Humanoid Walking Robot

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
in
Mechanical Engineering and Robotics Engineering
by

Andrew Curran

Kenneth Colpritt

Shannon Moffat

Mariah Sullivan

Date: 4/26/18

Popovic Labs

Worcester Polytechnic Institute

Professor Marko Popovic, Advisor

Professor Selçuk Guçeri, Co-Advisor

Abstract

Our project is focused on research and development of the application of Hydro Muscles to biologically inspired humanoid robots. Our team designed, researched, and developed a bio-inspired, bipedal walking robot to simulate the human gait cycle. The walking motion is actuated by Hydro Muscles, which are soft artificial muscles that are driven pneumatically or hydraulically to contract and expand longitudinally. These artificial muscles were were modeled to match mass scaled actuation of human muscles on a lower limb skeletal model to create a biologically authentic gait. Further extensions of this project would explore this robot's potential for clinical, prosthetic, military defense, and other applications.

Acknowledgements

This project would not have been possible without the support of our advisor, Professor Marko Popovic, and our co-advisor, Professor Selçuk Guçeri. We greatly appreciate their constructive feedback, as well as their advice for approaching problems along the way. We also benefited from the equipment and resources in Popovic Labs.

We would also like to thank Matthew Bowers, who guided us as a PhD student in Popovic Labs. He provided resources and background from previous relevant projects.

We would like to extend our gratitude to the Mechanical Engineering and Robotics Engineering Departments for their financial support, as well as their educational support. The team was able to use our academic backgrounds and learning from the past four years in approaching this extensive project.

Lastly, we would like to thank the other students working in Popovic Labs this year. We were able to create a support system for the late nights in the lab, and it was beneficial for us to collaborate with like-minded people.

Executive Summary

There exist many humanoid robots made for human interaction. However, typical rigid mechanisms present safety risks and are not as comfortable for humans to interact with. This has led to developments in soft robotics, with fluidic actuators becoming a popular mechanism for robot motion. There is a common principle that states that form follows function. If this is true, then in order to produce human motion, a human form should be used. Emulating the human musculoskeletal system will provide insight into the capabilities of bipedal and humanoid robots. The overarching goal of this project was to build a lifelike robot that can simulate the motions of the human gait cycle. We used a biomimetic approach to incorporate human anatomy into the mechanical design.

Hydro Muscles are a novel, soft robotic actuator developed by Professor Marko Popovic in Popovic Labs in 2014. They are used for fluidic linear actuation, in which hydraulic or pneumatic power may be used. Overall, the Hydro Muscle has many favorable characteristics. It is lightweight, small, efficient, soft, fast, similar to biological muscle, and cost-effective.

The gait cycle for bipedal human walking is a repetitive pattern that incorporates steps and strides, where a stride consists of two steps. A step is seen to start from the initial contact of one foot and then ending at the initial contact of the other, while a stride starts with the initial contact of one foot and ends with the initial contact of that same foot (Physiopedia, 2017). The gait cycle can be broken up into Heel Strike, Stance, Heel-Off, Pre-Swing (Toe-Off), Mid Swing, and Terminal Swing.

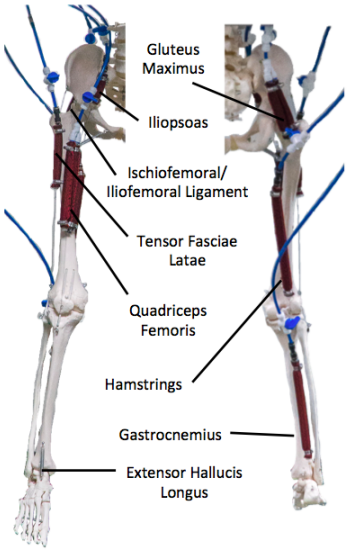

Figure 1: Muscle placements.

The amount of muscles in the lower limbs of humans is too extensive to be completely modeled using the HydroMuscles. For this

reason, the muscles must be simplified while still retaining anatomical integrity. The muscles that are most active during walking were selected, each analyzed for origin, insertion, length, force generation, and musculoskeletal function to be simplified into the segmental model provided by the HydroMuscles. There are six muscles total to be modeled, including the iliopsoas, tensor fasciae latae, quadriceps femoris, gluteus maximus, hamstrings (biceps femoris and semitendinosus), and gastrocnemius. There are also passive spring structures to model the extensor hallucis longus, as well as the iliofemoral and ischiofemoral ligaments.

A solenoid valve is an interface between pneumatic systems and electronic controllers. It is seen as a switch to send air to any pneumatic device, which can allow pneumatic control of temperature, flow, position, and pressure. We used these valves to apply a state machine system in which the valves would contract or expand a specific muscle depending on what phase of the gait cycle the robot found itself in. Connected to an Arduino, this allowed us to control the individual Hydro Muscles attached to the skeleton. In order to control the speed in which the muscles would contract, we used flow control valves. A flow control valve regulates the pressure or flow of air coming out of the hose side of the valve. This device allows us to better control the motion of the skeleton, allowing us to create a smoother gait cycle.

Sensors were used for transitions between gait cycle phases. IMUs were mounted to the lower leg to measure overall leg orientation, and conductive rubber stretch sensors were bridged across joints to measure joint angle. Thresholds were set for each state so that the program would know when to transition to the subsequent state.

While trying to anatomically resemble the muscles used in human walking, we were able to create a sufficient demonstration of Hydro Muscles actuating a bipedal robot to create a biomimetic walking gait cycle. We showed that there are strides to be made in the direction of biomimicry in robotics.

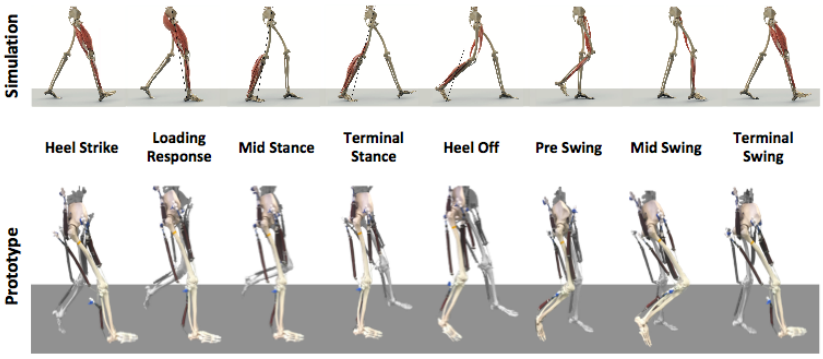

Figure 2: Gait cycle stages.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There exist many humanoid robots made for human interaction. However, typical rigid mechanisms present safety risks and are not as comfortable for humans to interact with. This has led to developments in soft robotics, with fluidic actuators becoming a popular mechanism for robot motion. This not only alleviates safety concerns, but also integrates robotic technology more acceptably into human tasks.

In addition, many robots are not able to achieve the same functions and tasks as humans with the same efficiency and precision. There is a common principle that states that form follows function. If this is true, then in order to produce human motion, a human form should be used. Emulating the human musculoskeletal system will provide insight into the capabilities of bipedal and humanoid robots.

The overarching goal of this project was to build a lifelike robot that can simulate the motions of the human gait cycle. We used a biomimetic approach to incorporate human anatomy into the mechanical design. There are about sixty muscles in the lower limbs. Some major muscle groups include the gluteal muscles, the iliopsoas, the quadriceps, the hamstrings, and the muscles that act on the ankle and foot. While many of these muscles are important for producing a walking motion, we will focus on only the most important ones that will be influencing our robot design.

In Chapter 2 we will introduce the background research we conducted to inform us about this project, including previous work in bipedal robots, existing artificial muscles, and an introduction to anatomy and the gait cycle. Chapter 3 will list the project objectives that we set out to achieve, and Chapter 4 will describe the methodology we applied to our design. Chapter 5 will detail the experiments we conducted and the results we obtained. Lastly, Chapter 6 will analyze the results in the context of our objectives, as well as address improvements and future implications of this work.

Chapter 2

Background

2.1 Hydro Muscles

Hydro Muscles are a novel, soft robotic actuator developed by Professor Marko Popovic in Popovic Labs in 2014. They are used for fluidic linear actuation, in which hydraulic or pneumatic power may be used.

The structure of a Hydro Muscle consists of a smooth elastic tube encased in a sleeve of soft, inelastic material. The tube is typically made of latex, while the sleeve is typically polyester. The tube is fixed with two caps, one of which has a valve that connects the actuator to the pressure source. When pressurized, the tubing stretches longitudinally, while the inelastic sleeve prevents radial expansion. When the muscle is depressurized, the tubing constricts to an unstretched length (Figure 2.1). The prevention of radial expansion increases the efficiency of actuation.

The Hydro Muscle is more typically used for exerting pulling forces, since it can tend to bow when exerting a pushing force. This can be solved by embedding the tube with a small rigid telescoping or with pressurized granular media to increase the critical bending force.

Figure 2.1: Expansion and contraction of Hydro Muscle (Sridar et. al., 2016)

Overall, the Hydro Muscle has many favorable characteristics. It is lightweight, small, efficient, soft, fast, similar to biological muscle, and cost-effective. It can be constructed with off-the-shelf materials in less than 10 minutes, and the materials for each muscle cost less than $10. The muscles can also be pressurized with ordinary tap water at standard household pressures of 0.59 MPa, or pressurized

pneumatically with compressed air at around 100 psi. Many muscles can be actuated from one source using flow and/or pressure control (Sridar et. al., 2016).

2.2 Other Artificial Muscles

There are many other various artificial muscles, including the Pneumatic Rubber Artificial Muscle (PRAM), PneuFlex, Fiber Reinforced Actuator (FRA), and the McKibben artificial muscle. The PRAM is made of a rubber tube with a fiber bellow surrounding it. The bellows are reinforced with fiber tape to limit the tube's expansion in the axial direction. This muscle is designed to bend and curve in the direction of the reinforced side (Figure 2.2a). It has mostly been used for applications in artificial hands and grasping tools (Noritsugu et. al., 2004).

The PneuFlex functions in a similar way, with a silicone tube consisting of a passive layer and an active layer. The passive layer is reinforced with woven fabric, while the active layer is reinforced with helically wound thread. When pressurized, the active layer expands more than the passive layer, causing the muscle to bend (Figure 2.2b). This muscle is also used in prosthetic hand applications (Deimel et. al., 2013).

The FRA is a rubber tube with a semi-circle cross-section. Woven fiberglass is glued to the flat face so that the muscle curves during pressurization. The range of motion and angle of bending of the muscle can be altered with Sure-Grip heat-shrink tubing (Figure 2.2c). This sleeve is made of polyolefin/polyester and can be used to shape the muscle into a variety of motions (Galloway et. al., 2013).

The McKibben artificial muscle is widely used in prosthetic hand applications. It is made of a rubber inner tube and a helically-woven shell. When pressurized, the McKibben muscle bulges and stiffens radially while contracting axially. When depressurized, it softens radially while elongating axially (Figure 2.2d). Due to the radial expansion and contraction, there is a loss of energy from the actuation to the load (Tondu et. al., 2000). It therefore has a lower efficiency than the Hydro Muscle, which only expands in the axial direction.

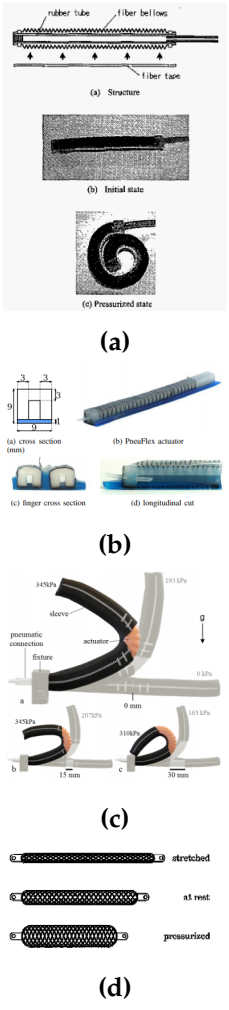

Figure 2.2: Commonly used artificial muscles.

