NEURAL NETWORK CONTROL OF A NEURAL PROSTHESIS TO ASSIST WITH GAIT FOR PEOPLE WITH MUSCLE WEAKNESS

A thesis proposal to the faculty of the Graduate School of
Western Carolina University in partial fulfillment of the
requirements for the degree of Master of Science in Technology

By
Pablo Valenzuela

Director: Dr. Martin Tanaka
Associate Professor
School of Engineering and Technology

Committee Members:
Dr. Paul Yanik, School of Engineering and Technology
Dr. David Hudson, Department of Physical Therapy

May 2021

Dedicado a mi mamá y mi papá,

Lourdes y Rubén Valenzuela

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family for all their support, especially my parents. Without them, I would not be the person I am today. Their guidance and encouragement have raised me to always follow my passions. I am extremely grateful for their love and the sacrifices they have made to prepare me better for the future. I would also like to thank my brothers for always keeping me with a positive mindset by providing a relaxed environment at home.

I would also like to acknowledge and thank my advisor, my committee members, and the program director for their continued assistance and encouragement during my graduate studies. I would like to especially thank my advisor, Dr. Martin Tanaka, for all the help and support he gave me while working on this project. Without your motivation and guidance, I would not have been able to accomplish all of this. I would also like to thank Dr. Paul Yanik for his advice while working on this project. Your knowledge and enthusiasm about machine learning helped me gain a better understanding of the topic and have inspired me to gain more knowledge and experience regarding artificial intelligence. Finally, I would like to thank Dr. David Hudson for his time and feedback. Without your help and expertise, I would not have been able to have data that helped develop this device. In addition, your feedback helped open more ideas that would improve the design of the device. Thank you all for your time and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF LISTINGS

ABSTRACT

NEURAL NETWORK CONTROL OF A NEURAL PROSTHESIS TO ASSIST WITH GAIT FOR PEOPLE WITH MUSCLE WEAKNESS

Pablo Joaquin Valenzuela

Western Carolina University (May 2021)

Director: Dr. Martin L. Tanaka

Studies show that about 1.7% of the US population live with some sort of paralysis which can reduce muscle function. Functional electrical stimulation (FES) has been widely used in the biomedical field to increase the functionality of atrophied muscles. The goal of this research was to design, build, and test a neural prosthesis that uses artificial electrical stimulation to improve gait in people with muscle weakness. The overall objectives of this project were to quantify the gait tracking performance of the 3rd generation prosthesis, and to develop the next generation model by implementing an artificial neural network that automatically controlled the electrical muscle stimulator. The 4th generation prosthesis was programmed to use sensor feedback from three inertial measurement units (IMUs) and four force sensitive resistors (FSRs) to predict the correct stimulation time. The IMUs were used to keep track of the leg movement during gait and the FSRs were used to track the force exerted by the foot at different stages of the gait cycle. Results showed that it was possible to program a highly accurate neural network from the received data of the sensors. After implementing the neural network and the stimulator device to the prosthesis, it was observed that the network correctly predicted when muscle contraction was required and was able to automatically send the stimulation signal.

CHAPTER I: INTRODUCTION

There are many functions that help humans carry out tasks during the day. Most of these relate to the mobility of the body or extremities. Unfortunately, per the World Health Organization, about 15% of the world's population lives with some sort of disability, and about 2% - 4% of those are related to mobility function [1], in the United States alone, about 1.7% (5.4 million people) of the population lives with some form of paralysis which results in difficulty moving the upper or lower limbs. Some of the leading causes of limb paralysis are strokes, spinal cord injury, multiple sclerosis, and cerebral palsy [2]. Paralysis can cause the functionality of a muscle to be reduced or not function at all. The treatment for these types of diseases can get expensive as the ability to move around to do simple tasks requires help or supervision. An innovative way to help improve muscle functionality at a relatively low cost is using electrical stimulation.

Electrical stimulation has been used in the therapeutic field to improve muscle function, sometimes decrease pain in a particular area, or to boost healing ability [3]. Functional electrical stimulation (FES), a form of electrical stimulation, sends a small electrical signal to the muscle that has been weakened and causes it to contract as it would during normal function [4]. Thus, FES tries to mimic the natural muscle stimulation that our bodies produce. The FES device sends signals though electrode wires which is similar to the body sending stimulation signals through the spinal cord and nerves [5][6]. These electrodes are attached to the outer layer of the skin through the use of patches. These patches produce a bridge to allow for the flow of the electrical signal.

The overall goal of this line of research is to design, build, and test a neural prosthesis that uses artificial electrical stimulation to improve gait in people with muscle weakness. Our strategy is to enhance the strength of muscular contraction in two muscles associated with plantarflexion, the gastrocnemius (GN) and the soleus (SL) muscles located in the back (calf section) of the leg. The device consists of a microcontroller, sensors to provide biofeedback, and a modified commercial FES device. This research spans multiple years and includes the work of several thesis students. The objective of this specific research project is to quantify the performance of the biofeedback sensors in the 3rd generation neural prosthesis (manual stimulation control) and to develop a 4th generation device that automatically controls the electrical muscle stimulator using an artificial neural network.

CHAPTER II: LITERATURE REVIEW

2.1 Human Gait and Gait Cycle

Human gait refers to the body movement produced when a person is walking. It considers the function of various muscles and joints that propels someone's center of gravity in a forward direction [7]. Gait varies from person to person, as each person's joints and muscles behave differently. Human gait is essential to how one moves around, and it is the building block for other functions such as running and jumping. For this study, only walking is considered as it is the primal element of how humans move.

The study of gait is essential to the understanding of human motion. Providing the quantification of this model can help solve many issues associated with muscle weakness. To begin studying gait, it must be divided into a gait cycle. The gait cycle is defined as the period from when a foot strikes the ground and then the same foot strikes the ground again after forward motion. A cycle is then divided into two phases, the stance phase, and the swing phase. The stance phase occurs when the foot is in contact with the ground, this phase holds about 60% of the total gait cycle. The swing phase occurs when the foot is mid-air or when it is not in contact with the ground. This accounts for the remaining 40% of the gait cycle, an image of this cycle can be seen in Figure 2.1 [8]. The stance phase is split into 5 different events. These events are heel strike (HS), foot flat (FF), mid stance (MT), terminal stance (TS) and toe off (TO). Heel strike occurs when the heel of the person first touches the ground. Foot flat is the sequential motion in which the weight of the person is shifted towards the leg that is in the stance phase. Mid stance occurs when the entire weight of the person is on the leg and foot that is contacting the ground. Once this occurs the foot then moves to the terminal stance portion in which the heel begins to rise eventually leading to toe off which is when the toe pushes off the ground to keep moving forward [8][9]. The cycle, on average, last one second. The stance phase accounts for 60% or roughly .6 seconds of the cycle, and the swing phase accounts for the remaining percent or time in the cycle. Some common equipment used to measure gait are inertial measurement units (IMUs), motion cameras, and other types of sensors [10][11].

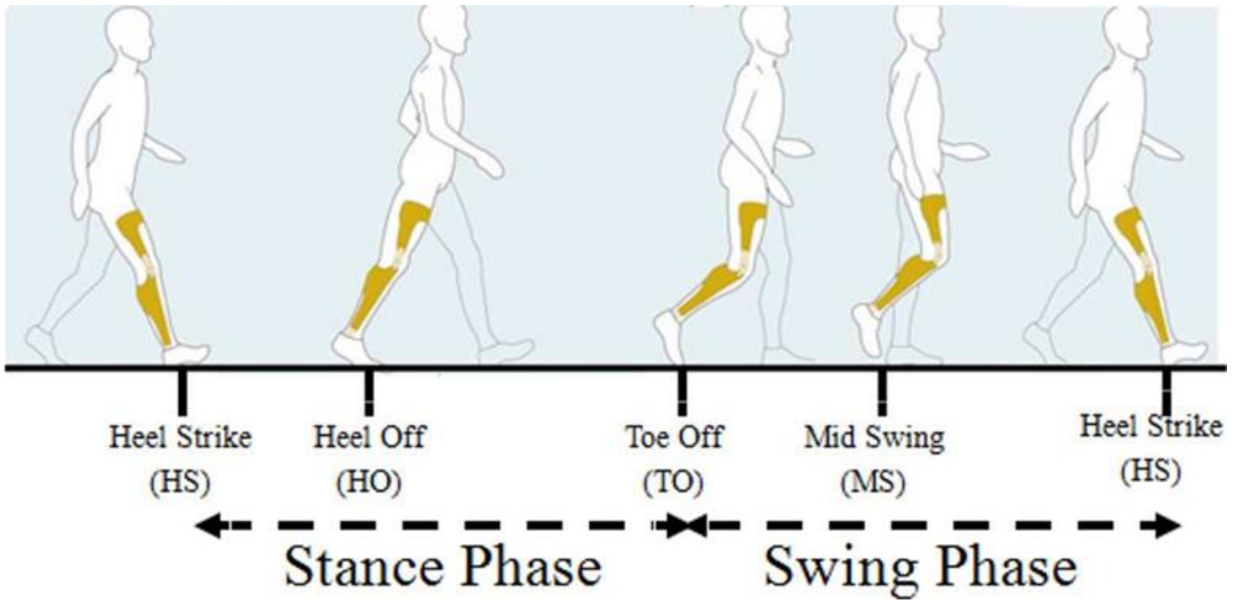


Figure 2.1: Gait cycle phases
(Diagram by Engineering in Medicine and Biology Society)[12]

2.2 Muscles in Gait Motion

There are many muscles associated with gait. Some of the different body sections associated with gait are the spine, pelvis, arms, and legs. All those body sections serve a different purpose. The upper body muscles help with the stability of one's stance when taking a step, or with carrying the forward motion for the body to move forward. The pelvis also provides stability when walking and acts as a pivot point for the lower limbs to carry out the motion of walking. However, out of all the body sections used to walk the most important is the lower limbs (legs). For the nature of gait analysis, the legs are split into three different sections: thigh, shank, and foot [13].

This research focused on the ankle because it acts as a hinge point for the foot and controls the plantarflexion and dorsiflexion of the foot. Plantarflexion is the motion in which the foot is pushed away from the body toward the ground, while dorsi-flexion is the motion that occurs when the top of the foot is pulled towards the shank [4]. During plantarflexion, the foot increases its angle from the anterior shank, while during dorsiflexion reduces the angle between the anterior shank and the foot. Understanding the movement of the ankle during plantarflexion and dorsiflexion is important to understanding gait, in particular events that occur during the gait cycle [14].

There are also important muscles that can help in this movement, these are the gastrocnemius muscle and the soleus muscles (Figure 2.2). The Gastrocnemius muscle is a muscle located in the posterior part of the shank. It is a muscle associated with the calf of the leg and has a lot of control over the plantar flexion of the foot and the flexion of the knee joint [15]. The Soleus muscle is also located in the posterior section of the shank and sits deep to the gastrocnemius muscle. The SL muscle also contributes to the control of the ankle and plantarflexion of the foot [15].

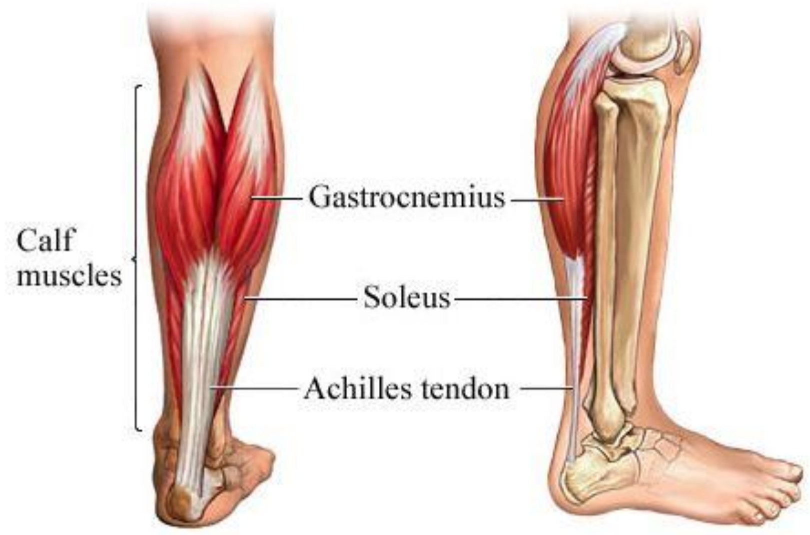


Figure 2.2: Location of the GN and SL muscles
(Image by Foot Pain Explored, https://www.foot-pain-explored.com/gastrocnemius.html)

2.3 Functional Electrical Stimulation

Functional Electrical Stimulation (FES) is widely used in the medical field to treat muscle atrophy and maximize a muscle's functionality. Electrical stimulation can be used to improve muscle function, decrease pain, or boost the muscle's healing ability [5][16]. Stimulation is given by sending electrical signals using electrodes. Electrodes may be attached to the outer surface of the skin and send electrical pulses which contract the muscle [17]. The output of the FES device to the electrodes can be increased or decreased to reach an optimal point at which the muscle is stimulated enough to contract but not so strong as to cause pain to the user as the feeling from electrical stimulation can be like the sensation of having a limb "fall sleep."

2.4 Artificial Neural Network

An artificial neural network (ANN) is a computational system that mimics the behavior of neurons in the brain. These systems are used for machine learning purposes to classify data. Just as the brain can identify different objects based on experiences, ANNs work with training data sets to identify or separate data. ANNs use computational nodes called artificial neurons, which can break down the data and identify patterns across the set. Neural networks are composed of layers that contain different numbers of artificial neurons that look for specific characteristics of the data to classify it. Each layer communicates with the subsequent layer in order to form an appropriate output [18], [19]. A simple architecture of an artificial neural network can be seen in Figure 2.3.

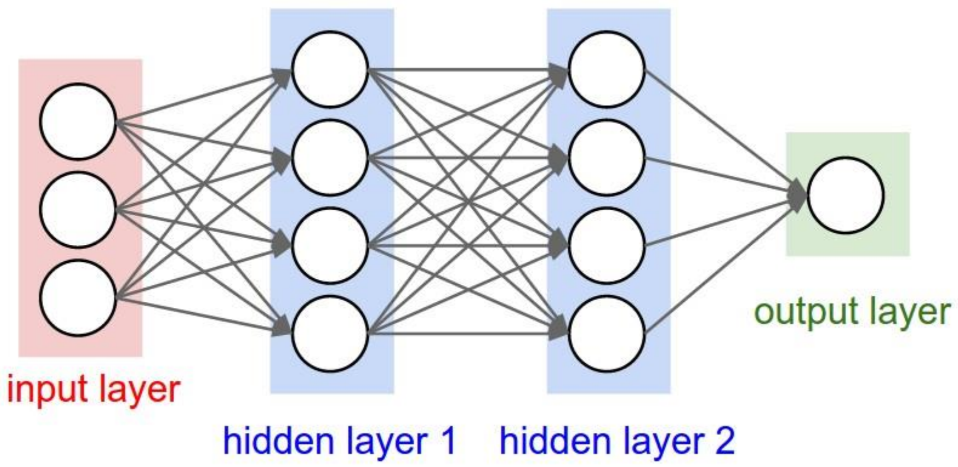


Figure 2.3: Architecture of an artificial neural network
(Image by CS231n GitHub) [20]

2.4.1 Network Layers

An ANN is commonly composed of at least 3 layers including an input layer, one or multiple hidden layer(s), and an output layer [21]. Each layer has a different function. The input layer receives a vector from the data set. The number of inputs (or features) will dictate how many neurons are used in the layer. The input layer will communicate with the first hidden layer.

