming less than full rank. Many efforts have been made to detect, and prevent the robot from moving into these particular configurations, but the calculations are often times too computationally intensive to be done in real time on anything less than a desktop computer. This makes these solution impractical for tether-less robots operating away from humans.

Chapter 4: Gait Parameters and Restrictedness

4.1 Introduction

This chapter describes the theories and parameters that govern the gait generation process for a hexapod robot. The framework developed within provided TigerBug with smooth, omnidirectional motion capabilities in the form of a fixed gait and a rule-based free gait. Both gaits implemented on this controller produce stable motion over smooth terrain, and provides a base framework which can be modified to work over uneven terrain, once the hardware on TigerBug is in place to do so.

4.2 Gait Parameters

Merriam-Webster defines a gait as a sequence of movements which allow directional progress to occur. In order to have a successful and efficient gait, certain physical parameters need to be considered. First the duty cycle of the leg, β, will have an impact on the speed of the gait and will have a reciprocating effect on the stability of the gait. The state of each leg, whether it is in stance or swing, is also very important in determining if a gait will be stable and efficient. If too many legs are in the air at once, the robot will not be able to maintain its posture, and will have a tendency to fall over. If too many legs are on the ground, there may be awkward pauses or gaps in motion when legs attempt to reposition themselves in order to continue the robot's current trajectory. This is leads into the importance of knowing the each leg's position to either its anterior or posterior neighbor, in order to avoid collisions, based on the current state of the leg.

4.2.1 Leg Duty Cycle

In order for any gait to be considered periodic, a leg must be in the same state, at the same time in the cycle, and each leg cycle must take precisely the same amount of time. In order for the aforementioned criteria to occur, each leg must have the same duty cycle. A leg's duty cycle is analogous to the duty cycle of a pulse width modulated signal. It is simply the ratio of time a given leg spends in stance mode during an entire leg cycle, as represented by Equation 10.

$$\beta = \frac{t_{stance}}{t_{stance} + t_{swing}} \qquad [10]$$

In general, the less time a leg spends in stance mode, the faster and more unstable the gait becomes. This means that as β decreases, the robot tends to move faster, but has a greater chance to either stumble, or fall over during transitions between swing and stance mode. For a hexapod, the fastest, statically stable gait is a tripod gait. In a tripod gait, two sets of three legs are 180 degrees out of phase with one another in terms of leg cycle; i.e. when one set is at a certain point in stance mode, the other set is at the equivalent point in swing mode. This means that the tripod gait is fast because only three legs are ever on the ground at any given time, but also means that the robot is the least stable. The legs of a tripod gait have a duty cycle of 0.5, due to each leg spending equal time on the ground as it does in the air.

A duty cycle below 0.5 begins to cause instability in a walking system for the sole fact that legs are spending more time in the air than they are on the ground. This suggests that at a certain point during the gait cycle, there will be no legs on the ground.

This is not unheard of, especially in nature. When an animal is attempting to move very quickly, there is often a point during its gait where none of its legs are on the ground. In order to not fall over, the animal usually propels itself upwards slightly, in order to give the swinging legs time to return to the ground as they transition to stance mode. If a force is imparted on the animal during this transition time, the animal will usually stumble and fall over. Therefore this is not typically seen in robot walking due stability being a higher priority than speed in most applications.

4.2.2 Gait Stability

In order to ensure a useful and efficient gait, stability of the robot is often prioritized over many other parameters of the system. In order for a robot to remain upright, the center of mass of the robot must lie within a region that is created by the legs that are in stance mode at any given point during the gait cycle. This region is often called the triangle of stability for a hexapod robot.

The terms swing and stance mode have been thrown around quite often, but what exactly are these modes referring to? If a leg is in stance mode, it is on the ground, providing support for the entire robot and attempting to propel the robot in the desired direction of motion. The longer a leg stays in stance mode, the closer it becomes to the edge of its workspace and to its posterior neighbor. The closer a leg is to the edge of its workspace, the less it is able to support due to the increased torque on each of the leg's joints caused by the foot being further from the center of mass. In order to avoid excessive joint damage from over torqueing the joint servos, the legs are switched into swing mode. During swing mode, the leg is not on the ground, and is being moved closer to its anterior

neighbor in an attempt to put the leg in a more favorable load bearing position, once the foot is placed back down.

However, if transitioning a leg to swing mode will cause a violation of the stability triangle that leg cannot be put into swing mode. This leg will instead have to wait for another leg to be transitioned to stance mode, becoming a load bearing leg and satisfying the triangle of stability, thereby allowing the other leg to transition safely to swing mode.

Figure 11: Shows a stable system where, three legs are in stance mode (black) and three are in swing mode (white). If any leg is switched from stance to swing in the current configuration without another leg switching from swing stance, the system will become unstable, and fall over.

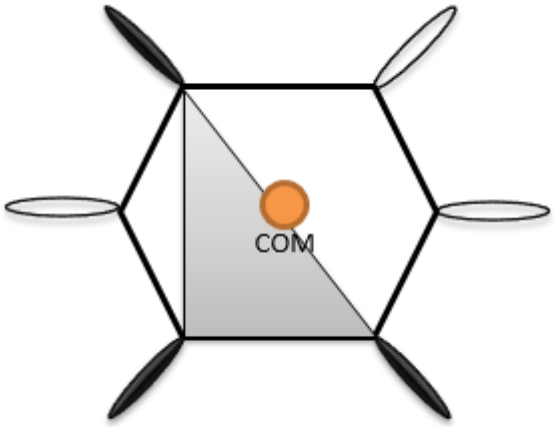

Figure 12: Shows a system which violates the triangle of stability. Note that the center of mass, COM, does not lie fully within the stability region.

4.2.3 Anterior vs Posterior

Within this work, the words anterior and posterior are used to define the legs that reside either in front of or behind the current leg that is being addressed. If leg one is being referred to, then leg 2 is the anterior leg, while leg 6 is the posterior leg. This concept becomes very important during the discussion of restrictedness. Depending on the state of the leg it will either be moving towards its anterior or posterior leg, and will therefore have an increasing restrictedness with respect to the neighboring leg it is moving towards, while having a decreasing restrictedness with respect to the leg it is moving away from.

4.3 Fixed Gait vs Free Gait

Fixed and free gaits are the two most common gaits implemented on all form of walking robots. While a fixed gait is usually favored for most applications, the free gait is extremely useful for those application in which a fixed gait fails. Both gait styles were implemented on TigerBug over the course of this project.

4.3.1 Fixed Gait

The defining feature of a fixed gait is the preprogrammed, periodic leg motion that is provided ahead of time for the robot to follow. As discussed in the duty cycle section, each leg arrives at the same spot in the leg cycle at the same time during the cycle, for every cycle that is performed during the gait. This can provide the robot with a very smooth and efficient form of locomotion that can be easily tuned for either speed or stability.

A fixed gait is simple to implement due to its predetermined pattern, but is not a very accommodating approach to locomotion. Due to the inherent periodicity of this gait style, robots that it is implemented on often have a hard time dealing with uneven and unpredictable terrain. If the robot encounters a slightly lowered portion of the environment, a leg transitioning from swing to stance mode will not know that it has to move further down in order to contact the ground. Therefore as the transition occurs, the robot can become unstable due to having too many legs in the air, and violating the stability triangle. Conversely, if a leg collides with an object in the environment or even with another leg on the system during either mode, a servo could become over torqued and burn out.

4.3.2 Free Gait

A free gait provides a very liberal and open-ended approach to robot locomotion. In a true free gait there is nothing that drives a legs movements other than the direction in which the robot intends to move. This is not a very efficient approach to locomotion due to the high probability of instability, and no guaranteed propulsion of the robot. Therefore

most researches tend to introduce a rule-based free gait. This modified version of a free-gait uses a set of rules to determine when a leg should be transitioned from stance to swing mode and vice-versa. The rule set governing the free gait can be anything from environmental input through sensors, to physical joint constraints, or workspace reachability.