2.3 Existing Bipedal Walking Robots

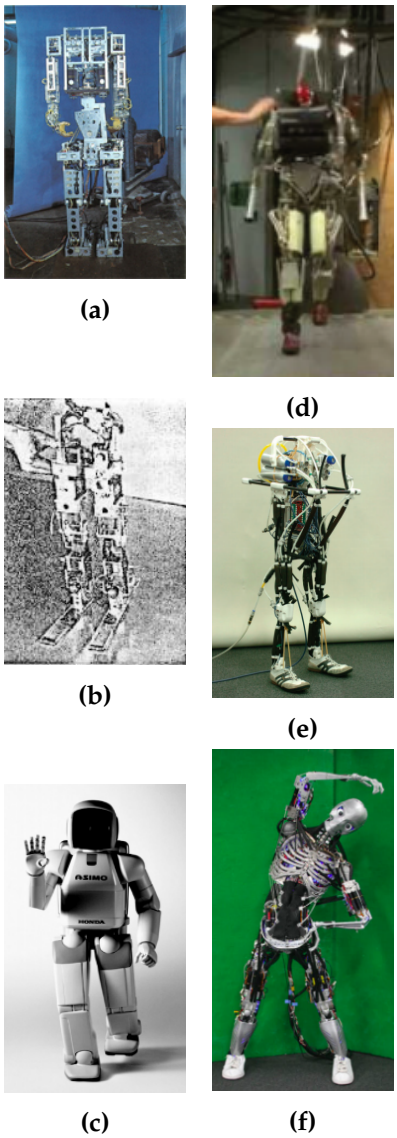

Figure 2.3: Existing bipedal walking robots.

Legged machines have existed since the late 19th century; however, the first full-scale, anthropomorphic, bipedal walking robot was created by Dr. Ichiro Kato in 1973, called WABOT 1 (Figure 2.3a) (Popovic, 2014). It was powered hydraulically and walked in a shuffling motion with disproportionately large feet (Biped humanoid robot group). This robot was statically stable.

The first bipedal robots with dynamic balance were developed by Hirofumi Miura and Isao Shimoyama in 1984 (Figure 2.3b). The BIPER robots had controlled balance based on principles of inverted pendulum motion (Popovic, 2014).

Honda's ASIMO robot was capable of walking forward, walking backward, balancing on one foot, walking up and down stairs, running, and more (Figure 2.3c). However, ASIMO did not have a very biological gait due to its strategy of keeping its center of pressure in the center of the support base. This required the robot to walk with bent knees, rather than execute a lifelike gait (Popovic, 2014).

Another advanced humanoid robot is PETMAN, which can achieve faster speeds than ASIMO and has a more biological gait (Figure 2.3d). It is lifesize and has controlled balance even when shoved (PETMAN).

At the University of Tokyo, a prototype for a pneumatically-powered musculoskeletal robot was developed (Figure 2.3e). This robot used McKibben muscles for actuation. Two models were made: the original used a biological ankle joint, while the subsequent iteration simplified the lower leg to an elastic blade (Niiyama et. al., 2018).

Another project at the University of Tokyo is the robot Kenshiro (Figure 2.3f). Kenshiro is a cable-driven humanoid mimetic robot with 100 actuators. The robot is meant to mimic realistic human joint and muscle arrangements and motions (Nakanishi et. al., 2012).

2.4 Anatomy

There are about sixty muscles in the lower limbs. Some major muscle groups include the gluteal muscles, the iliopsoas, the quadriceps, the hamstrings, and the muscles that act on the ankle and foot. While many of these muscles are important for producing a walking motion, we will focus on only the most important ones that will be influencing our robot design.

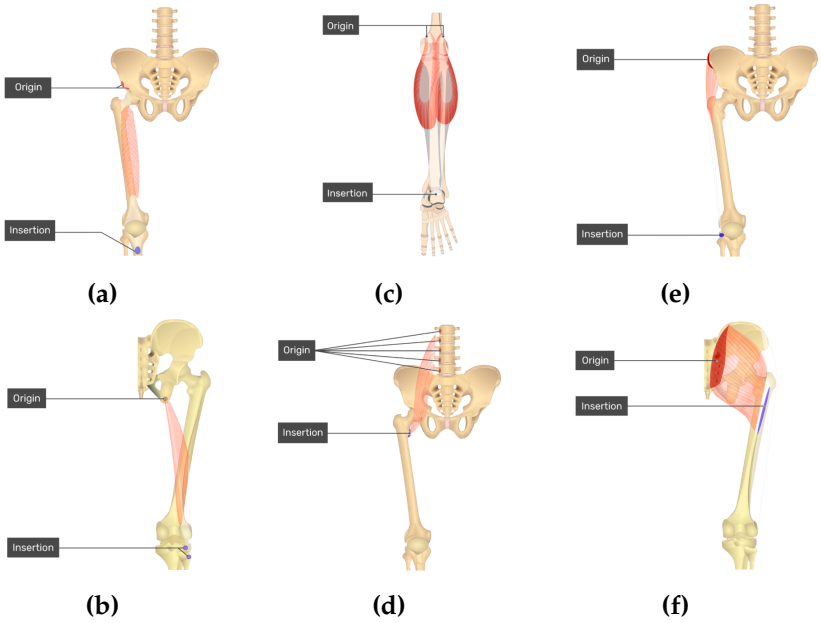

Figure 2.4: Origin and insertion points for relevant muscles.

The quadriceps femoris flexes the thigh at the hip, as well as extends the knee. The most relevant muscle in this group is the rectus femoris, which has a proximal attachment at the anterior inferior iliac spine, an insertion point at the patella, and a distal attachment to the tibial tuberosity (Figure 2.4a).

The hamstrings act antagonistically to the quadriceps. Two of the important muscles in this group are the biceps femoris and the semitendinosus muscle, which both originate at the ischial tuberosity of the ox coxa. The biceps femoris inserts at the head of the fibula, while the semitendinosus inserts at the proximal tibia (Figure 2.4b).

The gastrocnemius muscle plantar flexes the foot and flexes the knee. It has proximal attachments at the medial and lateral condyles of the femur and distally attaches to the posterior calcaneus (Figure 2.4c).

Psoas major is the main muscle of the iliopsoas. It flexes and laterally rotates the thigh at the hip. The psoas major originates from the bases of the transverse processes of L1-L5 in the spine, and it inserts at the lesser trochanter of the femur (Figure 2.4d).

The tensor fasciae latae muscle flexes and abducts the thigh at the hip. It has a

proximal attachment at the anterior iliac crest and a distal attachment at the lateral condyle of the tibia (Figure 2.4e).

The gluteus maximus is the major muscle from the gluteal group. It extends, laterally rotates, and abducts the thigh at the hip. It originates along the surface of the ilium below the posterior gluteal line and inserts at the gluteal tuberosity of the femur and iliotibial tract (Figure 2.4f).

All figures and muscle information was informed by getbodysmart.com (Muscular System).

2.5 Gait Cycle

2.5.1 Overview

The gait cycle for bipedal human walking is a repetitive pattern that incorporates steps and strides, where a stride consists of two steps. A step is seen to start from the initial contact of one foot and then ending at the initial contact of the other, while a stride starts with the initial contact of one foot and ends with the initial contact of that same foot (Physiopedia, 2017). The gait cycle is made up of two steps or one stride (Bogey, 2016).

During walking, while the body is engaged in forward momentum, one limb provides support while the other limb moves forward in preparation to become the support limb (Bogey, 2016). The classification of the gait cycle is broken down into the stance phase, which takes up 60% of the gait cycle, and the swing phase, which takes up the remaining 40% (Ekka, 2016). The stance phase can be further broken down into initial double limb stance (10%), single limb stance (20%), and terminal double limb stance (10%). During double stance periods, the two limbs normally do not share the load equally (Bogey, 2016).

With different walking conditions, slight variations can occur in the overall percentages of stance and swing in the gait cycle. As walking velocity increases, the duration of the stance phase components decrease. The elimination of double stance periods is the mark of the transition from walking to running (Bogey, 2016).

Walking can be broken down to occur in a sequence of six steps: activation of the gait command within the central nervous system, transmission of the gait command to the peripheral nervous system, contraction of muscles, force generation, motion of joints, and ground reaction force generation (Physiopedia, 2017). The more indebted classification of gait can be broken up into eight phases (Physiopedia, 2017).

1. Initial Contact (Heel Strike)
2. Loading Response (Flat Foot)

3. Midstance
4. Terminal Stance (Heel Off)
5. Pre-Swing (Toe Off)
6. Initial Swing
7. Mid Swing
8. Terminal Swing

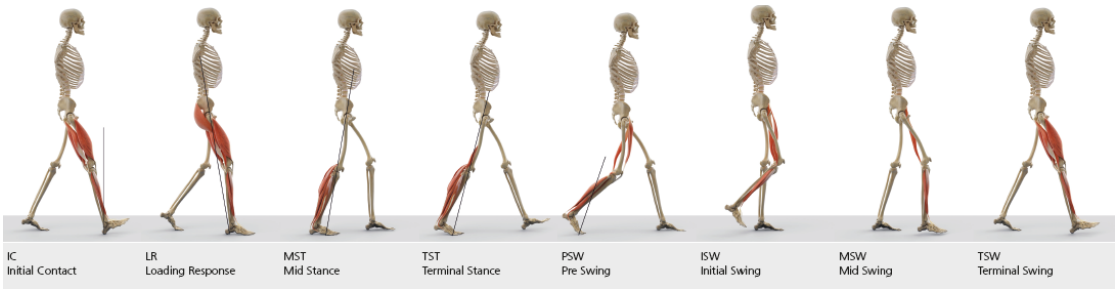

Figure 2.5: Complete gait cycle (Streifeneder, 2017).

2.5.2 Stance Phase

The stance phase consists of a sequence of five phases: Initial Contact, Loading Response, Midstance, Terminal Stance, and Pre-Swing.

Initial Contact is the first phase of double support, which begins with the heel's initial contact with the floor. The hip is at roughly 30 degrees of flexion, the knee is beginning to flex, and the ankle is neutrally positioned. At this point, the body weight is shifted towards the stance limb (Physiopedia, 2017). In this phase, muscles that see activity are the quadriceps femoris, tibialis anterior, gluteus medius, gluteus maximus, and ischiocrurale muskulatur (Streifeneder, 2017).

Loading Response occurs directly after initial contact, which is when the foot is in full contact with the floor. The knee is at a flexion of 15 degrees, and the ankles move into a plantar flexion of roughly 5-10 degrees. At this point the hip moves into extension, allowing the weight of the body to shift onto the stance limb (Physiopedia, 2017). Muscles that see activity are quadriceps femoris, tibialis anterior, gluteus medius, gluteus maximus, adductor magnus, tensor fasciae latae, tibialis posterior, and peroneus longus (Streifeneder, 2017).

In Midstance, the body passes over the stance limb, meaning the leg is approaching the vertical position. Here the knee starts to extend, the hip continues to extend, the ankle will move into slight dorsiflexion, and the trunk is seen in a neutral position of rotation (Physiopedia, 2017). In this phase, muscles that see activity are the gastrocnemius and soleus (Streifeneder, 2017).

Terminal Stance occurs when the heel is about to lift off the floor. The ankle is in dorsiflexion at the start and moves into plantar flexion. The hip is in hyperextension, and the knee is extending while preparing to flex (Physiopedia, 2017). Muscles that see activity are the gastrocnemius, soleus, flexor digitorum longus, flexor hallucis longus, tibialis posterior, peroneus longus, and peroneus brevis (Streifeneder, 2017).

Pre-Swing describes the process of the toes leaving the floor. In this time frame, the opposite foot is in its Loading Response phase. This marks the end of stance phase and the beginning of the swing phase. The toes will enter hyperextension while the ankles begin to plantar flex. Meanwhile, the knee keeps towards flexion. At the end of this phase, the hip starts flexing (Physiopedia, 2017). Muscles that see activity are the gastrocnemius, soleus, rectus femoris, and adductor longus (Streifeneder, 2017).