The number of hidden layer neurons is determined by the number of inputs. A common guideline is to have less than twice the number of input neurons (Eq. 1), in this equation n is the number of input neurons [22], [23].

$$\#\ of\ hidden\ layer\ neurons = 2n - 1 \qquad \text{(Eq. 1)}$$

where n is the number of input nodes. The hidden layer then communicates with other hidden layers or the output layer. The output layer is where the classification occurs. The number

function [23]. For this research, one output node is used, with an activation threshold set to allow for two defined classifications.

2.4.2 Artificial Neurons

Artificial neurons are the components of a layer in the neural network. Their job is to mimic the behavior of a neuron in a human brain [20][24]. Just as a human neuron takes the electrical impulses in the brain to form an action, the artificial neurons take in signals and sum them up to obtain an output *(z)* value that is passed to the activation function. The architecture of a single neuron can be seen in Figure 2.4.

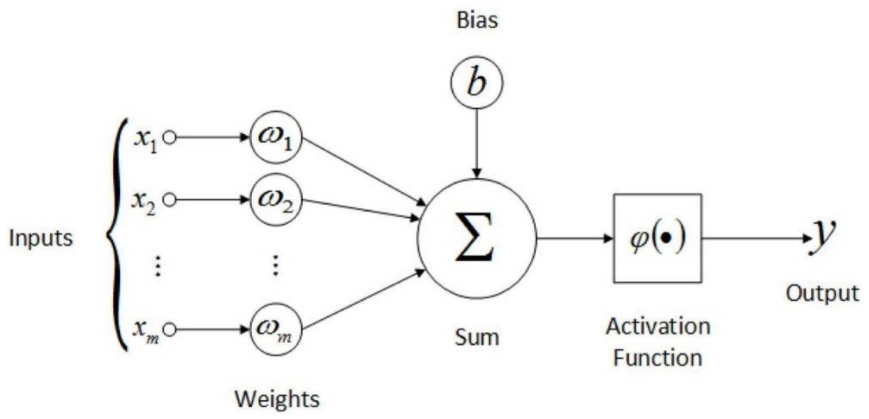


Figure 2.4: Architecture of an artificial neuron
(Image by Research Gate,)[25]

The figure shows the process of feed-forward propagation. The *z* value is calculated first from the summation of all the weights times their corresponding inputs plus a bias term. The *z* value equation can be seen in .

$$z = \sum_{i=0}^{n} w_i x_i + b \qquad \text{(Eq.2)}$$

where w_i is the weight variable for input *i*, x_i is input *i*, *b* is the bias term (typically 1 or 0), and *n* is the number of neurons. The *z* value is then passed to the activation function to obtain the output of the neuron. For this research, the sigmoid activation is used. The sigmoid activation function is a mathematical function that is characterized by its 'S' shape [26]. The sigmoid curve shape and its equation can be seen in Figure 2.5.

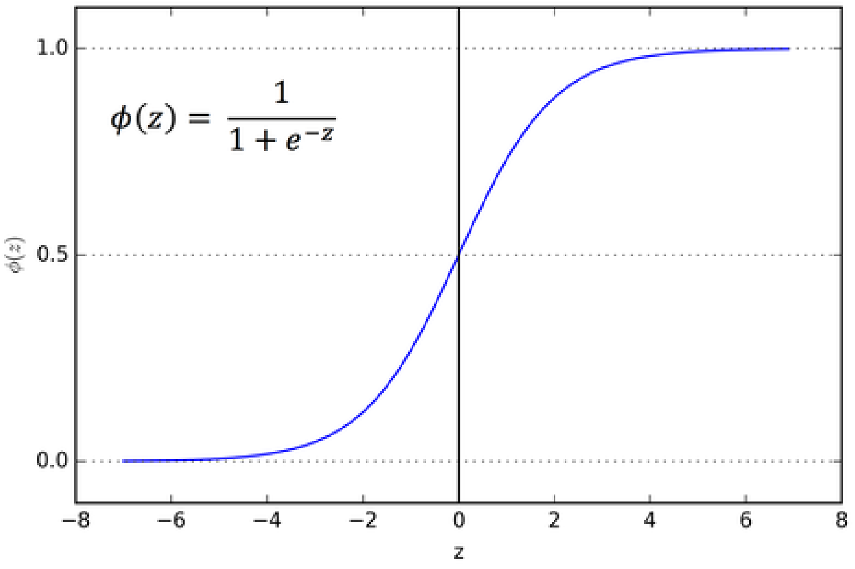


Figure 2.5: Sigmoid logistic function graph and equation
(Image by Towards Data Science)[26]

The output of the neuron is passed to the next layer as the input. In the output layer the value obtained from the sigmoid function is used to classify the current input. In a multiclassification problem with multiple output nodes the class may be decided by which output node has the highest value. In a binary classification problem, the classification can be condensed to one output neuron where a chosen value on [0, 1] denotes the classification threshold. If a value is above the threshold, then the input may be classified as class 1, and if the value is below the threshold, then it can be classified as class 2.

During an initial training process, backpropagation occurs. Back propagation is the process through which the neural network can adjust its weights to fit a particular set of data [27]. After the feed forward process calculates the output value, it is compared to the expected classification value. The difference in values is then assessed and weights are tuned to better fit the model. During training, the process of feed-forward propagation and back propagation occur after each iteration until the total network error over the training data set is less than a desired value [18] [27].

2.4.3 Sklearn Multilayer Perceptron Classifier

The Scikit-learn, also known as sklearn, is an open machine learning source library designed for the Python programming language. This library contains many classification, regression, and clustering algorithms that are useful for machine learning problems [28]. These algorithms have been programmed in an optimized manner that reduces the error compared to a simple machine learning algorithm designed and implemented from the start.

One of the many neural network models contained in sklearn is the multi-layer perceptron (an artificial neuron). The multi-layer perceptron algorithm is a type of learning algorithm that takes inputs and creates weights and bias components to classify data to the desired output value [28], [29]. This algorithm can be imported to Python code. It has a user-friendly interface to create the multi-layer perceptron model.

2.5 Python Coding Language

Python is a common programming language that has many applications. Python “is an interpreted, interactive, object-oriented programming language” [30]. Its simple syntax allows for users to program in fewer lines than other programing languages [31]. Python also has many libraries due to the common use in programing of the language, this creates versatility in the syntax of the language which allows new users to have an easier time learning the language and for advanced users to have multiple ways to approach real-world applications.

Qualities that make python a user-friendly language are its support for third-party tools that allow for easier learning and programming. This language also works seamlessly with different operating platforms. It can be extensible and embeddable which means that pieces of C/C++ can be combined with the code to improve its functionality [31]. This language is also object oriented which makes structuring easier since a user is not writing procedures but instead creating objects that contain data and functions to carry out a task [32].

2.6 Inertial Measurement Unit (IMU) Sensor

An Inertial Measurement Unit (IMU) is an electronic device that measures different forces to determine orientation. IMUs uses accelerometers, gyroscopes, and sometimes a magnetometer to obtain directional measurements. For this research four IMUs are used to determine the angles of the pelvis, thigh, shank, and foot at different stages of the gait cycle.

The IMU consists of different electronic and mechanical components to function, an image of this can be seen in Figure 2.6. There are different types of IMUs that revolve around degrees of freedom. The degrees of freedom are dictated by the components in the IMU such as the accelerometer, the gyroscope, and the magnetometer. A triaxial accelerometer measures acceleration in the three axes of motion and it can also be used to measure gravity taken as a downward force. The accelerometer is accurate with real-time readings; however, it is not recommended to use for distance calculations [33], [34].

The gyroscope also offers three degrees of freedom to the IMU. The gyroscope measures the angular velocity about the three axes, by providing this information the gyroscope can help determine the orientation of an object. The main difference between the gyroscope and the accelerometer is that the gyroscope does not have a frame of positional reference, instead it has a reference based on angular velocity [35]. By taking gravity as a force the accelerometer can understand some positioning, but the gyroscope does not take gravity into account therefore making it hard to find a starting position. Mixing both the accelerometer data and the gyroscope data can be helpful in determining the angular position of an object [36].

Another piece that can be seen in the IMU is the magnetometer, which uses the earth's magnetic field to determine the heading of the device. It basically works as a compass that finds magnetic north. Using these data along with the accelerometer and the gyroscope can provide the motion, orientation and heading of the device, which is helpful when trying to determine the different ranges of the IMU [37]. IMUs are also known to be inexpensive which helps to minimize the cost of developing a neural prosthesis for gait assistance.

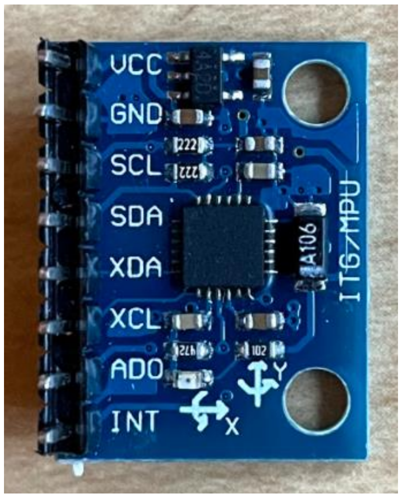


Figure 2.6: Inertial measurement unit
(photo by Pablo Valenzuela)

2.7 Cross-Correlation Analysis

Cross-correlation is a signal processing method that compares the similarity between two signals [38]. Cross-correlation is also known as the sliding dot product and it is used for distinct applications such as pattern recognition, electron tomography, and signal searching [39]. By using cross-correlation, two signals can be compared to quantify a shift in the x-coordinate direction, in this case time, and correlation factors.

In this research, cross-correlation can be used to determine the time delay associated with gyroscope data from the IMU. The data provided by this method is used to see when the curves from different trials are best aligned which validates the model that will be used to collect data for the artificial neural network training.

2.8 Third Generation Model and Previous Design

The third-generation device is composed of the Electrical Muscle Stimulator (EMS) with one of the lead wires turned into a switch for manual triggering of the signal used during trials

(Figure 2.7), the model of the EMS is the EMS-5000. Other devices that are used are also four IMUs implemented into the foot, the shank, and the hip, the current IMU is a 6 degree of freedom IMU with an accelerometer and a gyroscope, the model is MPU-6050. Four force sensitive resistors (FSRs) are implanted into a shoe insole (Figure 2.8) to measure the pressure of the heel and toes of the foot at different stages of the gait cycle, the model of these sensors is FSR-402.

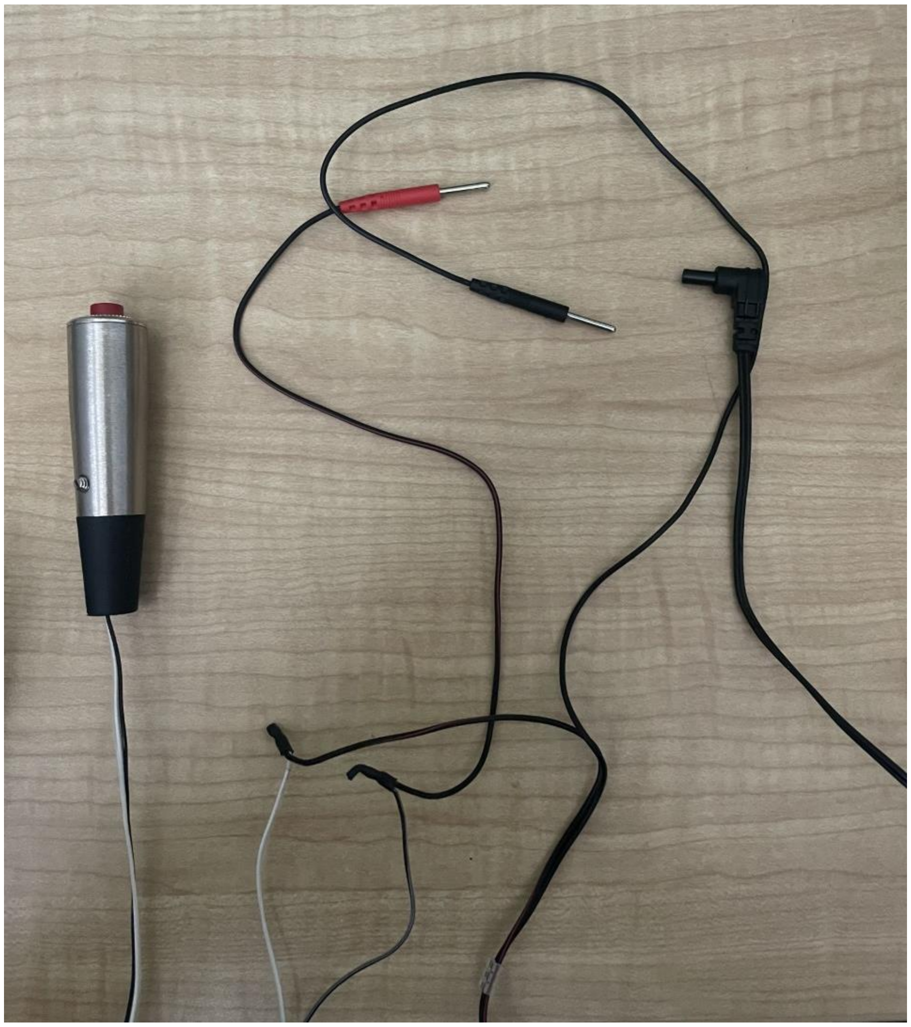


Figure 2.7: Lead wired adapted with manual trigger (photo by Pablo Valenzuela)

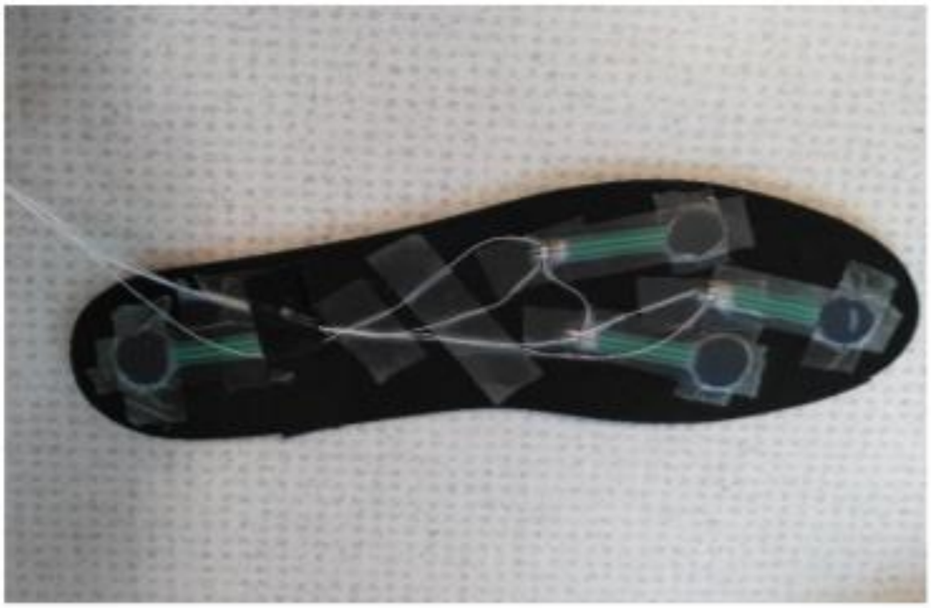


Figure 2.8: Insole containing the four FSRs
(Photo by Prem)

Another component of the third-generation device are the microcontrollers. The microcontroller that is being used is the Raspberry Pi 3, this microcontroller is used due to its capabilities when storing and computing data from a script that was written in Python. Some of these components were put into a housing that was attached to the waist of the person along with the EMS device, and the sensors were put on the person’s body [40].