The main advantage of the free gait over the fixed gait is its ability to adapt many different types of terrain without having to modify any code or parameters. With proper sensor feedback, the robot can know when a leg has been successfully placed back on the ground and allow other legs to switch to swing mode. It is also possible to know if a leg has collided with an object and to stop moving that leg along its current trajectory, and wait for an opportunity to switch its current mode. This gait can also be used over smooth terrain, and will often converge to the fastest stable gait possible; for a hexapod this is a tripod gait. The down side to this gait, however, is the complexity of the algorithm development. But once the algorithm has been developed, no other development needs to be done for a wide variety of environmental changes.

4.4 Restrictedness

Restrictedness was the parameter set used to drive the rule-based free gait implemented on TigerBug. Restrictedness is a measurement of a specific leg's ability to change its current location or mode based on a set of input parameters and the modes of said leg's posterior and anterior neighbors. This concept was developed by Fielding in 2002 and was implemented on a hexapod robot with a stick insect leg configuration.

4.4.1 Quantizing Restrictedness

The restrictedness value of a leg can take on any value depending upon the mathematical equation used to model each input into the restrictedness model. The symbol R_i^x can be used to describe the lack of freedom of the i[th] leg at any given time [10]. In general, if R_i^x is equivalent to its smallest possible value, as governed by the modeling equation, the leg is not restricted to any degree, and can move freely in any direction. Likewise if R_i^x is equivalent or greater than its largest possible value, as governed by the modeling equation, the leg is completely restricted and cannot move in any direction.

For this research, Fielding's suggestion of an exponential function to model restrictedness was used. This requires that each restrictedness parameter have a quantifiable minimum and maximum value such that as the parameter approaches either extreme, the exponential function approaches positive or negative infinity. An exponential function is an exceptional choice for modeling restrictedness due to its smooth nature over the entire range of a given parameter, and for its rapidly increasing nature as a limit is approached. As a parameter approaches one of its limits, the restrictedness model signals that with each additional move, that leg has a stronger and stronger desire to switch modes in order to make it less restricted. Equation 11 shows the exponential function used to model restrictedness during this research.

$$R_{i,k}^x = e^{\varepsilon^*(q_k - q_{k\max})} \rightarrow q > \frac{q_{k\max} + q_{k\min}}{2}$$
$$R_{i,k}^x = e^{-\varepsilon^*(q_k - q_{k\min})} \rightarrow q \leq \frac{q_{k\max} + q_{k\min}}{2} \qquad [11]$$

$$\varepsilon = \frac{\ln(0.1)}{\Delta q_k} \qquad [12]$$

In Equation 11, $R_{i,k}^{x}$ represents the restrictedness of a given parameter of a given leg. This does not represent the restrictedness of the entire leg, only the restrictedness value of the given parameter. The variable q_k represents the current value of the parameter being calculated, and likewise $q_{k,min}$ and $q_{k,max}$ represent the minimum and maximum possible values that the given parameter can take on. Fielding suggests limiting the value of restrictedness for each parameter between 0 and 1. In order to do this, it is suggested to use a smoothing factor, ε. The smoothing factor will determine how abruptly restrictedness will take effect. If ε is used as in Equation 12, $R_{i,k}^{x}$ will increase smoothly over a range of values from 0.1 to 1, without any abrupt jumps in restrictedness.

The lower the smoothing value is decreased, the value inside of the natural log, the more abruptly restrictedness takes effect. This means that as the parameter q_k approaches its minimum or maximum value, a small change in the parameter towards the limiting value can cause a very large and sudden change in restrictedness. Otherwise if the smoothing value is increased, the more linear the restrictedness model appears. Therefore any change in a parameter towards a limit will have a proportionally linear effect on the restrictedness value calculated. This can be seen in Figures 13 through 15 below.

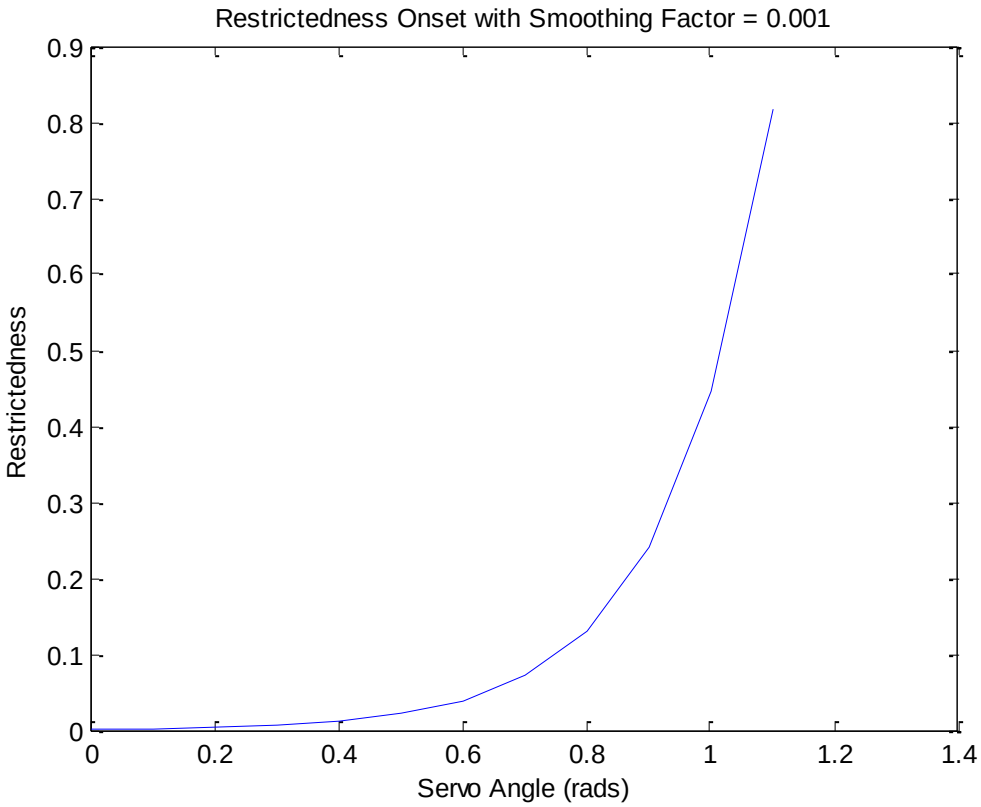

Figure 13: Restrictedness model with a smoothing factor of 0.001. Note the rapid increase of restrictedness as the parameter gets closer to its limit.

