Push Recovery: A Machine Learning Approach to Reactive Stepping

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Jennifer Leigh Horton

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2013

Master's Examination Committee:

Professor Yuan Zheng, Advisor

Professor David Orin

Abstract

When robots are integrated into the real world, chances are they will not be able to completely avoid situations in which they are bumped or pushed unexpectedly. In these situations, the robot could potentially damage itself, damage its surroundings, or fail to perform its tasking unless it is able to take active countermeasures to prevent or recover from falling. One such countermeasure, referred to as reactive stepping, involves a robot taking a series of steps in order to regain balance and recover from a push. Research into reactive stepping typically focuses on choosing which step to take.

This thesis proposes a machine learning approach to reactive stepping. This approach leverages neural networks to calculate a series of steps that return the robot to a stable position. It was theorized that the robot would become stable if it always chose the step resulting in the highest reduction of energy. Theories were tested using a compass model that incorporated parameters and constraints realistic of an actual humanoid robot. The machine learning approach using neural networks performed favorably in both computation time and push recovery effectiveness when compared with the linear least squares, nearest interpolation, and linear interpolation methods. Results showed that when using neural networks to calculate the best step for an arbitrary push within the defined range, the compass model was able to successfully recover from 97% of the pushes applied. The procedure was kept very general and could be used to implement reactive stepping on physical robots, or other robot models.

This thesis is dedicated to Ryan Kay.

Acknowledgments

I would like to thank my advisor, Dr. Yuan Zheng, for providing me with the opportunity to work on this project. I would also like to express my gratitude to Dr. Manoj Srinivasan for providing guidance on MATLAB simulations that was integral to the success of this project. Additionally, I would like to thank Dr. David Orin for taking the time to serve on my committee.

I would like to thank my parents, James and Pamela Horton, for their love and support throughout my education. I would also like to thank Ryan Kay for his help in developing this thesis and for always being there for me. Finally, I would like to thank my friends for all of the wonderful experiences over the last six years at The Ohio State University.

Vita

June 2007……..Colonel Zadok Magruder High School

June 2008 to September 2009……...Intern, Lockheed Martin Corporation

June 2010 to August 2012…….....Intern, The Johns Hopkins University Applied Physics Lab

March 2012……...B.S. Electrical and Computer Engineering, The Ohio State University

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Background

Robotics has become a very popular field of study among electrical engineers, mechanical engineers, and computer scientists. Aside from their broad nature and technical complexity, researchers are often interested in robotics due to the immense benefits they could offer humans.

Major robotic research efforts are taking place to develop both commercial and military robots. Some of the earliest commercial robots included industrial robots that were used for tasks such as pick and place, painting, and welding. Often times, industrial robots are favored over humans for performing these tasks due to their accuracy, speed, and lack of fatigue. Other robots have been developed to perform tasks that humans either do not want to perform or do not have the time to perform on a regular basis. Examples of these robots include the Lawnbott [1] and the iRobot Roomba vacuum cleaning robot [2]. One of the more recent developments is the application of robots to the surgical field. Surgical robots, as shown in Figure 1, have the potential to allow remote surgeries, which will permit any number of specialized surgeons to be chosen rather than limiting patients

to those surgeons physically present. They also allow for smaller and more precise incisions during procedures [3].

Figure 1: The da Vinci Surgical System Made by Intuitive Surgical [4]

One of the latest military robots is DARPA's LS3 pack mule robot. LS3 is currently being developed in order to address the large amount of equipment that soldiers are required to carry. If the pack mule robot is able to reduce the load that soldiers must carry, soft tissue injuries and fatigue in soldiers could be reduced [5]. Another place where robots prove useful is in situations that are too dangerous for humans, such as bomb disposal. Bomb disposal robots, as shown in Figure 2, are given tasks that range from moving the bomb to a safe location for detonation to disarming the device.

Figure 2: The TALON Bomb Disposal Robot [6]

Similarly, in response to disasters such as the Fukushima Daiichi nuclear disaster and the Deepwater Horizon oil spill, DARPA is currently sponsoring a challenge to develop disaster response robots. The goal of these robots is to perform necessary tasks in environments that may be unsafe for humans, such as those experiencing structural instability or nuclear contamination [7]. The aforementioned situations are just a few examples of the many valuable applications that are pushing researchers to take an active interest in the field of robotics.

Modern land-based robots can be classified into two major categories: robotic manipulators and mobile robots. Robotic manipulators, as shown in Figure 3, include a non-mobile base, one or more limbs consisting of several links connected by rotational or translational joints, and one or more end effectors used to complete tasks. Most industrial robots and surgical robots are classified as robotic manipulators.

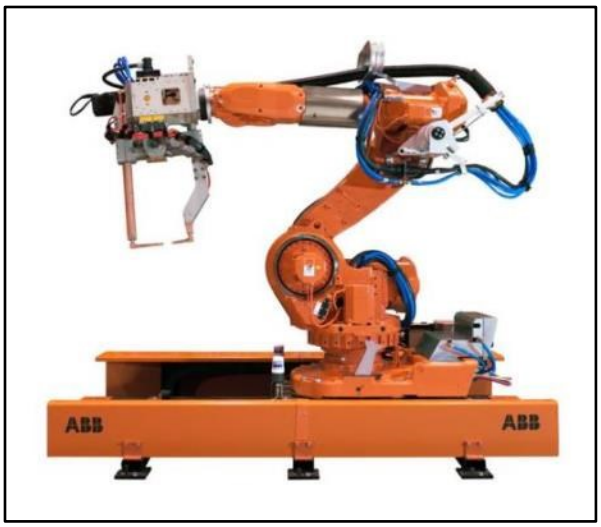

Figure 3: A Robotic Welder [8]

Mobile robots can physically move their location while in operation and can be classified into three major categories: wheeled robots, treaded robots, and legged robots. Wheeled robots, as shown in Figure 4, are comprised of a mobile chassis that moves its location through the use of motorized wheels. Treaded robots are similar but include a continuous track that connects the wheels on each side of the robot. Among mobile robots, wheeled and treaded robots are popular choices due to the ease of development and control. They also have the advantage of being able to travel at relatively high speeds. However, wheeled robots are limited in the fact that they usually need a continuous area of even terrain over which to drive. While treaded robots can handle slight gaps or minor levels of uneven terrain, most are incapable of climbing stairs or walking on discrete footholds such as stepping stones [9].

Figure 4: The Mars Rover [10]

Legged robots, as shown in Figure 5, include a base that moves its location through the use of linkages connected by rotational or translational joints. These linkages often resemble the legs of either animals or humans. Legged robots are able to overcome some of the limitations of wheeled robots, but they generally travel at low speeds and are extremely difficult to develop due to their complex motions and naturally unstable nature.

Figure 5: BigDog, A Quadrupedal Robot Created by Boston Dynamics [11]

A subset of the legged robot category includes robots with only two legs, or bipedal robots, as shown in Figure 6. In addition to the benefits of legged robots mentioned above, bipedal robots also have several other advantages. One such advantage is that the length of a bipedal robot is typically smaller than a robot with three or more legs, which allows them to navigate tight turns more easily. Bipedal robots also have the potential to use less energy as they will have fewer actuators than robots with three or more legs [9]. Another advantage is that the development of bipedal robot motions can greatly benefit from their close resemblance to humans. Since bipedal robots bear a close resemblance to humans, much of what has been learned about human locomotion can be leveraged for use in the development of bipedal robot locomotion. Finally, studying human locomotion for the purposes of developing bipedal robots can lead to an increased understanding of human movement, which can aid in physical therapy and prosthetic development.

Figure 6: ASIMO, A Bipedal Robot Created by Honda [12]

It is expected that bipedal robots will be able to assimilate into the human world more easily than other types of robots. People are more comfortable interacting with robots if they resemble humans. The world has also been developed for use by humans. The size of doorways and walkways, the shape of tools, the placement of pedals in a car, and the height of doorknobs and light switches were all specifically designed to be convenient for human use. If robots are developed to be human-like, then the environment will not have to be modified in order for it to be convenient for use by robots [13].

1.2 Motivation

Unfortunately, even with leveraging the benefits of their resemblance to humans, bipedal robots are still extremely challenging to develop, mostly due to issues with stability. With only two legs on the ground on which to balance, bipedal robots are naturally very unstable. If the motions are carefully planned out, many bipedal robots are able to stand and walk stably in controlled, isolated environments. However, when operating in the real world, robots will meet additional challenges such as being bumped or pushed unexpectedly. While the frequency of these occurrences may be reduced through the use of vision strategies to avoid objects, some situations in which a robot may be bumped or pushed are unavoidable. In these situations, the robot is likely to fall if it does not take active countermeasures.

Falls can be devastating to a robot for several reasons. First and foremost, most robots are very fragile and very expensive, so a fall could result in the need to make very costly repairs. A robot could also break or hurt an object or person that it falls into. Another issue to consider is if a robot falls and cannot stand back up, it will be rendered useless and unable to complete its assigned tasks. Even if the robot is able to stand back up, it will have taken longer to complete its assigned tasks due to the extra time associated with standing up.

It is widely accepted that these issues must be addressed before robots are able to assume a prominent role in society. A summary of previous research pertaining to this topic is included in the next section.

1.3 Literature Review

When a robot is pushed, it must first recognize that it has in fact been pushed and identify how hard it has been pushed in order to decide on an appropriate response. To a human, these may seem like trivial tasks, but fall prediction for a robot is actually very complex and has been approached many different ways in previous research. Renner and Behnke recorded a sequence of parameters during undisturbed walking, and a push was considered to have been detected if the robot's sensor readings deviated from that sequence. They found that this approach typically recorded less false positives than using the distance between the zero-moment point and the edge of the support polygon to detect instability [14]. Hohn et al. used pattern recognition along with sensor data, such as the

angles and velocities of the legs, to recognize and classify falls. While the pattern recognition was only trained using pushes, it was also able to recognize when the robot began stumbling [15].

Once a robot has predicted a fall, it should attempt to recover from the push and avoid falling. There are three different forms of push recovery, which are commonly referred to as ankle strategy, hip strategy, and reactive stepping.

When the push is small, ankle strategy allows the robot to regain balance simply by applying a torque at the ankle. Yi et al. indirectly implemented ankle strategy by controlling the zero moment point of the robot in real time. Indirectly implementing ankle strategy was found to be more effective than using direct control of the ankle due to the tendency of the feet to tip before ankle control could affect the position of the robot [16]. Stephens implemented ankle strategy with the goal of returning the leg to a vertical position, thereby causing the leg to act like a stiff inverted pendulum. Results of the study found that this method of control accurately reflected hip strategy as performed by human subjects [17].

If the push is too large to recover using ankle strategy, hip strategy allows the robot to regain balance by applying a torque at the hip. Yi et al. applied a torque to accelerate the center of mass, thereby counteracting the movement caused by the push. While results were promising, they did not take arm motions into account for fear of damaging the physical robot, so further investigation may be necessary [16]. Lee and Goswami

implemented a controller that prioritized the required linear momentum to remain balanced while sacrificing the required angular momentum if necessary. This caused the robot to bend at the hip in order to avoid moving forward. The paper suggests that better results may have been achieved if the controller was adjusted to prioritize some aspects of both linear and angular moment rather than just linear momentum [18].

If the push is too large to recover from using hip strategy, reactive stepping allows the robot to take steps in order to reach a more stable position. Pratt et al. introduced the concept of a Capture Point, which is a point on the ground such that if the robot steps on that point, then the kinetic energy of the robot will be able to both become and remain at zero. The article only addressed situations where the robot could become stable through one single step [19]. In contrast, Pratt and Tedrake estimated N-Step Capture Points by using a brute-force search algorithm [20]. A major drawback to using Capture Points to determine the step location is that a Capture Point is estimated based on the current state of the robot, which will change as the robot performs the step. As such, a Capture Point must be recalculated and the trajectory must be changed several times as the step is being taken [20, 21].

Yun and Goswami expanded on the work from [18] to include a controller that prioritized the required angular momentum to remain balanced while sacrificing the required linear momentum if necessary. In contrast to the controller from [18], this new controller caused the robot to take a step in order to avoid bending at the hip. Their controller determined the step location by modeling the robot as a passive rimless wheel and

choosing to step at a location which would cause the wheel to stop when the spoke was in a vertical position. Unlike a Capture Point, the calculation of this step location was predictive, so the step location only needed to be calculated once during the stepping process. One drawback to this method is that it did not present a recovery solution when the computed step location could not be reached by the robot within a single step [21].

If a robot is unable to recover from the push using the ankle strategy, hip strategy, or reactive stepping, then a fall is unavoidable for the robot. If a fall is unavoidable, the robot should implement a fall control strategy. Fall control strategies may concentrate on either minimizing damage to the robot's surrounding or minimizing damage to the robot itself. In the work by Yun et al., the robot either began to take a step or generated angular momentum in order to change the robot's fall direction away from delicate surrounding objects. The proposed method approximated the robot's future states using a simple inverted pendulum, which due to its simplicity, may have resulted in approximation errors [22]. Lee and Goswami proposed rotating the swing leg and the trunk in order to change the fall direction of the robot such that it would fall on its impact absorbing backpack, thereby reducing damage to the robot. Using this technique, the robot was able to successfully fall on its backpack when falling backwards or sideways but not when falling forward [23].

If the robot has fallen over, then the robot should have a strategy for standing back up to prevent it from being rendered useless by the fall. It should be noted however that in order to successfully implement a standing-up strategy, the robot must remain undamaged either through a robust design or one of the fall control strategies discussed previously. Much of the leading research on standing-up routines in robots has been conducted during the development of soccer playing robots for the RoboCup Humanoid League [24]. One such RoboCup competitor, DARwIn-OP, is capable of successfully implementing the standing-up routine shown in Figure 7 [25].

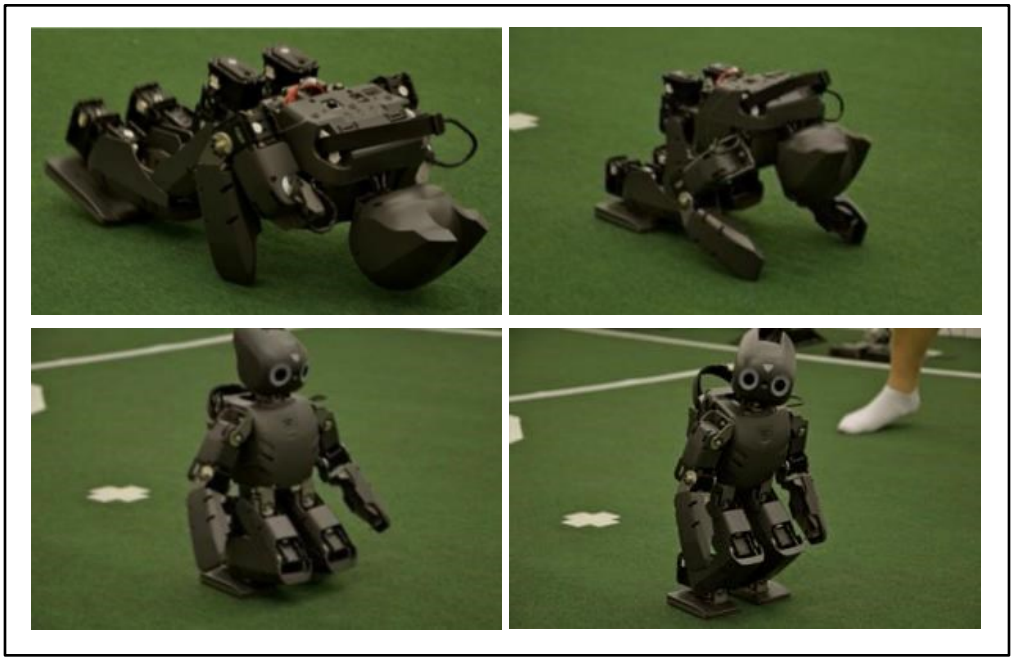

Figure 7: DARwIn-OP Executing His Standing Up Strategy [25]

1.4 Thesis Organization

This thesis introduces a new approach to the reactive stepping method in which a robot uses machine learning to calculate the step that will result in the highest energy reduction. The thesis is organized as follows.

Chapter 2 gives an overview of the compass robot, which is how the bipedal robot will be represented within this thesis. This chapter will include the equations of motion and the changes in leg velocities that occur when taking a step. Chapter 3 details the MATLAB model that was created to simulate a walking compass robot. This chapter will define the parameters of the system, specify the conditions that make the model invalid, and explain the different states of the simulation.

Chapter 4 explains the Energy Reduction Method for choosing the best step a robot can take to recover from a push. Chapter 5 specifies how the Energy Reduction Method could be implemented on a robot using various interpolation methods. Chapter 6 discusses the effectiveness of the various interpolation methods. Chapter 7 summarizes the thesis and outlines how the work could be expanded in the future.

CHAPTER 2

Compass Robot

2.1 Introduction

The compass robot and subsequent equations used within this thesis were borrowed from the *Springer Handbook of Robotics* [26], but they are also very widely used by many in the field of robotics. The compass robot, shown in Figure 8, is a simple version of a bipedal robot that uses two knee-less legs connected by a revolute joint at the hip.