2.5.3 Swing Phase

The swing phase consists of three stages: Initial Swing, Mid Swing, and Late Swing.

Initial Swing is when the reference leg starts swinging forward (Ekka, 2016). The hip begins by extending to roughly 10 degrees, then flexes from the contraction of the iliopsoas muscle. The knee then flexes within a range of 40-60 degrees while the ankle moves from 20 degrees of plantar flexion towards dorsiflexion to end at a neutral position (Physiopedia, 2017). In this phase, muscles that see activity are the extensor hallucis longus, flexor hallucis longus, sartorius, iliacus and tibialis anterior (Streifeneder, 2017).

Mid Swing occurs when the reference leg passes below the body (Ekka, 2016). From the contraction of the adductors, the hip flexes to 30 degrees, while the ankle becomes dorsiflexed due to the tibialis anterior muscle contraction. The knee will initially flex 60 degrees but then will extend roughly 30 degrees from the contraction of the sartorius muscle (Physiopedia, 2017). Muscles that see activity in this phase are the semimembranosus, semitendinosus, biceps femoris, and tibialis anterior (Streifeneder, 2017).

Late Swing concludes the gait cycle, which is when the leg starts to slow down in preparation for Initial Contact (Ekka, 2016). This starts with hip flexion ranging from 25-30 degrees. The knee is extended and the ankle is in a neutral position (Physiopedia, 2017). In this phase, muscles that see activity are the quadriceps femoris, semimembranosus, semitendinosus, biceps femoris, and tibialis anterior (Streifeneder,

2017).

2.5.4 Gait Analysis and Abnormal Gait

For a successful gait cycle, the reference limb must meet three tasks: weight acceptance, single limb support, and limb advancement. Successful weight acceptance entails that the leg did not collapse from first contact, while single limb support ensures that the leg has the ability to hold up the body while in forward motion. Limb advancement is successful when the limb is able to swing forward and initiate the gait cycle again (Hernandez, 2002).

Gait analysis can help determine factors like walking speed, step length, cadence, symmetry, stability, and angle of joints (Hernandez, 2002).

An altered gait pattern due to weakness, deformities, or impairments is known as a pathological gait. Pathological gait patterns that are musculoskeletal in nature are normally caused by joint alignment, tissue imbalance, or bony abnormalities (Physiopedia, 2017). Some common neurological causes of pathological gait are Hemiplegic Gait, Diplegic Gait, Parkinsonian Gait, Ataxic Gait, Myopathic Gait, and Neuropathic Gait.

Chapter 3

Project Objectives

The scope of our project is to design, research, and develop a bio-inspired, bipedal walking robot. We will be using HydroMuscles for actuation to produce a human-like gait.

This project is focused on research and development of the application of HydroMuscles to bipedal robots. Further extensions of this project can explore this robot's potential for clinical, prosthetic, military defense, and other applications.

Our objectives for this project are as follows:

1. To design a functional bio-inspired bipedal robot actuated by HydroMuscles.
2. To demonstrate the application of HydroMuscles in bipedal walking robots.
3. To design a biomimetic robot to simulate motion inspired by the phases of the human gait cycle (stance, loading, swing, heel strike).
4. To actuate a minimum of 12 HydroMuscles for motion.
5. To implement an umbilical system to store all power components in order to minimize the weight of the robot.
6. To implement a control system for balance to the extent that a supportive spring does not exert a force greater than 20% of the weight of the structure for one complete gait cycle.
7. To have the robot walk at a speed of at least 0.25 m/s (average human = 1.1 m/s), with a stride length (two complete steps) of at least 0.4 meters (average human = 0.8-1 m).
8. To have the robot initiate a stride from either foot.

9. To construct a robotic platform for future research on biomimicry and gait kinematics, as well as HydroMuscle actuation and controls.

Chapter 4

Methodology

4.1 Materials and Design

4.1.1 Hydro Muscles

The Hydro Muscles were made out of the typical materials of surgical latex tubing and polyester sheathing. We used tubing with 1/2" OD and 1/4" ID, with sheathing taken from Uber Hose that fits snugly with the tubing. We actuated the muscles pneumatically, using barbed plugs for the connection to the pneumatic tubing. Gorilla glue was used to prevent any leakage through the plugs, and clear nail polish was used to prevent the sheathing from unraveling at the ends. The sheathing was clamped to the ends of the muscles. Thick fishing line was tied at the end of the plug to allow for easy attachment to the skeleton. Figure 4.1 shows an exploded view of the materials, as well as a picture of a completed muscle.

Figure 4.1: Materials to construct Hydro Muscles.

The procedure to construct the Hydro Muscles is detailed below:

1. Cut the latex tubing to the required length, ensuring a clean, flat cut. Angled cuts may lead to leakages.

2. Cut the hose sheathing 1/4" longer than the latex tubing for clearance.

3. Insert the latex tubing into the sheathing
4. Slide worm clamps on either end of the tubing
5. Insert the barbed hose fitting and barbed plug on each end of the tubing, but not completely. Coat the edge of the fitting or plug with super glue before completely inserting.
6. Ensure that the clamps are around the tubing where the fitting or plug is inserted. Tighten the clamps as tightly as possible to prevent leakages.
7. Seal the ends of the sheathing with clear nail polish or super glue to prevent fraying.
8. Use a needle tool to create two small holes on the end of the flat surface of the barbed plug. Thread fishing wire through to create a loop for attachment to tendons.

4.1.2 Skeleton

Figure 4.2: Bungee cord ligaments.

The skeleton that we selected was the Frank model from 3B Scientific, which has constraints at the joints to allow for lifelike degrees of freedom and motion (Figure 4.3). We chose this skeleton since the lifelike motion would allow for a more accurate gait cycle when we actuated the limbs.

To attach the muscles to the skeleton, we used threaded inserts and eye hooks. The threaded inserts we used were embedded into 5/32" holes at each of the attachment points for the muscles (discussed in section 4.2.1). We then used Spiderwire to tie the ends of the muscles to their respective eye hooks. The threaded inserts allowed us to easily attach and detach individual muscles during testing.

While the skeleton's motion was very lifelike, it was very bow-legged at rest (Figure 4.4c). This would impair the robot's ability to simulate an accurate gait, so we used bungee cords to rotate the hips out. We attached eye hooks to points on the skeleton so that the bungee cord would function similarly to the iliofemoral and ischiofemoral ligaments (Figure 4.4a-b). This was successful in rotating the hips so that the legs were parallel to each other (Figure 4.4d).

Figure 4.3: Eye hook and threaded insert.

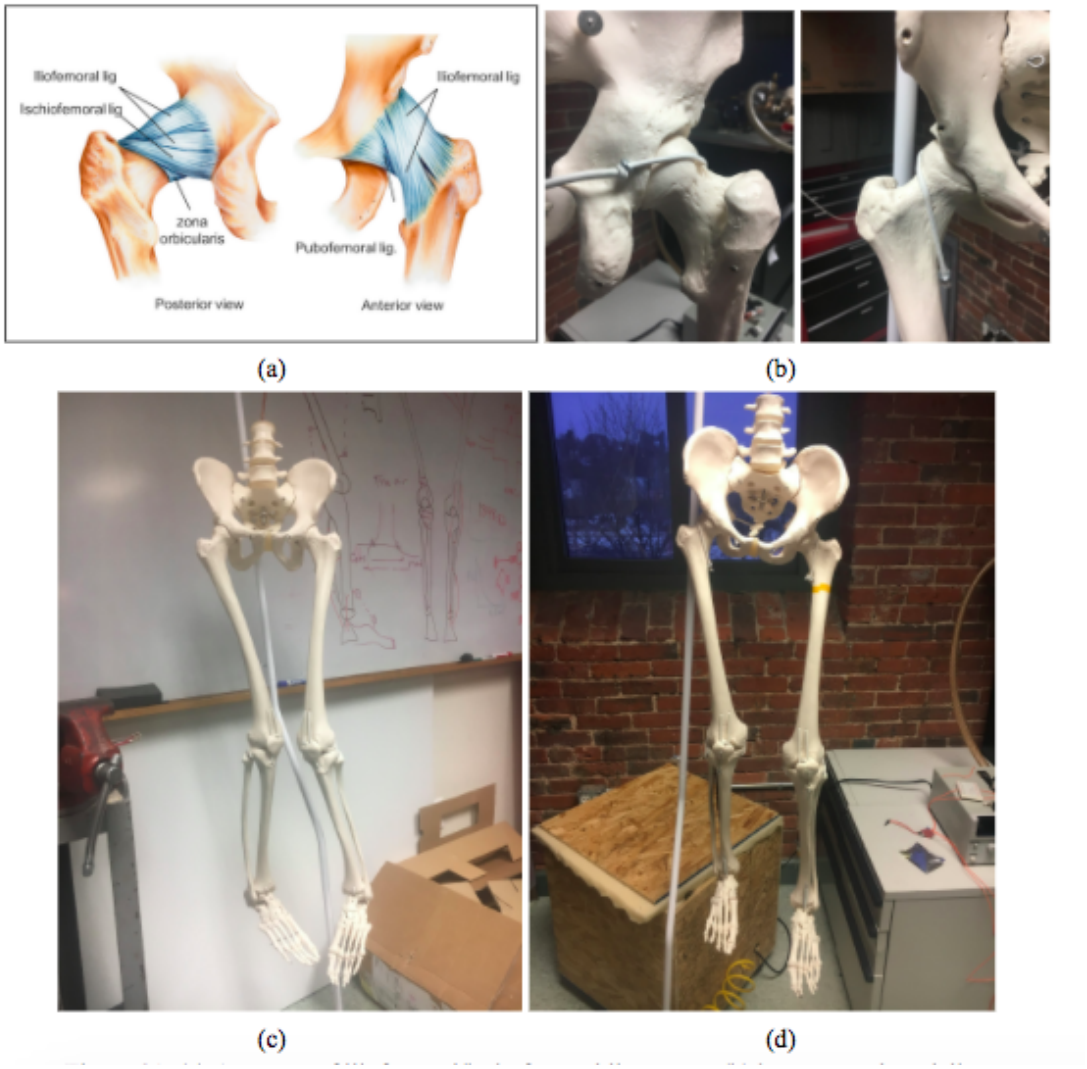

Figure 4.4: Skeleton components.

4.1.3 Umbilical System

Our umbilical system consisted of the pneumatic system and the structural system. We used a compressed air tank capable of up to 150 psi, for which we constructed a sound-reducing box for use in the lab (Figure 4.5). We operated the air tank at 100 psi for our purposes, since this pressure will give strong forces but will not cause the muscles to burst.

Figure 4.5: Air compressor tank.

Figure 4.6: Manifold.

We used 1/4" diameter pneumatic tubing and push connects for easy connections. The push connects were either single or branched for doubled-up muscles. To streamline the connections from the air tank to the twelve muscles we are actuating, we used a manifold with one inlet and several outlets (Figure 4.6). This allowed for a more neatly assembled umbilical system, as well as reduced the risk of pressure loss through long lengths of tubing.

The equipment was placed on a stand made from 80/20 T-slotted aluminum framing. A platform on the top held all of the pneumatic and electrical equipment needed to actuate the muscles. A load cell was also bolted to the framing to hold up the skeleton on a treadmill. We used a treadmill since the pneumatic system is not easily movable, and it provides a consistent speed for gait. The treadmill we chose is foldable for easy transportation, and operates at 500W. The belt is 14" wide and 39.4" long. It has a speed range of 1 km/h (0.28 m/s) to 10 km/h (2.78 m/s). This meets our minimum objective of a walking speed of 0.25 m/s.