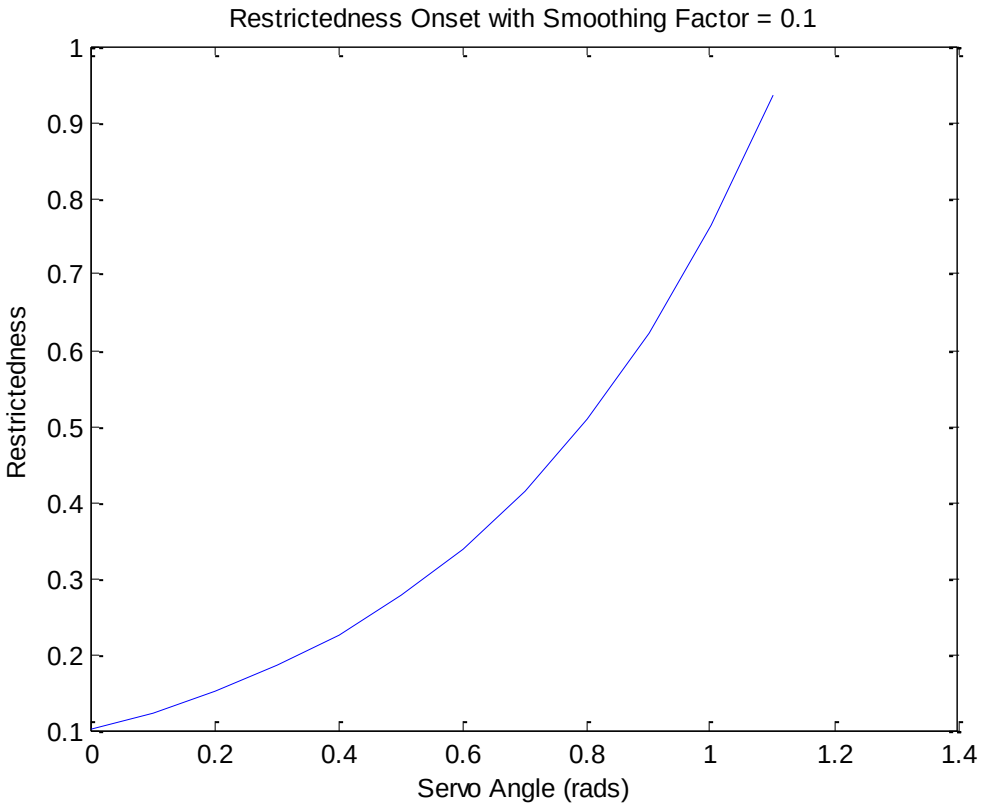

Figure 14: Restrictedness model using a smoothing factor of 0.1. Note the more gradual onset of restrictedness as compared to the model in Figure 13 above. This was the model used exclusively during this research.

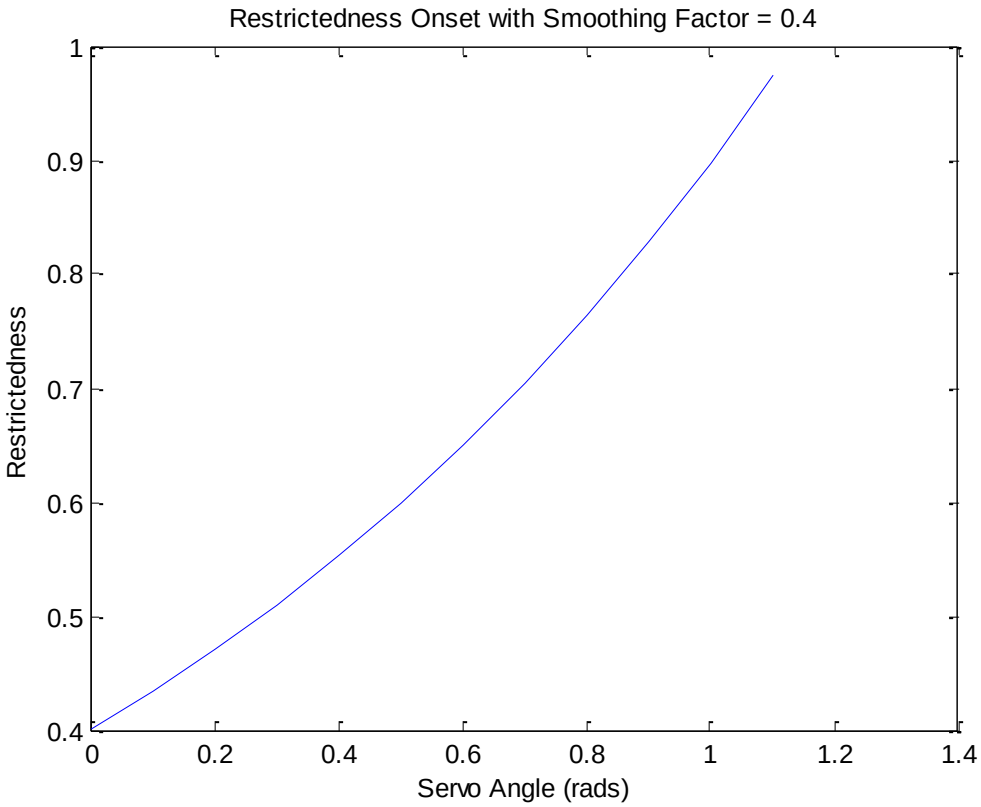

Figure 15: Restrictedness model using a smoothing factor of 0.4. Note the linear trend of restrictedness as the parameter approaches its limit. This will cause very little increase in urgency for switching as each movement has approximately the same differential increase in restrictedness.

Now that it is understood how restrictedness is calculated for each individual restrictedness parameter, the entire restrictedness of a leg can be calculated. This is simply done through superposition by summing the restrictedness of each parameter governing a legs total restrictedness. This is shown in Equation 13 below.

$$R_i^x = \sum_{k=1}^{n} s_{k,m} * R_{i,k}^x \qquad [13]$$

The term $s_{k,m}$ allows for scaling or complete elimination of a restrictedness parameter to be possible during the total restrictedness calculation. This comes in handy when a parameter is solely dependent upon the current leg to be moving towards a leg it is not

currently moving towards. In other words, if a restrictedness parameter is the distance between a leg's foot and its anterior leg's foot, and the leg is moving towards the posterior leg, this parameter can be ignored and $s_{k,m}$ can be set to 0 since there is no danger in colliding with the anterior leg's foot.

4.4.2 Restrictedness Parameters

The following list is a set of possible parameters that can be used to model the total restrictedness of a leg. These parameters can theoretically be applied to any legged robot with all revolute joints. This is not a comprehensive list, as there is a seemingly infinite set of possibilities based on sensor availability and implementation.

$R^x(\vartheta_1)$: Represents the restrictedness of joint one, the hip joint on TigerBug, based on a combination of physical and software limits. The upper and lower limits were set to protect against collision with neighboring legs to some degree even though the legs can still collide.

$R^x(\vartheta_2)$: Represents the restrictedness of joint two, the knee joint on TigerBug, based on a combination of physical and software limits. The upper and lower limits were set to avoid the physical limits of the servo and prevent over torqueing.

$R^x(\vartheta_3)$: Represents the restrictedness of joint three, the ankle joint on TigerBug, based on a combination of physical and software limits. The upper and lower limits were set to avoid both physical damage to the servo and leg and to avoid problematic positions for the inverse kinematic solution. The maximum limit was set to avoid the foot from folding too far under itself and causing instability and slipping. Likewise the minimum limit was

set to avoid slipping and the edge of the robot's work space, specifically when walking in the X direction.

$R^x(a)$: Represents the restrictedness based on the Euclidean distance between the current leg's foot and its anterior neighbor's foot. This parameter had its limits set in order to avoid collisions between feet and possible damage to servos.

$R^x(p)$: Represents the restrictedness based on the Euclidean distance between the foot of the current leg and its posterior neighbor's foot. This parameter had its limits set in order to avoid collisions between feet and possible damage to servos.

The above parameters were the chosen parameters to drive the restrictedness model of TigerBug during this research. The following are other possible parameters that can be implemented on most other hexapods.

$R^x(aA)$: Represents the restrictedness based on the Euclidean distance between the current leg's ankle and its anterior neighbor. This parameter would have its limits set in order to avoid collisions between feet and possible damage to servos.

$R^x(pA)$: Represents the restrictedness based on the Euclidean distance between the current leg's ankle and its posterior neighbor. This parameter would have its limits set in order to avoid collisions between feet and possible damage to servos.

$R^x(f)$: Represents the restrictedness based on a force sensor's feedback to the controller representing the current force the foot of a leg is applying to something. The limits would

be set to determine if the leg was on the ground or in the air in order to adapt to uneven terrain.