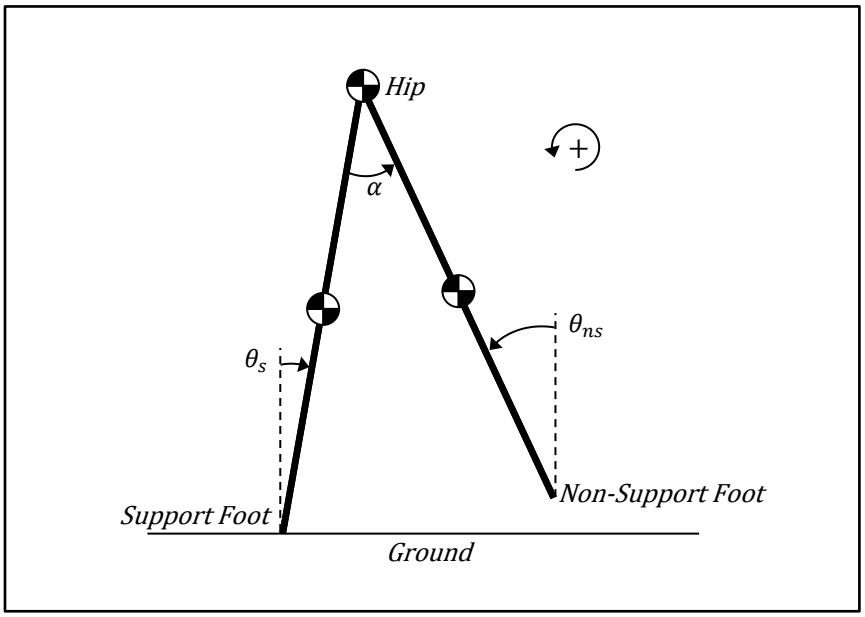

Figure 8: Compass Robot

At all points in time, one foot of the robot is in contact with the ground, and the other foot is free to swing as part of a walking motion. The leg whose foot is in contact with the ground is called the support leg, while the leg whose foot is free to swing is called the non-support leg. When a step occurs during walking, the support leg and non-support leg switch instantaneously, so there is never a double support phase during which time both feet would be in contact with the ground.

The position of any point on the robot relative to the position of the support foot can be uniquely determined by the angle of the support leg, θ_s , and the angle of the non-support leg, θ_{ns}. As such, the state of the compass robot is defined in Equation 1 as q.

$$q = \begin{bmatrix} \theta_{ns} \\ \theta_s \end{bmatrix} \quad (1)$$

The inner leg angle, α, is also a useful value for analysis, but it is solely defined by θ_s and θ_{ns}. The movement of the robot can be represented by the angular velocity and angular acceleration of each leg. The angular velocity of the support leg is defined as $\dot{\theta}_s$, and the angular velocity of the non-support leg is defined as $\dot{\theta}_{ns}$. The angular acceleration of the support leg is defined as $\ddot{\theta}_s$, and the angular acceleration of the non-support leg is defined as $\ddot{\theta}_{ns}$.

The following sections derive the necessary equations to represent the compass robot, which includes position equations, energy equations, equations of motion, and transition equations.

2.2 Position Equations

In order to determine the position of the support foot (x_s, y_s), the position of the hip (x_h, y_h), and the position of the non-support foot (x_{ns}, y_{ns}), the dimensions of the legs were defined as shown in Figure 9.

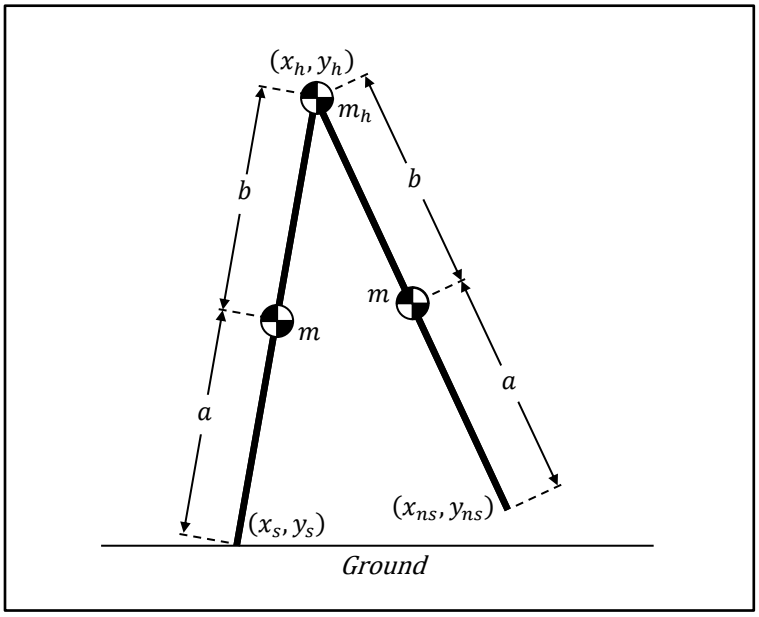

Figure 9: Compass Robot Dimensions

The mass of the robot was chosen to be a point mass, m, located in the middle of each leg with an additional point mass, m_h, located at the hip. As can be seen in the image above, the legs are identical in both length and mass. The length from the foot to the center of mass of each leg is a constant, a. The length from the hip to the center of mass of each leg is a constant, b. The overall length of the leg consists of both of these quantities as shown in Equation 2.

$$\ell = a + b \tag{2}$$

The horizontal position of the support foot, x_s, begins at zero but changes at the end of each step when the support leg and non-support leg switch. Since the support foot is always in contact with the ground, the height of the support foot, y_s , is always zero. As shown in Equation 3, the position of the hip can be defined relative to the position of the support foot as a function of θ_s.

$$(x_h, y_h) = (x_s - \ell \sin(\theta_s), y_s + \ell \cos(\theta_s)) \quad (3)$$

Similarly, as shown in Equation 4, the position of the non-support foot can be defined relative to the position of the hip as a function of θ_{ns}.

$$(x_{ns}, y_{ns}) = (x_h + \ell \sin(\theta_{ns}), y_h - \ell \cos(\theta_{ns})) \quad (4)$$

2.3 Energy Equations

The total energy of the robot consists of kinetic energy and potential energy. The kinetic energy is a function of the velocities of each mass, $\vec{v}_s$, $\vec{v}_h$, and $\vec{v}_{ns}$, which are shown in Figure 10.

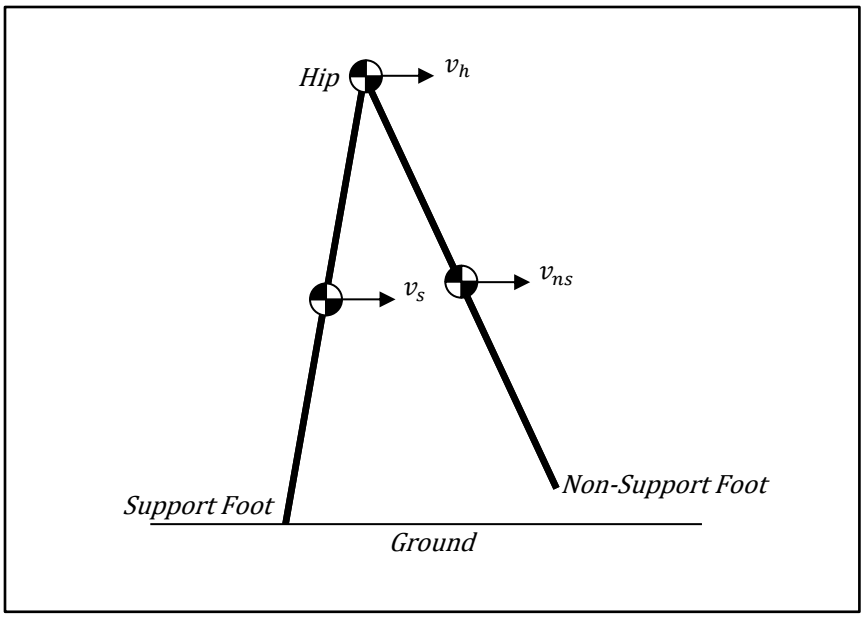

Figure 10: Compass Robot Velocities

According to *Engineering Mechanics: Dynamics*, the velocities of two points, A and B, on a rigid body in planar motion are related by Equation 5. Within this equation, ω_{AB} is the angular velocity of the rigid body, and $\vec{r}_{B/A}$ is a vector defining the location of point B relative to point A [27].

$$\vec{v}_B = \vec{v}_A + \omega_{AB}\, x\, \vec{r}_{B/A} \quad (5)$$

Using the equation above and the fact that the support foot is not moving, $\vec{v}_s$ can be calculated as a function of θ_s and $\dot{\theta}_s$. The equation for $\vec{v}_s$ is included in Equation 6.

$$\vec{v}_s = \dot{\theta}_s \hat{k}\, x\, (-a\sin(\theta_s)\hat{\imath} + a\cos(\theta_s)\hat{\jmath}) = -\dot{\theta}_s a\cos(\theta_s)\,\hat{\imath} - \dot{\theta}_s a\sin(\theta_s)\,\hat{\jmath} \quad (6)$$

The movement of the mass at the hip is completely determined by the movement of the support leg. Using the same method as above, $\vec{v}_h$ can be calculated as a function of θ_s and $\dot{\theta}_s$. The equation for $\vec{v}_h$ is included in Equation 7.

$$\vec{v}_h = \dot{\theta}_s \hat{k}\, x\, (-\ell \sin(\theta_s)\hat{\imath} + \ell \cos(\theta_s)\hat{\jmath}) = -\dot{\theta}_s \ell \cos(\theta_s)\, \hat{\imath} - \dot{\theta}_s \ell \sin(\theta_s)\, \hat{\jmath} \quad (7)$$

Again using the rigid body velocity equation, $\vec{v}_{ns}$ can be found relative to $\vec{v}_h$ as a function of θ_{ns} and $\dot{\theta}_{ns}$. The equation for $\vec{v}_{ns}$ is included in Equation 8. Unlike the other two velocities, $\vec{v}_{ns}$ uses length b, because this velocity is being defined relative to $\vec{v}_h$ rather than $\vec{v}_s$.

$$\begin{aligned} \vec{v}_{ns} &= \vec{v}_h + \dot{\theta}_{ns} \hat{k}\, x\, (b \sin(\theta_{ns})\hat{\imath} - b \cos(\theta_{ns})\hat{\jmath}) \\ &= \vec{v}_h + \dot{\theta}_{ns} b \cos(\theta_{ns})\, \hat{\imath} + \dot{\theta}_{ns} b \sin(\theta_{ns})\, \hat{\jmath} \end{aligned} \quad (8)$$

The equation for $\vec{v}_h$ found previously can be inserted into the $\vec{v}_{ns}$ equation to obtain Equation 9.

$$\begin{aligned} \vec{v}_{ns} = {} & \left(-\dot{\theta}_s \ell \cos(\theta_s) + \dot{\theta}_{ns} b \cos(\theta_{ns})\right) \hat{\imath} \\ & + \left(-\dot{\theta}_s \ell \sin(\theta_s) + \dot{\theta}_{ns} b \sin(\theta_{ns})\right) \hat{\jmath} \end{aligned} \quad (9)$$

The kinetic energy of an object is equal to one-half of its mass times its velocity squared. As shown in Equation 10, the total kinetic energy of the system is calculated as the sum of the kinetic energies of each mass in the system.

$$KE(q,\dot{q}) = \frac{1}{2}m\|\vec{v}_s\|^2 + \frac{1}{2}m_h\|\vec{v}_h\|^2 + \frac{1}{2}m\|\vec{v}_{ns}\|^2 \quad (10)$$

The potential energy of an object is equal to its mass times gravity times its height above a reference point. The reference point is generally defined with the ground at zero height. As with the total kinetic energy, the potential energies of each mass are computed and summed to find the potential energy of the entire system, shown in Equation 11.

$$\begin{aligned} PE(q) = mga\cos(\theta_s) + m_h g\ell\cos(\theta_s) \\ + mg(\ell\cos(\theta_s) - b\cos(\theta_{ns})) \end{aligned} \quad (11)$$

As shown in Equation 12, the total energy of the robot is the sum of these two energy components.

$$TE(q,\dot{q}) = KE(q,\dot{q}) + PE(q) \quad (12)$$

2.4 Equations of Motion

The compass robot is a simple planar double pendulum whose equations of motion can be derived using the Euler-Lagrange approach. This approach is well defined and commonly used in the study of dynamics. As such, the following derivation of the equations of motion was obtained from [28]. The Euler-Lagrange approach begins with defining the Lagrangian as the difference between the kinetic and potential energies of the system, see Equation 13.

$$Lagrangian: L(q, \dot{q}) = KE(q, \dot{q}) - PE(q) \quad (13)$$

As shown in Equation 14, the Lagrangian is then differentiated and set equal to the input torques of the system. If no input torques are applied at the joints and the system is passive, the right hand side of the equation becomes zero. If torques are applied at the joints making the system active, then the right hand side of the equation is u, with u being a vector of the input joint torques as defined in Equation 15.

$$\frac{d}{dt}\left(\frac{dL}{d\dot{q}}\right) - \frac{dL}{dq} = u \quad (14)$$

$$u = \begin{bmatrix} \tau_{ns} \\ \tau_s \end{bmatrix} \quad (15)$$

The torque on the support leg, τ_s, is the torque applied at the ankle. A torque cannot be applied at the ankle of the non-support leg, because the non-support leg is not in contact with the ground. Therefore, the torque on the non-support leg, τ_{ns}, is equivalent to applying a torque at the hip. Once Equation 14 has been computed, it can be rearranged into the form shown in Equation 16, with the matrices M, N, and G defined in Equations 17 through 19 respectively.

$$M(q)\ddot{q} + N(q,\dot{q}) + G(q) = u \tag{16}$$

$$M(q) = \begin{bmatrix} mb^2 & -m\ell b\cos(\theta_s - \theta_{ns}) \\ -m\ell b\cos(\theta_s - \theta_{ns}) & m_h\ell^2 + m(\ell^2 + a^2) \end{bmatrix} \tag{17}$$

$$N(q,\dot{q}) = \begin{bmatrix} 0 & m\ell b\dot{\theta}_s\sin(\theta_s - \theta_{ns}) \\ -m\ell b\dot{\theta}_{ns}\sin(\theta_s - \theta_{ns}) & 0 \end{bmatrix} \tag{18}$$

$$G(q) = \begin{bmatrix} mgb\sin(\theta_{ns}) \\ -\big(m_h\ell + m(a+\ell)\big)g\sin(\theta_s) \end{bmatrix} \tag{19}$$

Equation 16 can be further rearranged to solve for the angular accelerations of each leg as shown in Equation 20. It is useful to have the equation in this form, because if the angular acceleration equations are known, they can be integrated to find the angular velocities and angular positions of the system at any time.

$$\ddot{q} = -M^{-1}(q) * (N(q,\dot{q})\dot{q} + G(q) - u) \tag{20}$$

2.5 Transition Equations

When the non-support foot hits the ground triggering the completion of a step, the robot experiences a force due to impact with the ground. This impact causes a decrease in the kinetic energy of the robot, thereby causing a decrease in the total energy as well. The impact is considered to be instantaneous, and a double support phase during which both feet would be in contact with the ground is not considered. When the impact occurs, the support foot and non-support foot switch instantaneously, and the velocities of the legs change due to the energy lost during impact. The derivations of the equations for the change in the velocities during impact, called the transition equations, were obtained from [28].

Because the impact only occurs when both feet are on the ground making θ_s and θ_{ns} equal and opposite, the robot's position can be defined solely by the inner leg angle, α. The equation for α is included in Equation 21.

$$\alpha = (\theta_{ns} - \theta_s) \quad (21)$$

The velocities after impact can be calculated as a function of the velocities and angles before impact using the conservation of angular momentum. The post-impact state and the pre-impact state can be related by Equation 22, which can be rearranged to solve for the post-impact state as shown in Equation 23. The matrices Q^- and Q^+ are defined in Equations 24 and 25 respectively.

$$Q^+(\alpha)\dot{q}^+ = Q^-(\alpha)\dot{q}^- \quad (22)$$

$$\dot{q}^+ = {Q^+}^{-1}(\alpha)Q^-(\alpha)\dot{q}^- \quad (23)$$

$$Q^-(\alpha) = \begin{bmatrix} -mab & -mab + (m_h\ell^2 + 2ma\ell)\cos(\alpha) \\ 0 & -mab \end{bmatrix} \quad (24)$$

$$Q^+(\alpha) = \begin{bmatrix} mb(b - \ell\cos(\alpha)) & m\ell(\ell - b\cos(\alpha)) + ma^2 + m_h\ell^2 \\ mb^2 & -mb\ell\cos(\alpha) \end{bmatrix} \quad (25)$$

CHAPTER 3

MATLAB Model

3.1 Introduction

The first step in testing methods of push recovery for the compass robot was to create a computer model using MATLAB. In order to implement the model, realistic values for a bipedal robot needed to be selected for each parameter. The process for deciding these parameters is described in the next section. Following that, a discussion of the different scenarios that would make the simulation fail is included. The final portion of this chapter describes the different states of push recovery that were implemented during the simulation.

3.2 Determining System Parameters

3.2.1 Physical Parameters

In order to ensure that the simulation would be realistic, the compass model was modeled after HUBO-2, a humanoid robot sometimes referred to as Jaemi HUBO or KHR4. A picture of HUBO-2 is included in Figure 11.

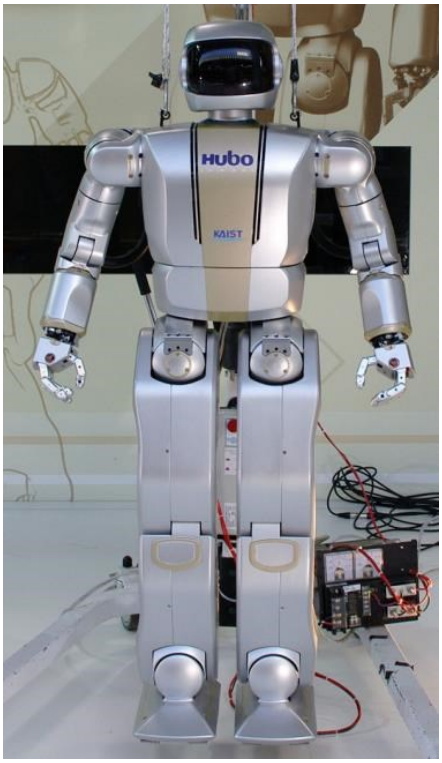

Figure 11: HUBO-2, A Humanoid Robot [29]

According to the HUBO-2 Manual [30], the length of HUBO-2's leg from the floor to the hip is equal to 695.38 mm. To approximate this dimension, the leg length in the model was chosen to be .7 m.