4.2 Muscles

4.2.1 Placement

The amount of muscles in the lower limbs of humans is too extensive to be completely modeled using the HydroMuscles. For this reason, the muscles must be simplified while still retaining anatomical integrity. The muscles that are most active during walking were selected, each analyzed for origin, insertion, length, force generation, and musculoskeletal function to be simplified into the segmental model provided by the HydroMuscles. There are six muscles total to be modeled, including the iliopsoas,

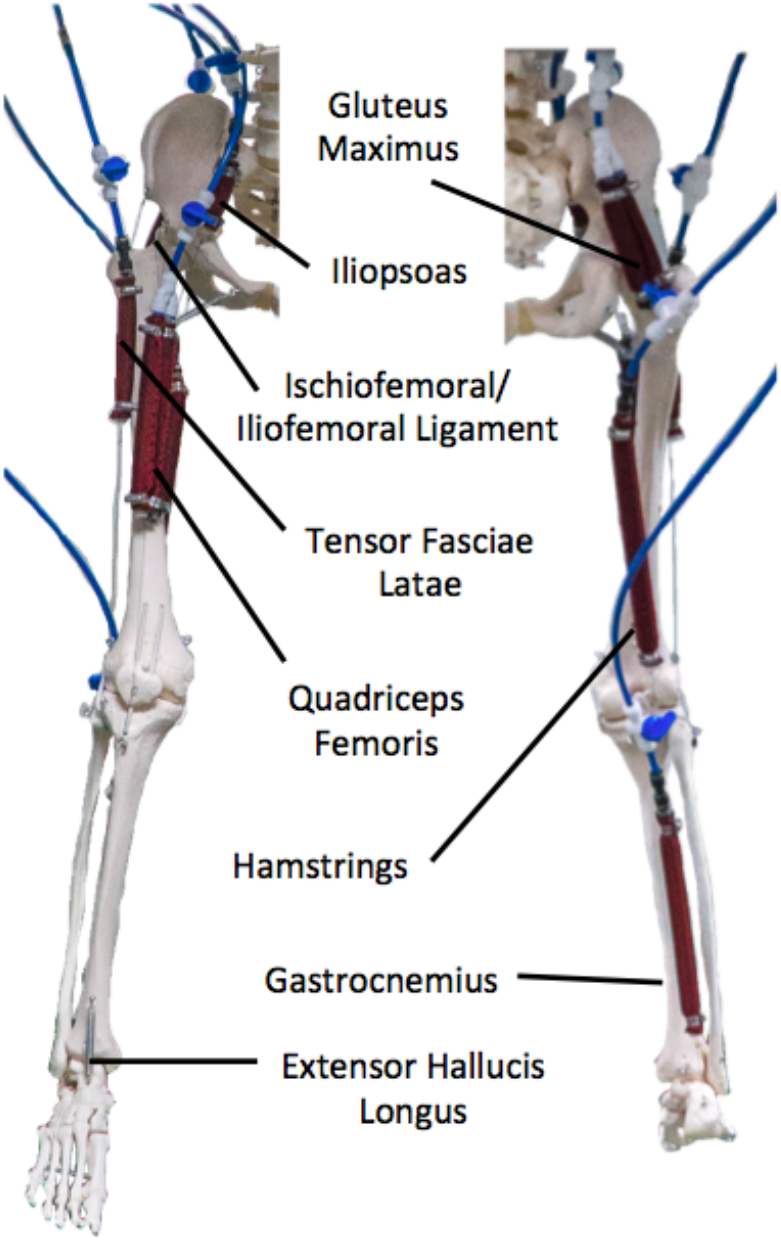

Figure 4.7: Muscle placements.

tensor fasciae latae, quadriceps femoris, gluteus maximus, hamstrings (biceps femoris and semitendinosus), and gastrocnemius. There are also passive spring structures to model the extensor hallucis longus, as well as the iliofemoral and ischiofemoral ligaments. A general model of these muscles on the skeleton is shown in Figure 4.7.

Table 4.1 details the function of each of these muscles on joint motion when they are expanded versus contracted.

Table 4.1: Muscle functions.

Muscle	Position	Joint	Action (Elongated)	Action (Contracted)
Iliopsoas	Anterior	Hip	Extend	Flex
Tensor Fasciae Latae	Anterior	Hip	Extend	Flex
Quadriceps Femoris	Anterior	Hip (Knee)	Extend (Flex)	Flex (Extend)
Gluteus Maximus	Posterior	Hip	Flex	Extend
Hamstrings	Posterior	Knee	Extend	Flex
Gastrocnemius	Posterior	Knee	Extend	Flex

4.2.2 Dimensions and Force Calculations

In order to determine the dimensions of each muscle, we had to determine the desired contraction lengths. We found the range of joint angles for the hip, knee, and ankle during walking; these values can be found in Table 4.2.

Table 4.2: Joint ranges of motion.

Joint	Flexion (Degrees)	Extension (Degrees)
Hip	25	-18
Knee	-60	0
Ankle	-16	10

For each muscle, the relevant joints were moved to the flexion and extension angles to determine the desired change in length. We used a linear regression to determine the relationship between the latex tube's original length and its change in length (Appendix A). The force applied by each muscle was also found. The force calculation accounted for the change in cross-sectional area of the inside of the tube, as well as for

the change in wall thickness as the tubing stretches. The force is represented by the following equations:

$$F(p, \epsilon) = p[A_M - c(\epsilon)A_{RW}] - Ec(\epsilon)A_{RW}\epsilon \tag{4.1}$$

$$c(\epsilon) = [1 - (1 + \epsilon)^{-0.5}]^2 \tag{4.2}$$

In equation 4.1, A_M is the muscle area defined by the outer diameter, A_{RW} is the cross-sectional area of the contracted latex tubing, E is the Young's modulus of the tube (1.34 MPa), p is the pressure (100 psi), and ε is the strain. Equation 4.2 shows the factor of strain that affects the force of the muscle (Sridar et. al., 2016). The following table shows the contracted length and contraction force of each muscle.

Table 4.3: Initial length and force of each muscle.

Muscle	Contracted Length (in)	Force (lb)
Iliopsoas	2.00	17.27
Tensor Fasciae Latae	2.30	16.74
Quadriceps Femoris	5.90	16.22
Gluteus Maximus	2.23	17.37
Hamstrings	5.90	16.22
Gastrocnemius	4.07	16.83

4.3 Sensors and Control

4.3.1 Inertial Measurement Units and Conductive Rubber

In order to be able to control the robot, the team needed some way to know how it was oriented throughout the stages of the gait cycle. One way we did this was by using IMUs. The IMUs we used had built-in gyroscopes and accelerometers, allowing us to use public code written by Jeff Rowberg (Appendix B). Some modifications were made in order to allow multiple IMUs to be used on one microcontroller. The IMUs communicated using I2C. In Rowberg's code, there are multiple different outputs that are possible with our IMUs, and we chose to calculate the pitch and roll. We attached the IMUs just below the knee due to the larger change of angle relative to the ground. Once we had the IMUs attached, we had to figure out what threshold values we needed. These values would let the microcontroller know when it should change

certain valves to continue the stages of the gait cycle. These values were determined by testing the robot in the air, and were then adjusted with fine tuning when the robot was in contact with the treadmill.

Conductive rubber stretch sensors were used in a simpler fashion. They were placed on joints that only have two degrees of freedom: behind the knee and from the ankle to the calf. In our model, the ankle only moved up or down, making measuring the angle much simpler. The sensors are given a voltage, and when they are stretched, the measured voltage increases. This is how we determined what position the legs were in, and therefore what phase of the gait cycle the robot was in.

4.3.2 5-to-2-Way Solenoid Valves

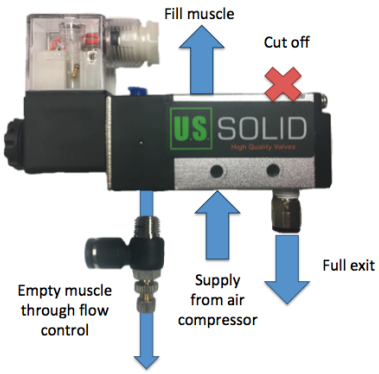

Figure 4.8: Solenoid valve.

A solenoid valve is an interface between pneumatic systems and electronic controllers. It is seen as a switch to send air to any pneumatic device, which can allow pneumatic control of temperature, flow, position, and pressure. The type we utilized for this project was a 5/2 valve (Figure 4.8). This means that it has five ports: one supply port for the air inlet to supply pressure, two exhaust ports for air to be regulated during its flow out, two ports that power the system, and two flow positions (US Solid, 2018).

We used these valves to apply a state machine system in which the valves would contract or expand specific muscles depending on what phase of the gait cycle the robot found itself in. Connected to an Arduino, this allowed us to control the individual Hydro Muscles attached to the skeleton. The reason we chose to use the 5 way 2 position valves was for the possibility of using one valve to actuate antagonistic muscles. We did not implement these antagonistic pairs in our design, but it will be possible for future iterations.

4.3.3 Flow Control Valves

In order to control the speed in which the muscles would contract we used flow control valves. A flow control valve regulates the pressure or flow of air coming out of the hose side of the valve (Figure 4.9). This device allows us to better control the motion of the skeleton, allowing us to create a smoother gait cycle. Each muscle was tested individually to see what the correct position of

Figure 4.9: Flow control valve.

the flow control valve should be for each muscle. These values were used as a baseline since the muscle was acting alone. Once other muscles were working together, the team realized that the motion was too fast and the flow rate needed to be slowed down to accommodate for extra force created from the other muscles.

4.3.4 Arduino Mega

The board we decided to use was the Arduino MEGA 2560. The reason we chose this board was because of the 54 digital I/O pins. Each valve needs its own digital pin for the MOSFET drivers, and each motor also needed its own digital port as well. Using this board allows future projects the ability to expand without having to find a new board and rewire the current setup. Figure 4.10 lists the spec sheets for the Arduino MEGA.

Technical Specs

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
USB Host Chip	MAX3421E
Length	101.52 mm
Width	53.3 mm
Weight	36 g

Figure 4.10: Arduino MEGA Specifications.

4.4 State Machine for Gait Actuation

With the human gait cycle being a series of stages, our team decided to use a state machine. The state machine consisted of each stage of the gait cycle, as well as a resting state where all of the muscles were expanded. This made testing much easier because it allowed us to see certain stages repeatedly, without interference from other stages. Once we had a stage moving like we wanted it to, we moved onto the next stage. The next step was to get fluid motion throughout the gait cycle. We did this by slowing all of the flow control valves, as well as using the sensors to determine when the position was at the correct point, meaning it was time to switch to the next state. Using a state machine made debugging much simpler because of the ability to isolate certain stages.

Chapter 5

Experiments

The following section describes iterations of testing performed in order to develop the movement of the actuated legs, developing the device from a single moving leg into a fully walking system.

The setup procedure in order to use the robot is as follows:

1. Turn on the compressor and set it to 90-100 psi, ensuring that it is connected to the pneumatic system located at the top of the stand.
2. Power the arduino by USB with a computer.
3. Supply 12V of power to the solenoid valves.
4. Release the cut offs from the valves to the muscles, allowing them to go to the standby state in which all muscles are expanded.
5. Attach muscles to corresponding threaded inserts (Note: ensure that cutoffs to the muscles are closed before altering the program or shutting down power, as the valves will turn off and all muscles will contract simultaneously).

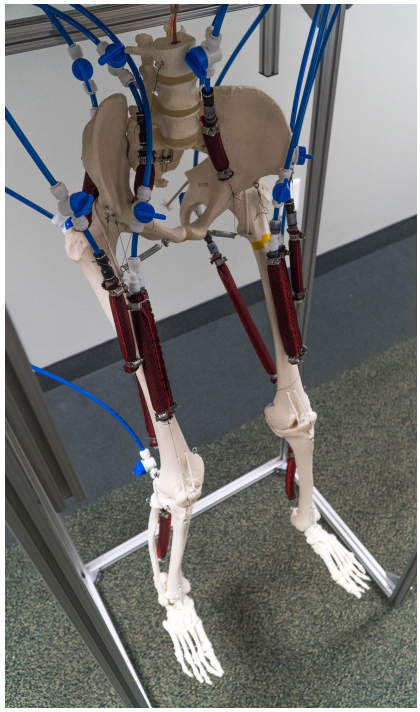

Figure 5.1: Setup of robot.