4.4.3 Basic Algorithm

Now that the concept of restrictedness is understood, a basic algorithm utilizing this concept can be developed. Based on the stability principles discussed previously, it was determined that two neighboring legs cannot be in swing mode at the same time without the robot becoming unstable. Therefore in order for a leg to transfer from stance to swing mode, both its anterior and posterior neighbors need to also be in stance mode. Also the leg looking to be switched into swing mode should be somewhere near the maximum allowable restrictedness value to avoid a leg being put into swing mode when another leg may be in a more restricted configuration and has a stronger need to go into swing mode.

Transitioning from swing mode to stance mode is much less restricted in terms of conditions that need to be satisfied for the leg to do so. Pretty much the leg can be considered in stance mode as soon as it has taken a foothold back on the ground and can provide load bearing support. Pseudo code for the above may look as follows:

```
for leg in range(0,6):
        if (legMode ==stance and legRestrct ~ maxRestrct and
        antLegMode==stance and postLegMode==stance):
                legMode = swing

        elif(legMode == swing and leg == onGround):
                legMode = stance
```

Adaptations necessary for implementation on TigerBug will be introduced in the next section. Even though the basic algorithm seems to cover most of the scenarios, during testing some problematic situations were uncovered.

4.5 Algorithm Adjustments

The previous section discussed the theory behind restrictedness and how it can be used in as a rule set for a rule-based free gait. The previous section also detailed a basic set of pseudo code for implementing this rule-based free gait. This section looks into scenarios that were uncovered during implementation that had to be addressed before the free gait would work as intended.

4.5.1 Switching to Swing Mode

Originally it was thought that looking at the total restrictedness of a leg was a good measure for determining if a leg should be switched out of stance mode and into swing mode. During testing this proved detrimental when legs were switching to swing mode, even though the restrictedness was decreasing due to the leg being in stance mode. To combat this, the difference between the legs current total restrictedness and its previous total restrictedness was calculated. If the value was positive, that means that the restrictedness is increasing due to the current movement, and the leg would flag that it needed to be switched to swing mode. If the value was negative, that would mean that current movement was causing the total restrictedness of the leg to decrease, and therefore the leg should not flag to be switched into swing mode. Regardless of this calculation, the leg would still check that it and both of its neighbors were in stance mode before attempting to be switched into swing mode.

4.5.2 When to Allow Movement in Stance Mode

Even if a leg cannot be switched into swing mode due to the current state of the legs in the system, that leg may not be able to move in stance mode due to it being over

restricted. It is important to not move this leg any further until it can be switched into swing mode to avoid any physical damage to the system. Therefore before allowing movement to take place while the leg is in stance mode, it checks the current restrictedness of the leg, and only allows movement if the current restrictedness is less than the maximum allowable restrictedness.

The only exception to this rule is if the leg has just come of out swing mode. In this scenario the leg may still have a restricted value due to its proximity to the anterior leg, or to a joint angle that was caused from the swing trajectory. If this is the case, the leg is allowed to move once in stance mode. Once this happens the decision discussed in 4.5.1 takes over to check if the restrictedness has decreased. If it has, the leg is allowed to continue in stance mode; otherwise it flags to be switched back into swing mode as soon as possible.

4.5.3 Swing Trajectory

When a leg was set to swing mode, it would follow a predetermined trajectory to attempt to move the leg to a less restricted position. The trajectory chosen was based on the equation for a sine wave. This provided the leg with a more realistic and smooth motion than if a hard coded, square path was taken. The equation would calculate the height of the foot based on the total distance traveled during the swing trajectory up to that point along the path. The differential change between this height and the previous height would then be calculated and feed into the inverse kinematics functions in order to move the foot to the correct height. The swing trajectory would cover a set amount of distance in

either the x or y direction as specified by parameters in the code, and the reciprocal of this would be used to calculate the necessary period of the sine wave, ω.

$$curHght = amp * \sin(\omega * diffChng_{x,y}) \quad [14]$$

4.5.4 Least Recently Used Algorithm

Fielding mentioned that giving certain legs priority in switching to swing mode may prove useful as it did for his implementation. During testing it was observed that sometimes a leg would come out of swing mode and then immediately go back into swing mode, causing the leg to reach an undesired position, and causing the system to fail. To rectify this, a least recently used algorithm, LRU, was implemented to ensure that a leg that hasn't been in swing mode recently had the first opportunity to switch into swing mode before any other leg.

The LRU algorithm was accomplished through the use of a first in, first out (FIFO) stack. All of the legs would be loaded into the stack, and the first one would be popped off. That leg would then be checked to see if it met the criteria for switching to swing mode, and if it did, the leg would be switched and that would be pushed to the bottom of the stack. Otherwise that leg would be appended to the bottom of the stack and be checked again once it reaches the top of the stack.

Chapter 5: Results

5.1 Experimental Setup

During all of the experiments performed during this research, the environmental setup was the same. The robot was powered by its own batteries that were always charged prior to starting any experiment. The SSC-32 servo controller was attached to a laptop through a serial cable, and a serial to USB adapter in order to receive servo commands from either the fixed or free gait scripts.

During initial testing, a severe decrease in gait performance was noticed if any of the mounting hardware on any of the legs or body was loose. Therefore, special care was taken to ensure that all hardware was tightened properly prior to any experiment being run. During these initial trials, it was also noticed that the feet often had trouble slipping on smoother surfaces, such as the linoleum floor of the lab. To ensure more accurate results, all walking tests were performed on a raised, carpeted platform. The platform was relatively level, with minimal changes in foot hold positions.

During walking tests, a line was struck on the platform that the robot would start behind. The robot would then move its legs to their initial position and hold them there for two seconds while the robot was adjusted to the correct starting position. From there no outside influences were imposed upon the robot and it was allowed to walk on its own for a predetermined number of algorithm cycles. Once the robot completed its walking routine, the final position was marked with a piece of tape, and the distance was measured with a tape measure and recorded.

5.2 Fixed Gait Walking

5.2.1 Fixed Gait Omnidirectional Movement

The first task that was accomplished from the proposed work was omnidirectional walking and posture control. Once the inverse kinematic solution was solved for as outlined in Chapter 3, differential rotations and translations of the body were tested about all three axes. This was accomplished through inputs of differential changes in and around any of the three axes through the keyboard on the laptop to the posture control function. The posture control function would first look at the current joint angles of all of the legs, and perform the desired differential change with respect to the current joint angles using the inverse kinematic solution. Doing all differential changes with respect to the current body posture allows for compounded motions to occur. Figure 16 show rotations about the x, y, z axes respectively. Figure 17 show the robot translated along all three axes. Figure 18 shows a body posture using compounded rotations and translations.

Figure 16: Differential rotations about the x, y, and z axes

Figure 17: Differential translations about the x, y, and z axes

Figure 18: A rotation about –z axis, translations along y axis, and a rotation about +x axis

All of the motions achieved above were done so very smoothly using the proposed kinematics approach. It is also important to note that these results prove that the kinematic equations implemented are correct. If these equations were incorrect, the feet of would leave the ground, and the robot would become unstable and ultimately fail to support itself.

Once the differential motion about each axis was completed, the proposed fixed gait approach was implemented. Once again the fixed gait function took in direction parameters from the laptop's keyboard and directed the robot in which direction to walk. In the end the robot was able to walk in three directions, x, y, and rotation about the center of its body, rz. Walking was accomplished by assigning two groups of three legs to move in opposite directions at the same time, known as a tripod gait. A differential stride length was fed into the algorithm, along with a foot height were fed into the function allowing for the gait parameters to be changed on the fly as the user saw fit.

A differential change in positon for each foot on the system was provided based on the desired direction of motion. This differential change was made with respect to the current joint angles. This allowed for the robot to be able to walk regardless of the robots current body posture. Once fully implemented, the robot was able to walk with the body rotated around any of the three axes.

5.2.2 Turning about a Radius

In order for the robot to be useful in difficult environments, turning while walking is a very important feature for the robot to poses. To do this, walking in both the rz and x or y direction at the same time must be possible. This was successfully implemented on TigerBug, providing the robot with the ability to turn exactly on the center of its body, or over radii of varying sizes. Figures 19 and 20 show the robot turning around two different radii.