The overall mass of HUBO-2 is 45 kg [29]. In the model, this total mass was distributed as 10 kg in each of the legs and 25 kg located at the hip. The mass of each leg was represented as a point mass located in the center of the leg, so the constants a and b were both equal to one-half of the leg length as shown in Equation 26.

$$a = b = \frac{\ell}{2} \tag{26}$$

3.2.2 Initial Values

The simulation was started with the robot in a stationary position, which means that $\dot{\theta}_s$ and $\dot{\theta}_{ns}$ were both zero. It was assumed that people often stand with one foot slightly in front of the other while in a stationary position. When being pushed forward, the forward foot would be the support foot. Based on this assumption, the initial positions, θ_{ns} and θ_s, of the robot were assumed to be $-2°$ and $2°$ respectively.

3.2.3 Machine Limits

Due to physical limitations, humans have a maximum inner leg angle, α_{max}, that can be realistically achieved. When both feet are on the ground, α is a function of the stride length, d, as shown in Figure 12.

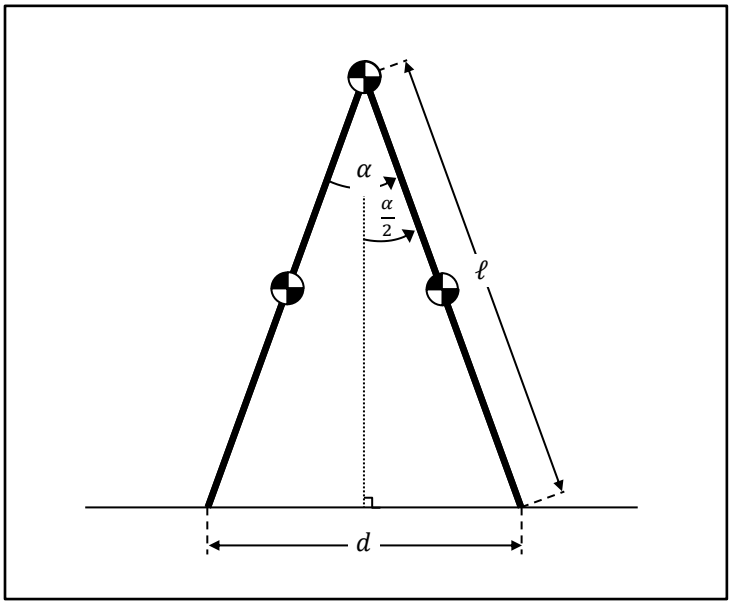

Figure 12: Relationship Between Stride Length and Inner Leg Angle

The equation relating the stride length to the inner leg angle is included in Equation 27.

$$d = 2\ell \sin\left(\frac{\alpha}{2}\right) \; or \; \alpha = 2\sin^{-1}\left(\frac{d}{2\ell}\right) \quad (27)$$

Due to the availability of information, the maximum stride length of HUBO was used to approximate the maximum stride length of HUBO-2. The stride length of HUBO can vary from 0 to 64 cm [31]. Using this information, α_{max} of HUBO-2 was approximated as $54.41°$. This calculation is included in Equation 28.

$$\alpha_{max} = 2\sin^{-1}\left(\frac{.64}{2 * .7}\right) = 54.41° \quad (28)$$

To be conservative, α_{max} was set as $50°$ in the model. The next parameter that needed to be defined was the maximum torque. As shown in Equation 29, torque can be computed as power divided by angular velocity.

$$\tau = \frac{p}{\omega} \quad (29)$$

It was assumed that when recording HUBO-2's maximum speed, HUBO-2 would be taking steps of the maximum stride length, d_{max}. By equating the time it takes the center of mass to travel linear distance d_{max} with the time it takes the non-support leg to travel angular distance α_{max}, the maximum angular velocity of the hip can be approximated. Equation 30 shows this approximation with v_{max} representing the maximum linear

velocity of the robot's center of mass and ω_{max} representing the maximum angular velocity of the robot's hip motor.

$$\omega_{max} = \alpha_{max} \frac{v_{max}}{d_{max}} \quad (30)$$

By combining Equations 27, 29, and 30, the equation for the maximum torque was approximated as shown in Equation 31.

$$\tau_{max} = \frac{p * 2\ell \sin\left(\frac{\alpha_{max}}{2}\right)}{\alpha_{max} * v} \quad (31)$$

Due to the availability of information, the hip motor power of HUBO was used to approximate the hip motor power of HUBO-2. HUBO's hip motor has a power of 90 watts in the pitch direction [31]. This power can be equivalently expressed in different units as shown in Equation 32.

$$p = 90\ watts = 90\ \frac{kg * m^2}{s^2} * \frac{rad}{s} \quad (32)$$

HUBO-2's linear walking speed ranges from zero to 1.4 km/hr [29]. To be conservative, a slightly slower speed of 1.25 km/hr was used. The maximum linear speed can be equivalently expressed in different units as shown in Equation 33.

$$v_{max} = \frac{1.25\ km}{hr} * \frac{1000\ meters}{1\ km} * \frac{1\ hr}{3600\ sec} = 0.3472\ \frac{meters}{sec} \quad (33)$$

By substituting in the appropriate values, the maximum torque was approximated as 175.75 Nm. This calculation is included in Equation 34.

$$\tau_{max} = \frac{90\ \frac{kg * m^2}{s^2} * \frac{rad}{s} * 2 * .7 \sin\left(\frac{.87266}{2}\right) \frac{m}{step}}{.87266\ \frac{rad}{s} * .3472\ \frac{m}{s}} = 175.7486\ Nm \quad (34)$$

Again, to be conservative, τ_{max} was set as 170 Nm in the model. During walking, people tend to keep their feet relatively close to the ground rather than performing an unnaturally high step. It was estimated that a normal human keeps their foot within 6 inches of the floor at all times while walking. Assuming a 6 foot tall person whose legs make up half of their height, the ratio of maximum foot lift to leg length was calculated as 16.67%. Using this ratio and a leg length of 0.7 m, the maximum height the foot could be lifted off the ground in the model was approximated as 0.12 m.

3.2.4 Push Parameters

When subjected to a small push, a robot would theoretically be able to apply a small opposing torque at its ankles, thereby restricting motion between its legs and feet. As with the hip motor power, the ankle motor power of HUBO was used to approximate the ankle motor power of HUBO-2. HUBO's ankle motor has a power of 90 watts in the pitch direction [31]. Because this is the same power as HUBO's hip, it was concluded that the maximum torque at the ankle would also be 170 Nm. Even if the robot was able to apply a torque capable of keeping its ankles rigid, it could still fall over by tipping about the front toes as shown in Figure 13.

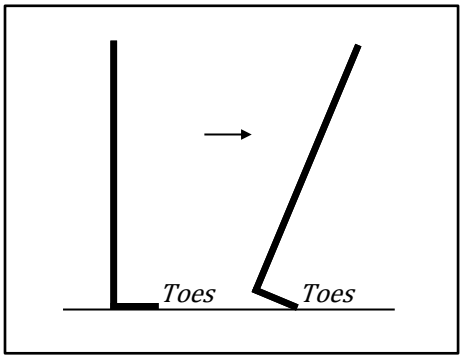

Figure 13: Tipping About the Front Toes when its Ankles are Rigid

Although the compass robot does not have feet, this calculation was performed in order to determine the range of pushes that the actual HUBO-2 robot would be able to recover from using only a torque at its ankles. HUBO-2's foot length, f, is 220 mm [30], which was approximated as .25 m. Figure 14 details the free-body diagram of the robot when the robot is rigid at its ankles and standing with both legs together.

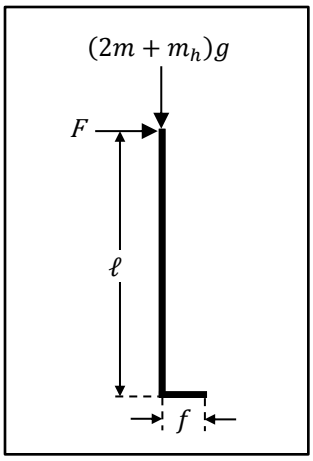

Figure 14: Free Body Diagram when Ankles are Rigid

The robot will tip if the moment from the push about the front toe is greater than the moment due to the weight about the front toe. Therefore, the point at which the robot will begin to tip is the point where the moments are exactly equal as expressed in Equation 35.

$$F\ell = (2m + m_h)gf \qquad (35)$$

Solving this equation for F and plugging in the relevant values, the force that it would take to tip the robot was calculated as 157.66 N. This calculation is included in Equation 36. By multiplying this force by a leg length of 0.7 m, it was determined that this push was equivalent to applying a torque of 100.36 Nm at the ankles. Based on this calculation, HUBO-2 should be able to withstand pushes with equivalent torques of up to 100.36 Nm just by applying a torque at the ankles.

$$F = \frac{(2m + m_h)gf}{\ell} = \frac{45 * 9.81 * .25}{.7} = 157.66\ N \quad (36)$$

Assuming the robot requires reactive stepping is a much safer option than assuming the ankle will be able to apply a sufficiently large torque to prevent falling. Since it is very important that HUBO-2, or other similar bipedal robots, do not fall over, a safety factor of approximately 1.5 was applied. This means that it was assumed the robot can only withstand pushes with equivalent torques of up to 60 Nm just by applying a torque at the ankles. Using the results of this calculation and arbitrarily picking an upper bound, it was decided to test the model for pushes with equivalent torques in the range of 60 Nm to 130 Nm. Lastly, it was assumed that a push applied to the robot would be brief, and as such, the duration of the push was chosen to be 0.2 seconds.

3.2.5 Walking Parameters

In the model, there were three parameters that affected the way a step was taken. The first parameter was the desired step size, $\alpha_{desired}$. As discussed previously, α_{max} was set equal to $50°$, so the range of $\alpha_{desired}$ in the model was defined as $-45°$ to $45°$. The maximum was set at $45°$ rather than the absolute maximum to allow for a slight overshoot without exceeding the maximum limit of the system. Negative $\alpha_{desired}$ represented the situation when the non-support foot was behind the support foot at the time of the step, and positive $\alpha_{desired}$ represented the opposite orientation.

The next parameter was the push-off constant. At the beginning of a step, humans push off of the ground with their back leg. In reality, this force is a linear force applied on the bottom of the foot, but it was represented in the model as a torque on the support leg. This torque should only be applied for a short duration, so it was assumed to be 0.1 seconds.

The last parameter was the swing constant. During walking, a torque must be applied at the hip in order to swing the leg to the desired position. As shown in Equation 37, this torque was applied using proportional control on the inner leg angle. It was determined experimentally that all pushes within the set range could be successfully recovered from with a swing constant in the range of 0 to 180 on the first step and a swing constant equal to 60 on all other steps.

$$\tau_{ns} = SwingConstant * (\alpha - \alpha_{desired}) \quad (37)$$

During times when α was to be held constant, the non-support leg needed to change its angle at the same rate as the support leg. To accomplish this, a torque proportional to the difference between the angular velocities of the two legs was applied at the hip. The equation for this torque is included in Equation 38. The constant, k, was determined using trial and error by noting if the chosen value resulted in α remaining constant for a certain period of time. Using this method, it was determined that k should be set as 500.

$$\tau_{ns} = k * (\dot{\theta}_s - \dot{\theta}_{ns}) \quad (38)$$

3.3 Invalid Situations

During the simulation, invalid situations could occur very easily by exceeding a machine limit of the robot, taking a step of the incorrect size, moving unnaturally, or falling over. It should be noted that most situations could only occur when the model was in a certain state. The following sections describe each of the invalid situations in detail.

3.3.1 Exceeding Machine Limits

In modeling a physical robot, there were machine limits that had to be accounted for in the simulation. For example, each motor on the robot had a maximum torque that it was capable of applying. τ_s was an input to the system that was always defined such that it was less than τ_{max}. As a result, τ_s did not need to be checked for validity. τ_{ns} was determined by proportional control, so the robot could have tried to apply τ_{ns} such that it exceeded τ_{max}. The maximum torque was exceeded and the simulation became invalid if the condition in Equation 39 became true.

$$\tau_{ns} \geq \tau_{max} \quad (39)$$

Due to flexibility limitations, humans can only take steps of a limited size. To ensure that the model only performed motions similar to a human being pushed, the simulation became invalid if the condition in Equation 40 became true.

$$\alpha > \alpha_{max} \quad (40)$$

3.3.2 Incorrect Step Size

If the robot attempted to take a step of a specific size but took a step of another size instead, the simulation became invalid, because it did not follow the desired motion. This would occur if the robot was incapable of reaching $\alpha_{desired}$ given its current state or if the robot reached $\alpha_{desired}$ but was incapable of holding α constant until the foot reached the ground for the step. For example, take the situation where the robot attempted to take a step of size A but instead took a step of size B. This scenario would be stored the same as the scenario where the robot attempted to take a step of size B and achieved the desired step size.

To eliminate this ambiguity, a step was only valid if it was the intended step size. As such, the simulation became invalid if the condition in Equation 41 became true. The first part of the condition is the requirement for taking a step while the second part of the condition enforces that α at the time of the step must be within a certain tolerance of $\alpha_{desired}$.

$$y_{ns} = 0 \;\&\&\; |\alpha_{desired} - \alpha| \geq \alpha_{tolerance} \quad (41)$$

3.3.3 Moving Unnaturally

When walking naturally, a person tends to keep their non-support foot close to the ground rather than raise it up unnaturally high as shown in Figure 15.

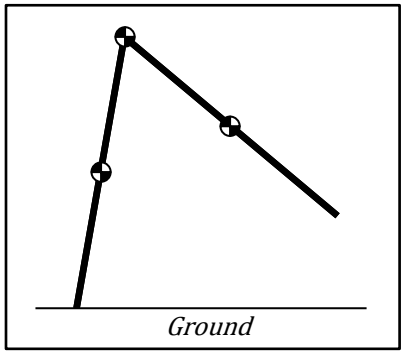

Figure 15: Unnaturally High Step

To enforce this, if y_{ns} ever went above a certain threshold as shown in Equation 42, the simulation became invalid.

$$y_{ns} \geq y_{max} \quad (42)$$

While walking, a human attempts to be efficient and therefore would swing their leg directly to the position they want. For example, the motion in Figure 16 would be unnatural as a human would not initially swing their leg backwards if taking a forward step.

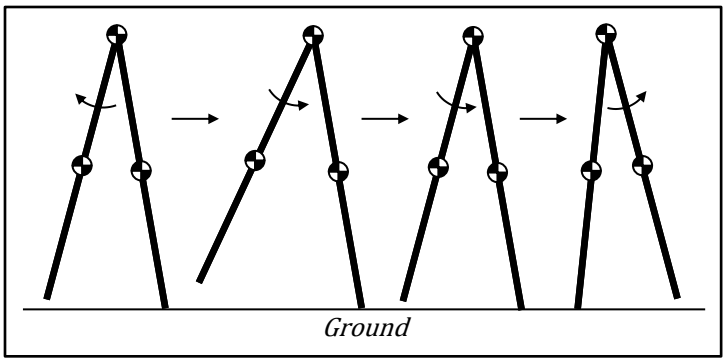

Figure 16: Unnatural Swinging of the Leg in the Opposite Direction of the Step

Similarly, a human would not initially swing their leg forward if taking a backward step. To enforce this, the simulation became invalid if $\dot{\theta}_{ns}$ changed direction while the leg was swinging. It was determined that $\dot{\theta}_{ns}$ was changing directions if the condition in Equation 43 was true.

$$\dot{\theta}_{ns} = 0 \tag{43}$$

Another type of step that would be unnatural for a human is a step where both the support foot and the non-support foot are in front of the hip as shown in Figure 17.

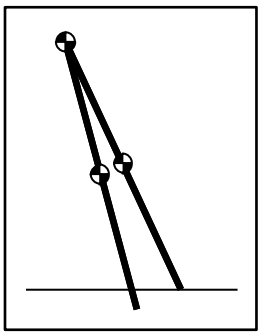

Figure 17: Unnatural Position with Both Feet in Front of the Hip

To enforce that the robot would not choose to walk in this unnatural manner, the simulation became invalid if the condition in Equation 44 became true.

$$x_{ns} > x_h \;\&\&\; x_s > x_h \qquad (44)$$

When taking a step, the support leg of the human moves in the direction of the step as shown in Figure 18.

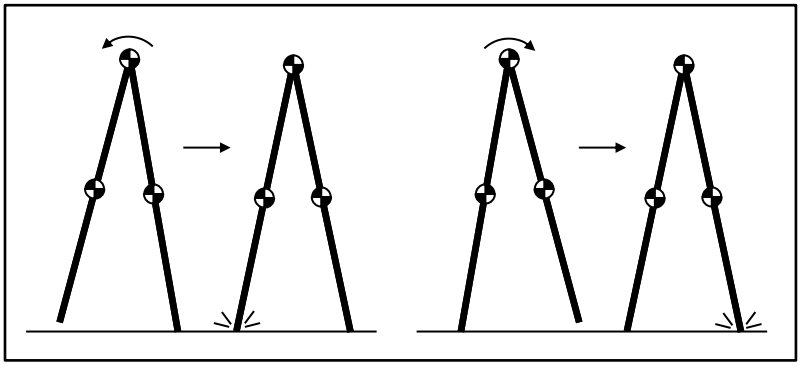

Figure 18: Support Leg Moving in the Direction of the Step

In the model, there were situations in which this was not the case, so the simulation became invalid whenever the condition in Equation 45 became true. The first part of the condition determined if the robot was taking a step. The second part of the condition determined if the support leg was moving in the direction of the step. Within the condition, the angular velocity was compared to 0.01 rather than 0, because it was assumed that a leg with such a small velocity would have been moving slowly enough to still be considered valid.

$$y_{ns} = 0 \;\&\&\; \Big(\big(\theta_s > 0 \;\&\&\; \dot{\theta}_s < 0.01\big) \;||\; \big(\theta_s < 0 \;\&\&\; \dot{\theta}_s > 0.01\big) \Big) \qquad (45)$$

3.3.4 Falling Over

Since the goal was for the robot to remain standing, the simulation became invalid if the robot fell over. It was determined that the robot would fall over if $|\theta_s|$ became larger than half of α_{max}. If $|\theta_s|$ was greater than half of α_{max}, then the non-support foot of the robot was already below ground, and there were only two ways in which the foot could have been brought above ground to take a valid step. The first way was for the robot to apply an opposing torque to the ankle of the support leg as shown in Figure 19.