5.1 Single Leg Suspended Actuation

The robot was suspended in air from the load cell,
attached to the stand for motion and actuation testing. The purpose of the test was to determine if the single leg could achieve the required motion of the gait cycle for a single step. The robot hips were held vertically to minimize swaying. Since only one leg was actuated, only six of the twelve solenoid valves were in use for this experiment. The uploaded state machine allowed the leg to be taken through each phase of the gait cycle. Each time a button was pushed, the leg would move to the next position in the gait cycle for a single step.

After the setup procedure was completed, the experiment progressed as follows:

Starting from heel strike, the operator clicked the button connected to the breadboard. This allowed the operator to cycle through the gait cycle phase by phase. At each phase, the operator examined the position of the single leg and compared it to the model of a single leg gait for that specific phase to determine if the leg was in the proper position. If the leg was found to not be in the proper position, then muscles would be expanded, contracted, or changed so that the leg would more represent a human gait.

Utilizing this simple configuration, the muscle position, leg extension, and actuation speed were adjusted and observations were recorded and analyzed. Muscle actuation for a single leg was successful. The operator was able to actuate all muscles in the single leg, and air was able to enter and leave the system easily. The leg was able to perform the basic movements of a single leg for a human basic gait. However, the leg did not seem to be have enough toe clearance during swing.

5.2 Double Leg Suspended Actuation

The robot, similarly to Experiment 1, was suspended in mid air from the load cell for actuating tests utilizing the pneumatic system. The purpose of this experiment was to determine if the robot could achieve the required motion of the gait cycle for a single stride, encompassing one step from each leg. Since both legs were being actuated, the robot was left to swing in mid air off the load cell. Further, since both legs were in use for this experiment, all twelve solenoid valves were operated. The uploaded state machine allowed the legs to be taken through each phase of the gait cycle. Again, each time the button was pushed, the legs would move to the next position in the gait cycle for a single stride.

After the setup procedure was completed, the experiment progressed as follows:

Starting from heel strike, the operator clicked the button connected to the breadboard. This allowed the operator to cycle through the gait cycle phase by phase. At

each phase, the operator examined the position of both legs and compared it to the model in the gait cycle for that specific phase. Muscles were adjusted if the position did not resemble the model closely enough. Utilizing this simple configuration, the muscle position, leg extension, and actuation speed were adjusted and observations were recorded and analyzed.

Muscle actuation of both of the legs were successful. The system allowed all the muscles to be actuated and a basic gait cycle was performed in mid air. However, the swaying of the robot in mid air made it difficult to determine if there was enough toe clearance through the whole gait cycle.

5.3 Movement on Treadmill

The robot was positioned so that the feet made contact with the treadmill, while still having the spine tethered to the load cell. The purpose of this experiment was to have the legs walk on the treadmill to perform the basic motion of the gait cycle. The hips were lightly tethered to the side of the stand in order to prevent rotation of the skeleton. The new uploaded state machine code allowed the legs to perform multiple gait cycles in a row in order to allow the skeleton to walk in time with the treadmill. State transitions were simply manually timed after several initial tests.

After the setup procedure was completed, the experiment progressed as follows:

Starting from heel strike, the operator clicked the button, which initiated the gait cycle starting with heel strike. It was then up to the operator to adjust the speed of the legs to walk in time with the treadmill, prevent the feet from dragging, and determine if any muscles needed to be altered in order for the robot to best mimic the human gait cycle in time with the treadmill.

Where initially the legs were found to drag during the swing phase, alteration of the tensor fasciae allowed for the required clearance to move more fluidly on the treadmill. The robot was able to walk in time with the treadmill at 1 km/h, while supporting itself in forward motion with assistance from the load cell.

5.4 Movement Utilizing Sensor Information

Similarly to the last experiment, the robot was positioned so that the feet made contact with the treadmill, while still having the spine tethered to the load cell. However, for this experiment, IMUs and Conductive Rubber Stretch Sensors were attached to the skeleton. The purpose of this experiment was for the legs to walk utilizing position feedback from the IMUs and stretch sensors in order to transition between states in the hopes that this would create a smoother gait cycle. Two sets of stretch sensors were

attached to both legs: one bridging the femur and the tibia to measure the knee angle, and one bridging the tibia and the heel to measure the ankle angle. The IMUs were attached to the front of the tibia about an inch under the knee to give general position feedback. Thresholds were set for each state so that the program would know when to transition to the subsequent state. These thresholds were obtained from running the treadmill and then setting the state machine to certain states. With the legs having some force acting against them, which was different from it suspending, we were able to correctly determine what the sensor values were. These values needed to change slightly throughout testing, but this was finalized after many consecutive runs of the same state.

Chapter 6

Results

The robot was able to walk through the stages of the human gait cycle with relatively smooth motion using sensor feedback. The six muscles that were selected were effective in moving the legs to the desired positions for each phase (Figure 6.1).

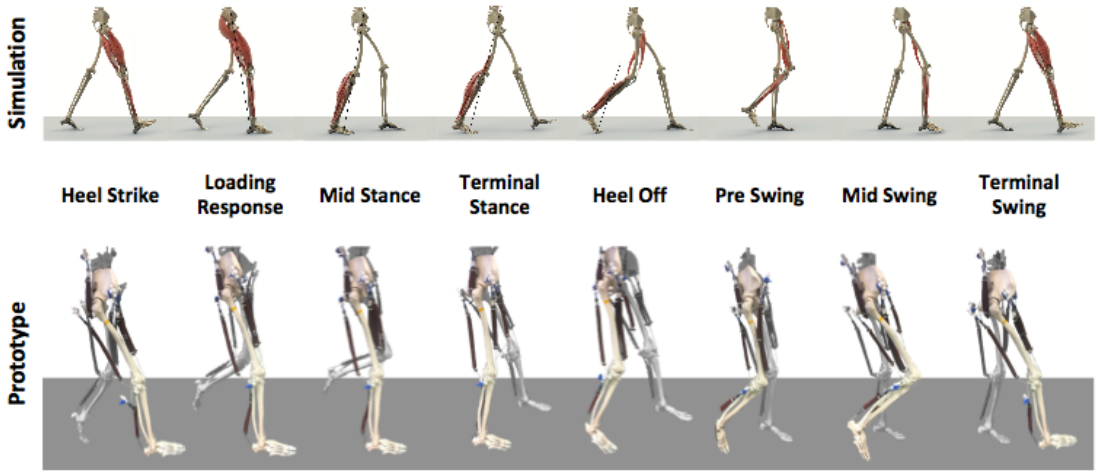

Figure 6.1: Gait cycle stages.

No balance control was implemented. The skeleton was able to support its own weight when the legs were straight, but it could not balance on its own during the entire gait cycle. The hips were lightly tethered to prevent rotation during walking.

The IMU and stretch sensor thresholds can be viewed in the code (Appendix B).

Chapter 7

Conclusion

While trying to anatomically resemble the muscles used in human walking, we were able to create a sufficient demonstration of Hydro Muscles actuating a bipedal robot to create a biomimetic walking gait cycle. We showed that there are strides to be made in the direction of biomimicry in robotics.

Some immediate next steps in continuing to improve this project would be to make small adjustments to the muscle lengths and tendon lengths to truly refine the motion to the best of the ability of the current build. Seeing as we achieved the motion so late in the timeline of our project, it is relatively rudimentary when looking to the future. This means that there is room for improvement to be made in the motion of the robot even in the current state of the build.

Another consideration that would have a positive impact on the motion would be to make the tendon material more biomimetic. We are currently using braided fishing line, which worked for our purposes as it could handle high bursts of tension. However, using an elastic material would make the motion more smooth, and attaching the material used for tendons over an area instead of a point would better define the degrees of freedom of the joints and make the motion more lifelike.

The next great consideration would be balance. We designed the build of the skeleton to easily allow for implementation of a balance system since we had originally planned to address it in more depth. As we got closer to completing our project, our group, along with our advisors, concluded that implementing a balance system with feedback and control would be a lofty feat. It was therefore decided that this would be outside the scope of our project. Motion was more heavily emphasized within the scope of our project, as our group focused more on the mechanical build and anatomical mimicry.

Another consideration to make regarding balance would be to somehow implement something that resembles the upper body in either mass or balance manage-

ment. Two-thirds of the body mass is typically located at about two-thirds of the body height. This means that the body acts similarly to an "inverted pendulum," which is unstable when the body is perturbed (Winter et. al., 1990). Without this extra body mass above the hips, our model currently needs attachment points at the hips in order to maintain orientation.

Once the use of Hydro Muscles is perfected, there are many applications such as assistive devices or replacements for human muscles. Since Hydro Muscles are soft, it is safer and more comfortable to wear than hard, rigid structures. Hydro Muscles can provide humans with the ability to move and feel with the same functionality as human muscles.

References

Biped humanoid robot group. Retrieved from http://www.takanishi.mech.waseda.ac.jp/top/research/wabian/previous_reserch/previous_research.htm

Blazquez, I. (2002), Dec 3-last update, Foot Pronation and Supination. Retrieved from http://www.angelfire.com/la/Ivan/gait.html

Bogey, R. (2016), Aug 17-last update, Gait Analysis. Retrieved from https://emedicine.medscape.com/article/320160-overview

Deimel, R., & Brock, O. (2013). "A compliant hand based on a novel pneumatic actuator," 2013 IEEE International Conference on Robotics and Automatic (ICRA), pp. 2039-2045. Retrieved from https://www.robotics.tu-berlin.de/fileadmin/fg170/Publikationen_pdf/2013-icra13_Deimel_Brock.pdf

Ekka, S.S. (2016), Concept of Gait Cycle Explained. Retrieved from https://www.physiocapsule.com/2016/10/concept-of-gait-cycle-explained.html

Galloway, K. C., Polygerinos, P., Walsh, C. J., & Wood, R. J. (2013). "Mechanically programmable bend radius for fiber-reinforced soft actuators," IEEE. Retrieved from https://pdfs.semanticscholar.org/cdee/a0c1d123bf32c8ef5c1782dfe0346c4446c8.pdf

Lewis, C. L., & Sahrmann, S. A. (2016) "Effect of posture on hip angles and moments during gait," Man Ther, pp. 1-15. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4286484/

Muscular system. Retrieved from https://www.getbodysmart.com/muscular-system

Nakanishi, Y., & Asano, Y. (2012) "Design concept of detail musculoskeletal humanoid 'Kenshiro' - Toward a real human body musculoskeletal simulator," IEEE-RAS International Conference on Humanoid Robots. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6651491

Niiyama, R., & Kuniyoshi, Y. (2018). "A pneumatic biped with an artificial musculoskeletal system." Retrieved from http://www.isi.imi.i.u-tokyo.ac.jp/~niiyama/pdf/Niiyama2008_AMAM2008_Biped.pdf

Noritsugu, T., Yamamoto, H., Sasaki, D., & Takaiwa, M. (2004). "Wearable

power assist device for hand grasping using pneumatic artificial rubber muscle," SICE Annual Conference in Sapporo, pp 420-425. Retrieved from https://pdfs.semanticscholar.org/4a7e/23f64fd5a645dfef7b5d65151b7f2f6a634d.pdf

PETMAN (Protection ensemble test mannequin) humanoid military robot. Retrieved from https://www.army-technology.com/projects/petman/

Physiopedia (2017), Gait. Retrieved from https://www.physio-pedia.com/Gait

Sridar, S., Majeika, C. J., Schaffer, P., Bowers, M., Ueda, S., Barth, A. J., Sorrells, J. L., Wu, J. T., Hunt, T. R., & Popovic, M. (2016). "Hydro Muscle - a novel soft fluidic actuator," 2016 IEEE International Conference on Robotics and Automation (ICRA), pp 4104-4021. Retrieved from http://users.wpi.edu/~mpopovic/pages/ICRA16_2771_FI_Final.pdf

Streifeneder (2017), The eight phases of human gait cycle. Retrieved from https://www.streifeneder.com/downloads/o.p./400w43_e_poster_gangphasen_druck.pdf

Tondu, B., & Lopez, P. (2000). "McKibben artificial muscle robot actuators," IEEE Control Systems Magazine, pp. 15-36. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=833638

US Solid. (2018). 1/4" 5 way 2 position pneumatic electric solenoid valve DC 24 V. Retrieved from https://ussolid.com/1-4-5-way-2-position-pneumatic-electric-solenoid-valve-dc-24-v.html.html

Verrelst, B., Ham, R. V., Vanderborght, B., Lefeber, D., & Daerden, F. (2006). "Second generation pleated pneumatic artificial muscle and its robotic applications," Advanced Robotics. Retrieved from https://www.semanticscholar.org/paper/Second-generation-pleated-pneumatic-artificial-mus-Verrelst-Ham/94cf5b394b7b3e36dfb37464b551b85f00aed044

Winter, D. A., Ruder, G. K., & MacKinnon, C. D. (1990) "Control of balance of upper body during gait," Multiple Muscle Systems, pp. 534-535. Retrieved from https://link.springer.com/chapter/10.1007%2F978-1-4613-9030-5_33#citeas

Appendix A

Muscle Length Linear Regression

The relationship between initial contracted length and change in length to the total expanded length is linear. Figure A.1 shows this regression line for several tested muscle lengths.