Figure 19: Robot turning around a radius of π/15 radians while walking in x with strides of 15 mm

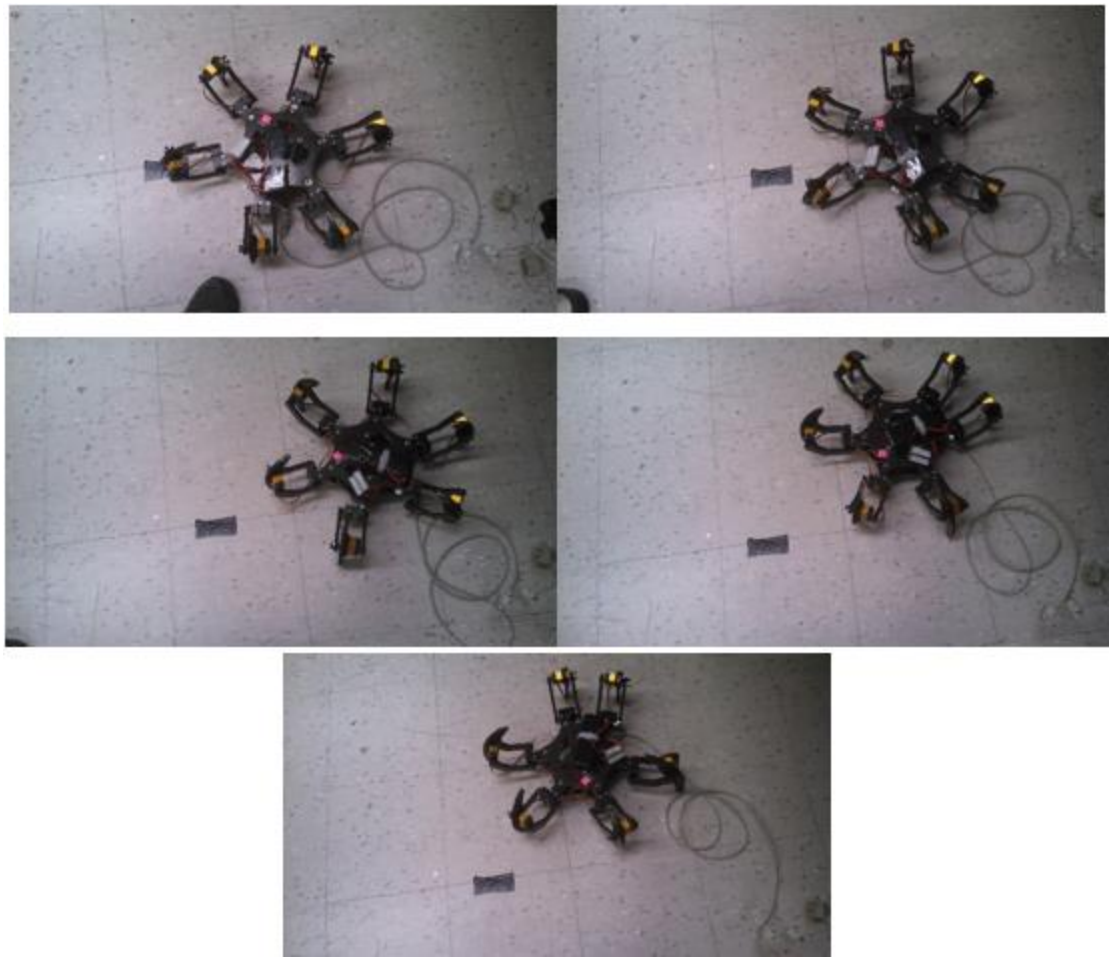

Figure 20: Robot turn around a radius of π/12 radians while walking in x with strides of 15 mm

It was noticed during the experiment it was quite easy to have TigerBug perform turns of varying degrees of tightness. In order to control the radius around which TigerBug would turn, amplitude of the rz motion would need to be changed. The larger the differential change around rz during the gait, the tighter the radius of the turn would be. The smaller the differential change around rz during the gait, the looser the radius of the turn performed by TigerBug would be.

5.3 Rule Based Free Gait

5.3.1 Implementation of the Free Gait

After successful implementation of the fixed tripod gait on TigerBug, it was time to attempt to implement the rule-based free gait using restrictedness. The restrictedness parameters chosen to represent the state of each leg were the hip, knee, and ankle servo angles, and the Euclidian distances between the anterior and posterior feet of each leg.

In order to ensure that the restrictedness algorithm was performing correctly, each parameter was swept from its minimum to its maximum value and the results were plotted using MATLAB. From the output, it was clear that each parameter sweep produced an exponential function. Figure 21 shows a sweep over a hip servos allotted angles. Note how the restrictedness decreases as the servo is relieved from its minimum constraint, and begins to rise again as it approaches the maximum constraint.

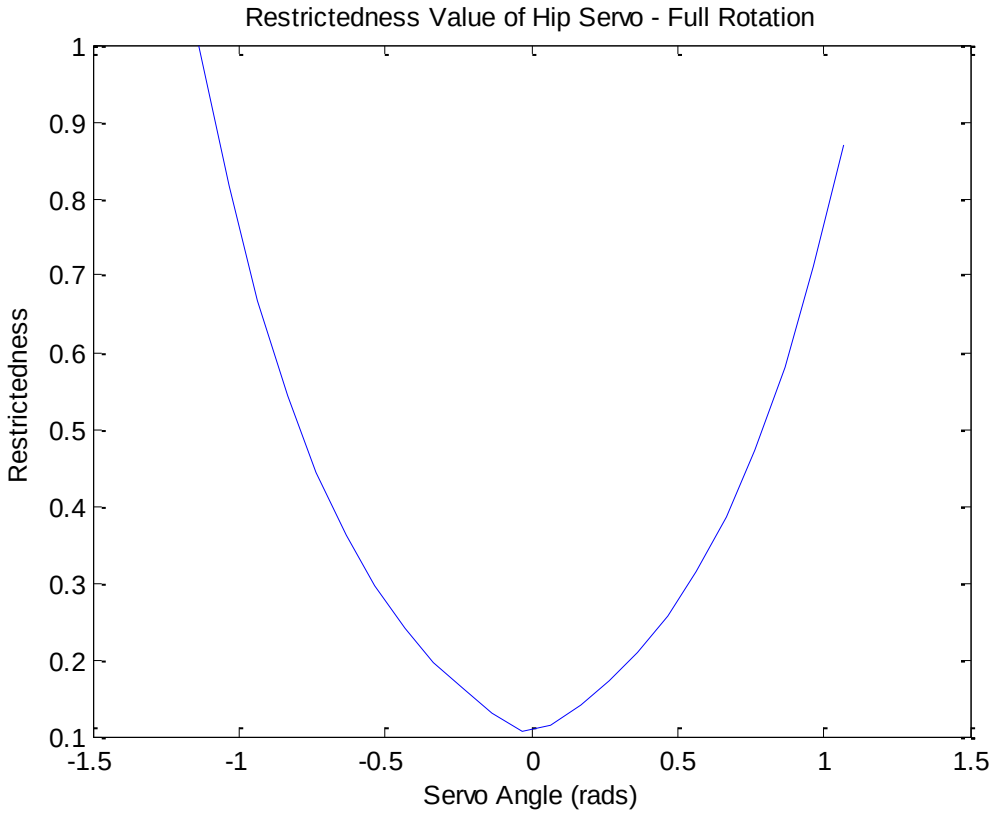

Figure 21: Restrictedness of a hip servo over its allotted angular range.

During initial experiments the TigerBug's legs were quickly becoming over restricted, and were not able to become unrestricted. This was a problem as legs that were transitioning from swing mode into stance mode would not move, causing all legs to eventually halt their motion and never continue. In order to rectify these issues, two solutions were implemented.