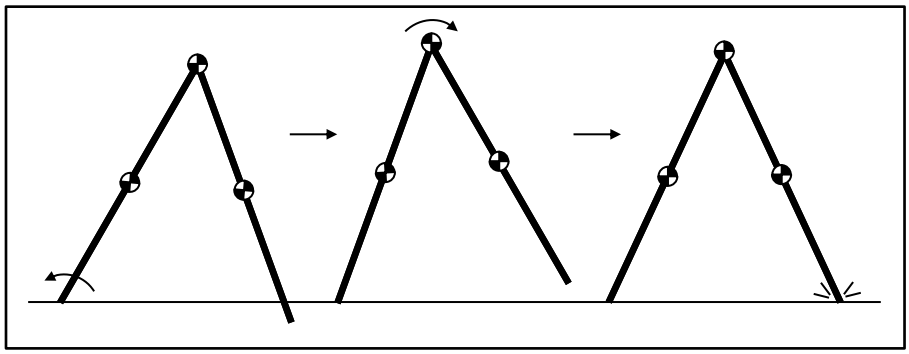

Figure 19: Applying an Opposing Ankle Torque to Take a Valid Step

Applying this ankle torque would have been unnatural in reference to normal human motion. The second way was for the robot to swing the non-support foot above ground to take a proper step as shown in Figure 20.

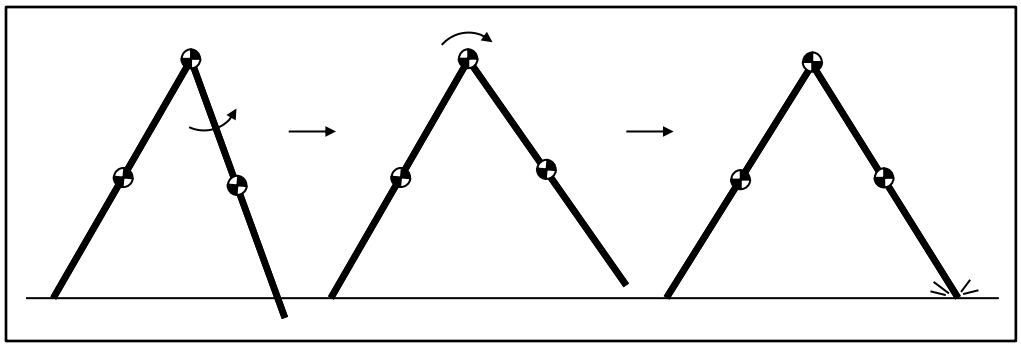

Figure 20: Swinging the Non-Support Foot above Ground to Take a Valid Step

However, if $|\theta_s|$ was greater than half of α_{max}, this would have resulted in a violation of the α_{max} criterion defined previously. Therefore, if the condition in Equation 46 became true, the robot was considered to have fallen over, and the simulation was considered invalid.

$$|\theta_s| \geq \frac{\alpha_{max}}{2} \quad (46)$$

3.4 States

The compass robot simulation, whose motions are summarized in Figure 21, began with a robot at rest being subjected to an external push. The robot then took a step of a certain size and speed. Next, it pushed off the ground with its back foot in order to take another step of a new size and speed. The robot continued to take steps in this manner until it came to rest.

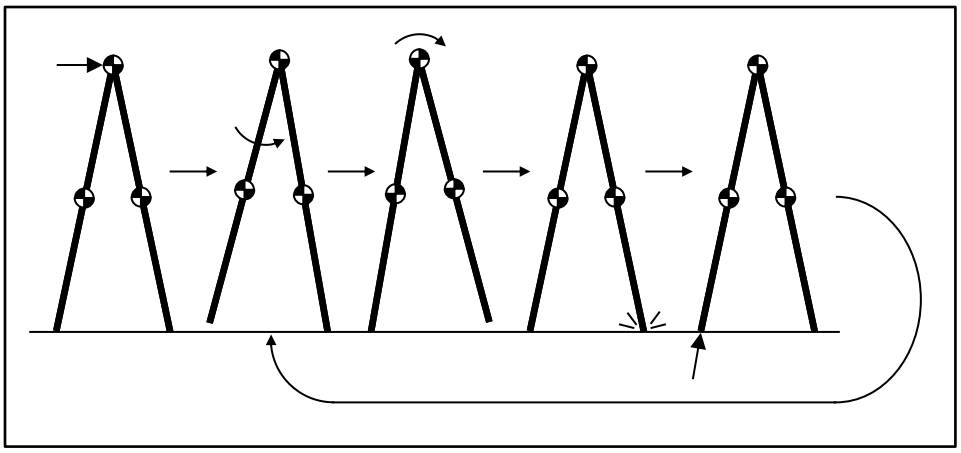

Figure 21: Overview of Compass Robot Simulation Motions

It should be noted that while only forward steps are shown in the sequence above, the robot could have chosen to take a step in which the non-support foot remains behind the support foot. This primarily happened when the robot was close to being stopped and began teetering back and forth while quickly changing support legs.

The overall process was split up into several different states. The state diagram for the system is included as Figure 22. With the exception of the stopped state, invalid state, and transition state, each state used MATLAB's *ode45* function to solve the compass robot equations of motion. Event detection was used to accurately stop the simulation of the current state when any of the events shown on the state diagram occurred. Each of these states and events are explained in further detail in the following sections.

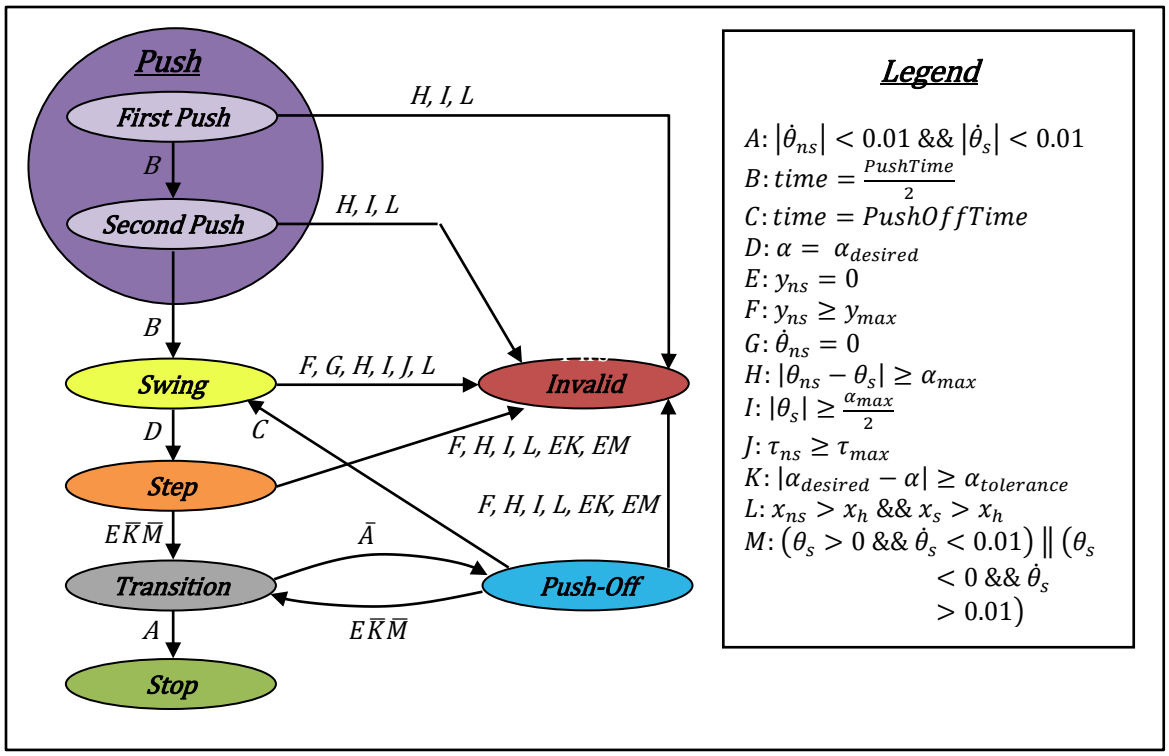

Figure 22: Compass Robot Simulation State Diagram

3.4.1 Push State

The simulation began in the push state when the robot at rest was subjected to an external force. In the model, a push applied at the hip was represented as an equivalent torque applied to the support leg. The push was assumed to begin with a small magnitude force which increased as full contact was made and then decreased again as the object moved away from the person and contact weakened. To represent this, the equivalent torque increased from zero to some chosen value linearly in time, and it then decreased back to zero linearly in the same amount of time as shown in Figure 23.

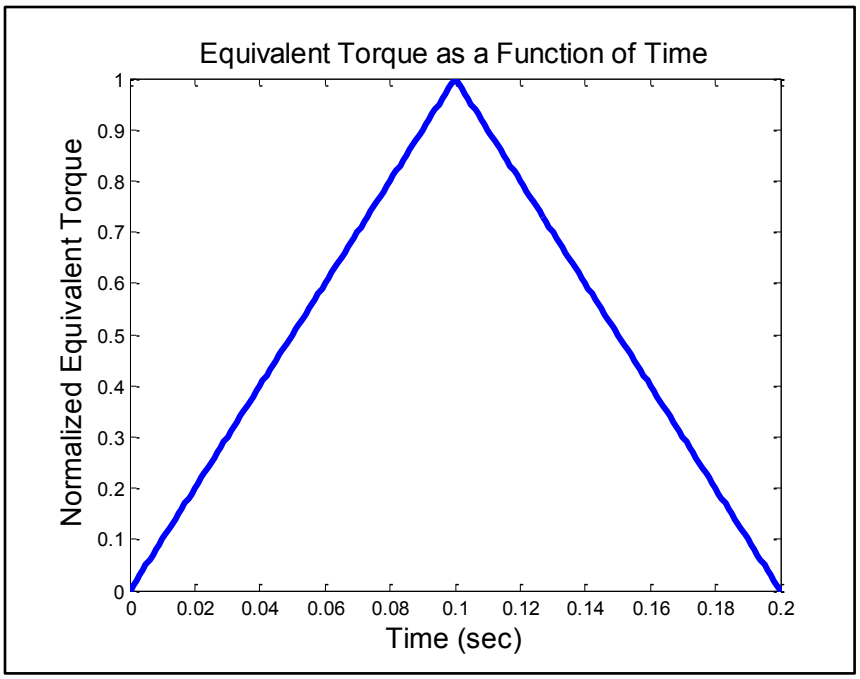

Figure 23: Equivalent Torque as a Function of Time

Because of this assumption, the push state could easily be split into two sub-states. The first push state encompassed the system while the torque was increasing, and the second push state encompassed the system while the torque was decreasing. These sub-states are further explained in the following sections.

3.4.1.1 First Push State

The first push state corresponds to the first half of the push when the applied torque was increasing. During the first push state, the torque on the support leg was applied according to Equation 47.

$$\tau_s = -\frac{Equivalent\ Torque}{Push\ Time/2} * time \quad (47)$$

It was assumed that when pushed, a person would naturally apply a torque at their hip that keeps α constant for a brief period of time. The angle remained constant as long as the angular velocities of the two legs were kept equal. The velocities of the two legs were kept equal by applying a torque at the hip using the proportional control shown in Equation 48.

$$\tau_{ns} = k * \left(\dot{\theta}_s - \dot{\theta}_{ns}\right) \quad (48)$$

The first push state lasted for half of the total push time unless the simulation became invalid first. When in the first push state, the simulation could have become invalid through conditions H, I, and L as defined in Figure 22. During the first push state, the y_{max} limit was not enforced, because the height of the non-support foot was caused by the push rather than the robot consciously lifting the foot too high. There was also no τ_{max} limit enforced during this state, because the current angle was just being maintained. In an actual mechanical system, it is often difficult to back drive a motor, so the motor should be able to maintain the current angle without exceeding its torque limitations. While using proportional control to maintain α, $\dot{\theta}_{ns}$ was allowed to change directions as needed without triggering the invalid condition discussed previously. Once the simulation completed the first push state with no invalid conditions, the simulation entered the second push state.

3.4.1.2 Second Push State

The second push state was entered if the first push state reached its time limit without the simulation becoming invalid. The second push state was used to implement the second half of the push when the applied torque was decreasing. During the second push state, the torque on the support leg was applied according to Equation 49.

$$\tau_s = \frac{Equivalent\ Torque}{Push\ Time/2} * time - 2 * Equivalent\ Torque \quad (49)$$

It was assumed that when humans are pushed, α is only held constant for a small period of time compared to the total push time. Therefore, α was only kept constant during the first push state. This means that during the second push state, τ_{ns} was zero.

Similar to the first push state, the second push state ended when it reached half of the total push time or an invalid condition. The push time was the same as in the first push state, and the invalid states were the same as well. Once the simulation completed the second push state with no invalid conditions, it entered the swing state.

3.4.2 Swing State

The simulation entered the swing state after reaching the time limit during the second push state or the push-off state, which will be discussed shortly. It was assumed that during the swing state, the support leg should only move as a result of momentum, so τ_s was set equal to zero. Simultaneously, the non-support leg was moved to the desired position by applying a torque at the hip. This torque was implemented such that it started with a large magnitude, which was reduced as the leg approached the desired position. The hip torque, shown in Equation 50, used proportional control and varied in magnitude according to the swing constant, which was chosen and idealized for each step.

$$\tau_{ns} = Swing\ Constant * (\alpha - \alpha_{desired}) \quad (50)$$

The desired step size, $\alpha_{desired}$, was another parameter of the system that was chosen and idealized for each step. Since the robot was knee-less and the legs were the same lengths, the non-support foot needed to pass beneath the ground in order to take a forward step. Therefore, the swing state did not end when the non-support foot contacted the ground. This phenomenon is generally considered acceptable, because in physical compass robots like the one shown in Figure 24, the leg is often shortened or folded sideways during the swing state so that the robot can walk properly without the foot passing below the ground [32].

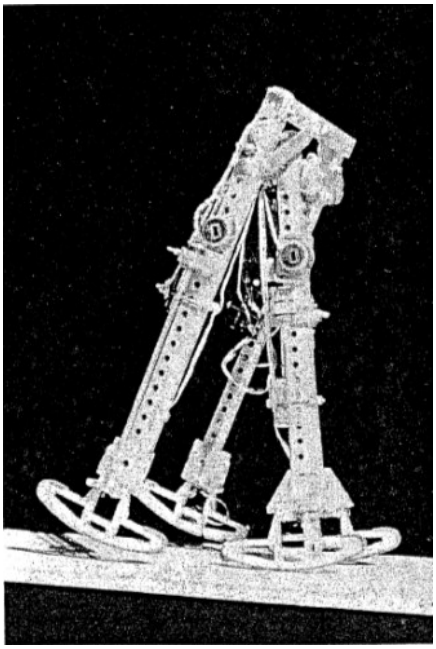

Figure 24: Physical Compass Robot [32]

Additionally, if the robot took a step before $\alpha_{desired}$ was reached, the simulation became invalid due to the invalid step size condition. Therefore, the swing state only ended if α became equal to $\alpha_{desired}$ or the simulation became invalid. While in the swing state, the simulation could have become invalid through conditions F, G, H, I, J, and L as defined in Figure 22. If α reached $\alpha_{desired}$ and no invalid conditions were true, then the robot proceeded to the step state in order to complete its current step.

3.4.3 Step State

The step state was entered after reaching $\alpha_{desired}$ during the swing state. It was assumed that during the step state, the support leg should only move as a result of momentum, so τ_s was set equal to zero. Also during the step state, α was held constant at $\alpha_{desired}$ until

the non-support foot made contact with the ground. The angle was held constant by applying a hip torque using proportional control on the angular velocities of the legs as shown in Equation 51.

$$\tau_{ns} = k * (\dot{\theta}_s - \dot{\theta}_{ns}) \quad (51)$$

The step state ended when the height of the non-support foot became zero from a position above ground. This was considered the end of the robot's current step. The step state also could have ended with the simulation becoming invalid through conditions F, H, I, L, EK, and EM as defined in Figure 22. If the robot's current step ended and no invalid conditions were true, then the robot proceeded to the transition state.

3.4.4 Transition State

Once the robot completed a valid step during either the step state or push-off state, the system entered the transition state. The first step of the transition state was to correct the small error usually associated with *ode45* event detection. Before entering the transition state, the *ode45* solver detected that y_{ns} was zero. However, due to precision limitations while solving, y_{ns} was generally not zero exactly. While the inaccuracies were very small, chosen to be on the order of 10^{-9}, they did have an effect on the system. Because a double support phase was not being considered, the robot experienced several almost instantaneous steps teetering back and forth between the two feet when it was almost stopped. In this situation, if one of the feet started below zero due to the small error, then

the robot would fall rather than taking the next immediate step. As a result, the model had to ensure after every step that y_{ns} was exactly zero. While correcting the small y_{ns} error, α was kept constant, and the robot was rotated slightly about the support foot as shown in Figure 25.