CHAPTER III: METHODS

This section will cover how the neural prosthesis was designed, tested, and analyzed. There were three different parts to design the fourth-generation prosthesis. The first step was to measure the accuracy of the biofeedback sensor data collected by the third-generation device. The second step was to enable the microcontroller to control the timing of the electrical stimulation. The third step was to build an artificial neural network to determine when to apply the stimulation based on sensor data and program it into the device.

3.1 Third Generation Prosthesis Performance Validation

The first step to build the fourth-generation prosthesis was to validate the results recorded by the previous iteration of the model. The goal of the third-generation prosthesis was to create a device that was able to accurately record the angle measurements of the lower extremities during a gait cycle, store these data, and communicate wirelessly with a nearby computer. Accuracy of the device was tested by comparing results to those collected using the Qualisys Miqus M3 camera motion capture system (Qualisys Americas, Chicago, IL, USA) [41]. The cross-correlation function in MATLAB (MathWorks Inc., Natick, Massachusetts) [42] was used to study the data to observe if there was any shift between data types. Cross correlating the results quantified the delay percentage from the IMUs to the camera.

The cross-correlation study was conducted in MATLAB. A program was built that used the IMU data and the camera data separately to calculate the lag (shift) between both curves along the x-axis. The shift that was calculated served to identify the gait percentage delay from the IMUs compared to the camera data. The percentage delay was used to calculate time delay between the IMUs and the camera, and to adjust the delayed curve. The adjustment helped to visualize similarities and differences between both curves if no lag was present.

There were 14 total gait cycles that were collected by the 3rd generation prosthesis. Each gait cycle was compared to its camera counterpart in which the correlation value and lag were collected. The two functions used in the algorithm were *xcorr* and *circshift*. The *xcorr* function calculated the correlation values between each curve and gave the output of correlation values and lag. The higher the correlation value the greater the correlation between the two curves, which in turn made the lag value smaller. The code proceeded to find the highest lag value and point index which gave the required numbers to assess the percentage delay. The same percentage was then calculated from the total time of the gait cycle. The *circshift* function was used to shift the array of points in a circular manner. The indices of the array shifted circularly depending on the lag value calculated by the *xcorr* function. If the number was positive the array indices shifted to the right, and if lag value was negative then the indices shifted to the left. A complete list of the MATLAB cross-correlation code can be seen in Appendix A.

3.2 Enabling Microcontroller Control of EMS Device

The EMS device needed modifications to be integrated with the microcontroller. The EMS device's original function (programmed-based functionality) stimulated the muscle for a set number of seconds then turned off the stimulation for a set number of seconds. In order for the push button to be used to control the stimulation timing, the FES device needed to be constantly "ON", and the circuit needed to be completed using a push button switch (user-based functionality). The programmed-based EMS device contained two potentiometers that controlled the time of contraction and relaxation; both potentiometers can be seen in Figure 3.1 marked by the red circles. The time set by the potentiometers were a problem because the contraction needed to occur around 60% of the gait cycle which varies from person to person. The time set by the EMS device was also constant meaning that if a person was not walking the contraction still occurred. To change the EMS device from the programmed function to a user-need function both potentiometers were removed.

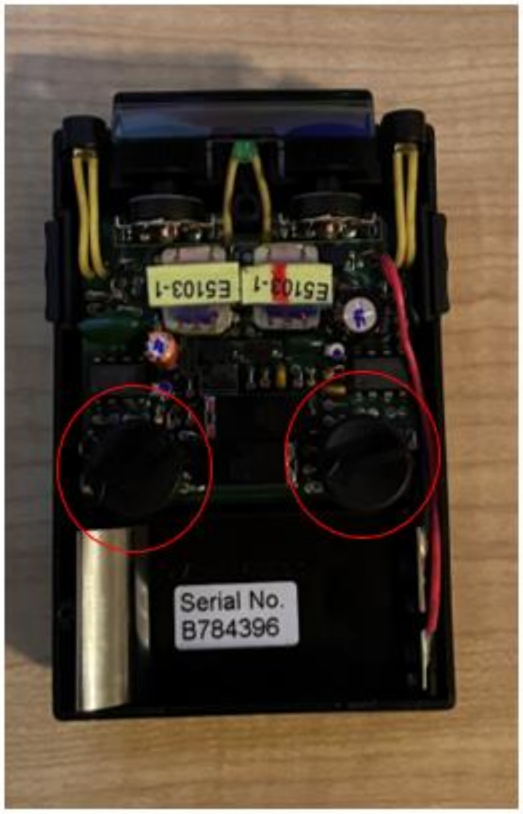


Figure 3.1: EMS unit control panel
(photo by Pablo Valenzuela)

The open loops left by removing the potentiometers were closed by soldering two wires that contained zero resistance (Figure 3.2). The wires were soldered into the positive and ground connections on the board to close the open circuits. This device modification was tested on an oscilloscope and it was observed that the wires set the device to the lowest time setting of contraction and relaxation. When the device was tested with the electrode patches on a person, it was observed that when the person pressed the button switch the muscle contraction occurred for one second and then relax for one second.

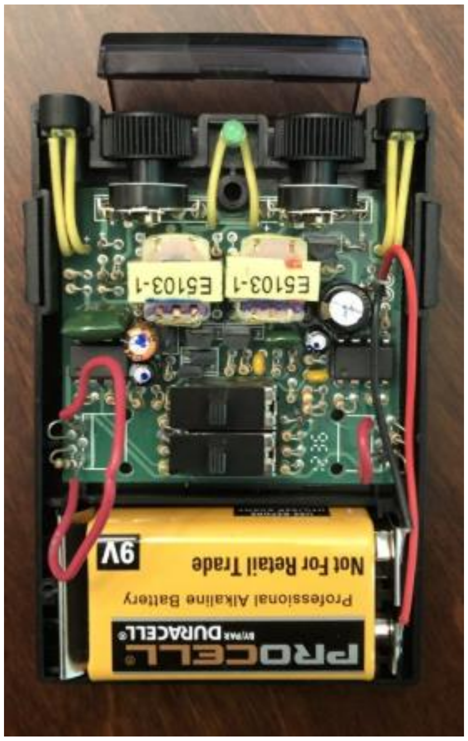


Figure 3.2: EMS circuit connected with 0 resistance wire
(photo by Pablo Valenzuela)

In order to fix the stimulation time error, the contraction side wire was cut in half. This left an open loop on the contraction side that closed when the button switch was pressed (Figure 3.3). After the changes were made, the device was again tested on the person and it was observed that the muscle contraction occurred when the button was pressed and then the muscle relaxed when the person stopped pressing the button.

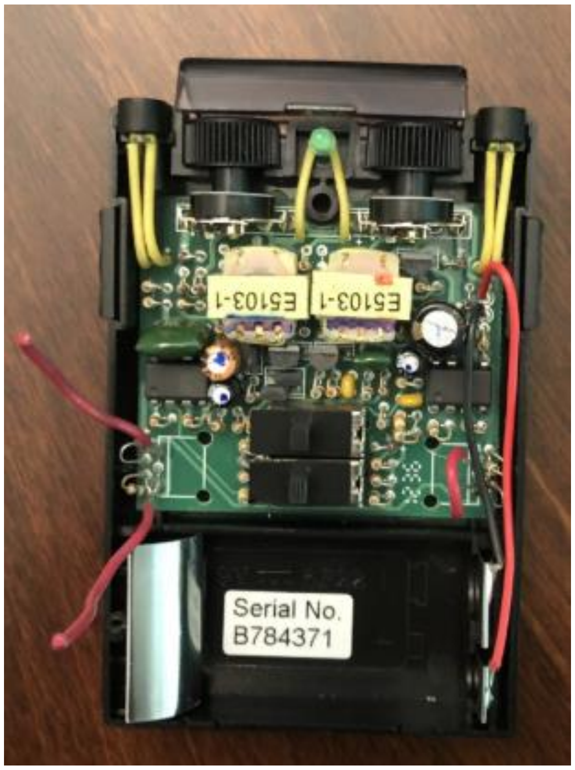


Figure 3.3: Contraction wire cut to create open circuit
(photo by Pablo Valenzuela)

After changing the functionality of the EMS device, the second modification was to change the push button to a solid-state switch so that it could be controlled by the microcontroller. The solid-state switch picked for this was the model LH1500AT (Figure 3.4). The solid-state switch was used to control the contraction signal based on script that opened and closed the circuit replacing a person pressing and letting go of the button. The solid-state switch turned on and off based on the electrical output of a pin from the RPi3.

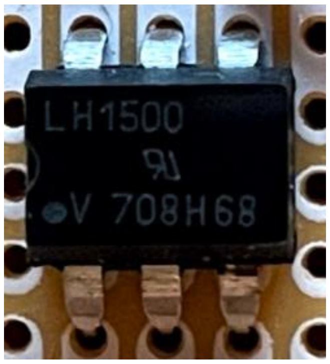


Figure 3.4: LH 1500AT solid-state switch
(photo by Pablo Valenzuela)

A model of the connections between the RPi3, solid-state switch and EMS device can be seen in Figure 3.5. After all the connections were made, a script was written to test the functionality of the device. The script seen in Listing 1 starts the program that requires user input. Entering a value of “1” causes the 35th pin of the RPi3 to output 3.3 volts, closing the solid-state switch. When the user enters a value of “0”, the voltage on the pin will drop to zero, opening the solid-state switch. Entering a value of “5” shuts off the script.

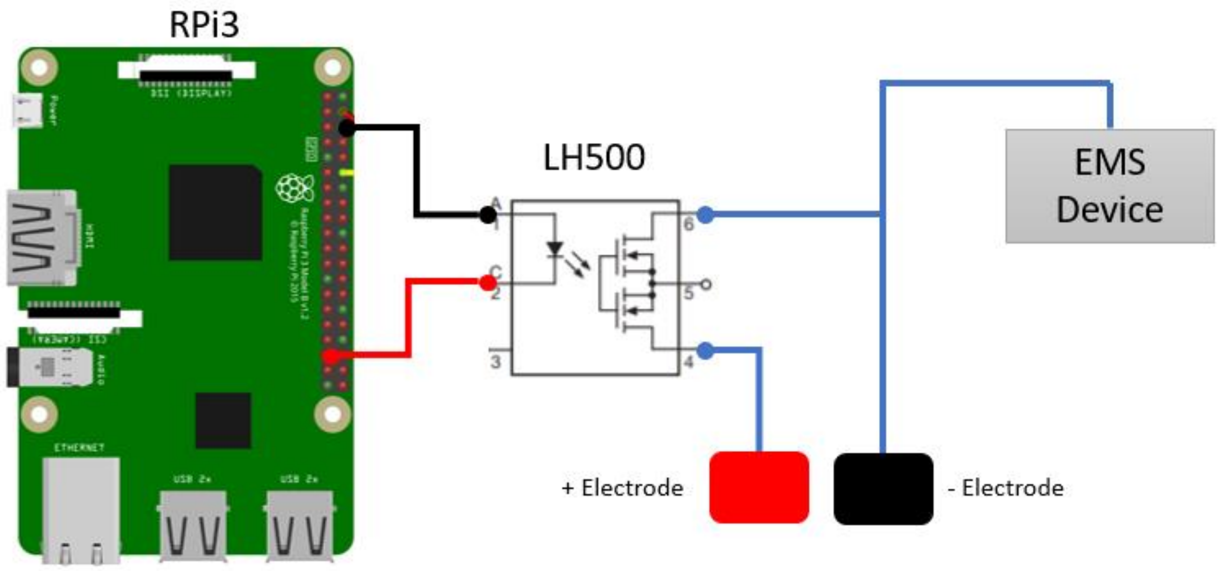
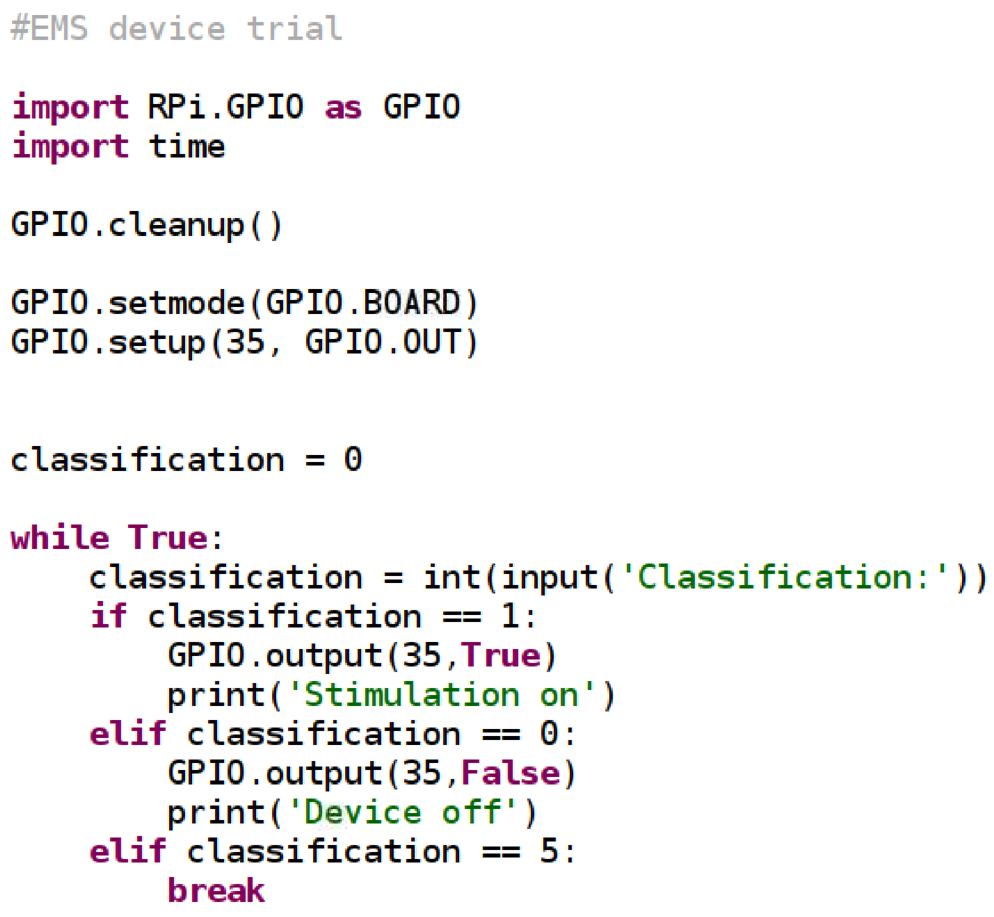


Figure 3.5: Connections model of solid-state switch, RPi3, and EMS device



```
#EMS device trial

import RPi.GPIO as GPIO
import time

GPIO.cleanup()

GPIO.setmode(GPIO.BOARD)
GPIO.setup(35, GPIO.OUT)


classification = 0

while True:
    classification = int(input('Classification:'))
    if classification == 1:
        GPIO.output(35,True)
        print('Stimulation on')
    elif classification == 0:
        GPIO.output(35,False)
        print('Device off')
    elif classification == 5:
        break
```

Listing 1: Script to test new EMS device

Like the previous prototype devices this was also tested by connecting the electrode pads to the muscle on a person. The EMS device was turned on and the written code was activated. The user then input a series of numbers to test if the device worked as intended. It was observed that the prototype device contracted when the user-input was “1” and turned off when the input was “0.” This final EMS model and code base was later implemented to the final script to create the 4th generation prosthesis.