First, fewer parameters were used to calculate restrictedness. Because the implemented algorithm only allows a minimum value of 0.1 for any restrictedness parameter, the minimum restrictedness of a leg is 0.1 *nParams, where nParams is the number of parameters being used to calculate restrictedness. Since a limit on restrictedness of 0.8 was found to be among the most optimal, using anywhere near 8 parameters would cause

an issue due to the restrictedness of a leg being near or at 1 from the beginning, assuming that all parameters were in their least restrictive state. Therefore it was decided to use four restrictedness parameters in order to calculate leg's restrictedness. This allowed for enough parameters to be utilized in order to give a realistic description of the leg's current state, while still allowing the leg to move a reasonable amount during stance mode.

Second, legs that were just entering stance mode from swing mode were allowed to move regardless of how restricted they were upon landing. Logically, if a leg is restricted, moving it in the opposite direction should relieve it of being restricted. Therefore when a leg was transitioned out of swing mode and into stance mode the leg was allowed to be moved in the direction of the stance mode once, and the restrictedness was checked again on the next iteration. Usually this was enough to allow the leg to become unrestricted and continue to move in stance mode, allowing forward progress of the gait to continue.

The first implementation of the free gait did not utilize the LRU algorithm. During these trails it was noticed that some legs would transition to stance mode, and immediately be put back into swing mode without moving in the direction dictated by stance mode at all. This was causing legs to move to undesired and ill-mannered positions due to the use of a predetermined swing trajectory that was not checking the current foot position against its workspace and certain legs always being checked first for transitions. To combat this, the LRU algorithm was implemented to ensure that legs that had not been used in a while were checked for transitions first, before legs that were used more recently. The

implementation of this algorithm helped significantly with legs going into ill –mannered positions, and practically eliminated all cases of it.

5.3.2 Gait Efficiency Test

The first test performed on the rule based free gait, was one that attempted to sweep certain gait parameters and test how far the robot was able to move under each of the combinations over a set number of algorithm cycles. The further the distance traveled, the more efficient that particular parameter combination. To test this, a robot was started on the same spot of the raised carpeted platform and was allowed to walk for 250 algorithm cycles, i.e. the restrictedness of each leg was calculated 250 times before the trail was over. Table 3 shows the parameter sweeps and the total distance traveled for each parameter combination. From left to right the parameters are, maximum restrictedness, differential leg motion, total swing distance.

Table 3: Parameter sweeps for testing gait efficiencies

Parameters:				Distance Traveled:	
0.6	5	60		25.4	cm
0.6	5	80		26.988	cm
0.6	6	60		29.528	cm
0.75	5	60		21.908	cm
0.75	5	80		27.94	cm
0.75	6	60		29.845	cm
0.9	5	60		28.575	cm
0.9	5	80		29.21	cm
0.9	6	60		31.433	cm
0.95	5	60		22.86	cm
0.95	5	80		28.575	cm
0.95	6	60		33.02	cm

From Table 3 it can be seen that the parameter set that was able to travel the furthest was the last set of parameters tested. This result is not surprising as this parameter set offered the largest maximum restrictedness value, along with the largest differential change possible for each leg movement. It is also interesting that for each set of parameters for a given maximum restrictedness value, the distance covered over the set time increased in every case. This indicates that maximum swing distance and the amount of differential displacement for each leg movement has a larger effect than a higher maximum restrictedness value. During these trails it was also noted that the most visually appealing, and often times most efficient in terms of less slippage, parameter sets were often those that converged to a tripod gait more quickly.

5.4 Convergence Test

5.4.1 Criteria for Convergence

The final test performed on the proposed work was to test how quickly the rule-based free gait was able to converge to one of the standard fixed gaits, i.e. tripod, wave, ripple, etc. gait. In order to test for this, the free gait function was initialized and the robot was once again set on the carpeted platform. The robot was allowed to walk forward in the y direction until one of the existing periodic gait patterns emerged. During this process the amount of time before convergence as well as the number of algorithm cycles before convergence was recorded. In order for the gait to be considered to have converged, the assumed converged pattern had to happen at least 4 times in row. This experiment was repeated for all possible combinations of starting leg positions with regards to the legs starting in either stance or swing mode.

5.4.2 Convergence Test Results

Fielding discovered, through his work with rule-base free gaits using restrictedness, that the gait tended to converge to a tripod gait over flat terrain, and a wave gait over uneven terrain. These results appear to make sense as a tripod gait is faster, yet less stable than a wave gait, making it the most efficient gait over even, predictable terrain, while a wave gait provides the extra stability needed to maneuver efficiently over rough terrain. It was found during this set of experiments that a tripod gait was always converged to. However, at some points during the convergence to the predicted tripod gait, characteristics of a wave gait could be seen, especially during the convergences that took a longer time to complete. Table 4 below shows the results of the convergence experiment.

Table 4: Convergence test results

# swing legs:	# stance legs:	time conv:		cycles conv:	
0	6	138.5	sec	367	cycles
1	5	78	sec	213	cycles
2	4	46	sec	115	cycles
3	3	64	sec	177	cycles
4	2	40	sec	101	cycles
5	1	56	sec	147	cycles
6	0	68	sec	181	cycles

From Table 4, it can be seen that the combination of starting positions of the legs that yielded the quickest convergence time was the situation where 4 legs were started in swing mode and two were started in stance mode. This particular combination only took 40 seconds to converge to the predicted tripod gait.

In 2002, Fielding was able to get convergence to a tripod gait within 36 seconds; however, there may be a number of factors governing the slight difference in results. It is

possible that the different languages used to implement the free gait could have introduced delays. Python, which was used here, is a scripting language that does not pre-compile the script. It also reallocates memory for every function call which takes a lot of processor time. Another possible source of discrepancy would be the speeds of the motors implemented on each robot. It is possible that the servo motors implemented on TigerBot were not as fast as those implemented on Hamlet. Due to these, and numerous other possible time delays, it is believed that reporting the number of algorithm iterations would be a more accurate measure of how quickly a free gait is able to converge to regular gait from a complete random mess of motions.

Chapter 6: Conclusion

6.1 Algorithm Summary

In the end, two algorithms were successfully implemented in order to achieve omnidirectional walking in the hexapod, TigerBug. The first algorithm implemented was the fixed gait version of a tripod gait; the second was a rule-based free gait using restrictedness as the rule set. Posture control was also implemented in order to provide TigerBug with future adaptability to walking on gradients and over uneven terrain.

6.1.1 Fixed Gait

The fixed gait provides a simple solution for walking over flat terrain. A tripod gait was chosen for the fixed gait implementation because it is the fastest statically stable gait possible on a hexapod robot. Legs are changed between stance and swing modes based on a fixed amount of differential change in one direction and are always switched at the exact same point in time during the gait cycle, making the gait periodic.

A fixed gait like this does not take into account any environmental factors or the position of other legs in the system. Therefore this gait is poorly adapted to uneven terrain. Implementing a fixed gait on uneven terrain often leads to stumbling and/or dragging of legs and can cause severe damage to hardware components on the robot.

This algorithm can also be easily adapted to any other gait style through simple variable changes within the code. This allows the user to choose between speed and stability during different tests or environmental conditions.

6.1.2 Free Gait

A rule-based free gait is one of the best gait styles for adapting to uneven and unpredictable terrain. Using a set of rules, the legs in the system change state based on internal and environmental factors being fed back into the controller. The rule set chosen in this research was restrictedness which takes in multiple inputs based on the physical limitations of the leg and decides how free the leg is to move in its current state.

Since TigerBug was not outfitted with any environmental sensors, such as force sensors, the restrictedness was based off of joint angles and foot position of each leg. Once a leg became too restricted during stance mode to move any further, it was transitioned into swing mode at the first possible opportunity that would not cause the system to become unstable.