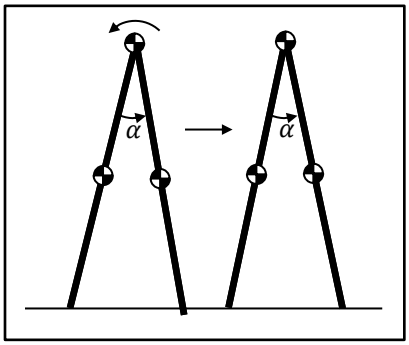

Figure 25: Correcting Small Inaccuracies in the Height of the Foot

To perform this correction, θ_s and θ_{ns} were adjusted to be equal and opposite with magnitudes of $\frac{\alpha}{2}$, and the location of the support foot was not changed. Based on these values, the new locations of the hip and non-support foot were calculated according to Equations 52 and 53 respectively. Adjustments were only made to the geometry of the robot, so the angular velocities of the legs were not changed. Once the correction was performed, the system proceeded with the rest of the transition state.

$$(x_h, y_h) = \left(x_s - \ell \sin\left(-\frac{\alpha}{2}\right), y_s + \ell \cos\left(-\frac{\alpha}{2}\right)\right) \quad (52)$$

$$(x_{ns}, y_{ns}) = \left(x_s - \ell \sin\left(-\frac{\alpha}{2}\right) + \ell \sin\left(\frac{\alpha}{2}\right), \right.$$
$$\left. y_s + \ell \cos\left(-\frac{\alpha}{2}\right) - \ell \cos\left(\frac{\alpha}{2}\right)\right) \quad (53)$$

During the transition state, the energy of the system was reduced due to the impact of the non-support foot with the ground. Using the transition equations derived previously, the new angular velocities were calculated based on the positions and angular velocities immediately before impact. The transition was considered instantaneous, so the new angular velocities were used as the initial conditions of the next state. During the transition state, the definition of the support leg and non-support leg were also switched.

If the new angular velocities were both below a set threshold, then the robot was considered to have stopped. A threshold value was necessary, because the velocities of the robot would never permanently become and remain zero without a double support phase. If the angular velocity of either leg was above the designated threshold, the system entered the push-off state to begin taking the next step.

3.4.5 Push-Off State

The push-off state followed the transition state in situations where the robot did not stop. Therefore, this state occurred at the beginning of each step except for the initial step after the robot was pushed. At the beginning of a step, a human pushes off the ground with their non-support foot in order to gain the necessary momentum for the step. The push-

off action was represented in the model by applying a torque at the ankle of the support foot while holding α constant. This torque could vary in magnitude and was a parameter of the system that was chosen and idealized for each step. A torque at the hip, as detailed in Equation 54, was applied using proportional control on the angular velocities to hold α constant.

$$\tau_{ns} = k * \left(\dot{\theta}_s - \dot{\theta}_{ns}\right) \quad (54)$$

The push-off state ended when the push-off time limit was reached, the step was completed, or the simulation was determined to be invalid. A step was considered complete if the height of the non-support foot became zero from above ground. Unlike the swing state, steps could be completed during the push-off state, because there was no need for the non-support leg to swing underground. Completing steps during the push-off state was actually very common, because when the robot neared stopping, it often teetered back and forth taking nearly immediate steps that occurred within the push-off time limit. The invalid states that could have been reached from the push-off state were F, H, I, L, EK, and EM as defined in Figure 22.

If the push-off time limit was reached, then the robot proceeded to the swing state. If the step was completed and no invalid conditions were true, then the robot proceeded to the transition state.

3.4.6 Stopped State

The stopped state followed the transition state in situations where the robot stopped. Once the robot reached the stopped state, it had successfully recovered from the push, and the simulation was ended.

CHAPTER 4

Energy Reduction Method

4.1 Introduction

When humans are pushed, they will naturally take strategically placed steps in order to return to a stable position. While this is easy for humans, a robot does not intuitively know what step to take. Due to the complex dynamics of the system and the fact that it may take more than one step to fully recover, it is difficult to calculate the ideal step that a robot should take in order to recover from a push.

When a robot is stationary, the kinetic energy of the system will be zero. Due to this fact, it was decided that the goal of push recovery should be for the robot to reduce its kinetic energy to zero. While walking, the robot's kinetic energy is reduced each time the foot impacts the ground. It was originally hypothesized that the robot should choose to take the step that results in the highest reduction of kinetic energy. If performed on every step, this would eventually result in the robot reducing its kinetic energy to zero, thereby successfully recovering from the push. The next sections detail how energy reduction should be defined and investigate whether total energy or kinetic energy should be used when implementing the reduction.

4.2 Reduction in Energy

Applying the transition equations caused immediate changes to the kinetic energy of the robot. While taking a step, energy was added to the system during the push-off and swing states. In order to ensure the maximum reduction in energy, the step with the largest net change in energy across the entire step was the correct choice, not necessarily the step with the largest net change across the transition equations. For example, take the scenario where step A adds 5 units of energy to the system with a reduction of 2 through the impact, and step B adds 10 units of energy to the system with a reduction of 5 through the impact. Step A results in a system with 3 units of energy, and step B results in a system with 5 units of energy. This example shows that step A should be chosen even though the net change in energy across the transition equations is greater in step B.

4.3 Kinetic Energy versus Total Energy

Once this theory was implemented, it was investigated whether the method should use kinetic energy or total energy to choose the best step. When the robot began a step with a small α as shown in Figure 26, only a small amount of kinetic energy was converted to potential energy as the height of each mass relative to the ground was only slightly increased.

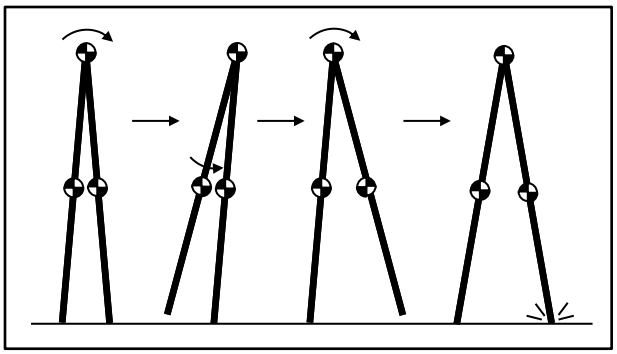

Figure 26: Motion when Starting with a Small α

This means that when the support leg became vertical, it most likely had kinetic energy remaining and continued moving in the forward direction. When this was the case, the robot had to take a forward step in order to avoid falling over. In order to take a forward step, a torque was applied to the non-support leg, thereby adding energy to the system. In comparison, if the robot had a slightly larger kinetic energy but was in the starting configuration shown in Figure 27, it would have a smaller potential energy and could have a smaller total energy. If the total energy was smaller, the robot moved forward, reached the point where all the kinetic energy had been converted to potential energy, and then fell slowly backward due to gravity.

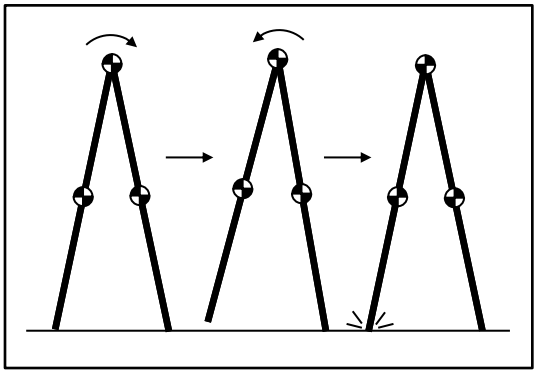

Figure 27: Motion When Starting with a Large α

When falling backward, very little if any torque needed to be applied to the non-support leg, so very little if any energy was added to the system. When compared to the first configuration, the second configuration began with a larger kinetic energy and a smaller total energy. Since the step required very little energy to be added, the second configuration ended with a smaller kinetic energy and a smaller total energy than the first configuration. Based on these observations, it was decided that a robot should choose the step that results in the lowest total energy rather than the lowest kinetic energy.

Note that the lowest possible total energy for a robot is when the kinetic energy is zero and the potential energy is at a minimum. The lowest potential energy of the compass robot occurs when each of the masses is as close to the ground as possible, which requires α to be as large as possible. However, within the model, α was limited to a maximum of $50°$, so the lowest possible potential energy would be as presented in Equation 55.

$$PE = mga\cos(-25°) + m_h g\ell\cos(-25°) + mg(\ell\cos(-25°) - b\cos(25°)) \quad (55)$$

4.4 Feasibility of the Energy Reduction Method

To test the Energy Reduction Method, a push with an equivalent torque of 130 Nm was applied, and the robot responded by taking a step with a specified push-off constant, swing constant, and $\alpha_{desired}$. The same push was applied for each of the possible

parameter combinations, and the total energy after the transition was recorded. Once all the combinations had been run, the combination that resulted in the lowest total energy was chosen as the best first step. Then, using the ending angular velocities and positions from the best first step as the initial values of the second step, the same process was followed to find the best second step. Once the best second step was determined, the best third step was determined and so on until the robot came to a stop or became invalid. The results in Table 1 show for a push with an equivalent torque of 130 Nm, the robot was able to come to a stop using the Energy Reduction Method.

Step #	θ_{ns}°	θ_s°	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-13.1	-5.1	-167	-101.9	0	180	45	240.2
2	-23.2	23.2	31.9	-75.9	0	60	-25	235.1
3	12.2	-12.2	0.2	0.5	STOPPED			

Table 1: Energy Reduction Method Results for a Push with an Equivalent Torque of 130 Nm

Once the best step was found for a push with an equivalent torque of 130 Nm, the process was repeated to find the best step for pushes with equivalent torques ranging from 60 Nm to 130 Nm counting by 10 Nm. Each of these pushes successfully came to a stop, and the results are included in Appendix A as Tables 2 through 8 respectively. Since the robot was able to come to a stop after each of the pushes within the defined range, the Energy Reduction Method was shown to be a feasible method for push recovery.

CHAPTER 5

Implementation of the Energy Reduction Method

5.1 Introduction

While the Energy Reduction Method was shown to be feasible in the previous section, a falling robot cannot try all types of steps in order to determine the best step. A falling robot only gets one shot at push recovery, so it needs to be able to compute the one best step to take. Simply storing the ideal step sequence for each push was not a practical option for several reasons. If the robot was subjected to a push between one of the stored values, then it would not have a sequence stored to know how to proceed. Another reason is if there were any errors in the first step due to natural mechanical variations, then the pre-computed second step would not be very accurate. Any slight inaccuracies on the first step or two could have a major impact on later steps as the errors would increase with each step.

Rather than storing the ideal step sequence for each push, a table was generated in MATLAB that contained the best step for any starting configuration. This table will be referred to as the Best Step Lookup Table. A starting configuration consisted of $\theta_{ns}, \theta_s, \dot{\theta}_{ns},$ and $\dot{\theta}_s$. Since the robot would have no prior knowledge of the magnitude or duration of the push, starting configurations for the first step were defined immediately

after the push ended. Starting configurations for all subsequent steps were defined immediately after the transition state. This table eliminated the difficulties associated with an unknown push magnitude and reduced the effect of slight inaccuracies due to natural mechanical variations. The following sections describe how the data for this table was generated and how the robot could interpolate within this table.

5.2 Data Generation

For each starting configuration, the MATLAB simulation was run for one step using all possible parameter combinations, and the results of those steps were recorded for selection of the best step. To decide on the range of starting configurations to run through the simulation, the results from the Feasibility of the Energy Reduction Method section were studied. The feasibility data was separated into three groups: first steps, non-first steps with θ_{ns} greater than zero, and non-first steps with θ_{ns} less than zero. The data was separated into these three categories, because data ranges were very different for each of these three categories.

The biggest difference between the sets was that non-first steps began with both feet on the ground, so θ_{ns} was always equal and opposite θ_s. In contrast, during first steps, the robot decided how to step after the end of the push, so one of the feet was already off of the ground. Also during first steps, the robot did not push off of the ground, so the push-off constant was always zero. Another difference was that if θ_{ns} was greater than zero, then the robot had to take a full step backward in order to reach a negative $\alpha_{desired}$.

Since the robot would never need to take a full step backward in order to recover from a forward push, these situations were not considered. Lastly, because the robot was always being pushed forward, first steps always began with negative values for θ_{ns} and θ_s

The minimum and maximum of each variable was calculated for every category. For the first steps, θ_{ns} and θ_s were incremented by $1.25°$, and $\dot{\theta}_{ns}$ and $\dot{\theta}_s$ were incremented by approximately $1\ \frac{deg}{sec}$. The swing constant was incremented by 10, and the push-off constant was always 0. Within the Energy Reduction Method results, $\alpha_{desired}$ was always chosen to be $45°$ on the first step, so no increment was necessary for $\alpha_{desired}$. For the non-first steps, θ_{ns} and θ_s were incremented by $0.625°$, and $\dot{\theta}_{ns}$ and $\dot{\theta}_s$ were incremented by approximately $0.5\ \frac{deg}{sec}$. The push-off constant was incremented by 2.5 Nm, and $\alpha_{desired}$ was incremented by $0.625°$. For the non-first steps, it was discovered during testing that a swing constant of 60 was sufficient for all pushes within the defined range. Defining the swing constant as 60 eliminated one parameter, thereby making it easier to gather the training data.

The simulation was run with every combination of the seven different variables ranging from their respective minimums to their respective maximums and counting by their respective increments. After removing any data from simulations that resulted in invalid steps, the first set had 1,520,034 data points. Similarly, the second set had 1,974,423 data points, and the third set had 27,234,664 data points.

Once all the data had been generated, the best step was chosen for each unique combination of θ_{ns}, θ_s, $\dot{\theta}_{ns}$, and $\dot{\theta}_s$ using the Energy Reduction Method. If multiple steps resulted in the same total energy, the step with the lowest push-off constant was favored as this would require the robot's motors to do less work. If there was a tie between multiple steps with the same push-off constant, the step with the lowest swing constant was favored. If both the swing constant and the push-off constant were the same, then the step with the smallest $\alpha_{desired}$ was favored as this would result in a more natural standing position. After applying the Energy Reduction Method, the first set had 262,665 data points. Similarly, the second set had 21,538 data points, and third set had 148,341 data points.

Once the best step had been chosen for each starting configuration, the data was stored in the Best Step Lookup Table. Since θ_{ns}, θ_s, $\dot{\theta}_{ns}$, and $\dot{\theta}_s$ are continuous rather than discrete variables, it was impossible to store every possible starting configuration in the table. If the robot found itself in one of the starting configurations that was not in the table, it would have to interpolate between values in the table to find the best step. For this project, four different interpolation methods were implemented and compared against each other. The four methods, further described in the following sections, were linear interpolation, nearest interpolation, linear least squares, and neural networks.

5.3 Linear Interpolation

The first method used to calculate the best step given the Best Step Lookup Table was linear interpolation. Linear interpolation can be explained very simply in the 2D case. In the 2D case, the lookup table has a set of y output values and each output's corresponding x input value. When computing the output y_2 for a given input x_2 that is not actually in the lookup table, the closest points, x_1 and x_3, on either side of x_2 are found in the table instead. A line is drawn between (x_1, y_1) and (x_3, y_3), and the output value y_2 is the value of the line at position x_2. A pictorial representation of this process is shown in Figure 28. In this example, six data points were created using the equation y=x, which is represented by the blue dotted line. A small amount of random noise was added to each of the data points, which were plotted as black dots. The values between these data points were generated using linear interpolation and are shown using the solid black line.

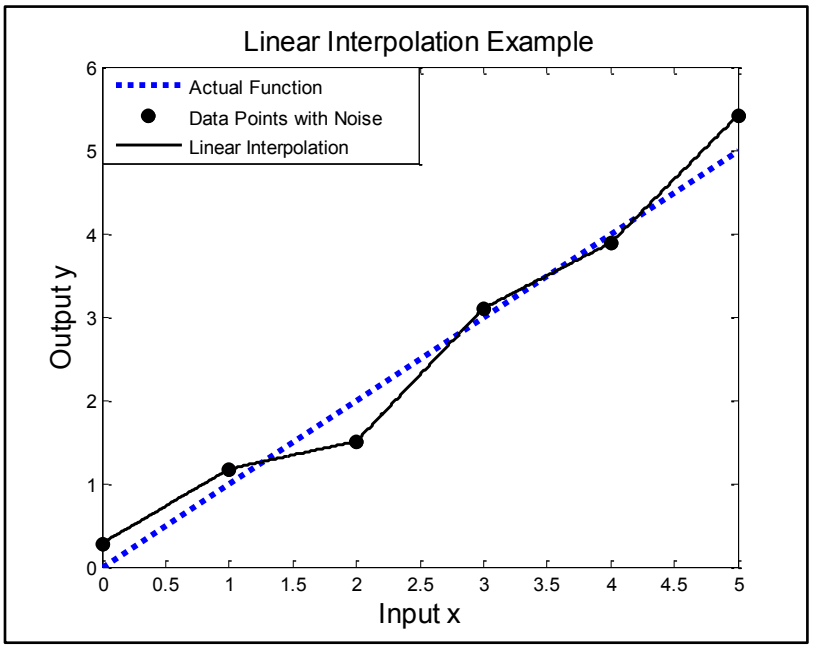

Figure 28: Linear Interpolation Example

This concept can be extended to multidimensional tables such as the Best Step Lookup Table, which has 4 inputs making it a 5D interpolation problem. Since the simulation was implemented in MATLAB, the *griddatan* function was used to perform the linear interpolation.

Linear interpolation, while simple, has many drawbacks. This method is typically slow as the entire data table must be searched to find the points closest to the point of interest. It also uses a large amount of memory as the entire data table must be stored. Interpolations of this type usually produce reasonably accurate results as long as the data table is very large and has very small increments between input values. Unfortunately, increasing the size of the data table to improve accuracy will also increase the computation time and memory requirement.

5.4 Nearest Interpolation

The next method used to calculate the best step given the Best Step Lookup Table was nearest interpolation. In the 2D case, the lookup table has a set of y output values and each output's corresponding x input value. When computing the output y_1 for a given input x_1 that is not actually in the lookup table, the closest point, x_2, is found in the table instead. The output value y_1 is set equal to the output value y_2. Using the same data points from the linear interpolation example, nearest interpolation was performed, and the results are shown pictorially in Figure 29.