3.3 Neural Network Development

The artificial neural network is the main component of the project as it brings all the other components together to automate the FES output. The neural network was written in Python so it could be easily uploaded to the Raspberry Pi. The Python code was written on a Jupyter notebook file. Jupyter notebook is an open-source web tool that can be used to run live code and obtain an output immediately, making errors easier to locate and fix.

Data collection trials were performed with the third-generation neural prosthesis to create training and testing data sets. The data collected had eight inputs, four inputs for the IMUs, and four inputs for the FSRs. In order to see how the sensors were effective, three artificial neural network codes were created. The first neural network considered all eight inputs, the second model only took the four FSR data inputs that measured force, and the last one only took the four IMU data inputs that measured angles of the foot, shank, thigh, and pelvis. These are described in sections 3.3.1 through 3.3.3.

3.3.1 Data Collection to Train and Test Neural Network

Data was collected from a healthy individual in a study approved by the IRB (Institutional Review Board). The participant was screened for any health conditions and once approved signed an informed consent form prior to testing. During the study, the participant and the researcher practiced COVID-19 safety guidelines. The participant and researcher wore masks at all times, devices were sanitized using isopropyl alcohol (disinfectant which would not damage the electrical components), and a distance of at least 6 ft was maintained when possible. The 3rd generation device was attached to the participant, one IMU on the foot, another on the shank, one on the thigh and the last one at the pelvis, all IMUs were attached with tape since only small portions of the body were used as attachment points (Figure 3.6 and Figure 3.7). The shoe insert containing the FSRs were placed into the participant's left shoe. The participant was asked to walk around with the device attached to become comfortable wearing the device prior to performing the trial runs.

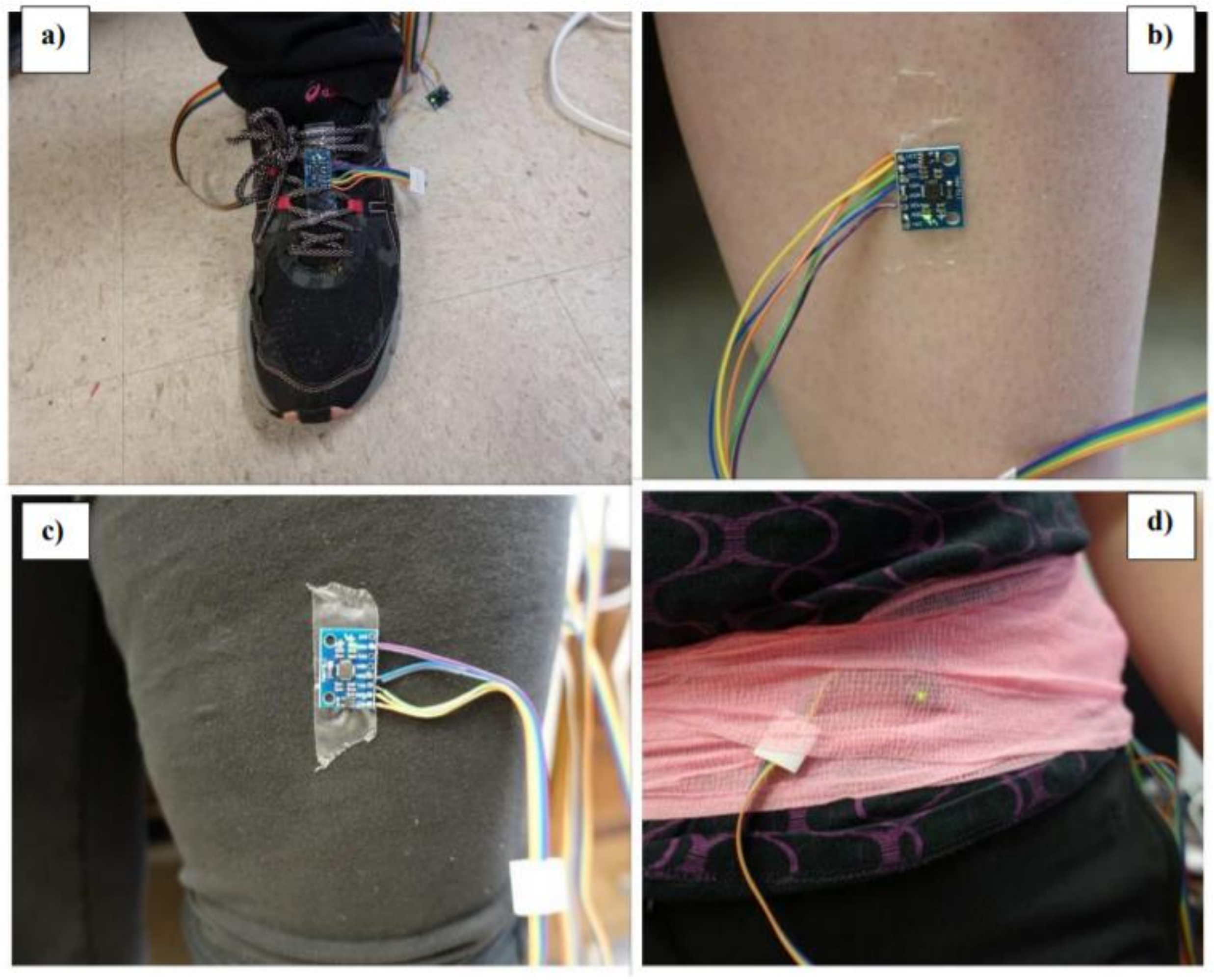


Figure 3.6: IMU placement points
Placement of IMU on the foot (a), shank (b), thigh (c), and pelvis (d) (Photos by Martin Tanaka)

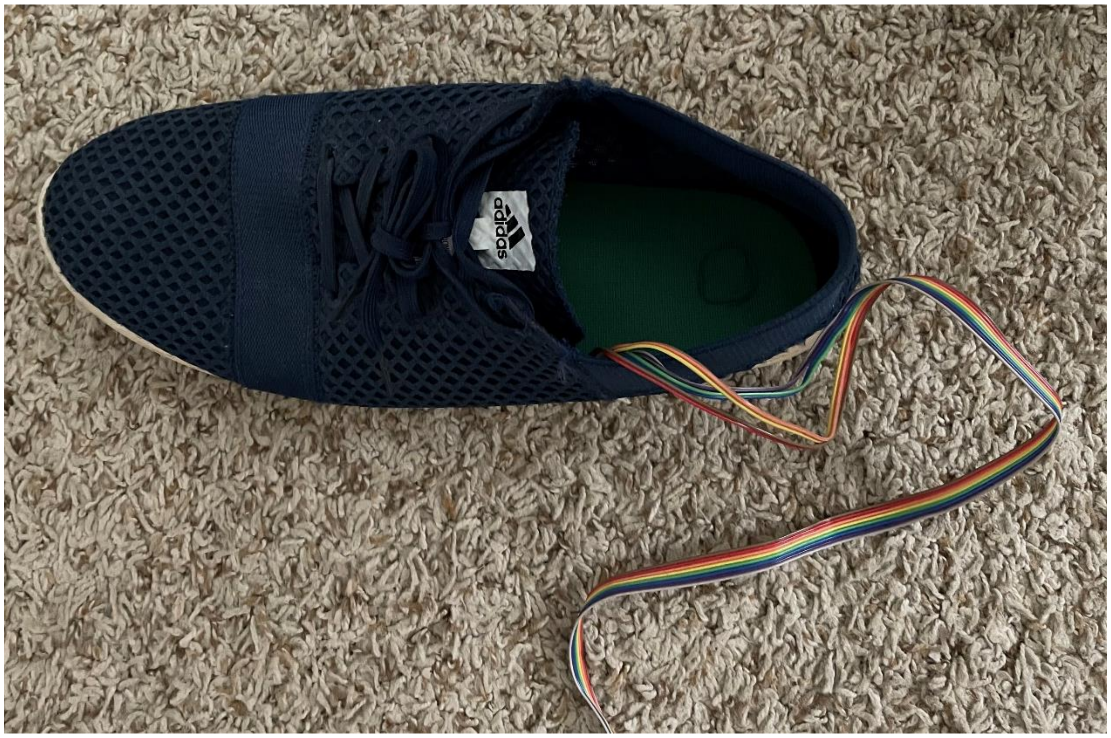


Figure 3.7: FSRs shoe insert placement
(photo by Pablo Valenzuela)

Once the participant felt comfortable walking around with the device on, the Raspberry Pi was powered on. The RPI3 was used in a wireless manner that connected to the computer via Wi-fi, the program that was used to view the Raspberry Pi 3 screen was VNC Viewer. VNC Viewer allowed the researcher to see the main screen of the RPi3 and be able to run the program that collected the data, this also allowed them to have full control of any other capabilities of the RPi3 during the trial. Once everything was connected and turned on (Figure 3.8), the participant was asked to do a few test runs. During the test runs the participant was asked to walk around, stand still, and keep walking. This was done to ensure that the program and the device were working properly before any data was collected.

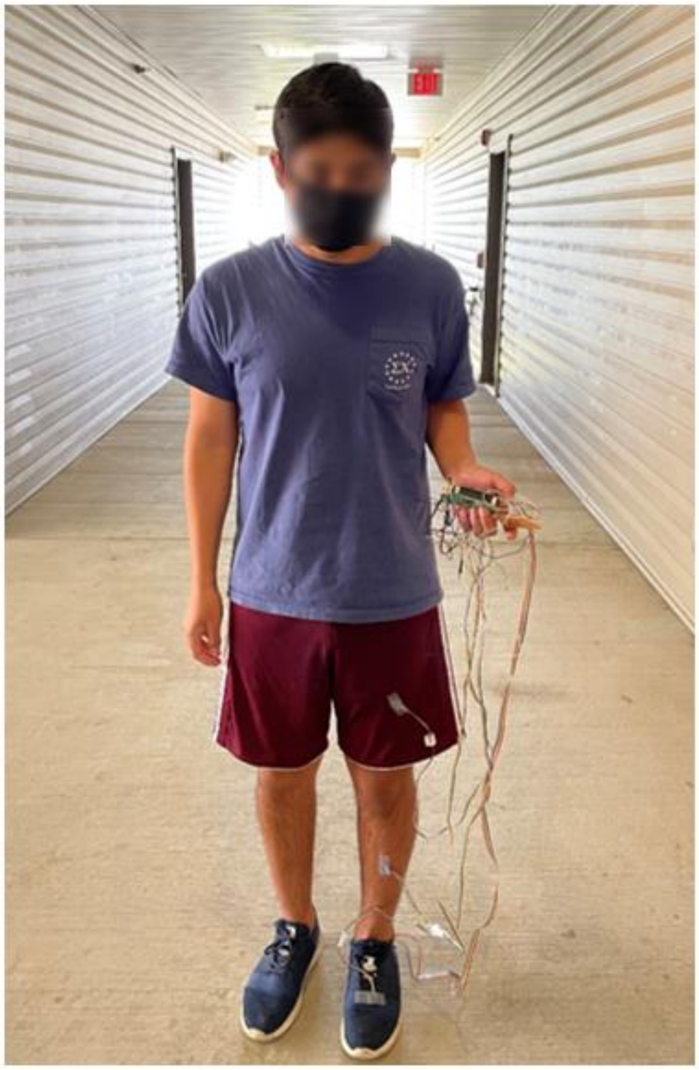


Figure 3.8: Participant device set-up
(photo by Pablo Valenzuela)

After the device was calibrated and debugged, the researcher informed the participant about the conditions for each trial. The participant was instructed to walk at three different speeds, normal, fast, and slow on a flat surface for a distance of 10 meters. This was used to determine if the device could perform well under varying speeds of gait, and to see if it had a problem recognizing the stimulation time when the walking speed changed. The normal speed trials served as the control and were most likely to be used in case of major errors in the other trials. The flat surface served as a control since this made certain that the data points were accurate and there was no skewness in angle measurements due to going uphill or downhill. A total of 12 trials were collected, three at each speed. A total of 55 gait cycles were collected with each trial having between 4-5 gait cycles. After each trial, the program was stopped and saved to a CSV file that was used later to create the training and testing data sets. After all trials were collected, all data files were examined for any mistakes to see if any other trials needed to be done.

3.3.2 Training and Testing Data Preparation

After the twelve trials were collected and saved as a CSV file, the training and data sets were created. Only nine of the trials were used to create these sets since the three other trials had errors in the data that could potentially crash the neural network code when it was in training. Each trial was then examined to see the starting point of the first gait cycle and the last point of the last gait cycle. All the trial data were then extracted and put into either the training data set or the testing data set. Once all the data was transferred into either the training or testing data set, the data points were classified. Since this study only contained 2 classes, a classification of "0" or "1" was used. The "0" classification meant that the device remained off, and a "1" meant that the device should send the stimulation signal. This format of classification also made the neural network calculations easier since the sigmoid logistic function has a threshold between 0 and 1 allowing for the neural network model to have a single output node.

Each data point of the training and test sets was manually classified with either a "1" or a "0" depending on the values that were gathered from the sensors. These classifications served as the target values to train the network and then test its accuracy. The data was classified to stimulate at 60% of the gait cycle, however this classification can be adjusted later to account for other events of the gait cycle. There was also a pattern from the sensor data that made it easier to classify each point. The foot weight transfer was visible on the FSRs from the heel to the toe and then when the foot was off the ground to start the swing phase. An example of the described pattern can be seen in Figure 3.9.