The developed rule-based free gait algorithm can be easily adapted for any type of input for future expansions which will allow TigerBug to maneuver through difficult, uneven terrain. Sensors can be easily integrated with the current code structure which will provide TigerBug with the needed environmental feedback to navigate this new type of terrain.

6.1.3 Posture Control

Posture control during walking is an integral part in uneven terrain adaptability of a walking robot system. The current algorithm allows the robot to walk with the body in different postures allowing the robot to maneuver on a gradient. A function was also developed to take in measurements from an inclinometer and have the body compensate its posture to counteract the gradient it was on, making the body parallel to the surface.

This was only tested with the fixed tripod gait, but adaptability to the rule based free gait would not be difficult. Since the free gait already looks at leg position relative to its neighboring legs as well as physical joint limits to decide restrictedness, the implementation of walking under different postures should not be difficult.

6.2 Summary

6.2.1 Work Completed

The following objectives were completed during this research:

1. Recalibrated and tuned up the circular hexapod robot TigerBug. Including developing an angle to PWM conversion for the servos currently implemented on each joint.

2. Developed a simpler and more straight forward approach to the forward and inverse kinematics on TigerBug than was previously developed.

3. Researched current legged robot systems and the advantages and disadvantages surrounding each system. Also researched current locomotion strategies for each legged robot.

4. Adapted Fielding's concept of restrictedness to a circular hexapod to control the step sequencing of the robot.

5. Implemented a fixed tripod gait that was controlled through user input on the keyboard. The fixed gait was also able to walk in many different body postures.

6. Implemented and tested a rule-based free gait using restrictedness on TigerBug.

7. Developed a swing trajectory to be used in the rule-based free gait based on a sine function using the desired total swing distance to calculate ω.

8. Implemented posture control using inclinometer to allow TigerBug to adapt to gradients about the x and y axes.

6.2.2 Capabilities

The implementation of restrictedness provides TigerBug with ability to maneuver over uneven terrain. This style of gait controller has a number of advantages over its fixed gait counterpart.

1. Restrictedness allows for omnidirectional walking under many different leg configurations due to the lack of a preprogrammed leg pattern. The ability to change direction at any point in the gait cycle is key for adaptability in uneven terrain where the robot may need to avoid an object in the environment during the middle of a gait cycle.

2. Free gaits are vital for locomotion over unpredictable terrain. The ability to adapt to random or nonexistent footholds is crucial for walking over uneven terrain. The free gait will also converge to the best gait based on the input to the restrictedness algorithm. If the inputs correspond to even terrain, the gait will converge to a tripod gait, but they correspond to uneven terrain the gait will converge to a wave or ripple gait.

3. It is easy to adapt the restrictedness to changes in hardware on TigerBug or even implement the algorithm on an entirely different robot all together. Changing the

algorithm to accommodate new sensors for environmental input is as simple as providing the algorithm with minimum and maximum values for the sensors and adding another input to the array of inputs for the restrictedness function. This is extremely simple when compared to other gait algorithms which may need to be retrained or the parameter set completely reassessed in terms of a neural controller or a graph search controller when it comes to introducing new hardware or sensors to the robot.

6.3 Future Work

6.3.1 Updated Foot Design

During testing, it was noticed that the current design of the link connecting the ankle joint to the ground was not as structurally stable as desired. Because this link is only attached to the servo horn with all of the servo's weight hanging off of the back of the link, deflections in this link was noticed when a leg was in the support phase. This often led to legs not being able to provide proper support for the body and causing awkwardness during walking at times due to slipping.

To fix this a new link should be developed that will support the servo on both sides and come to a point underneath the servo's center of mass. This will provide better support for the servo and cause less deflection of the leg during standing mode. This design will also induce less stress on the hardware connecting the servo to the leg and the servo horn itself.

6.3.2 Optimization Algorithm

In order to obtain the most efficient walking performance out of the rule-based free gait, an optimization algorithm could be developed to sweep a variable space of all of the gait parameters effecting gait performance and determining the best combination of gait parameters to produce the most efficient locomotion for TigerBug. This could be implemented using a particle swarm optimization (PSO) algorithm which would use the gait parameters as inputs and the effective forward displacement of the robot under the current gait parameters as a fitness function. This could be implemented on the physical robot, or on the webots simulation developed by a previous student.

6.3.3 Tetherless Functionality

A key functionality to a fully autonomous robot is the ability to operate without being attached to a large, bulky computer. Implementing the current algorithms on a microcontroller would be a very suitable option to accomplish tetherless functionality. Due to the limited amount of memory and clock cycles on a microcontroller, some of the precision of the current algorithms would have to be sacrificed for speed optimization.

It would also be a wise decision to adapt the current python 2.7 code to a precompiled language such as C, C++, or embedded C. This will help with execution time as python is scripting language and allocates memory for each separate function call.

Wireless communication back to a computer acting as a server would also be a beneficial functionality for data collection as the robot operates in its environment. This could be done via wifi as RIT has a largely connected campus or via Bluetooth for shorter ranged applications.

6.3.4 Leg Prioritization Algorithms

Different leg prioritization algorithms, other than the LRU algorithm implemented here, would be an interesting study to perform in order to better understand the effect of leg prioritization on gait convergence. During testing it was seen that the faster a gait converged, the more efficient it was in terms of distance covered over a specific time. The two specific priority algorithm families that would be most useful for this functionality would be fixed and adaptive priority algorithms.

REFERENCES

[1] Dunlop, G.; Fielding, M., "Omnidirectional Hexapod Walking and Efficient Gaits Using Restrictedness," The Internal Journal of Robotics Research, Sep. 23, 2004

[2] Ahmed, M.; Billah, M.; Farhana, S., "Walking Hexapod Robot in Disaster Recovery: Developing Algorithm for Terrain Negotiation and Navigation," World Academy of Science, Engineering and Technology, vol.2, pp. 307-312, 2008

[3] Chen, W.; Duan, X.; Liu, J.; Yu, S., "Tripod Gait Planning and Kinematic Analysis of a Hexapod Robot," IEEE International Conference on Control and Automation, pp.1850-1855, 9-11, Dec. 2009

[4] Almeida, A.; Barreto, A.; Dias, J.; Menezes, P.; Trigo, A., "Kinematic and Dynamic Modeling of a Six Legged Robot," Insituto de Sistemas e Robotica, Leiria, Portugal

[5] Grepl, R., "Extended Kinematics for Control of Quadruped Robot," Institute of Solid Mechanics, Mechatronics, and Biomechanics, Faculty of Mechanical Engineering, Brno University of Technology, Czech Republic

[6] Shkolnik, A.; Tedrake, R., "Inverse Kinematics for a Pointed-Foot Quadruped Robot with Dynamic Redundancy Resolution," IEEE International Conference on Robotics and Automation, pp.4331-4336, 10-14, Apr. 2007

[7] Konno, A.; Ouezdou, F.; Sellaouti, R., "Design of a 3 DOFs Parallel Actuated Mechanism for a Biped Hip Joint," IEEE Conference on Robotics and Automation, pp. 1161-1166, May 2002

[8] Baerlocher, R., "Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures," Ecole Polytechnique Federale De Lausanne, 2001

[9] Hajaibadi, M., "Analytical Workspace, Inverse Kinematics, and Foot Based Stability of Hexapod Walking Robots," Worcester Polytechnic Institute, May 2013

[10] Fielding, M., "Omnidirectional Gait Generating Algorithm for Hexapod Robot," University of Canterbury, New Zealand, June 2002

[11] Jayarajan, K.; Pal, P., "A Free Gait Generalized Motion," IEEE Transactions on Robotics and Automation, pp. 597-600, vol.6, No.5, Oct 1990

[12] Estermera, J.; de Santos, P., "Generating Continuous Free Crab Gaits for Quadruped Robots on Irregular Terrain," IEEE Transactions on Robotics and Automation, pp. 1067-1076, vol.21, No.6, Dec 2005

[13] Buehler, M.; Groff, R.; Koditschek, D.; Weingarten, J., "Gait Generation and Optimization for Legged Robots," University of Michigan, Center for Intellegent Machines

[14] Niku, S., “Differential Motions and Velocities-3.8 Calculating the Jacobian,” *An Introduction to Robotics Analysis, Systems, Applications,* Upper Saddle River, N.J.: Prentice Hall, 2001. Print.