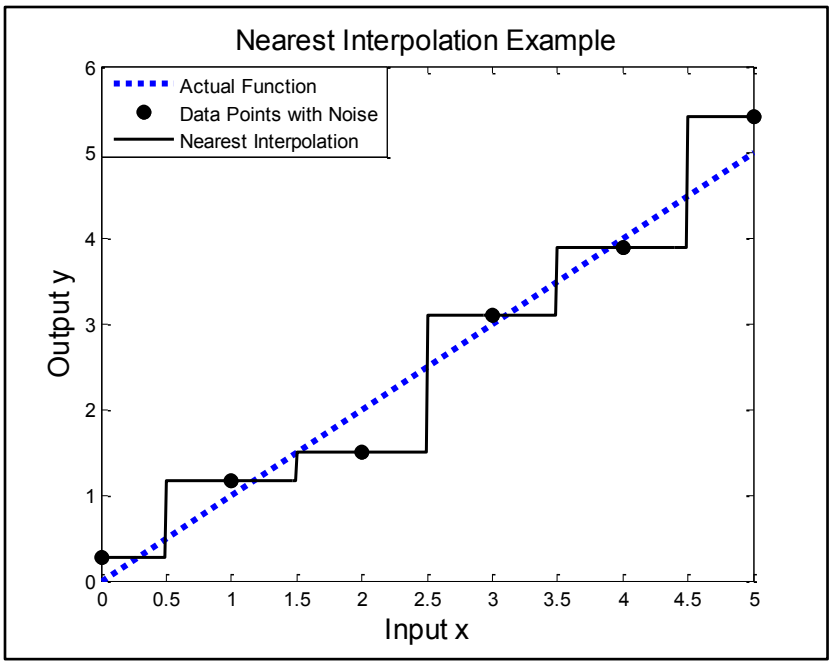

Figure 29: Nearest Interpolation Example

As with linear interpolation, the 2D example can be extended to a multidimensional table. Once again, MATLAB's *griddatan* function was used to perform the interpolation. Nearest interpolation experiences many of the same drawbacks as linear interpolation. These include having a slow computation time and using a large amount of memory. While accuracy is increased with more data points, nearest interpolation is generally less accurate than linear interpolation as the set of possible outputs is discrete rather than continuous.

5.5 Linear Least Squares

When in possession of a set of data, a relationship between the input and output variables can be found using linear least squares. This relationship is expressed in the form of Equation 56. Within this equation, the x's are the inputs, the m's are the slopes, b is the y-intercept, y is the output, and p is the number of input variables.

$$m_1 x_1 + m_2 x_2 + \cdots + m_p x_p + b = y \tag{56}$$

Given q points in the dataset, a system of equations can be expressed as shown in Equation 57. An exact solution to the system of equations is one in which the correct y value is calculated with no error for every point in the dataset. If there are fewer equations than unknowns, then the system is underdetermined, and an exact solution cannot be found. If there is the same number of equations as unknowns, then an exact solution can be found if and only if the dataset has a linear relationship with no noise. If there are more equations than unknowns, then the system is overdetermined, and an exact solution cannot be found.

$$\begin{gathered} m_1 x_{11} + m_2 x_{21} + \cdots + m_p x_{p1} + b = y_1 \\ m_1 x_{12} + m_2 x_{22} + \cdots + m_p x_{p2} + b = y_2 \\ \vdots \\ m_1 x_{1q} + m_2 x_{2q} + \cdots + m_p x_{pq} + b = y_q \end{gathered} \tag{57}$$

In most practical implementations, the dataset will be relatively large, so the system will be overdetermined. The dataset will also most likely contain noise. Since an exact solution cannot be found in these cases, the solution will be the equation that best approximates the dataset. This means that the goal of linear least squares will be to find an equation that minimizes the sum of the squares of the error for each data point. The first step in the linear least squares method is to represent the system of equations in matrix form as shown in Equation 58.

$$\begin{bmatrix} x_{11} & \cdots & x_{p1} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{1q} & \cdots & x_{pq} & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ \vdots \\ m_p \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix} \quad (58)$$

As shown in Equation 59, the matrices above can be represented algebraically as X, A, and Y respectively. X is not a square matrix, so the inverse of X cannot be determined. To solve this equation for A, both sides of the equation are first multiplied by X^T to obtain a square matrix and then multiplied by $(X^TX)^{-1}$. The value $(X^TX)^{-1}X^T$ is called the pseudo-inverse of X. These two steps are included in Equations 60 and 61 respectively, with the solution for A presented as Equation 62.

$$X * A = Y \quad (59)$$

$$X^TX * A = X^TY \quad (60)$$

$$(X^TX)^{-1}(X^TX) * A = (X^TX)^{-1}X^TY \quad (61)$$

$$A = (X^T X)^{-1} X^T Y \quad (62)$$

Using the same data points from the linear interpolation and nearest interpolation examples, linear least squares was performed, and the results are shown pictorially in Figure 30.

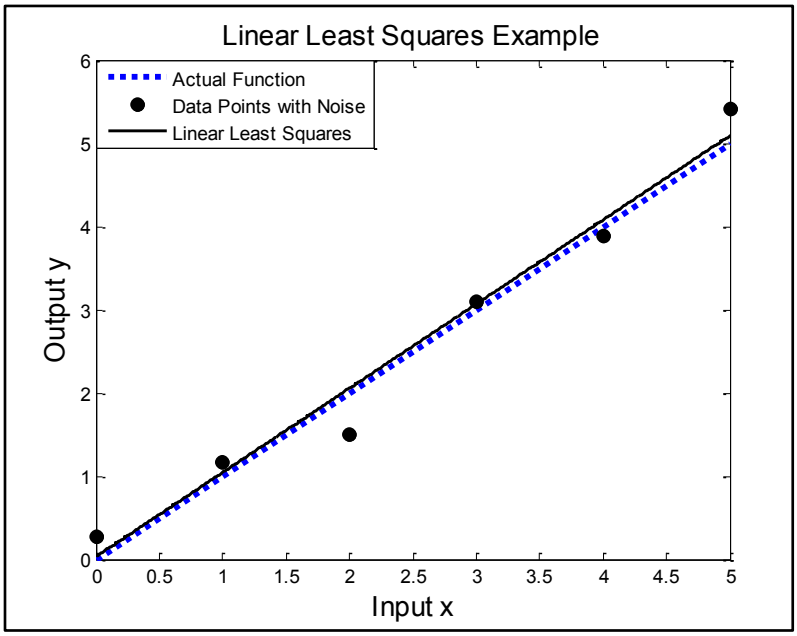

Figure 30: Linear Least Squares Example

In the 2D case, the linear least squares solution is referred to as the line of best fit, and the error for each data point is the vertical distance between the data point and the line when graphed. In the 3D case, the linear least squares solution will be a plane of best fit, and in the n-dimensional case, the linear least squares solution will be a hyperplane of best fit.

Once the linear least squares solution has been found, the equation can be used to compute the output for any input, regardless of whether it was in the original dataset. One caveat is that the linear least squares solution should only be used for interpolation rather than extrapolation. The equation should not be used for extrapolation, because there is no observed data to suggest that the pattern continues for values outside of the original range. Most real datasets do not continue to increase infinitely; they generally either level off or begin to decrease after a certain point.

There are several advantages to using linear least squares but also one major disadvantage. To begin with, the equation is very fast to implement as it only uses a relatively small number of simple mathematical operations. Also, storing the simple equation uses almost no memory. Linear least squares can often produce better results than linear interpolation, because it considers the overall pattern of the data. Therefore, linear least squares is less susceptible to noise than linear interpolation. However, if the input to output relationship is not linear, linear least squares will produce a very poor approximation.

5.6 Neural Networks

5.6.1 Introduction

The last method used to calculate the best step given the Best Step Lookup Table was neural networks. Neural networks are a machine learning technique used to fit a non-linear equation to a set of data. Although the method of computing the equation is very

different, the neural network itself is very similar to a line of best fit. Both predict an output from a given input, but neural networks differ in their ability to capture non-linear relationships. A neural network is generally represented by the structure shown in Figure 31.

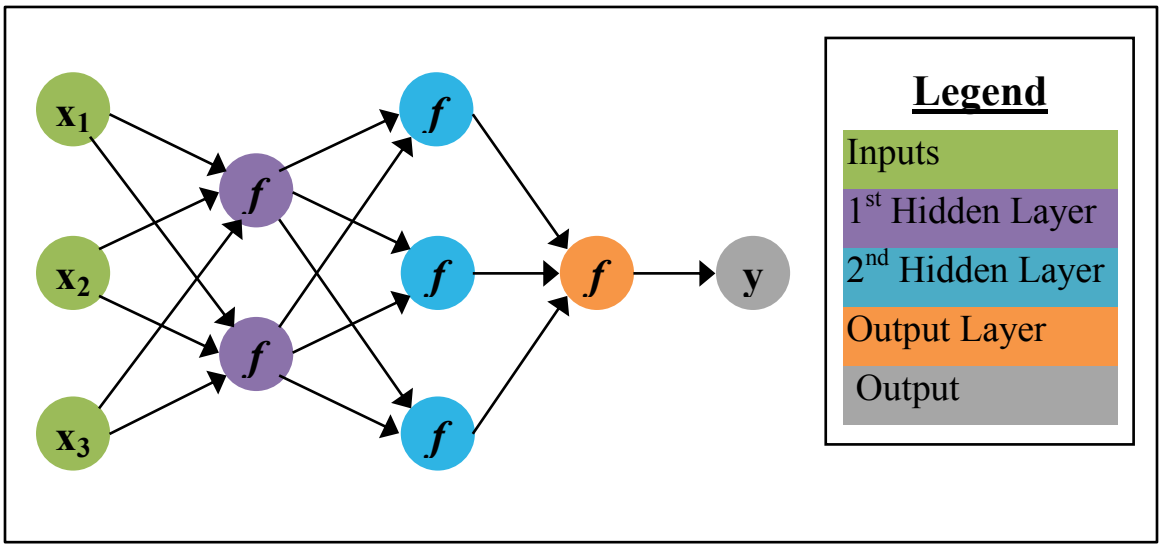

Figure 31: Neural Network Structure

A neural network is arranged in a series of nodes which are the circles seen above. Each node computes its own output by creating a linear combination of its inputs and applying some activation function. The activation function will be described in further detail later. The neural network's inputs are also the inputs to the first hidden layer. Neural networks can optionally include a second hidden layer whose inputs are the outputs of the first hidden layer. The outputs from the last hidden layer are fed into the output layer to compute the final output. Generally speaking, a neural network is nothing more than a large set of nested equations.

5.6.2 Training

The equation for the neural network is found using feedforward backpropagation training. To start the training process, the weights at each node are randomly declared. Each point from the dataset is fed forward through the neural network in order to determine the predicted value for that point given the current equation. The error of each point is determined as the difference between the predicted output and the actual output. By using partial derivatives to assign the blame of the errors to certain nodes, the weights are systematically adjusted. This process is then repeated for many iterations until the stopping criterion is met. Options for the stopping criterion are described in further detail later.

There are two different types of errors that are typically used during training: root mean squared error and mean absolute error. The word error generally means the difference between the actual and predicted values. Root mean squared error is the square root of the mean of the error for each data point squared. By squaring the error, it makes large errors even larger relatively, so outliers will have a much greater effect when fitting an equation. Squaring the errors also makes it so that negative and positive errors do not cancel each other out making an imperfect curve seem like a perfect fit. By taking the square root, it returns the error to the same units as the original data before the error was squared. Mean absolute error is the mean of the absolute value of the error for each data point. The mean absolute error also prevents negative and positive errors from cancelling each other out, but it does not put extra weight on the results of outliers.

5.6.3 Activation Functions

Typical activation functions used in neural networks include sigmoid, Gaussian, linear, and threshold functions as shown in Figure 32. A sigmoid function is a function that starts low and ends high with a smooth S shape. Two common sigmoid functions are the hyperbolic tangent function and logistic function. The Gaussian function is a smooth bell-shaped curve that is low on the two ends and high in the middle. The linear function is a straight line with a constant slope. The threshold function is equal to zero when less than a cutoff value and one when greater than the same cutoff value.

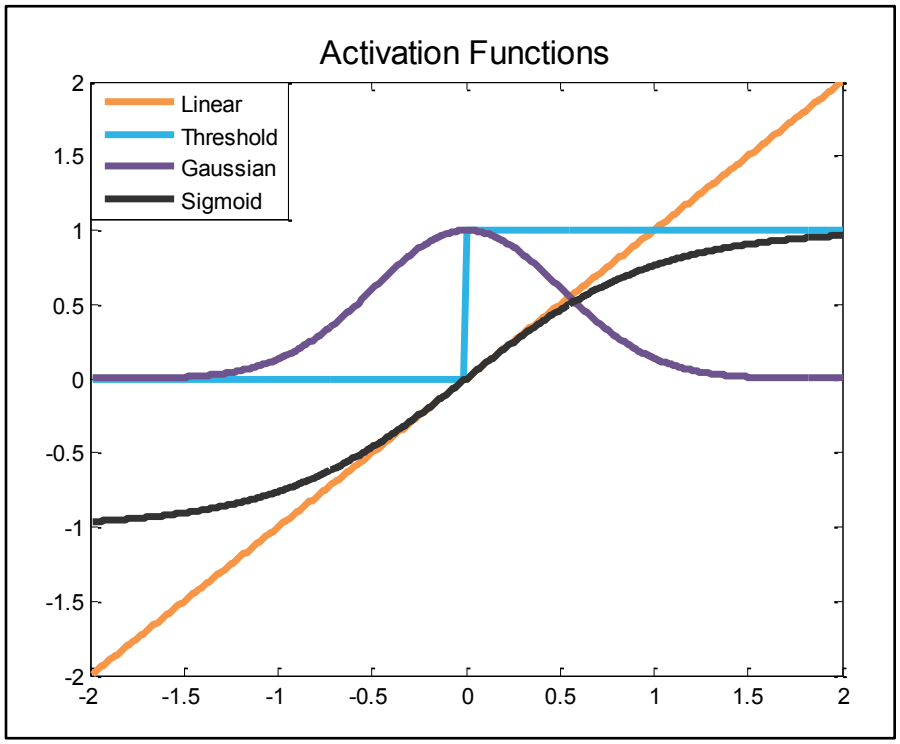

Figure 32: Typical Neural Network Activation Functions

A linear combination of other linear combinations will just create a different linear combination, so using a linear function in the hidden layers would make the neural

network linear as well. Generally, neural networks are used when non-linear functions are desired, so the linear function is very rarely used in hidden layers. In contrast, the linear function is the most common function used in the output layer, because it will not restrict the range of output values [33].

The threshold function can be used to introduce non-linearity to the equation, but it is difficult to use due to the discontinuity in the function. The discontinuity makes the function non-differentiable, so backpropagation of the error cannot be done as accurately.

This leaves the Gaussian and sigmoid functions, which are the two most common activation functions used in neural networks. These two functions produce favorable results as they are non-linear, differentiable, and bounded [33]. Either function may produce faster training time or more accurate results depending on the underlying pattern in the data. Separate neural networks can be trained with each activation function, and the neural network with the lowest validation error, described in more detail later, will be chosen.

5.6.4 Overfitting

The main concern while training a neural network is overfitting. When fitting a curve to a set of data, the error can generally get lower if a higher order polynomial is used. This is shown by the red function in Figure 33. The data, represented by black dots, was generated by adding a little bit of noise to the straight line shown in the figure.

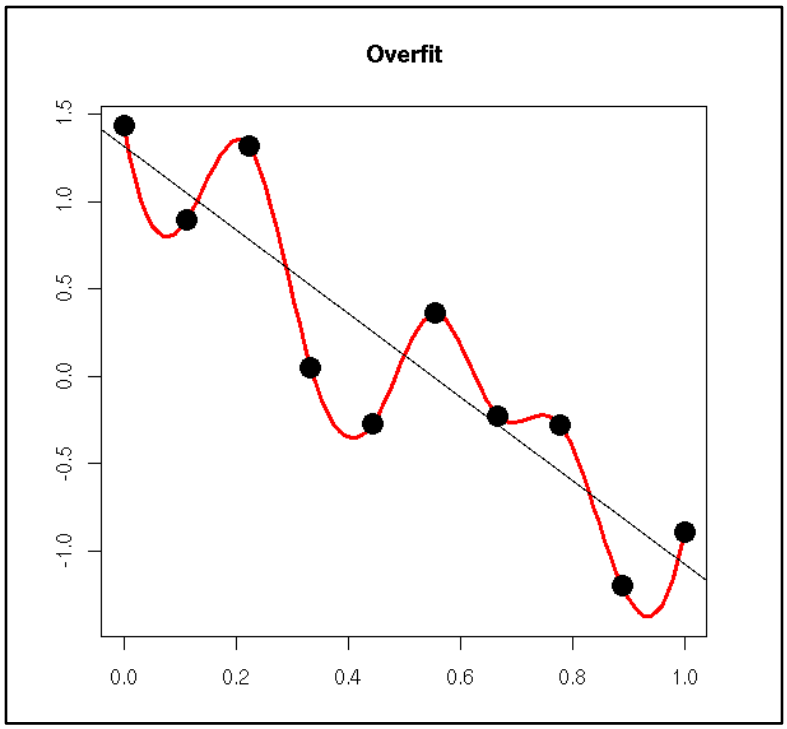

Figure 33: Example of Overfitting a Curve to a Dataset [34]

The error between the training data and the red function is zero, because the red function goes through each point exactly. However, when interpolating between points in the training data, the red function will be very inaccurate. This situation is called overfitting, because the red function has been fit overly well to the training data at the expense of generalizing well. Overfitting can be prevented by having a large training dataset, choosing the ideal number of nodes, and using an appropriate stopping criterion. The different options for stopping criteria and the ideal number of nodes are discussed in the following sections.

5.6.5 Stopping Criteria

The stopping criterion is what determines when to stop iterating through the feedforward backpropagation training process. The three most common stopping criteria are number of iterations, convergence, and early stopping.

The most straight-forward of the stopping criteria options is number of iterations. When using this criterion, the weights are adjusted a certain number of times before stopping the training. This method is simple but very ineffective, because the neural network may not be fully trained when the process is stopped. The number of iterations it will take to fully train the neural network will vary widely based on the number of nodes and layers, the amount of data points in the training set, the complexity of the underlying function, how fast the weights are adjusted, and the random initial values of each weight. With all of these variables affecting the number of iterations to fully train the neural network, it is difficult to choose just one number to use as an effective number of iterations.