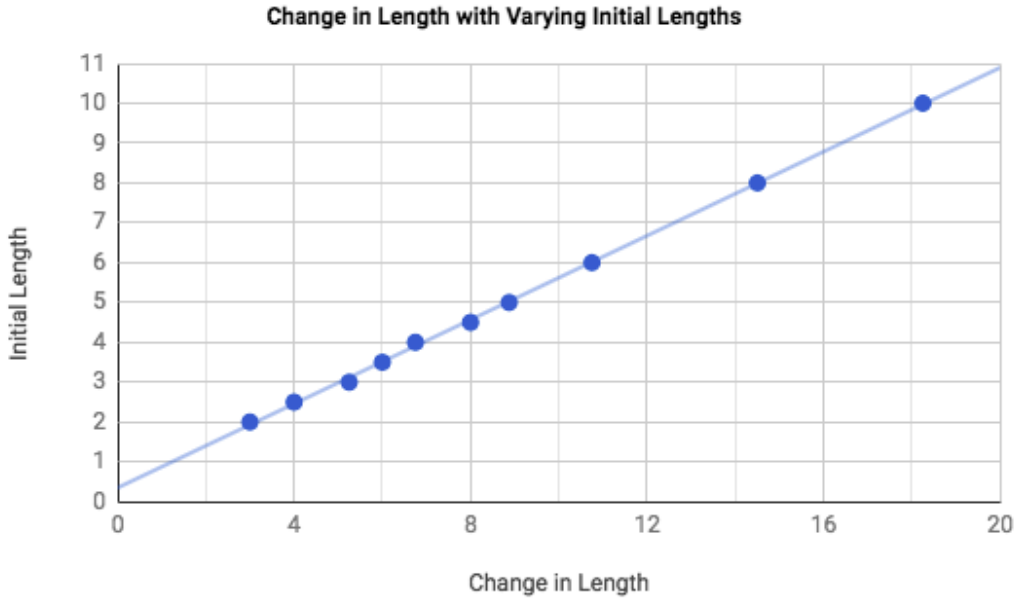

Figure A.1: Linear regression line for initial length versus change in length.

The table below shows the dimensions found from the linear regression for each muscle.

Table A.1: Muscle lengths found from linear regression.

Muscle	Change in Length (in)	Initial Length (in)	With 20% Safety Factor (in)
Iliopsoas	0.83	1.66	2.00
Tensor Fasciae Latae	1.05	1.91	2.30
Quadriceps Femoris	3.75	4.92	5.90
Gluteus Maximus	1.00	1.86	2.23
Hamstrings	3.75	4.92	5.90
Gastrocnemius	2.38	3.39	4.07