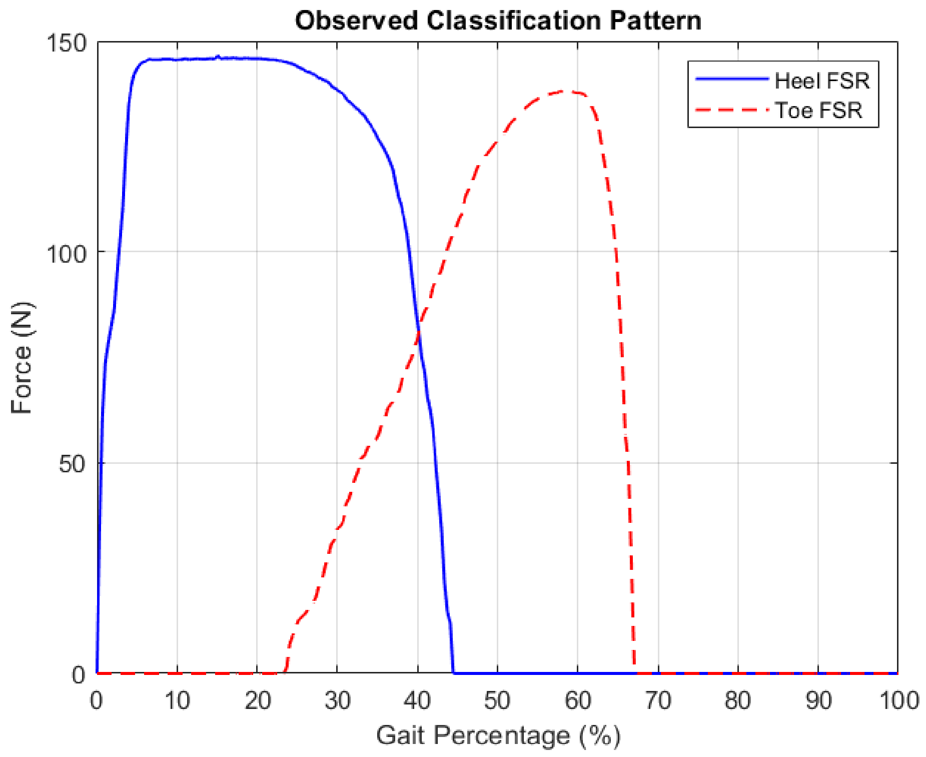


Figure 3.9: Observed gait pattern for classification

The heel and toe patterns were visible for each gait cycle since the cycle started at the heel and ended before the next heel strike. Based on this information, each point in that area of weight transfer (near the line intersection between both curves) to the point of highest force value on the toe FSR was given a classification of "1" while the other points were given a classification of "0." After the classification of the data, the neural network could be programmed to produce the weights that were later implemented into the final model.

3.3.3 Neural Network Modeling

As previously mentioned, there were a total of three neural networks that were programmed based on different data sets. The first model took all the eight sensor values, while the other two models only considered either the FSR sensors or the IMU sensors. Before coding the neural network, a model was created showing the architecture of the network. An example of this can be seen in Figure 3.10.

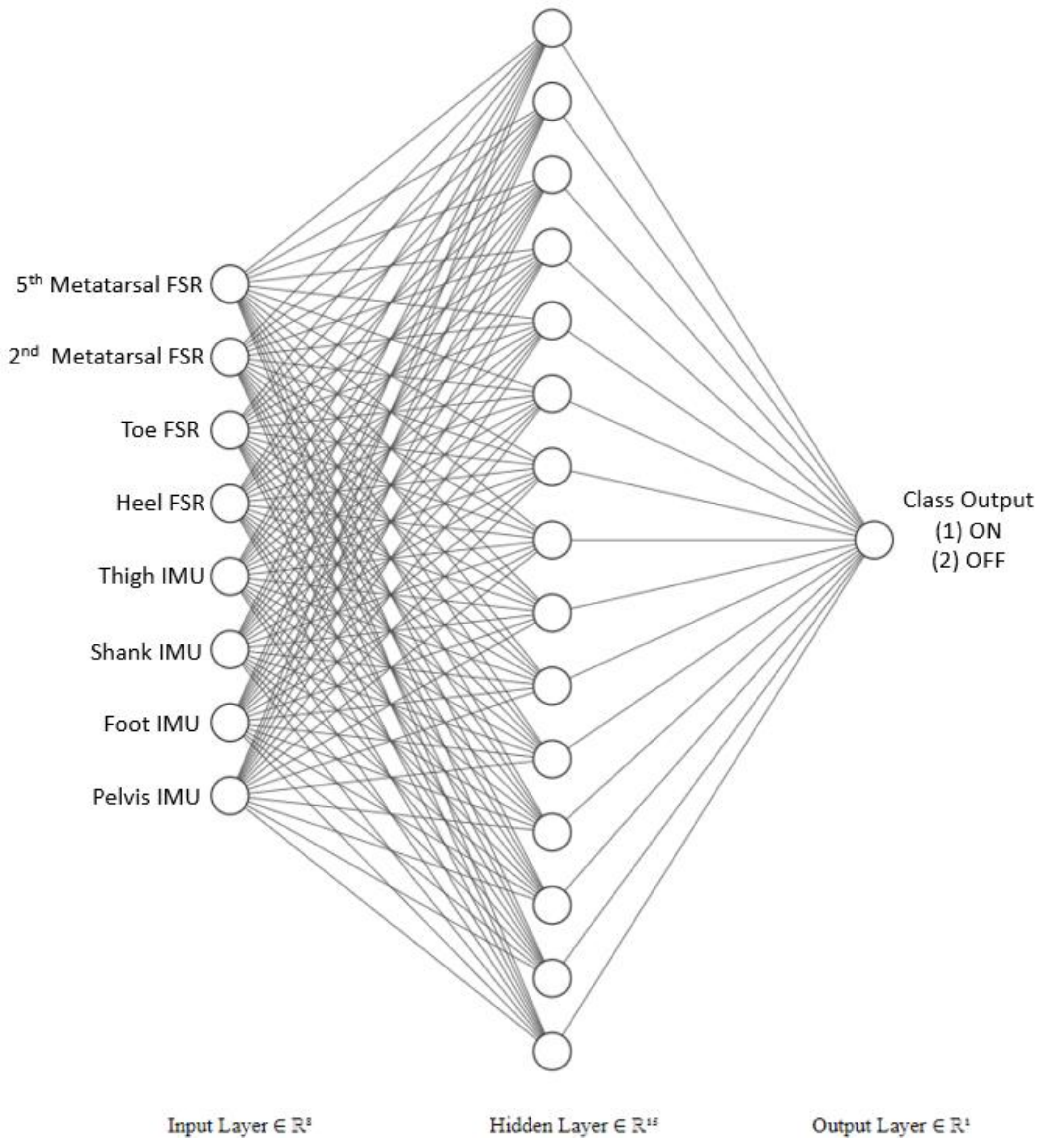
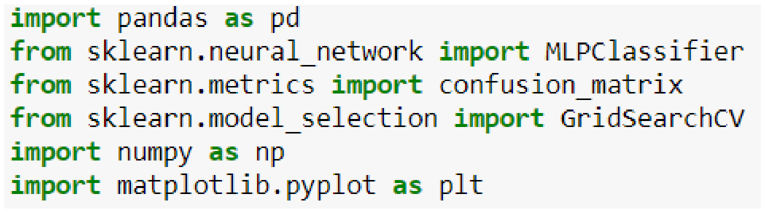


Figure 3.10: Neural network architecture
(produced using NN-SVG program - http://alexlenail.me/NN-SVG/index.html)

The neural network architecture shows the eight input nodes, one for each of the sensors that collect the data. There are 15 hidden layer nodes following the rules given in the literature [22], and one output node since this is a binary classification problem. It should be noted that this architecture was created assuming all sensors were equally important in classification. The architecture was altered appropriately for the other two models that only used either the FSR data or IMU data. The parts that changed were the input layer and the hidden layer. Instead of having 8 nodes, the input layer only had 4 nodes since only four sensors were used. As for the hidden layer there were 7 nodes, again this was decided based on the literature guidelines for building a neural network. The output node remained the same for all cases to accommodate the binary classification problem.

Once the architectures of the models were established, the code was developed in a Jupyter notebook. The first step was to import available functions and libraries to be able to read files or run programs. This is seen in Listing 2.

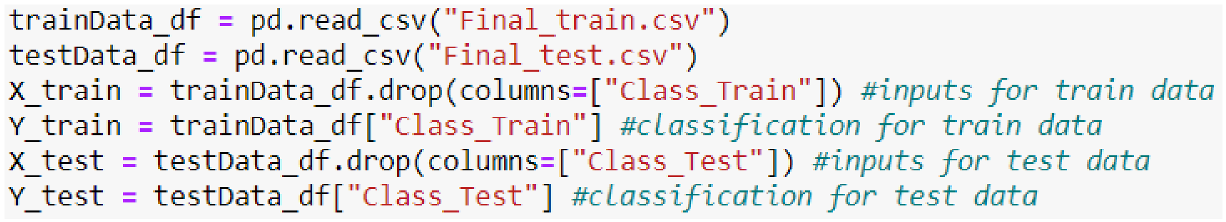


```python
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
import numpy as np
import matplotlib.pyplot as plt
```

Listing 2: Starting functions

The *import* command was used to bring in the functions and libraries that were required for the program to run. Importing *pandas* as *pd* brought in the pandas library which was used to read CSV files. To make coding easier it was given the name of *pd* so anytime the pandas library was called the term “pd” was used instead. The next three lines call for different sklearn modules, and from each module a certain function was called. The *MLPClassifier* function was used to create the neural network as this was the function that handled all the calculations and optimizations to create an accurate classifier. The *confusion_matrix* function was called to be used as a validation technique for the classifier. A confusion matrix was used to determine how many points were classified correctly and how many points were misclassified; an example of this can be seen in Table 1. The *GridSearchCV* function allowed the user to create lists with different conditions to test for the classifier. The next two lines import *numpy* and *matplotlib.pyplot*. *Numpy* is a numerical Python library that was used for mathematical operations, and *matplotlib.pyplot* was used to plot the loss functions of the classifier to see how error diminished with more iterations in the model, both of these functions can also be called by using the terms *np* and *plt* respectively.

After importing all the necessary functions, the variables that contained the data were created. This can be seen in Listing 3, all the variables on the left sides of the equal sign are the names that were used later in the code. Some other functions such as *read_csv* and *drop* were used to read the files and put them in an order that helped the classification algorithm.



```
trainData_df = pd.read_csv("Final_train.csv")
testData_df = pd.read_csv("Final_test.csv")
X_train = trainData_df.drop(columns=["Class_Train"]) #inputs for train data
Y_train = trainData_df["Class_Train"] #classification for train data
X_test = testData_df.drop(columns=["Class_Test"]) #inputs for test data
Y_test = testData_df["Class_Test"] #classification for test data
```

Listing 3: Variable set-up

The *read_csv* function is part of the pandas library that reads the CSV file that was created after collecting the data. The *.drop* function was used to drop certain columns of the data set to be used as another variable, in this case the column that was dropped was the classifications because the classifier needed to be a different variable to measure accuracy.

The *MLPClassifier* function was then used to set the initial conditions of the neural network, this can be seen in Listing 4. For this model of the neural network the hidden layer size was set to 15. The activation function was then defined to be the logistic sigmoid function. The batch size is set to auto to facilitate the prediction curve that classifies the points, having it set to auto will also help run less iterations. The maximum number of iterations was set to 10000 as it was expected that the neural network would converge before reaching that iteration. This was expected because the neural network used mini batches of data to adjust the weights allowing it to converge in less iterations. The verbose component allowed for the loss to be shown after each iteration when the code is activated. The random state variable set the seed, the starting point for any randomization that occurs in the model, that allowed for the results to be repeatable in case the model needed to be run again.

```
my_NN = MLPClassifier(hidden_layer_sizes = 15, activation='logistic',
                      batch_size='auto', max_iter=10000,verbose=True,
                      random_state=5)
```

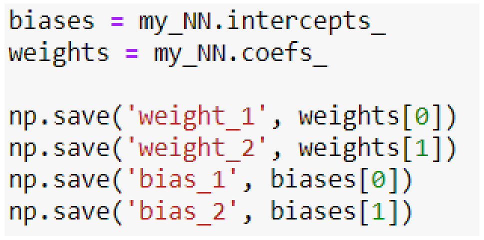
Listing 4: Set-up conditions for classifier

The *MLPClassifier* function does not activate the neural network but instead sets up the conditions that the neural network needs to create the weights and biases to correctly classify the data. The *.fit* function is the function that runs all the calculations for the neural network to work. This is seen in Listing 5.

```
NN_fit = my_NN.fit(X_train, Y_train)
```

Listing 5: Fitting the network

In this line a new variable is created to save the neural network model. The *my_NN* variable name is called again to fit the model based on the conditions that were previously set. The model fits the train data to the train data classifications, the X component accounts for all the sensor data, and the Y component is the actual classification that is expected. The next step was to set new variables to save the weight and bias values of each node, this can be seen below in Listing 6.



```
biases = my_NN.intercepts_
weights = my_NN.coefs_

np.save('weight_1', weights[0])
np.save('weight_2', weights[1])
np.save('bias_1', biases[0])
np.save('bias_2', biases[1])
```

Listing 6: Saving weights and biases

The biases (intercepts) and weights (coefficients) arrays were saved under new variables in the model, which were then saved as a singular array to be called on later in the neural prosthesis. Saving the variables as a single array allowed for the weights and biases to be used in other code scripts as long as the variables were in the same directory. Other components that were programmed were accuracy calculations of the model for the training and the testing data, a model of the confusion matrix to show the amount of correctly classified points, and a graph to show the loss function. A full script of the code can be seen in Appendix B.

The other models were built in a similar manner the only changes that were made were in the hidden layer size, which was set to 7 instead of 15, and the training and test sets that were used were different, but besides those two aspects the code followed the same pattern. After creating all the models, accuracy was assessed by the amount of correctly classified points. The classifications were extracted as an excel document and then plotted to determine the accuracy of the results. The results of each model were compared to the actual classifications (values set previously when making the training and test data sets), this helped to determine if all sensors were useful, or if a certain set of sensors did not have a strong impact on the classification of the data points, this comparison also pointed out which set of sensor data was at fault for some misclassifications.

3.3.4 Final Neural Network Model

After analyzing the data and curves, (Section 4.3), it was apparent that the IMUs were not as essential as the FSR sensors. Therefore, it was decided that one of the IMUs be removed to open a pin connection for the connection of the EMS device. Comparing at the IMU vs camera capture system data it was observed that the pelvis sensor contributed the least since it had no recognizable features that aided classification. Because of this, the pelvis IMU was removed from the system. Due to this observation, a new model was built that only considered 7 sensors. The sensors that were used were the four FSR sensors, and the remaining three IMUs. Figure 3.11 shows the topology for the final ANN model.