Another common stopping criterion is convergence. When using this criterion, the weights are adjusted until the values of the weights have converged, meaning that the values have stopped changing between iterations. While this method does not risk undertraining, it risks overtraining, because as the number of iterations is increased, the error on the training set will try to decrease until it levels off.

The third common stopping criterion is called early stopping. In order to prevent overtraining, some of the data is not included in the training set and becomes the

validation set. As seen in Figure 34, the error on the validation data will initially decrease as the neural network is trained and then increase as the neural network is overtrained. The error on the validation data begins to increase, because if a neural network is overtrained, it will perform poorly when predicting points not used during training, such as the validation data. When using the early stopping criterion, the neural network stops training once the error on the validation data begins to increase.

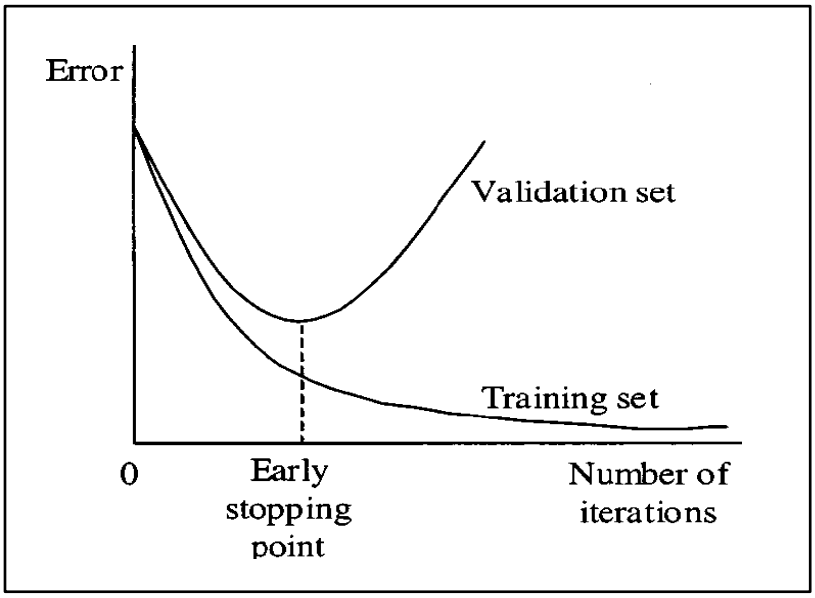

Figure 34: Early Stopping Method [35]

5.6.6 Number of Layers and Nodes

Most neural networks consist of only one or two hidden layers. Training three or more hidden layers is rarely ever necessary and is much more complex to train. For most applications, neural networks with only a single hidden layer are sufficient [36]. One

way to choose the number of hidden layers is to train single layer neural networks and then proceed to two layer neural networks if and only if the performance is unacceptable.

The ideal number of nodes varies for each situation and cannot be easily determined. If there are not enough nodes, then the neural network will not be complex enough to model the true function. If there are too many nodes, then the function is more likely to overfit the training data [36]. Since the validation data is not used during training, the error on the validation set can be used to aid in neural network selection. As the number of nodes is increased, the validation error will decrease initially and then increase due to overfitting. The ideal number of nodes occurs when the validation error is at a minimum. Multiple neural networks, each with a different number of nodes, must be trained in order to find the ideal number of nodes.

There are several rules of thumb for determining the ideal number of nodes, although their effectiveness is somewhat debated. One rule of thumb is that the number of nodes is between the number of inputs and the number of outputs. A second rule of thumb is that the number of nodes should be less than twice the number of inputs [36, 37]. When using the early stopping method, it is generally believed that the risk of overtraining is small, so contrary to these rules of thumb, a large number of nodes should be used [37].

CHAPTER 6

Results

6.1 Introduction

The effectiveness of each interpolation method was evaluated for pushes within the defined range. The following section discusses the neural network training results, which include selection of the activation function and number of nodes for each neural network. The remaining two sections explore the push recovery effectiveness and computation time of each interpolation technique.

6.2 Neural Network Training

The first step in the evaluation process was to train neural networks using the statistical software program JMP. During training, 75% of the data was included in the training set while the other 25% of the data was reserved for the validation set. The error was measured using root mean squared error for reasons previously discussed. The early stopping method was used to determine the number of iterations during training. To avoid finding a local minimum, each neural network was trained five separate times starting with different initial weights each time, and the neural network with the lowest validation error was returned. Neural networks were trained in 5 node increments until

the error on the validation set began increasing. This process was performed once using Gaussian activation functions and then again using sigmoid activation functions. Of these trained neural networks, the one with the lowest validation error was chosen for further evaluation within the simulation. By plotting the training results in Figure 35, it can be seen that a neural network with a tanh activation function and 45 nodes was chosen to calculate the swing constant when taking the first step.

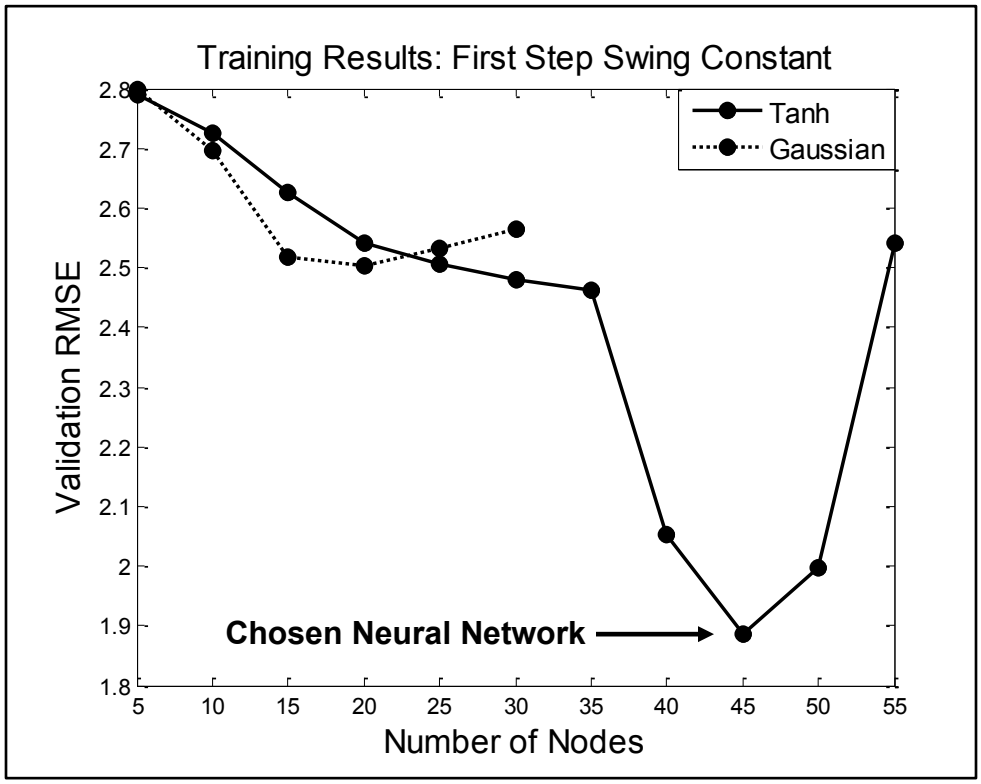

Figure 35: Training Results: First Step Swing Constant

As shown in Figure 36, the appropriate choice for calculating the push-off constant when taking non-first steps was a neural network with 30 nodes and a Gaussian activation function.

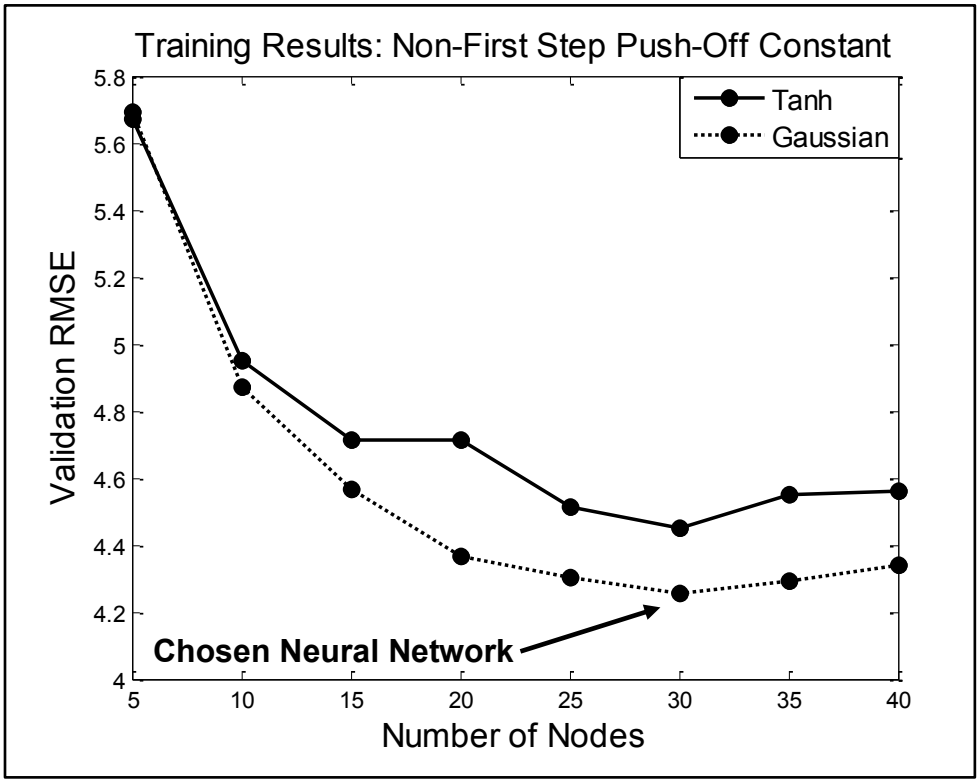

Figure 36: Training Results: Non-First Step Push-Off Constant

Similarly, Figure 37 shows that the appropriate choice for calculating $\alpha_{desired}$ when taking non-first steps was a neural network with 20 nodes and a tanh activation function.

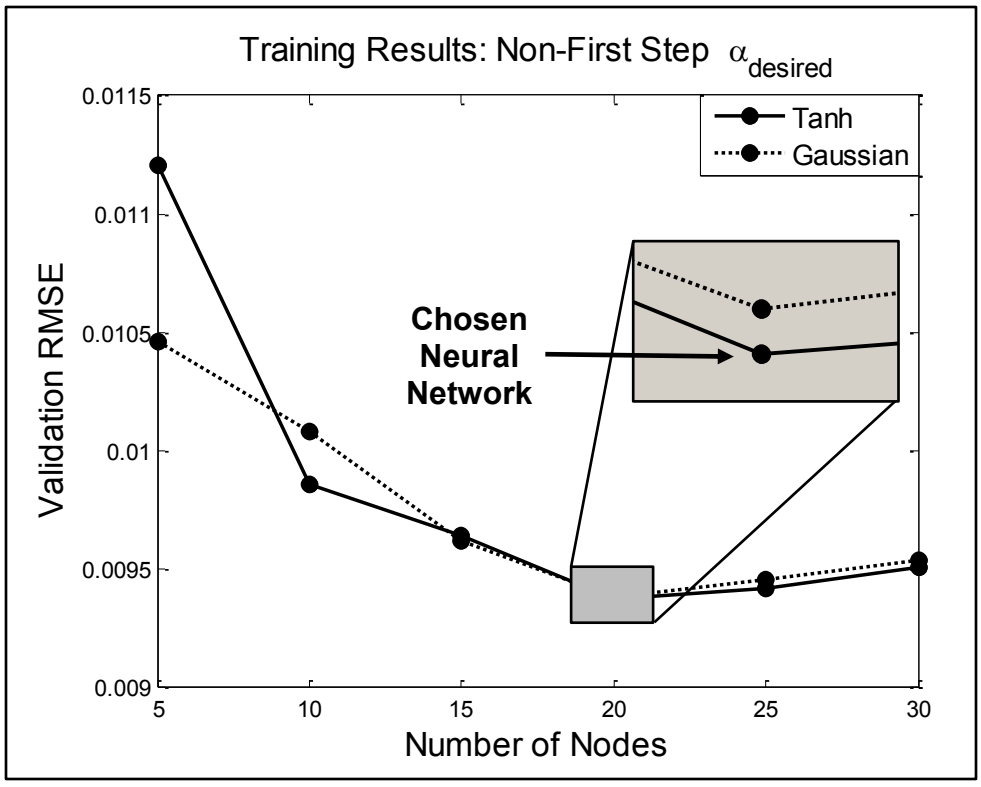

Figure 37: Training Results: Non-First Step $\alpha_{desired}$

6.3 Push Recovery Effectiveness

After the neural networks were trained, linear least squares equations were calculated for the first step swing constant, non-first step push-off constant, and non-first step $\alpha_{desired}$. There was no need to calculate equations for linear or nearest interpolation as these methods use the lookup table during implementation. It should be noted that for the linear and nearest interpolations, the dataset had to be reduced by a factor of ten in order to perform each simulation within a reasonable amount of time.

Each interpolation method was tested within the MATLAB simulation on pushes ranging from 60 Nm to 130 Nm with an increment of 1 Nm. For each interpolation method, the number of pushes that the robot recovered from was recorded, and a summary of these results is shown in Figure 38.

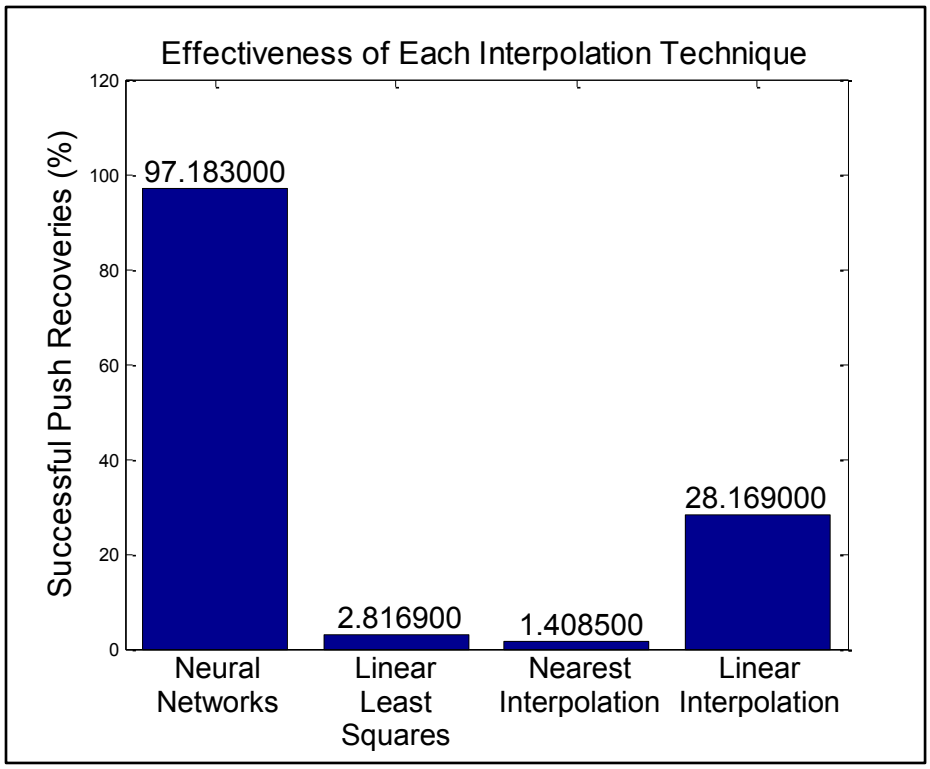

Figure 38: Effectiveness of Each Interpolation Technique

These results show that the least effective interpolation technique was nearest interpolation, which only recovered from approximately 1.4% of the pushes. Linear least squares also performed poorly by recovering from less than 3% of the pushes. This poor performance was anticipated due to the non-linear dynamics of the compass robot. Neural networks recovered from approximately 97% of the pushes, thereby making them the most effective interpolation technique. While its performance fell short of neural

networks, linear interpolation was able to successfully recover from roughly 28% of the pushes. This technique experienced limited success, because given enough data points, nonlinear functions can be reasonably approximated by a series of piecewise linear equations. Although 28% is relatively low, this would still be better than not having a push recovery strategy, and when compared to neural networks, linear interpolation was much simpler to implement.

6.4 Computation Time

While performing the simulations, the computation time for each step was also recorded, and a summary of these results is shown in Figure 39. The nearest interpolation and linear interpolation methods took approximately 117 and 73 seconds respectively to calculate the proper step based on the lookup table. By the time the step could be calculated using either of these methods, the robot would almost certainly have fallen over. Additionally, if more data was added to the table in an attempt to increase the accuracy of the results, these computation times would increase even more. In contrast, neural networks and linear least squares computed the step almost instantaneously, and their computation times would not increase as a result of adding more data. This would theoretically allow them to decide on an appropriate response in time to successfully recover from a push.