[15] Roboard, *Roboard Servo Motor RS-1270 Manual V1.01*, Taipei, Taiwan, May 2011

[16] Lynxmotion, *SSC-32 Manual*, Version 2.01XE, 16 June, 2010

[17] DH Parameter Illustration, *http://en.wikipedia.org/wiki/File:DHParameter.png*, 10 Nov, 2014

Appendices

Appendix A: Forward and Inverse Kinematic Equations

$$ {}^{i-1}T_i = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i}C_{\alpha_i} & S_{\theta_i}S_{\alpha_i} & a_i C_{\theta_i} \\ S_{\theta_i} & C_{\theta_i}C_{\alpha_i} & C_{\theta_i}S_{\alpha_i} & a_i S_{\theta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

where

θ_i, d_i, a_i, and α_i represent the Denavit-Heternberg Parameters for the given joint

$$ C_a = \cos(a) \quad and \quad C_{ab} = \cos(a-b) $$
$$ S_a = \sin(a) \quad and \quad S_{ab} = \sin(a-b) $$

Equation A1: Body Center to Foot Transformation for a Given Leg

$$ {}^{W}T_F = {}^{W}T_1 * {}^{1}T_2 * {}^{2}T_3 * {}^{3}T_F $$

${}^{W}T_1$ is the transformation from the world frame at the center of the body to the hip frame

${}^{1}T_2$ is the transformation from the hip frame to the knee frame

${}^{2}T_3$ is the transformation from the knee frame to the ankle frame

${}^{3}T_F$ is the transformation from the Ankle frame to the foot frame

${}^{W}T_F$ is the entire transformation from the body reference frame to the foot reference frame

Equation A2: Fourth Column of complete frame transformation

$$ {}^{W}T_{F}[:,4] = \begin{bmatrix} Dc_i + l_1(c_1c_i - s_1s_i) + c_2l_2(c_1c_i - s_1s_i) + l_3(c_1c_i - s_1s_i)c_{23} \\ Ds_i + l_1(c_1s_i + s_1c_i) + c_2l_2(c_1s_i + s_1c_i) + l_3(c_1s_i + s_1c_i)c_{23} \\ l_2s_2 + l_3s_{23} \\ 1 \end{bmatrix} $$

$$ where\ \ i = legNum * \frac{\pi}{3} $$

Appendix B: Pin Out and Angle to PWM Signals

Table B1: Servo Pin Out per Leg

Leg		Pin #
1	Hip	0
	Knee	1
	Ankle	10
2	Hip	17
	Knee	18
	Ankle	19
3	Hip	20
	Knee	21
	Ankle	22
4	Hip	23
	Knee	24
	Ankle	25
5	Hip	6
	Knee	7
	Ankle	8
6	Hip	3
	Knee	4
	Ankle	5

Table B2: Max, Min, and Center PWM Values per Leg

Leg		0	Max	Min
1	Hip	1500	1950	1050
	Knee	1470	1950	1050
	Ankle	1500	1950	1050
2	Hip	1470	1950	1050
	Knee	1500	1950	1050
	Ankle	1500	1950	1050
3	Hip	1500	1950	1050
	Knee	1475	1950	1050
	Ankle	1500	1950	1050
4	Hip	1500	1950	1050
	Knee	1475	1950	1050
	Ankle	1500	1950	1050
5	Hip	1570	1950	1050
	Knee	1500	1950	1050
	Ankle	1500	1950	1050
6	Hip	1500	1950	1050
	Knee	1475	1950	1050
	Ankle	1500	1950	1050

Equation B1: PWM calculation based on the center servo value and slope of the best fit line for a given leg

$$PWM = 286.48 * \theta_{joint,leg} + Center_{joint,leg}$$

Appendix C: Leg Frame Placement

Red = X Axis
Blue = Y Axis
Green = Z Axis

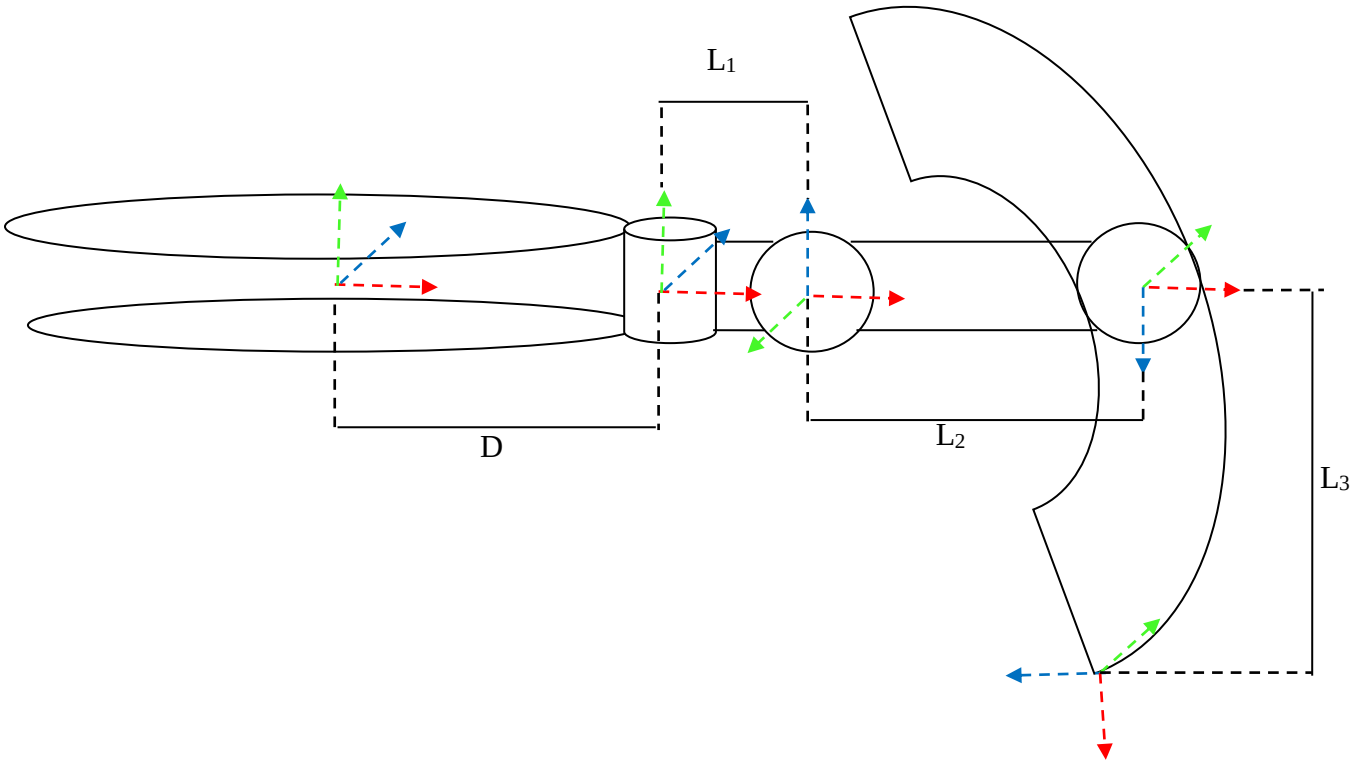

Table C1: Link Lengths

D	96 mm
L_1	42 mm
L_2	101 mm
L_3	106 mm