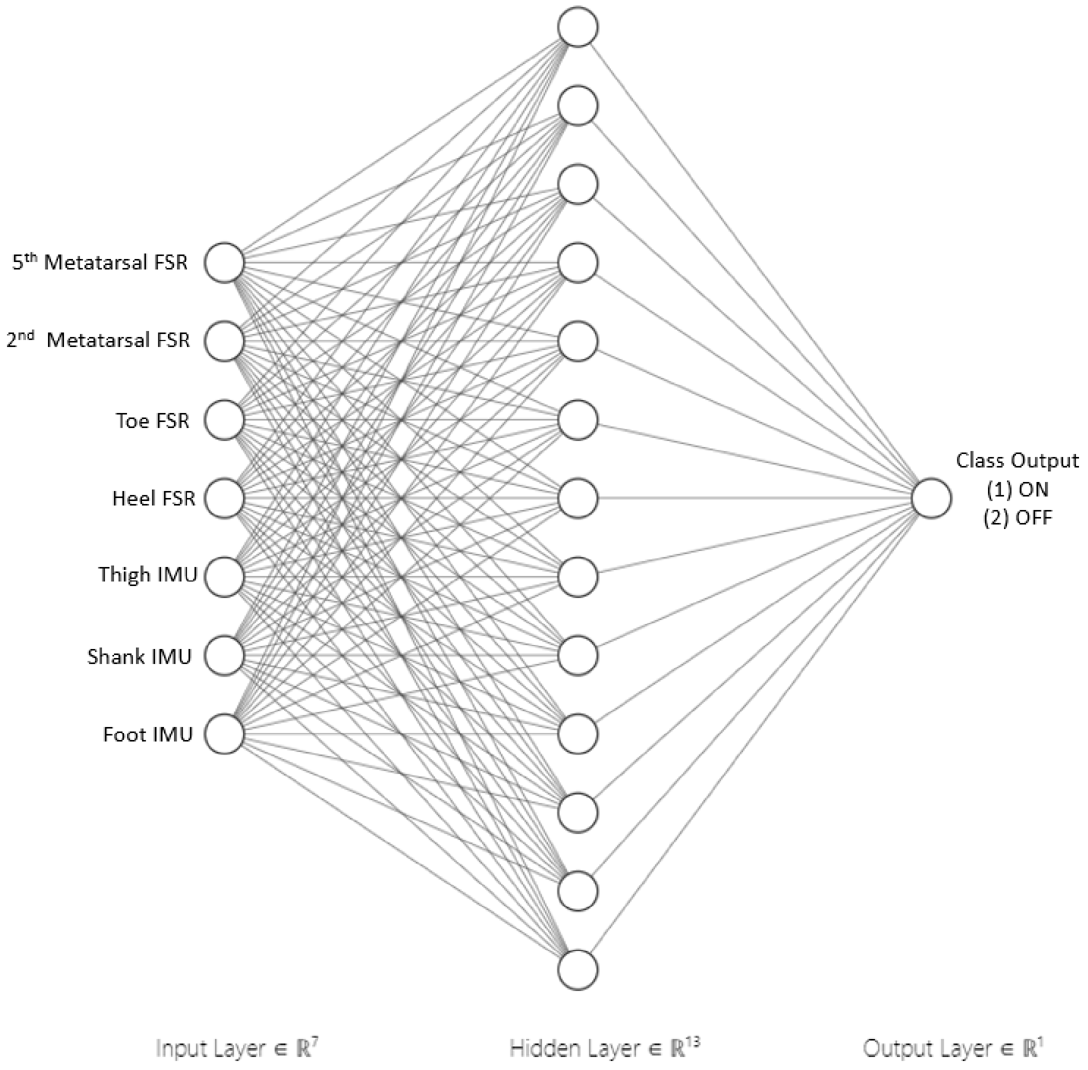


Figure 3.11: Final neural network model
(produced using NN-SVG program - http://alexlenail.me/NN-SVG/index.html)

The final architecture includes 7 input nodes, one for each of the sensors. There are 13 hidden layer nodes and one output node. The conditions of this model were set up in a similar way to the previous neural networks, but the hidden layer size was changed to 13 instead of 15. The rest of the code remained the same. The validity of the network was based on the accuracy of the model, which could be calculated after fitting the data set to the classifications. Validity was checked by the use of the *.score* function in the code, and by making a confusion matrix (Table 1) that showed the amount of misclassified points.

Final ANN Model		**Predicted**		
		Off	On	**Total**
Actual	OFF	2147	23	2170
	ON	7	90	97
			Error	1.10%

Table 1: Confusion matrix of final ANN model

The confusion matrix shows how many points there were in each class category (yellow column) and how many points out of each were misclassified. The goal of a classifier is to have the number zero in each of the red cells, and the total amount of points in each of the green cells. The final network had an accuracy of 98.9% but further validation was still performed to confirm the robustness of the model.

3.3.5 Neural Network Validation

Once the final neural network model was trained and tested, it was observed that it produced a 98.9% accuracy when classifying the original training and testing data sets. Further validation was also performed to confirm the robustness of the network model. A way to validate the network was to randomly slice the testing data samples and see if the accuracy changed. Randomizing the data samples and the order of classification helped to see if the varying subsets allowed the neural network converged in a similar manner to correctly fit the model. To test the model there were three different test data sets that were created apart from the original testing data set. These testing sets had a randomized order in their classification and gait percentage. Confusion matrices were used to observe if the final model kept the same accuracy. These results can be seen in Section 4.3.1. The results indicated that the neural network was fit correctly and could be used for the development of the fourth-generation prosthesis.

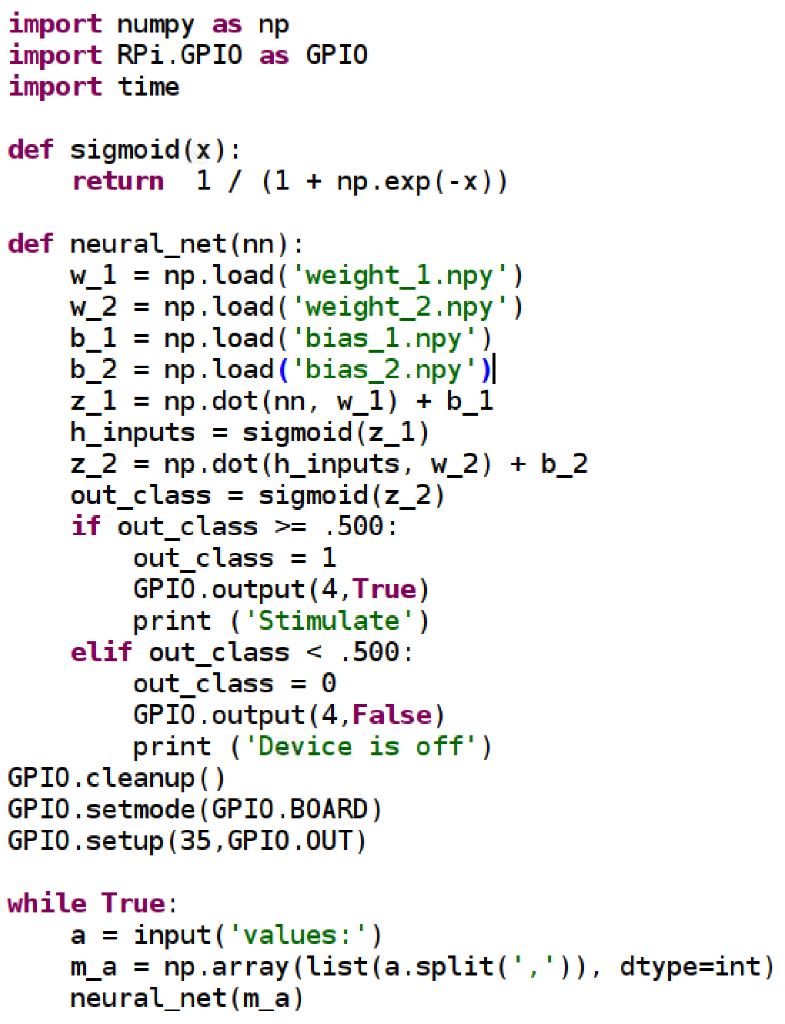
3.4 Fourth Generation Neural Prosthesis Development and Testing

The last step to make the 4th generation prosthesis was to integrate all the components that were previously mentioned. The first step was to integrate the neural network and the modified EMS device together. The code of the previous prosthesis mainly remained the same, but there were a few changes that allowed the data to be used by the neural network and then send the stimulation signal. After the code changes, the device was tested to see the accuracy of the fourth-generation prosthesis.

3.4.1 ANN Integration to Control the EMS Device

The development of the neural network was one of the key aspects to making the fourth-generation neural prosthesis. The neural network was the component that automated when the contraction signal was required and when it was not based on the data from the FSRs and the IMUs. Because of this it was essential to integrate the modified EMS device. In the previous modifications to the EMS device, we managed to allow the RPi3 to control the contraction signal based on the activation of the 35th GPIO pin when the user desired. The neural network combination served to automate that process and made it easier to integrate into the third-generation prosthesis code.

The first step was to write the feed-forward pass calculations portion of the neural network into the Python code on the microcontroller. The feed-forward pass is what takes the sensor readings and puts it through all the mathematical functions of the neural network to classify that data point or array. Listing 7 shows the script that was used to integrate both the forward pass calculations of the ANN and the integration of the EMS device.



```
import numpy as np
import RPi.GPIO as GPIO
import time

def sigmoid(x):
    return  1 / (1 + np.exp(-x))

def neural_net(nn):
    w_1 = np.load('weight_1.npy')
    w_2 = np.load('weight_2.npy')
    b_1 = np.load('bias_1.npy')
    b_2 = np.load('bias_2.npy')
    z_1 = np.dot(nn, w_1) + b_1
    h_inputs = sigmoid(z_1)
    z_2 = np.dot(h_inputs, w_2) + b_2
    out_class = sigmoid(z_2)
    if out_class >= .500:
        out_class = 1
        GPIO.output(4,True)
        print ('Stimulate')
    elif out_class < .500:
        out_class = 0
        GPIO.output(4,False)
        print ('Device is off')
GPIO.cleanup()
GPIO.setmode(GPIO.BOARD)
GPIO.setup(35,GPIO.OUT)

while True:
    a = input('values:')
    m_a = np.array(list(a.split(',')), dtype=int)
    neural_net(m_a)
```

Listing 7: EMS and ANN implementation script

First the important functions required to make the program were imported. The sigmoid function was then defined since this is the function that out ANN used to train. The forward pass is then defined as the function *neural_net(nn)*, where *neural_net* is the name of the function and *nn* is the variable or array that it takes in. The weights and biases that have been previously established by the neural network code are called back to help through the feed forward calculations.

The variables *z_1* and *z_2* calcualte the z value which is the sum of all the inputs times the weights and bias values. Once z is calculated, the *h_inputs* and *out_class* variables take the z value and put it into the sigmoid function which will return a value between 0 and 1. The next few lines set the conditions for the classification of the *out_class* value. The if statement in the script states that if the output value is equal or greater than .50 then it should be classified as a 1 (ON). Along with the classification, this data point should also turn on the 4th pin of the RPi3 which in turn will close the solid-state switch allowing for the EMS device to contract the muscle. To be more helpful a print statement was also used under this "if" condition so the researcher could see in print that the device was on. The *elif* function, which is short for "else if", states the opposite of the *if* statement. In this case if the output value is less than .50 then the designated class should be a 0 (OFF) and the 4th pin of the RPi3 will be turned off which opens the solid-state switch, so the muscle is not contracted anymore. A print statement was also used to show the user that the stimulation was off.

The next three lines are used to set up the GPIO pins of the RPi3. The while true statement allows the program to run until the program is manually stopped by the user. The code implemented after the while statement controlled the overall function of the script. In this case the user was asked to input an array of numbers that mimic the arrays that are collected by the seven sensors. The entered array was then turned into a *numpy* array that gets put into the *neural_net* function to classify.

The test for this device was similar to the test done to test the functionality of the EMS device but in this case, we used pin #4 instead of pin #35. Another difference was that instead of only asking for an input of a single digit, the input had to be an array of values. The arrays were obtained from the testing set since their classification was already known. The electrode patches were placed on a person's forearm, and the program was started. Once the programmed was started it prompted the user to input an array which was copied and pasted from the testing set and it was observed that when the inputs were a class 1 array the device turned on and kept on until another array was put in place that had a classification of 0. It should be noted that this process only occurred using the RPi3, the modified EMS device, and the forward-pass script. Because all of the components worked together, they could then be implemented into the previous generation neural prosthesis.

3.4.2 Implementation of all Modifications to the 3rd Generation Prosthesis

After the ANN and the EMS were combined in the previous script, these components had to be integrated into the third-generation prosthesis to create the fourth generation. There were two parts to construct the new generation device. One of the steps was to physically implement the solid-state switch circuit into the third-generation device and the next step was to implement the code developed in the previous section to the code that already existed.

The solid-state switch circuit needed to be implemented into the previous device because it was what controlled the EMS device contraction signal. As stated earlier, one of the sensors had to be removed because all the pins were being used by the previous prosthesis model. In this case we chose to remove the pelvis IMU because it did not contribute as much to the classifier. Once the IMU was dropped then the 35th pin was freed to be used as a connection to the EMS device. Figure 3.12 shows how the connections of the solid-state switch were changed to be integrated into the PCB board. The connections were similar to the connections made to the microcontroller.