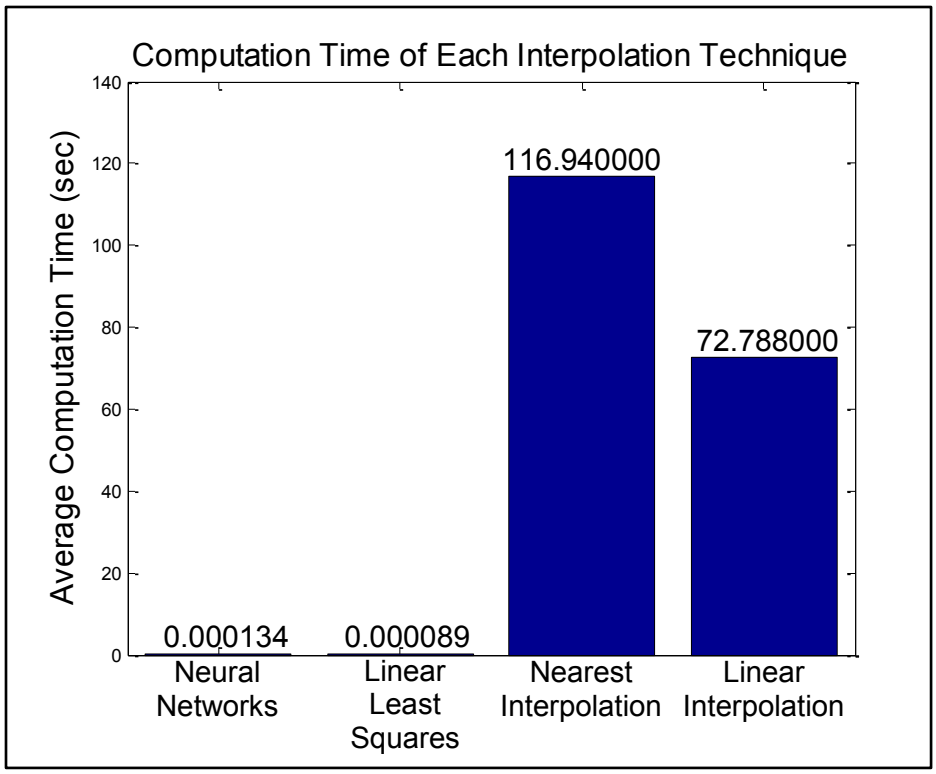

Figure 39: Computation Time of Each Interpolation Technique

CHAPTER 7

Summary and Conclusion

7.1 Summary and Conclusions

This thesis introduced a new approach to implementing the reactive stepping method. In order to test the new approach, a MATLAB model of a compass robot was created. The model included appropriate constraints and parameters that were realistic of an actual humanoid robot. It was then proposed that a robot should take the step which would result in the highest reduction in total energy. For each starting configuration, the simulation was run for every possible step, and the step that resulted in the lowest total energy was stored in the Best Step Lookup Table. In order to implement the Energy Reduction Method on a robot, interpolation within the Best Step Lookup Table was necessary. Neural networks, linear least squares, nearest interpolation, and linear interpolation were all tested as potential interpolation techniques. The performances of each technique were compared against each other on the basis of push recovery effectiveness and computation time.

Results showed that the computation times for nearest interpolation and linear interpolation were too slow for a robot to make a decision before falling over. In contrast, neural networks and linear least squares computed the step almost

instantaneously. The linear least squares and nearest interpolation techniques achieved very poor recovery rates. Linear interpolation experienced a mediocre recovery rate but had the advantage of being very simple to implement. In contrast, neural networks had a very high success rate but were much more difficult to implement due to the time and effort required to train each network.

When compared to previous reactive stepping approaches, the machine learning approach has many advantages. One advantage is that the calculated stepping location is predictive and therefore only has to be calculated once for each step. This approach also accounts for the situation in which a robot requires more than one step to recover from the push. Unlike an analytical solution that would be model specific, this approach is easily generalized. The same process as described in this thesis could be followed to implement reactive stepping on physical robots, or other robot models.

The main drawback to the machine learning approach is that a large amount of training data must be generated in order to train the neural networks. While the research as presented serves as a proof of concept, further testing should be performed on other, more complex robot models. In addition, the method should be tested on a physical robot in order to determine its true push recovery effectiveness.

7.2 Suggestions for Future Work

This thesis used a compass model to present a proof of concept for a machine learning approach to reactive stepping. The same process could be followed to implement reactive stepping on any robot model that can describe its steps using a relatively small number of parameters. This work could also be continued by implementing reactive stepping on a physical robot. Towards this goal, the researcher could either gather the training data by using a model or by using a physical robot.

If the training data is gathered using a computer model, the model should be made to accurately reflect the physical robot. It would need to include feet, knees, an upper body, appropriate masses, and appropriate dimensions. If the model is accurate enough, neural network equations trained using data from the model could theoretically be implemented directly on the physical robot. Unfortunately, developing a model accurate enough for these purposes could be extremely difficult.

Instead of using a model, the training data could be gathered by using the physical robot itself. Gathering training data using a physical robot would be much more difficult than using a computer simulation for several reasons. The first reason is that the researcher must prevent the robot from becoming damaged during steps that result in falls. This issue can be solved by appropriately tethering the robot or positioning the robot above a soft surface on which to fall. Another issue that would need to be addressed is that neural network performance generally improves with more training data. Increasing the amount of training data would be difficult when using a physical robot, because it usually cannot

be left to run autonomously for long periods of time. Kalyanakrishnan and Goswami ran into similar issues when proposing data collection on a physical robot for their supervised learning of fall prediction. They suggested building a test fixture to automate the process, thereby allowing a large amount of data to be collected without any human intervention [38]. Lastly, data collected on a physical robot will contain noise not present in a model. Therefore, it will most likely require more training data to accurately learn push recovery when using data from a physical robot.

Either of these avenues would present an interesting extension of the work performed within this thesis. When implemented on a physical robot, the machine learning approach to reactive stepping could provide a quick and effective way of deciding how and where to step in response to large pushes.

Appendix A: Energy Reduction Method Results

Step #	θ_{ns}°	θ_s°	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-6.4	-0.9	-66.5	-42.1	0	30	45	237.2
2	-22.5	22.5	20.1	-60.6	2.5	60	-32.5	231.2
3	16.1	-16.1	-6.5	-1.1	25	60	30	231
4	-16.1	16.1	0.5	-0.4	STOPPED			

Table 2: Energy Reduction Method Results for a Push with an Equivalent Torque of 60 Nm

Step #	θ_{ns}°	θ_s°	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-7.3	-1.5	-80.8	-50.7	0	40	45	237.6
2	-22.6	22.6	21.5	-62.5	2.5	60	-31.9	231.6
3	15.7	-15.7	0.2	0.9	0	60	28.8	231.4
4	-15.7	15.7	0.2	0.2	STOPPED			

Table 3: Energy Reduction Method Results for a Push with an Equivalent Torque of 70 Nm

Step #	$\theta_{ns}°$	$\theta_s°$	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-8.3	-2.1	-95.1	-59.2	0	50	45	238
2	-22.7	22.7	22.9	-64.6	5	60	-30.6	233.1
3	15.2	-15.2	-8.9	-5.1	25	60	27.5	232.8
4	-15.2	15.2	-1.2	-3.7	0	60	-33.1	232.4
5	15.2	-15.2	0.6	1.6	0	60	27.5	232
6	-15.1	15.1	0.1	-0.1	STOPPED			

Table 4: Energy Reduction Method Results for a Push with an Equivalent Torque of 80 Nm

Step #	$\theta_{ns}°$	$\theta_s°$	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-9.2	-2.7	-109.4	-67.8	0	70	45	238.4
2	-22.8	22.8	24.5	-66.5	2.5	60	-30	233.5
3	14.9	-14.9	-16.3	-5.2	20	60	27.5	233
4	-14.9	14.9	-0.3	-3.1	0	60	-32.5	232.5
5	14.9	-14.9	0.4	1.2	30	60	27.5	232.3
6	-14.9	14.9	0.2	0.1	STOPPED			

Table 5: Energy Reduction Method Results for a Push with an Equivalent Torque of 90 Nm

Step #	$\theta_{ns}°$	$\theta_s°$	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-10.2	-3.3	-123.8	-76.3	0	90	45	238.8
2	-22.9	22.9	26.1	-68.7	7.5	60	-28.1	233.6
3	13.8	-13.8	0.2	0.6	STOPPED			

Table 6: Energy Reduction Method Results for a Push with an Equivalent Torque of 100 Nm

Step #	θ_{ns}°	θ_s°	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-11.1	-3.9	-138.2	-84.9	0	110	45	239.3
2	-23	23	27.8	-71.1	0	60	-28.1	234.6
3	13.8	-13.8	2.5	4.9	0	60	25	234.1
4	-13.7	13.7	-1.4	-2.8	0	60	-30	233.7
5	13.7	-13.7	0.4	1.1	30	60	25	233.5
6	-13.7	13.7	0.2	0.1	STOPPED			

Table 7: Energy Reduction Method Results for a Push with an Equivalent Torque of 110 Nm

Step #	θ_{ns}°	θ_s°	$\dot{\theta}_{ns}\left(\frac{deg}{sec}\right)$	$\dot{\theta}_s\left(\frac{deg}{sec}\right)$	Push-Off Constant (Nm)	Swing Constant	$\alpha_{desired}$	Total Energy (N)
1	-12.1	-4.5	-152.5	-93.4	0	140	45	239.7
2	-23.1	23.1	29.7	-73.5	2.5	60	-26.3	235.5
3	12.8	-12.8	2.8	4.8	0	60	23.1	235
4	-12.8	12.8	-1.6	-2.9	0	60	-28.1	234.7
5	12.8	-12.8	0.6	1.2	5	60	23.1	234.4
6	-12.8	12.8	0.1	0	STOPPED			

Table 8: Energy Reduction Method Results for a Push with an Equivalent Torque of 120 Nm

References

[1] *LawnBott Robotic Mowers*. Kyodo America, 2013. Web. 9 April 2013. <http://www.lawnbott.com/>.

[2] *IRobot Roomba Vacuum Cleaning Robot*. IRobot Corporation, 2013. Web. 9 April 2013. <http://www.irobot.com/us/learn/home/roomba.aspx>.

[3] "Robotic Surgery." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc., 27 May 2013. Web. 2 April 2013. <http://en.wikipedia.org/wiki/Robotic_surgery>.

[4] K. Jensen. "Wisdom from the Workroom: David Robinson Shares Lessons in Product Development." Brigham Young University: Ira A. Fulton College of Engineering and Technology, 8 Apr. 2013. Web. 09 April 2013. <http://www.et.byu.edu/news/wisdom-workroom-david-robinson-shares-lessons-product-development>.

[5] T. Cronk. "Robot to Serve as Future Military's 'Pack Mule'." U.S. Department of Defense, 19 Dec. 2012. Web. 02 April 2013. <http://www.defense.gov/News/NewsArticle.aspx?ID=118838>

[6] "The Talon Bomb Disposal Robot Picks." Fine Art America, n.d. Web. 08 April 2013. <http://fineartamerica.com/featured/the-talon-bomb-disposal-robot-picks-stocktrek-images.html>.

[7] *DARPA Robotics Challenge*. Defense Advanced Research Projects Agency (DARPA), n.d. Web. 18 March 2013. <http://www.theroboticschallenge.org/aboutprogram.aspx>.

[8] "Industrial Robotics." Linkoping Center for Sensor Informatics and Control (LINK-SIC), n.d. Web. 08 April 2013. <http://www.linksic.isy.liu.se/?page=industrialrobotics-2>.

[9] S. Kajita. "Frequently Asked Questions about Biped Robots." National Institute of Advanced Industrial Science and Technology (AIST), 21 June 2011. Web. 28 March 2013. <http://staff.aist.go.jp/s.kajita/faq-e.html>.

[10] "Mars Rover." DuPont, n.d. Web. 14 April 2013. <http://www2.dupont.com/Media_Center/en_US/assets/images/vocus/NASA/pe_DCS_Mars_Rover_01.jpg>.

[11] "BigDog - The Most Advanced Rough-Terrain Robot on Earth." Boston Dynamics, 2013. Web. 7 April 2013. <http://bostondynamics.com/robot_bigdog.html>.

[12] "ASIMO Specifications." Honda, n.d. Web. 7 April 2013. <http://asimo.honda.com/asimo-specs/>.

[13] L. Raffensperger. "How Will Humans and Robots Coexist?" IEEE Spectrum, 25 Jan. 2013. Web. 02 April 2013. <http://spectrum.ieee.org/podcast/at-work/innovation/how-will-humans-and-robots-coexist>.

[14] R. Renner and S. Behnke. "Instability Detection and Fall Avoidance for a Humanoid using Attitude Sensors and Reflexes," *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* , vol., no., pp.2967,2973, 9-15 Oct. 2006

[15] O. Hohn, J. Gacnik, and W. Gerth. “Detection and Classification of Posture Instabilities of Bipedal Robots,” in *Proc. of the 8th Int. Conf. on Climbing and Walking Robots and the Support Technologies for Mobile Machines* – CLAWAR, 2005.

[16] S-J. Yi, B-T. Zhang, D. Hong, and D. Lee. "Learning Full Body Push Recovery Control for Small Humanoid Robots," *Robotics and Automation (ICRA), 2011 IEEE International Conference on* , vol., no., pp.2047,2052, 9-13 May 2011

[17] B. Stephens. "Integral Control of Humanoid Balance," *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on* , vol., no., pp.4020,4027, Oct. 29 2007-Nov. 2 2007

[18] S-H. Lee and A. Goswami. "Ground Reaction Force Control at Each Foot: A Momentum-Based Humanoid Balance Controller for Non-Level and Non-Stationary Ground," *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on* , vol., no., pp.3157,3162, 18-22 Oct. 2010

[19] J. Pratt, J. Carff, S. Drakunov, and A. Goswami. "Capture Point: A Step toward Humanoid Push Recovery," *Humanoid Robots, 2006 6th IEEE-RAS International Conference on* , vol., no., pp.200,207, 4-6 Dec. 2006

[20] J. Pratt and R. Tedrake. “Velocity Based Stability Margins for Fast Bipedal Walking,” in *First Ruperto Carola Symposium in the International Science Forum of the University of Heidelberg entitled “Fast Motions in Biomechanics and Robots”*, Heidelberg Germany, September 7-9 2005

[21] S-K. Yun and A. Goswami. "Momentum-Based Reactive Stepping Controller on Level and Non-Level Ground for Humanoid Robot Push Recovery," *Intelligent*

Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on , vol., no., pp.3943,3950, 25-30 Sept. 2011

[22] S-K. Yun, A. Goswami, and Y. Sakagami. "Safe Fall: Humanoid Robot Fall Direction Change through Intelligent Stepping and Inertia Shaping," *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on* , vol., no., pp.781,787, 12-17 May 2009

[23] S-H. Lee and A. Goswami. "Fall on Backpack: Damage Minimizing Humanoid Fall on Targeted Body Segment using Momentum Control," in *ASME Int. Design Engineering Tech. Conf.*, pp. 47153:1–10, 2011.

[24] J. Stuckler and S. Behnke. "Soccer Behaviors for Humanoid Robots," in *Proc. of the Workshop on Humanoid Soccer Robots of the IEEE-RAS Int. Conf. on Humanoid Robots, Genoa, Italy*, 2006, pp. 62–70.

[25] P. Miller. "Walking as Jazz: Virginia Tech Takes on the Biggest Challenge in Robotics." *The Verge*. Vox Media, Inc., 8 Nov. 2011. Web. 23 April 2013. <http://www.theverge.com/2011/11/8/2518608/walking-as-jazz-virginia-tech-robotics>.

[26] S. Kajita and B. Espiau. "Legged Robots." *Springer Handbook of Robotics*. Ed. Bruno Siciliano and Oussama Khatib. Berlin: Springer, 2008. 363-64. Print.

[27] A. Bedford and W. Fowler. "Planar Kinematics of Rigid Bodies." *Engineering Mechanics: Dynamics*. 5th ed. Upper Saddle River: Prentice Hall, 2008. 290-96. Print.

[28] A. Goswami, B. Thuilot, and B. Espiau. "Compass-Like Biped Robot Part I: Stability and Bifucation of Passive Gaits." Technical report, INRIA, No. 2996, Oct. 1996.

[29] D. George. "Thirteen Advanced Humanoid Robots Available for Sale." *Smashing Robotics*. N.p., 27 July 2012. Web. 06 Oct. 2012. <http://www.smashingrobotics.com/thirteen-advanced-for-sale>.

[30] D. Lofaro. "Jaemi Hubo (KHR4) Users Manual." Drexel University, 22 Dec. 2009. Web. 06 Oct. 2012. <http://www.pages.drexel.edu/~dml46/DASL/HUBO/JaemiHubo_Manual_KHR4_R1_2009-12-22_0236.pdf>.

[31] I-W. Park, J-Y. Kim, J. Lee, M-S. Kim, B-K. Cho, and J-H. Oh. "Development of Biped Humanoid Robots at the Humanoid Robot Research Center, Korea Advanced Institute of Science and Technology (KAIST)." *Humanoid Robots: Human-like Machines*. Ed. Matthias Hackel. Vienna: I-Tech Education and, 2007. 44-48. Print.

[32] T. McGeer, 1990a. "Passive Dynamic Walking," International Journal of Robotics Research, 9:62-82.

[33] W. Sarle. "Neural Network FAQ, Part 2 of 7: Learning." Periodic posting to the Usenet newsgroup comp.ai.neural-nets. N.p., 2002. Web. 9 July 2012. <ftp://ftp.sas.com/pub/neural/FAQ2.html#questions>.

[34] V. Zoonekynd. "Regression Problems -- and Their Solutions." N.p., 6 Jan. 2007. Web. 28 Feb. 2013. <http://zoonek2.free.fr/UNIX/48_R/11.html>.

[35] "Pricing and Hedging Derivative Securities with Neural Networks: Bayesian Regularization, Early Stopping, and Bagging." National Taiwan University of

Science and Technology: Department of Computer Science and Information Engineering, n.d. Web. 12 Dec. 2012. <http://neuron.csie.ntust.edu.tw/homework/94/neuron/Homework3/M9409204/discuss.htm>.

[36] J. Heaton. "Feedforward Neural Networks." *Introduction to Neural Networks for Java*. 2nd ed. St. Louis: Heaton Research, 2008. 157-59. Print.

[37] W. Sarle. "Neural Network FAQ, Part 3 of 7: Generalization." Periodic posting to the Usenet newsgroup comp.ai.neural-nets. N.p., 2002. Web. 9 July 2012. < ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hl >.

[38] S. Kalyanakrishnan and A. Goswami. “Learning to Predict Humanoid Fall,” *The International Journal of Humanoid Robotics*, vol. 8, no. 2, pp. 245–273, 2011.