Appendix B

Arduino Code

```
//#include <FiniteStateMachine.h>
#include <Servo.h>
#include <I2Cdev.h>

#include "helper_3dmath.h"
//#include "MPU6050.h"
#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050_9Axis_MotionApps41.h"

// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// 2016-05-14 github.com/eadf
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2016-05-14 - First revision
//Below is copyright from jrowberg IMU code from GitHub
/* ============================================
  I2Cdev device library code is placed under the MIT license
  Copyright (c) 2012, 2016 Jeff Rowberg
  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:
  The above copyright notice and this permission notice shall be included in
  all copies or substantial portions of the Software.
  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
  THE SOFTWARE.
  ===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050_Wrapper.h"
#include "TogglePin.h"
#include "DeathTimer.h"

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
```

```
#include "Wire.h"
#endif
#define OUTPUT_READABLE_PITCHROLL
#ifdef OUTPUT_TEAPOT
// Teapot demo can only output from one MPU6050
const bool useSecondMpu = false;
MPU6050_Array mpus(1);
#else
const bool useSecondMpu = true;
MPU6050_Array mpus(useSecondMpu ? 2 : 1);
#endif
#define AD0_PIN_0 38  // Connect this pin to the AD0 pin on MPU #0 WHICH IS RIGHT LEG
#define AD0_PIN_1 39  // Connect this pin to the AD0 pin on MPU #1 WHICH IS LEFT LEG
#define OUTPUT_SERIAL Serial

#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;         // [w, x, y, z]         quaternion container
VectorInt16 aa;       // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;   // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;  // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;  // [x, y, z]            gravity vector
float euler[3];       // [psi, theta, phi]    Euler angle container
float ypr[3];         // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '\r', '\n' };

TogglePin activityLed(LED_PIN, 100);
DeathTimer deathTimer(5000L);

Servo myservo;

#define rubberCordLK A9 //90 for mid swing and 115 for terminal
#define rubberCordRK A8 //128 for mid swing and 178 for terminal OR 104 and 120 when left for a while
#define rubberCordRH A10
#define rubberCordLH A11

int ledPin = 25; // choose the pin for the LED
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status
int ledFade = 26;
//int boardLED = 13;
int button = 29;
int currentTime = 0;
int buttonClicks = 0;
int testValvePin = 6;
//muscle pins
int lIliopsoas = 24;
int lGlute = 25;
int lHamstring = 28;
int lGastro = 26;
int lTensor = 23;
int lQuad = 27;
int rIliopsoas = 6;
int rGlute = 9;
int rHamstring = 11;
int rGastro = 10;
int rTensor = 7;
int rQuad = 5; //pin 8 is BAD

int rightStretchSensorKnee = 0;
int leftStretchSensorKnee = 0;
int rightStretchSensorHeel = 0;
int leftStretchSensorHeel = 0;
int rightIMUPitchKnee = 0;
int leftIMUPitchKnee = 0;
long rightIMU = 0;
long leftIMU = 0;



#define All_Expanded 0
```

```
#define Mid_Swing_Right 1
#define Terminal_Swing_Right 2
#define Heel_Strike_Right 3
#define Stance_Right_One 4
#define Stance_Right_Two 5
#define Stance_Right_Three 6
#define Heel_Off_Right 7
#define Toe_Off_Right 8

uint8_t fsm_state = All_Expanded;


void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);  // declare LED as output
  pinMode(inPin, INPUT);    // declare pushbutton as input
  pinMode(button, INPUT);
  pinMode(lIliopsoas, OUTPUT);
  pinMode(lGlute, OUTPUT);
  pinMode(lHamstring, OUTPUT);
  pinMode(lGastro, OUTPUT);
  pinMode(lTensor, OUTPUT);
  pinMode(lQuad, OUTPUT);
  pinMode(testValvePin, OUTPUT);
  pinMode(rIliopsoas, OUTPUT);
  pinMode(rGlute, OUTPUT);
  pinMode(rHamstring, OUTPUT);
  pinMode(rGastro, OUTPUT);
  pinMode(rTensor, OUTPUT);
  pinMode(rQuad, OUTPUT);
  pinMode(rubberCordRK, INPUT);
  pinMode(rubberCordLK, INPUT);
  pinMode(rubberCordLH, INPUT);
  pinMode(rubberCordRH, INPUT);
  myservo.attach(8);

  // join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
  Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

  // initialize serial communication
  // (115200 chosen because it is required for Teapot Demo output, but it's
  // really up to you depending on your project)
  Serial.begin(115200);

  while (!Serial); // wait for Leonardo enumeration, others continue immediately

  // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
  // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
  // the baud timing being too misaligned with processor ticks. You must use
  // 38400 or slower in these cases, or use some kind of external separate
  // crystal solution for the UART timer.

  // initialize device
  Serial.println(F("Initializing I2C devices..."));
  mpus.add(AD0_PIN_0);
  if (useSecondMpu) mpus.add(AD0_PIN_1);

  mpus.initialize();

  // configure LED for output
  pinMode(LED_PIN, OUTPUT);

  // verify connection
  Serial.println(F("Testing device connections..."));
  if (mpus.testConnection()) {
    Serial.println(F("MPU6050 connection successful"));
  } else {
    mpus.halt(F("MPU6050 connection failed, halting"));
  }
```

```
  // wait for ready
  Serial.println(F("\nSend any character to begin DMP programming and demo: "));
  while (Serial.available() && Serial.read())
    ; // empty buffer
  while (!Serial.available())
    activityLed.update(); // flash led while waiting for data
  while (Serial.available() && Serial.read())
    ; // empty buffer again
  activityLed.setPeriod(500); // slow down led to 2Hz

  // load and configure the DMP
  Serial.println(F("Initializing DMP..."));
  mpus.dmpInitialize();

  // supply your own gyro offsets here, scaled for min sensitivity
  MPU6050_Wrapper* currentMPU = mpus.select(0);
  currentMPU->_mpu.setXGyroOffset(220);
  currentMPU->_mpu.setYGyroOffset(76);
  currentMPU->_mpu.setZGyroOffset(-85);
  currentMPU->_mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
  if (useSecondMpu) {
    currentMPU = mpus.select(1);
    currentMPU->_mpu.setXGyroOffset(220);
    currentMPU->_mpu.setYGyroOffset(76);
    currentMPU->_mpu.setZGyroOffset(-85);
    currentMPU->_mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
  }
  mpus.programDmp(0);
  if (useSecondMpu)
    mpus.programDmp(1);
}

void loop() {

  //readIMU();
  //buttonTest();
  //noControl();
  //delay(50);
  //control();
  //stretchSensor();
  //delay(50);
  //myservo.write(80);
  //servoTest(180);
  //noControlActuation();
  //digitalWrite(rHamstring, HIGH);
  //delay(50);
  //noControlActuation();
  buttonClicks = digitalRead(button);
  while (buttonClicks == 1) {
    delay(1000);
    switch (fsm_state) {
      case All_Expanded:
        buttonClicks = digitalRead(button);
        while (buttonClicks == 0) {
          buttonClicks = digitalRead(button);
          digitalWrite(lGlute, HIGH);
          digitalWrite(lQuad, HIGH);
          digitalWrite(lIliopsoas, HIGH);
          digitalWrite(lHamstring, HIGH);
          digitalWrite(lTensor, HIGH);
          digitalWrite(lGastro, HIGH);
          digitalWrite(rGlute, HIGH);
          digitalWrite(rQuad, HIGH);
          digitalWrite(rIliopsoas, HIGH);
          digitalWrite(rHamstring, HIGH);
          digitalWrite(rTensor, HIGH);
          digitalWrite(rGastro, HIGH);
          Serial.println("Waiting to Start");
        }
        fsm_state = Heel_Strike_Right;
        break;

      case Mid_Swing_Right:
        //buttonClicks = digitalRead(button);
        rightStretchSensorKnee = analogRead(rubberCordRK);
```

```
  while (rightStretchSensorKnee > 113) {
    rightStretchSensorKnee = analogRead(rubberCordRK);
    digitalWrite(lIliopsoas, HIGH);
    digitalWrite(lTensor, HIGH);
    digitalWrite(lQuad, LOW);
    digitalWrite(lGlute, LOW);
    digitalWrite(lHamstring, HIGH);
    digitalWrite(lGastro, HIGH);
    digitalWrite(rIliopsoas, LOW);
    digitalWrite(rTensor, LOW);
    digitalWrite(rQuad, HIGH);
    digitalWrite(rGlute, HIGH);
    digitalWrite(rHamstring, LOW);
    digitalWrite(rGastro, LOW);
    Serial.println("MID-SWING");
  }
  fsm_state = Terminal_Swing_Right;
  break;
case Terminal_Swing_Right:
  rightStretchSensorKnee = analogRead(rubberCordRK);
  while (rightStretchSensorKnee < 127) {
    rightStretchSensorKnee = analogRead(rubberCordRK);
    digitalWrite(lIliopsoas, HIGH);
    digitalWrite(lTensor, HIGH);
    digitalWrite(lQuad, LOW);
    digitalWrite(lGlute, LOW);
    digitalWrite(lHamstring, HIGH);
    digitalWrite(lGastro, HIGH);
    digitalWrite(rIliopsoas, LOW);
    digitalWrite(rTensor, LOW);
    digitalWrite(rQuad, LOW);
    digitalWrite(rGlute, HIGH);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rGastro, HIGH);
    Serial.println("TERMINAL");
  }
  fsm_state = All_Expanded;
  break;

case Heel_Strike_Right:
  readIMU();
  while (rightIMU < -84) {
    readIMU();
    digitalWrite(lIliopsoas, HIGH);
    digitalWrite(lTensor, HIGH);
    digitalWrite(lQuad, HIGH);
    digitalWrite(lGlute, LOW);
    digitalWrite(lHamstring, HIGH);
    digitalWrite(lGastro, LOW);
    digitalWrite(rIliopsoas, HIGH);
    digitalWrite(rTensor, LOW);
    digitalWrite(rQuad, LOW);
    digitalWrite(rGlute, HIGH);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rGastro, HIGH);
    Serial.println("HEEL STRIKE");
  }
  fsm_state = Stance_Right_One;
  break;
case Stance_Right_One:    //toe off left
  while (leftIMU > -78) {
    readIMU();
    digitalWrite(lIliopsoas, LOW);
    digitalWrite(lTensor, HIGH);
    digitalWrite(lQuad, HIGH);
    digitalWrite(lGlute, LOW);
    digitalWrite(lHamstring, LOW);
    digitalWrite(lGastro, LOW);
    digitalWrite(rIliopsoas, HIGH);
    digitalWrite(rTensor, HIGH);
    digitalWrite(rQuad, LOW);
    digitalWrite(rGlute, LOW);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rGastro, HIGH);
    Serial.println("STANCE-1");
```

```
    Serial.println("Right IMU:    ");
    Serial.print(rightIMU);
  }
  fsm_state = Stance_Right_Two;
  break;
case Stance_Right_Two:    //mid swing left
  while (leftIMU < -52) {
    readIMU();
    digitalWrite(lIliopsoas, LOW);
    digitalWrite(lTensor, LOW);
    digitalWrite(lQuad, HIGH);
    digitalWrite(lGlute, HIGH);
    digitalWrite(lHamstring, LOW);
    digitalWrite(lGastro, LOW);
    digitalWrite(rIliopsoas, HIGH);
    digitalWrite(rTensor, HIGH);
    digitalWrite(rQuad, LOW);
    digitalWrite(rGlute, LOW);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rGastro, HIGH);
    Serial.println("STANCE-2");
    Serial.println("RIGHT IMU:   ");
    Serial.print(rightIMU);
  }
  fsm_state = Stance_Right_Three;
  break;
case Stance_Right_Three:  //terminal swing left
  while (leftIMU < -71 ) {
    readIMU();
    digitalWrite(lIliopsoas, LOW);
    digitalWrite(lTensor, LOW);
    digitalWrite(lQuad, LOW);
    digitalWrite(lGlute, HIGH);
    digitalWrite(lHamstring, HIGH);
    digitalWrite(lGastro, HIGH);
    digitalWrite(rIliopsoas, HIGH);
    digitalWrite(rTensor, HIGH);
    digitalWrite(rQuad, LOW);
    digitalWrite(rGlute, LOW);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rGastro, HIGH);
    Serial.println("STANCE-3");
  }
  fsm_state = Heel_Off_Right;
  break;
case Heel_Off_Right:
  rightStretchSensorHeel = analogRead(rubberCordRH);
  while (rightStretchSensorHeel > 183) {
    rightStretchSensorHeel = analogRead(rubberCordRH);
    digitalWrite(lIliopsoas, HIGH);
    digitalWrite(lTensor, LOW);
    digitalWrite(lQuad, LOW);
    digitalWrite(lGlute, HIGH);
    digitalWrite(lHamstring, HIGH);
    digitalWrite(lGastro, HIGH);
    digitalWrite(rIliopsoas, HIGH);
    digitalWrite(rTensor, HIGH);
    digitalWrite(rQuad, HIGH);
    digitalWrite(rGlute, LOW);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rGastro, LOW);
    Serial.println("HEEL OFF");
  }
  fsm_state = Toe_Off_Right;
  break;
case Toe_Off_Right:
  while (rightStretchSensorHeel > 174) {
    rightStretchSensorHeel = analogRead(rubberCordRH);
    digitalWrite(lIliopsoas, HIGH);
    digitalWrite(lTensor, HIGH);
    digitalWrite(lQuad, LOW);
    digitalWrite(lGlute, LOW);
    digitalWrite(lHamstring, HIGH);
    digitalWrite(lGastro, HIGH);
    digitalWrite(rIliopsoas, LOW);
```

```
          digitalWrite(rTensor, HIGH);
          digitalWrite(rQuad, HIGH);
          digitalWrite(rGlute, LOW);
          digitalWrite(rHamstring, LOW);
          digitalWrite(rGastro, LOW);
          Serial.println("TOE OFF");
        }
        fsm_state = Mid_Swing_Right;
        break;
    }
  }
}
void noControl() {
  val = digitalRead(button);  // read input value
  if (val == LOW) {          // check if the input is HIGH (button released)
    //digitalWrite(testValvePin, LOW); // turn LED OFF
    digitalWrite(lGlute, HIGH);
    digitalWrite(lQuad, HIGH);
    digitalWrite(lIliopsoas, HIGH);
    digitalWrite(lHamstring, HIGH);
    digitalWrite(lTensor, HIGH);
    digitalWrite(lGastro, HIGH);
    digitalWrite(rGlute, HIGH);
    digitalWrite(rQuad, HIGH);
    digitalWrite(rIliopsoas, HIGH);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rTensor, HIGH);
    digitalWrite(rGastro, HIGH);
  } else {
    //digitalWrite(testValvePin, HIGH);// turn LED ON
    digitalWrite(lGlute, HIGH);
    digitalWrite(lQuad, HIGH);
    digitalWrite(lIliopsoas, LOW);
    digitalWrite(lHamstring, LOW);
    digitalWrite(lTensor, HIGH);
    digitalWrite(lGastro, HIGH);
    digitalWrite(rGlute, LOW);
    digitalWrite(rQuad, LOW);
    digitalWrite(rIliopsoas, LOW);
    digitalWrite(rHamstring, HIGH);
    digitalWrite(rTensor, HIGH);
    digitalWrite(rGastro, HIGH);
  }
}

//above should fill muscle all the way when button is pressed/released
//below should technically fill half way

void control() {
  // read input value
  val = digitalRead(inPin);
  if (val == HIGH) {          // check if the input is HIGH (button released)