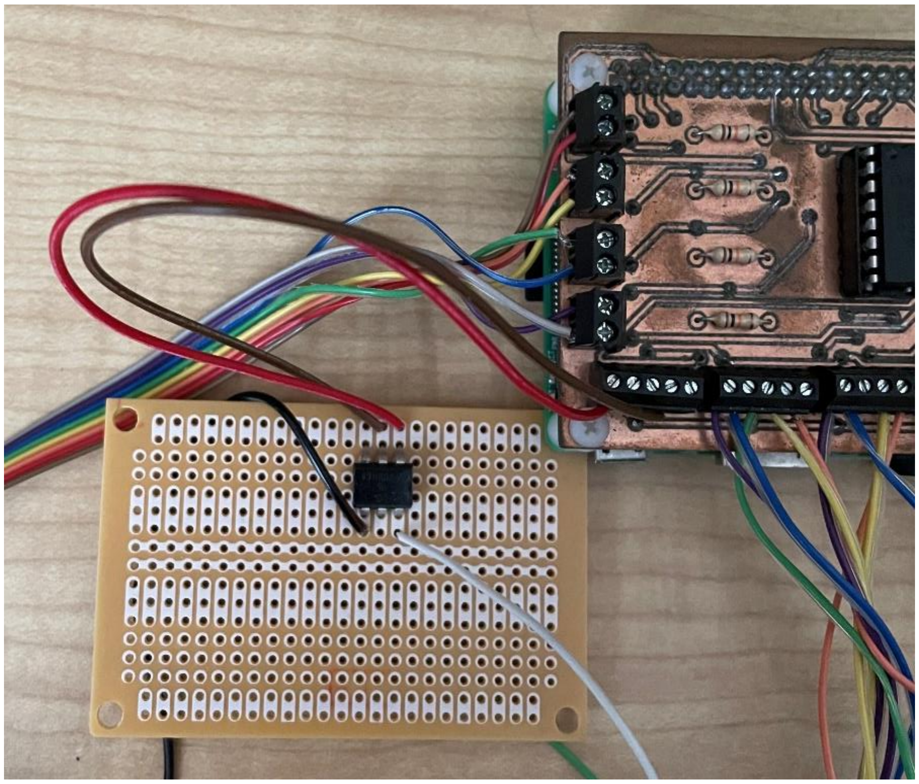
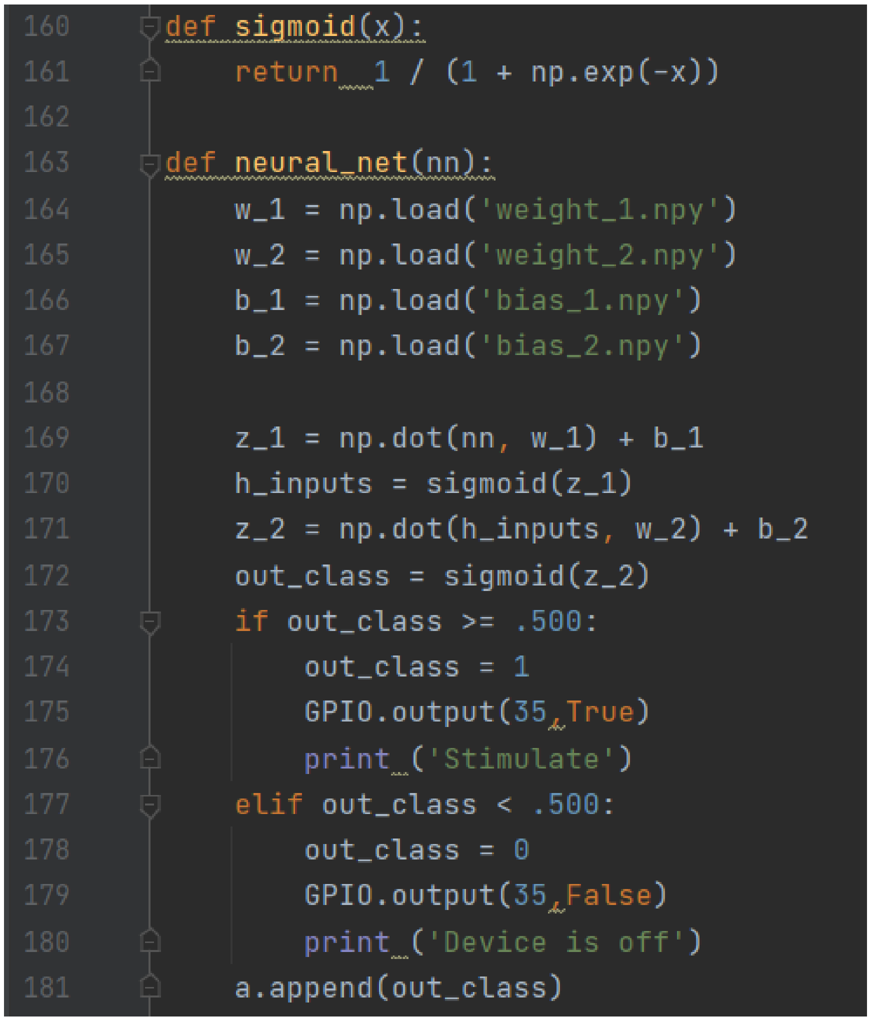


Figure 3.12: EMS integration to prosthesis
(Photo by Pablo Valenzuela)

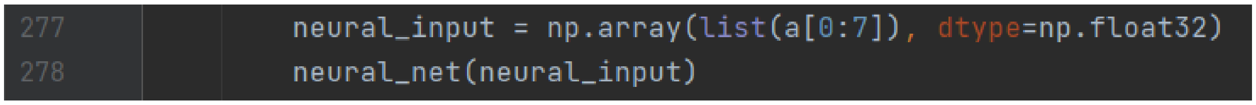
After the EMS device was integrated into the third-generation model, code changes were needed to enable the device to function. The code that was developed to combine the EMS device and ANN was used as the building block to modify the code of the third-generation prosthesis. Listing 8 shows how the code was implemented along with some changes.



```
160 def sigmoid(x):
161     return 1 / (1 + np.exp(-x))
162 
163 def neural_net(nn):
164     w_1 = np.load('weight_1.npy')
165     w_2 = np.load('weight_2.npy')
166     b_1 = np.load('bias_1.npy')
167     b_2 = np.load('bias_2.npy')
168 
169     z_1 = np.dot(nn, w_1) + b_1
170     h_inputs = sigmoid(z_1)
171     z_2 = np.dot(h_inputs, w_2) + b_2
172     out_class = sigmoid(z_2)
173     if out_class >= .500:
174         out_class = 1
175         GPIO.output(35,True)
176         print ('Stimulate')
177     elif out_class < .500:
178         out_class = 0
179         GPIO.output(35,False)
180         print ('Device is off')
181     a.append(out_class)
```

Listing 8: Feed-forward pass calculation

Lines 160-172 remained the same as the forward pass code for the combination of the EMS device and the ANN. The changes that were made to the code were in the *if* and *elif* statements. The changes that occurred in these sections were that the pin output changed to pin #35 instead of being kept at pin #4. Pin #35 was used because it was the pin that freed up after the pelvis IMU was removed. Another change that occurred was line 181 which appended the value that was predicted to the array that was stored in the CSV files. This made it easier to see the classifications of each point when the fourth-generation prosthesis was tested. Listing 9 shows the last two lines that were changed in the prosthesis code to make sure that all components worked together.



```
277            neural_input = np.array(list(a[0:7]), dtype=np.float32)
278            neural_net(neural_input)
```

Listing 9: Classification code for live trials

Lines 277 and 278 are lines that are introduced during the “try” part of the third-generation prosthesis code. The try command basically runs the script written underneath it like a while loop, this script can also be stopped from running when the user presses CTRL+C. After FSRs and IMUs are set up and used to collect a data point, an array is created to be input into the *neural_net* function that was defined previously in line 163. The *neural_input* variable takes in the list that is created by all the sensors and converts it to a *numpy* array which can be used by the forward pass script to classify the data point. Once all the code implementations were made, the device was ready for testing on a participant. Figure 3.13 shows the final model of the 4th generation neural prosthesis. The complete code for this model can be found in Appendix C.

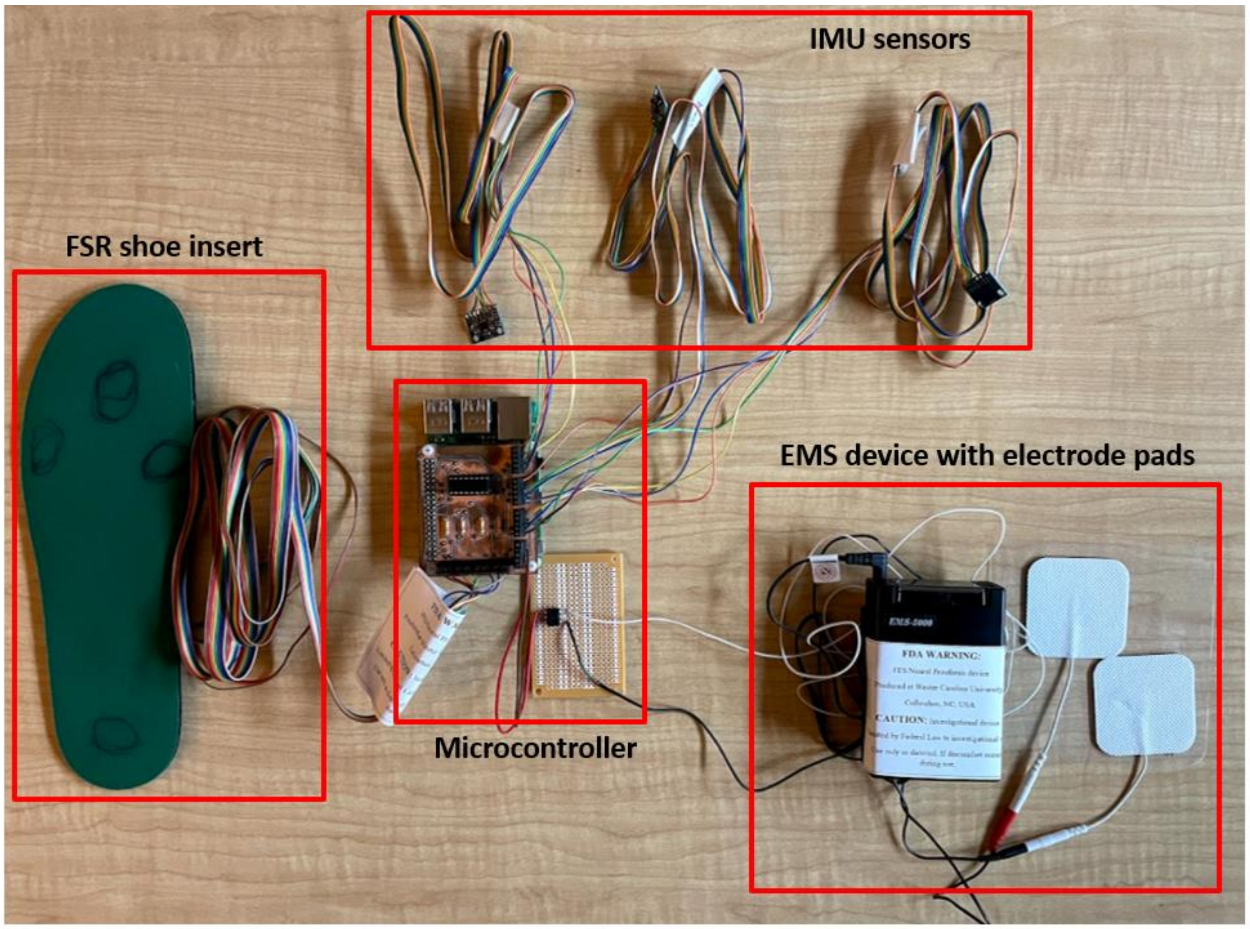


Figure 3.13: Fourth generation neural prosthesis device
(photo by Pablo Valenzuela)

3.4.3 Fourth Generation Device Testing

Once both the script and the EMS device were integrated, a test was performed to see if the device could stimulate the GN and SL muscles at the proper time. Testing the device followed a similar format to the trials done during data collection but in this case the EMS device was attached to the person, and the electrode patches were also attached to the person. This can be seen in Figure 3.14 and Figure 3.15.

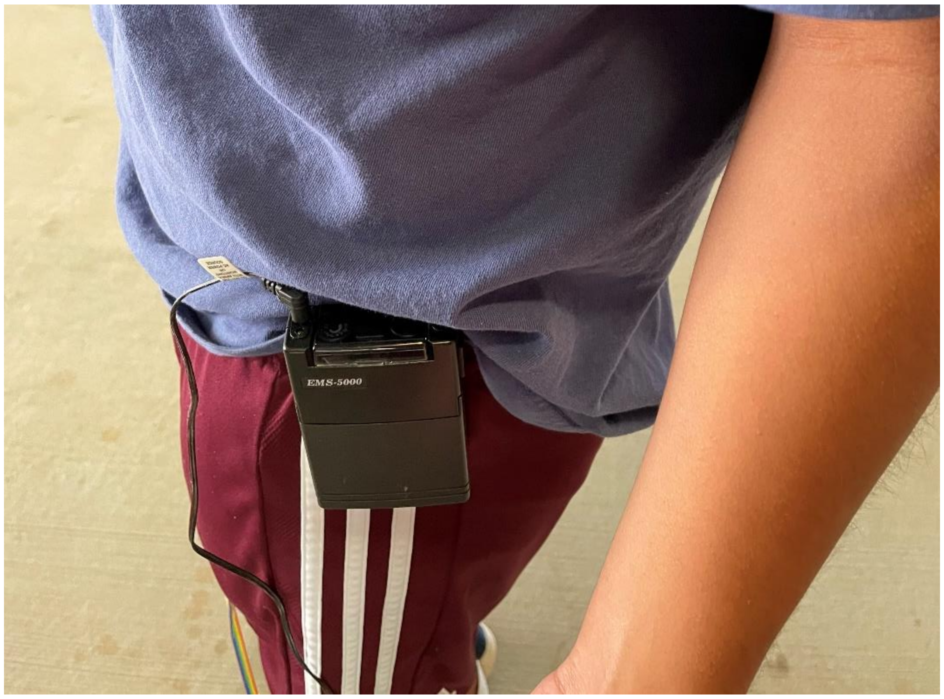


Figure 3.14: EMS attachment to participant
(photo by Pablo Valenzuela)

The EMS device was attached to the waist of the participant to have an easier path to the leg area, so the participant did not have to hold the device along with the prosthesis. Figure 3.15 shows the electrode patch placements that were used for the testing of the device. The electrodes were placed in the calf area where the GN and SL muscles are located.

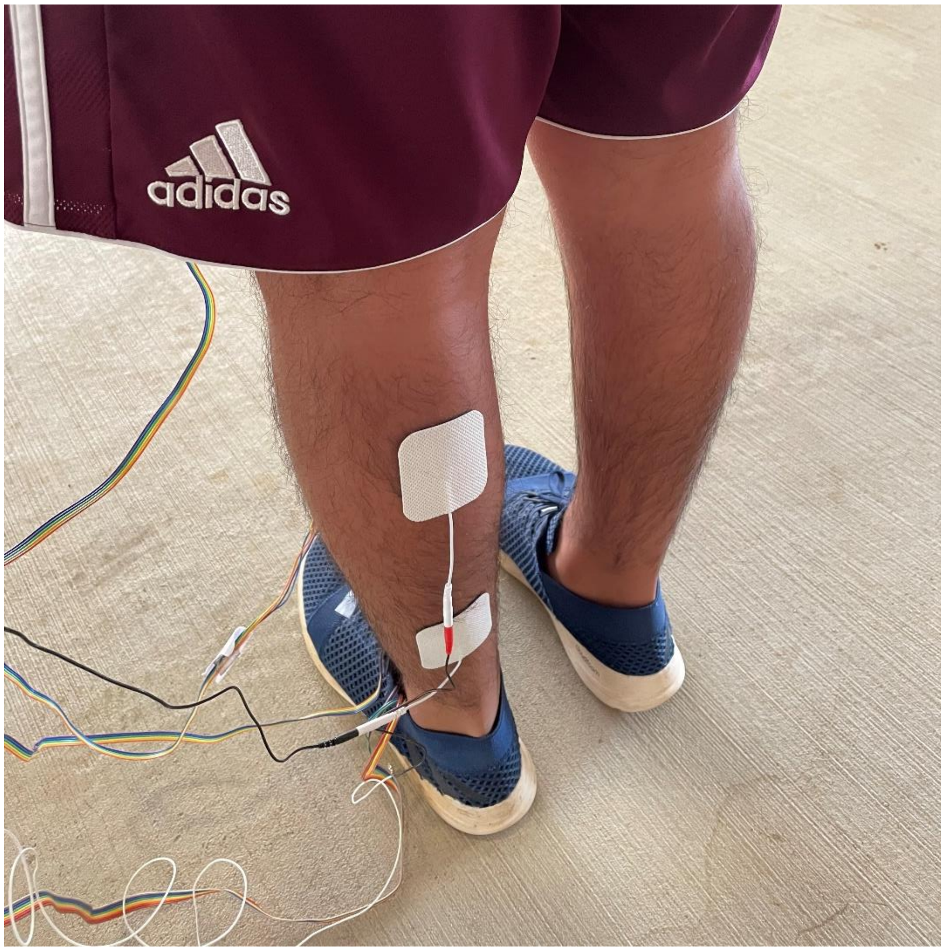


Figure 3.15: Electrode patch attachment locations
(photo by Pablo Valenzuela)

A total of four trials were performed during one testing session. Like the collecting data phase, the participant was given time to acclimate wearing the device prior to testing so that the gait cycle could be measured accurately. The participant was instructed to walk 10 meters at the three previously mentioned speeds (normal, fast, and slow). In addition, the participant was asked to note when the stimulation occurred since there was no easy way to see if the device worked besides the printouts on the main screen and the classifications recorded in the CSV file. The last test trial was used to really notice when the stimulation happened, for this particular trial the participant was asked to go through the motions of gait slowly and say when they felt the stimulation occur, this was compared to the screen printouts and the classification of the point to see if it accurately represented when the stimulation needed to occur. The participant stated that the stimulation occurred when the heel left the ground, and their weight was transferred to the toes which is when toe off happens. These tests were then graphed to see how closely the stimulation occurred to the approximate 60% of the gait cycle.