    digitalWrite(ledFade, LOW);  // turn LED OFF
  } else {
    //digitalWrite(ledPin, HIGH);  // turn LED ON
    analogWrite(ledFade, 124);
  }
}

void fade() {
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledFade, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }

  // fade out from max to min in increments of 5 points:
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5) {
    // sets the value (range from 0 to 255):
    analogWrite(ledFade, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}
```

```
void servoTest(int rpm) {
  currentTime = millis();
  while (1) {
    while ((millis() - currentTime) <= 5050) { //527 = one turn
      myservo.write(rpm);
    }
    myservo.write(94);     //94 is stop on black motors
    Serial.println(currentTime);
    break;
  }
  delay(5000);
}
/*
  void buttonTest() {
  val = digitalRead(button);  // read input value
  if (val == HIGH) {          // check if the input is HIGH (button released)
    digitalWrite(boardLED, LOW);  // turn LED OFF
  } else {
    digitalWrite(boardLED, HIGH);  // turn LED ON
  }
  }
*/
void stretchSensor() {
  int stretchValL;
  int stretchValR;
  stretchValL = analogRead(rubberCordLH);
  //stretchValR = analogRead(rubberCordRK);
  Serial.print("Analog reading Left ");
  Serial.println(stretchValL);                  //Print value
  delay(500);
  //Serial.print("Analog reading Right ");
  //Serial.println(stretchValR);
  //delay(500);
}

void noControlActuation() {
  while (1) {
    rightStretchSensorHeel = analogRead(rubberCordRK);
    buttonClicks = 0;
    delay(500);
    while (buttonClicks == 0) {
      rightStretchSensorHeel = analogRead(rubberCordRH);
      buttonClicks = digitalRead(button);
      digitalWrite(lTensor, HIGH);
      digitalWrite(lQuad, HIGH);
      digitalWrite(lIliopsoas, HIGH);
      digitalWrite(lGlute, HIGH);
      digitalWrite(lHamstring, HIGH);
      digitalWrite(lGastro, HIGH);
      digitalWrite(rGlute, HIGH);
      digitalWrite(rQuad, HIGH);
      digitalWrite(rIliopsoas, HIGH);
      digitalWrite(rHamstring, HIGH);
      digitalWrite(rTensor, HIGH);
      digitalWrite(rGastro, HIGH);
      Serial.println("EXPANDED");
    }
    delay(500);
    buttonClicks = 0;
    while (buttonClicks == 0) {
      //rightStretchSensorKnee = analogRead(rubberCordRK);
      leftStretchSensorHeel = analogRead(rubberCordLH);
      buttonClicks = digitalRead(button); //this is for left not right, switch
      digitalWrite(lIliopsoas, HIGH);
      digitalWrite(lTensor, HIGH);
      digitalWrite(lQuad, HIGH);
      digitalWrite(lGlute, LOW);
      digitalWrite(lHamstring, HIGH);
      digitalWrite(lGastro, LOW);
      digitalWrite(rIliopsoas, HIGH);
      digitalWrite(rTensor, LOW);
      digitalWrite(rQuad, LOW);
      digitalWrite(rGlute, HIGH);
      digitalWrite(rHamstring, HIGH);
```

```
  digitalWrite(rGastro, HIGH);
  Serial.println("HEEL STRIKE");
  //Serial.println(leftStretchSensorHeel); //162
  readIMU(); // right -82 left -76
}
delay(500);
buttonClicks = 0;
while (buttonClicks == 0) {
  leftStretchSensorHeel = analogRead(rubberCordLH);
  buttonClicks = digitalRead(button);
  digitalWrite(lIliopsoas, LOW);
  digitalWrite(lTensor, HIGH);
  digitalWrite(lQuad, HIGH);
  digitalWrite(lGlute, LOW);
  digitalWrite(lHamstring, LOW);
  digitalWrite(lGastro, LOW);
  digitalWrite(rIliopsoas, HIGH);
  digitalWrite(rTensor, HIGH);
  digitalWrite(rQuad, LOW);
  digitalWrite(rGlute, LOW);
  digitalWrite(rHamstring, HIGH);
  digitalWrite(rGastro, HIGH);
  Serial.println("SANCE-1"); //left toe off
  //Serial.println(leftStretchSensorHeel); //156
  readIMU(); //right -84 left -81
  //delay(30);
}
delay(500);
buttonClicks = 0;
while (buttonClicks == 0) {
  leftStretchSensorKnee = analogRead(rubberCordLK);
  buttonClicks = digitalRead(button);
  digitalWrite(lIliopsoas, LOW);
  digitalWrite(lTensor, LOW);
  digitalWrite(lQuad, HIGH);
  digitalWrite(lGlute, HIGH);
  digitalWrite(lHamstring, LOW);
  digitalWrite(lGastro, LOW);
  digitalWrite(rIliopsoas, HIGH);
  digitalWrite(rTensor, HIGH);
  digitalWrite(rQuad, LOW);
  digitalWrite(rGlute, LOW);
  digitalWrite(rHamstring, HIGH);
  digitalWrite(rGastro, HIGH);
  Serial.println("STANCE-2");
  Serial.println("left knee");
  //Serial.println(leftStretchSensorKnee); //left mid-swing 122
  readIMU(); //right -86 left -79
  //delay(30);
}
delay(500);
buttonClicks = 0;
while (buttonClicks == 0) {
  leftStretchSensorKnee = analogRead(rubberCordLK);
  buttonClicks = digitalRead(button);
  digitalWrite(lIliopsoas, LOW);
  digitalWrite(lTensor, LOW);
  digitalWrite(lQuad, LOW);
  digitalWrite(lGlute, HIGH);
  digitalWrite(lHamstring, HIGH);
  digitalWrite(lGastro, HIGH);
  digitalWrite(rIliopsoas, HIGH);
  digitalWrite(rTensor, HIGH);
  digitalWrite(rQuad, LOW);
  digitalWrite(rGlute, LOW);
  digitalWrite(rHamstring, HIGH);
  digitalWrite(rGastro, HIGH);
  Serial.println("STANCE-3"); //left terminal swing
  //Serial.println(leftStretchSensorKnee); //125 lknee
  readIMU(); //right -84 left -71
  //delay(30);
}
delay(500);
buttonClicks = 0;
while (buttonClicks == 0) {
```

```
  rightStretchSensorHeel = analogRead(rubberCordRH);
  buttonClicks = digitalRead(button);
  digitalWrite(lIliopsoas, HIGH);
  digitalWrite(lTensor, LOW );
  digitalWrite(lQuad, LOW);
  digitalWrite(lGlute, HIGH);
  digitalWrite(lHamstring, HIGH);
  digitalWrite(lGastro, HIGH);
  digitalWrite(rIliopsoas, HIGH);
  digitalWrite(rTensor, HIGH);
  digitalWrite(rQuad, HIGH);
  digitalWrite(rGlute, LOW);
  digitalWrite(rHamstring, HIGH);
  digitalWrite(rGastro, LOW);
  Serial.println("HEEL OFF");
  Serial.println("right heel");
  //Serial.println(rightStretchSensorHeel); //183
  readIMU(); //right -83 left -71
  //delay(30);
}
delay(500);
buttonClicks = 0;
while (buttonClicks == 0) {
  buttonClicks = digitalRead(button);
  rightStretchSensorHeel = analogRead(rubberCordRH);
  digitalWrite(lIliopsoas, HIGH);
  digitalWrite(lTensor, HIGH);
  digitalWrite(lQuad, LOW);
  digitalWrite(lGlute, LOW);
  digitalWrite(lHamstring, HIGH);
  digitalWrite(lGastro, HIGH);
  digitalWrite(rIliopsoas, LOW);
  digitalWrite(rTensor, HIGH);
  digitalWrite(rQuad, HIGH);
  digitalWrite(rGlute, LOW);
  digitalWrite(rHamstring, LOW);
  digitalWrite(rGastro, LOW);
  Serial.println("TOE OFF");
  Serial.println("right heel");
  Serial.println(rightStretchSensorHeel); //174
  readIMU(); //right -71 left -68
  //delay(30);
}
/*
  delay(500);
  buttonClicks = 0;
  while (buttonClicks == 0) {
  buttonClicks = digitalRead(button);
  digitalWrite(lIliopsoas, HIGH);
  digitalWrite(lTensor, HIGH);
  digitalWrite(lQuad, LOW);
  digitalWrite(lGlute, LOW);
  digitalWrite(lHamstring, HIGH);
  digitalWrite(lGastro, HIGH);
  digitalWrite(rIliopsoas, LOW);
  digitalWrite(rTensor, LOW);
  digitalWrite(rQuad, HIGH);
  digitalWrite(rGlute, HIGH);
  digitalWrite(rHamstring, LOW);
  digitalWrite(rGastro, LOW);
  Serial.println("MID-SWING-FINAL");
  //readIMU(); //right -52 left -63
  delay(30);
  }
*/

delay(500);
buttonClicks = 0;
while (buttonClicks == 0) {
  rightStretchSensorKnee = analogRead(rubberCordRK);
  //left
  buttonClicks = digitalRead(button);
  digitalWrite(lIliopsoas, HIGH);
  digitalWrite(lTensor, HIGH);
  digitalWrite(lQuad, LOW);
```

```
      digitalWrite(lGlute, LOW);
      digitalWrite(lHamstring, HIGH);
      digitalWrite(lGastro, HIGH);
      digitalWrite(rIliopsoas, LOW);
      digitalWrite(rTensor, LOW);
      digitalWrite(rQuad, HIGH);
      digitalWrite(rGlute, HIGH);
      digitalWrite(rHamstring, LOW);
      digitalWrite(rGastro, LOW);
      Serial.println("MID-SWING");
      Serial.println("right knee");
      //Serial.println(rightStretchSensorKnee); //128
      readIMU(); //right -51 left -62 ... -66 second
      //delay(30);
    }
    delay(500);
    buttonClicks = 0;
    while (buttonClicks == 0) {
      rightStretchSensorKnee = analogRead(rubberCordRK);
      buttonClicks = digitalRead(button);
      digitalWrite(lIliopsoas, HIGH);
      digitalWrite(lTensor, HIGH);
      digitalWrite(lQuad, LOW);
      digitalWrite(lGlute, LOW);
      digitalWrite(lHamstring, HIGH);
      digitalWrite(lGastro, HIGH);
      digitalWrite(rIliopsoas, LOW);
      digitalWrite(rTensor, LOW);
      digitalWrite(rQuad, LOW);
      digitalWrite(rGlute, HIGH);
      digitalWrite(rHamstring, HIGH);
      digitalWrite(rGastro, HIGH);
      Serial.println("TERMINAL");
      Serial.println("right knee");
      //Serial.println(rightStretchSensorKnee); //130
      readIMU(); //right -84 left -66
      //delay(30);
    }
  }
}

void handleMPUevent(uint8_t mpu) {

  MPU6050_Wrapper* currentMPU = mpus.select(mpu);
  // reset interrupt flag and get INT_STATUS byte
  currentMPU->getIntStatus();

  // check for overflow (this should never happen unless our code is too inefficient)
  if ((currentMPU->_mpuIntStatus & _BV(MPU6050_INTERRUPT_FIFO_OFLOW_BIT))
      || currentMPU->_fifoCount >= 1024) {
    // reset so we can continue cleanly
    currentMPU->resetFIFO();
    Serial.println(F("FIFO overflow!"));
    return;
  }
  // otherwise, check for DMP data ready interrupt (this should happen frequently)
  if (currentMPU->_mpuIntStatus & _BV(MPU6050_INTERRUPT_DMP_INT_BIT)) {

    // read and dump a packet if the queue contains more than one
    while (currentMPU->_fifoCount >= 2 * currentMPU->_packetSize) {
      // read and dump one sample
      Serial.print("DUMP"); // this trace will be removed soon
      currentMPU->getFIFOBytes(fifoBuffer);
    }

    // read a packet from FIFO
    currentMPU->getFIFOBytes(fifoBuffer);

#ifdef OUTPUT_READABLE_QUATERNION
    // display quaternion values in easy matrix form: w x y z
    currentMPU->_mpu.dmpGetQuaternion(&q, fifoBuffer);
    OUTPUT_SERIAL.print("quat:"); OUTPUT_SERIAL.print(mpu); OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(q.w);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(q.x);
```

```
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(q.y);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.println(q.z);
#endif

#ifdef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    currentMPU->_mpu.dmpGetQuaternion(&q, fifoBuffer);
    currentMPU->_mpu.dmpGetEuler(euler, &q);
    OUTPUT_SERIAL.print("euler:"); OUTPUT_SERIAL.print(mpu); OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(euler[0] * 180 / M_PI);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(euler[1] * 180 / M_PI);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.println(euler[2] * 180 / M_PI);
#endif

#if defined(OUTPUT_READABLE_YAWPITCHROLL) or defined(OUTPUT_READABLE_PITCHROLL)
    // display Euler angles in degrees
    currentMPU->_mpu.dmpGetQuaternion(&q, fifoBuffer);
    currentMPU->_mpu.dmpGetGravity(&gravity, &q);
    currentMPU->_mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
#if defined(OUTPUT_READABLE_YAWPITCHROLL)
    OUTPUT_SERIAL.print("y");
    delay(50);
#endif
    OUTPUT_SERIAL.print("pr:"); OUTPUT_SERIAL.print(mpu); OUTPUT_SERIAL.print("\t"); //does this
#if defined(OUTPUT_READABLE_YAWPITCHROLL)
    OUTPUT_SERIAL.print(ypr[0] * 180 / M_PI);
    OUTPUT_SERIAL.print("\t");
#endif
    OUTPUT_SERIAL.print(ypr[1] * 180 / M_PI);
    rightIMU = ypr[1] *180 / M_PI;
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.println(ypr[2] * 180 / M_PI);
    leftIMU = ypr[2] * 180 / M_PI;
#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    currentMPU->_mpu.dmpGetQuaternion(&q, fifoBuffer);
    currentMPU->_mpu.dmpGetAccel(&aa, fifoBuffer);
    currentMPU->_mpu.dmpGetGravity(&gravity, &q);
    currentMPU->_mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    OUTPUT_SERIAL.print("areal:"); OUTPUT_SERIAL.print(mpu); OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(aaReal.x);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(aaReal.y);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    currentMPU->_mpu.dmpGetQuaternion(&q, fifoBuffer);
    currentMPU->_mpu.dmpGetAccel(&aa, fifoBuffer);
    currentMPU->_mpu.dmpGetGravity(&gravity, &q);
    currentMPU->_mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    currentMPU->_mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    OUTPUT_SERIAL.print("aworld:"); OUTPUT_SERIAL.print(mpu); OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(aaWorld.x);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.print(aaWorld.y);
    OUTPUT_SERIAL.print("\t");
    OUTPUT_SERIAL.println(aaWorld.z);
#endif

#ifdef OUTPUT_TEAPOT
    // display quaternion values in InvenSense Teapot demo format:
    // Note that this does not differentialte between your mpus
    teapotPacket[2] = fifoBuffer[0];
    teapotPacket[3] = fifoBuffer[1];
    teapotPacket[4] = fifoBuffer[4];
```

```
    teapotPacket[5] = fifoBuffer[5];
    teapotPacket[6] = fifoBuffer[8];
    teapotPacket[7] = fifoBuffer[9];
    teapotPacket[8] = fifoBuffer[12];
    teapotPacket[9] = fifoBuffer[13];
    OUTPUT_SERIAL.write(teapotPacket, 14);
    teapotPacket[11]++;// packetCount, loops at 0xFF on purpose
#endif

  }
}

void readIMU() {
  static uint8_t mpu = 0;
  static MPU6050_Wrapper* currentMPU = NULL;
  if (useSecondMpu) {
    for (int i = 0; i < 2; i++) {
      mpu = (mpu + 1) % 2; // failed attempt at round robin
      currentMPU = mpus.select(mpu);
      if (currentMPU->isDue()) {
        handleMPUevent(mpu);
      }
    }
  } else {
    mpu = 0;
    currentMPU = mpus.select(mpu);
    if (currentMPU->isDue()) {
      handleMPUevent(mpu);
    }
  }

  // other program behavior stuff here
  // .
  // .
  // .
  // if you are really paranoid you can frequently test in between other
  // stuff to see if mpuInterrupt is true, and if so, "break;" from the
  // while() loop to immediately process the MPU data
  // .
  // .
  // .
  //Serial.println("Reached Here");
  activityLed.update();
  deathTimer.update();
}
```