CHAPTER IV: RESULTS

This section discusses the results that were obtained from the tests that were presented in the methods section. These results aid with the understanding of certain choices that were made during the development of the neural prosthesis and shows the accuracy of the final model as well as the accuracy of the 4th generation neural prosthesis.

4.1 Third Generation Neural Prosthesis Performance Results

Cross-correlation was used to validate the use of the 3rd generation prosthesis to collect the angles of the leg during gait cycles. The results measured by the prosthesis were done using a complimentary filter that had 98% weight on the gyroscope and 2% weight on the accelerometer. The results were then compared to the measurements collected by a motion capture system and analyzed for delay associated with the gyroscope. The figures in this section show the angle measurements of each joint. Figure 4.1 and Figure 4.2 show the cross-correlated ankle joint graphs and associated time delays. The ankle joint angles were calculated by subtracting foot IMU measurements from the shank IMU measurements. This image shows the average delay of all 14 gait cycles that were collected during testing trials of the 3rd generation neural prosthesis.

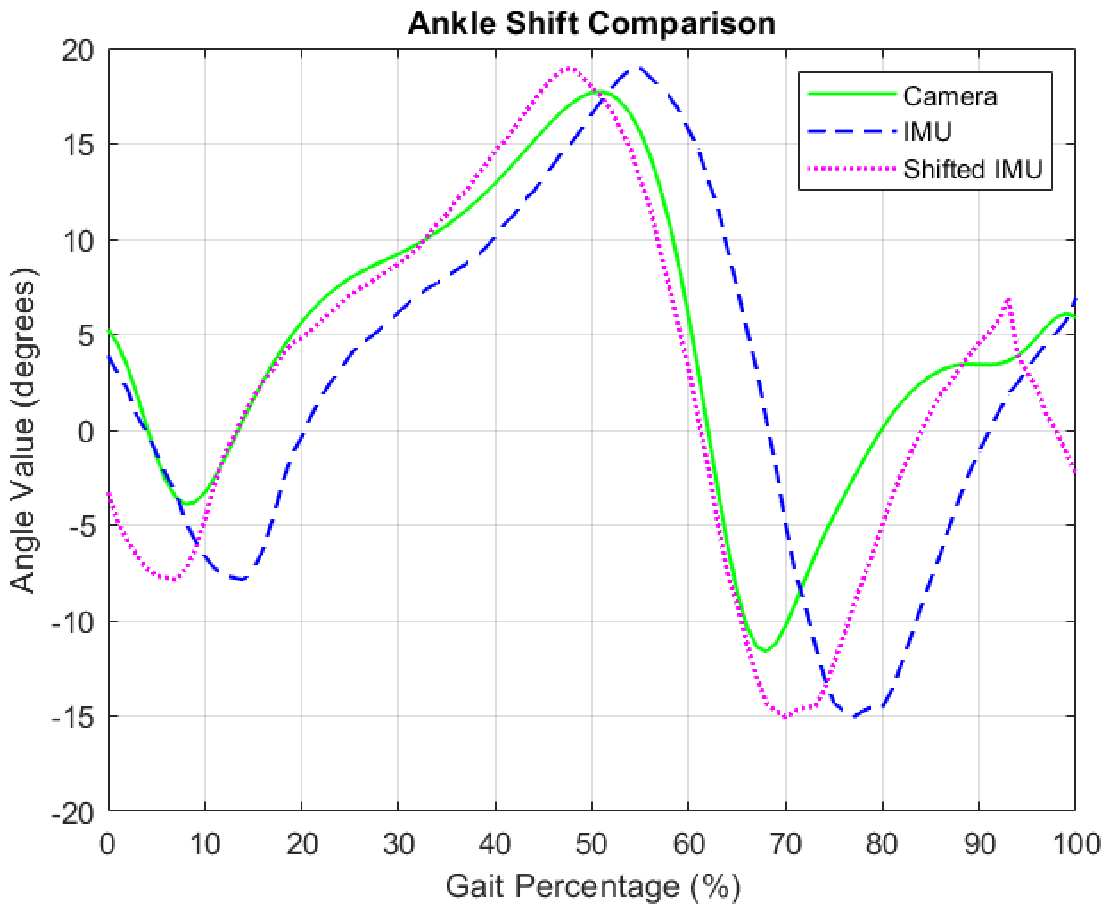
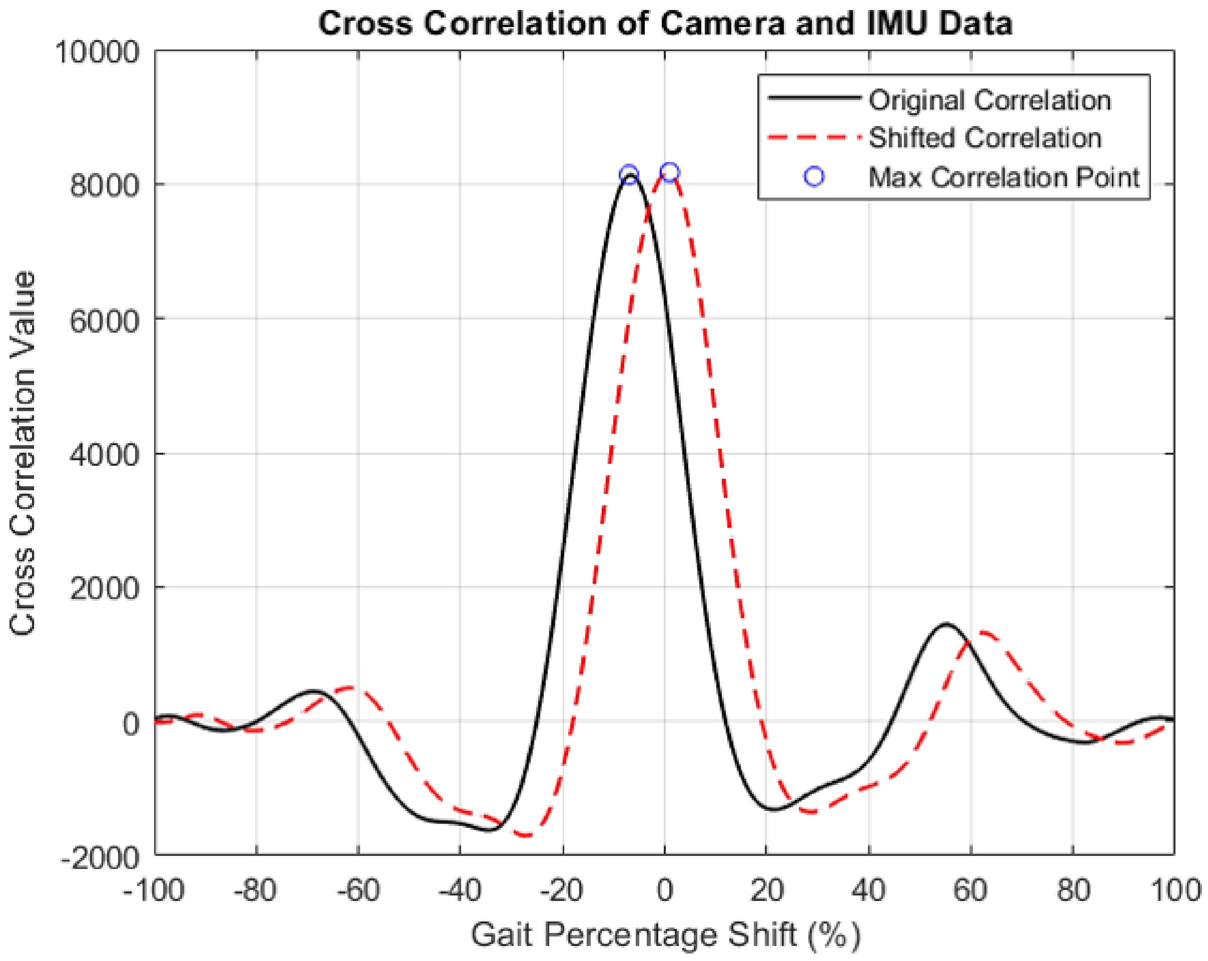


Figure 4.1: Ankle joint angle comparison

The ankle shift comparison shows the original plot of the average IMU ankle angle measurements over the gait cycle (blue dashed line), the average camera angle measurements (solid green line), and a shifted IMU (magenta dotted line) to show how the lines are more closely correlated after accounting for the delay. The camera and IMU curves follow a similar shape, from 0-10% it is visible that the unshifted data has the best fit, but data between 10-65% of the gait cycle the shifted IMU curve has a good fit, after 65% there is no close fit, but the lines seem parallel which suggests that, once shifted, the only difference is the magnitude of the angle values that were collected. The cross-correlation plot shows how the camera data and the IMU data are correlated with each other, as stated in the methods section the higher the value the greater the correlation between the curves. The figure shows the highest value of correlation and its percentage location delay as well as the time delay.



```
The lag delay is estimated to be: 7 percent to the left
The expected time delay is: 0.083300 seconds.
```

Figure 4.2: Ankle cross-correlation test

The ankle joint has many recognizable features in its curve which helped the cross-correlation algorithm to quantify the time delay. When compared to the control, the calculated gait delay was about 7%, about 83 ms. The delay associated with the IMU, along with the delay of the EMS device ramp up time were accounted for in the classification of the data that trained the neural network. This allowed for muscle stimulation timing to remain accurate when testing the final device.

In a similar way to the ankle angle measurements, the knee joint data points and curves were also averaged and cross-correlated, the knee joint measurements were calculated by subtracting the shank IMU angles from the thigh IMU angles. Figure 4.3 and Figure 4.4 show how the knee joint IMU data set compared to the data set collected by the camera. The colors of the curves are the same as with the ankle joint. Figure 4.4 shows the original correlation between the curves, the original correlation being left of the neutral zero point which indicates a delay in the IMU sensors, the delay for this joint is shown to be smaller than the ankle joint by half. The delay was shown to be 4% of the gait cycle time, which was calculated to be 45 ms. Figure 4.3 shows the angle measurements curves of both the IMUs and the camera and shows an adjusted IMU curve due to the delay. After the IMU curve was adjusted, the recognizable features look more closely related than the ankle joint results. This is possibly because the curves for the knee joint do not have such prevalent slope changes compared to the ankle joint data which can explain why there was a smaller percentage of delay. There is one steep curve in the knee graph which is the one that is best correlated once the curves are adjusted. All the curves seen in the knee joint graph seem to fit well once adjusted.

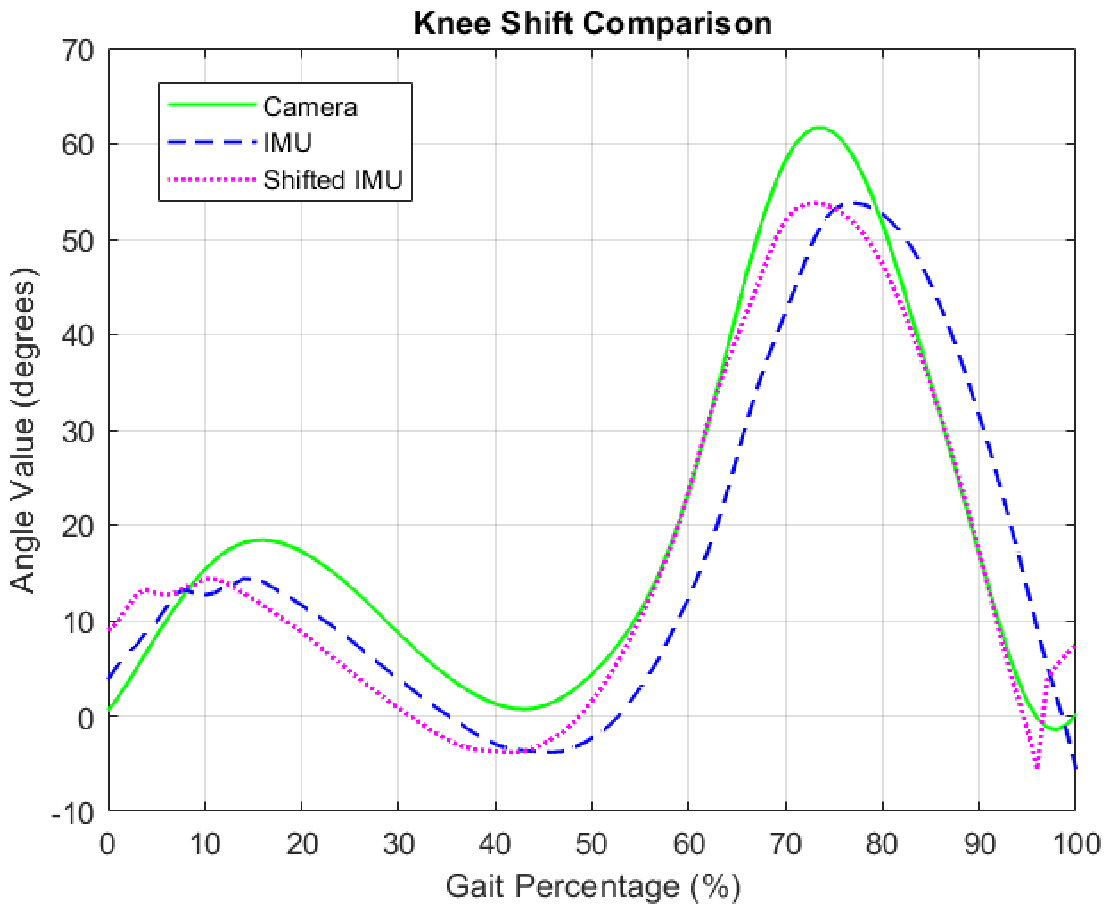
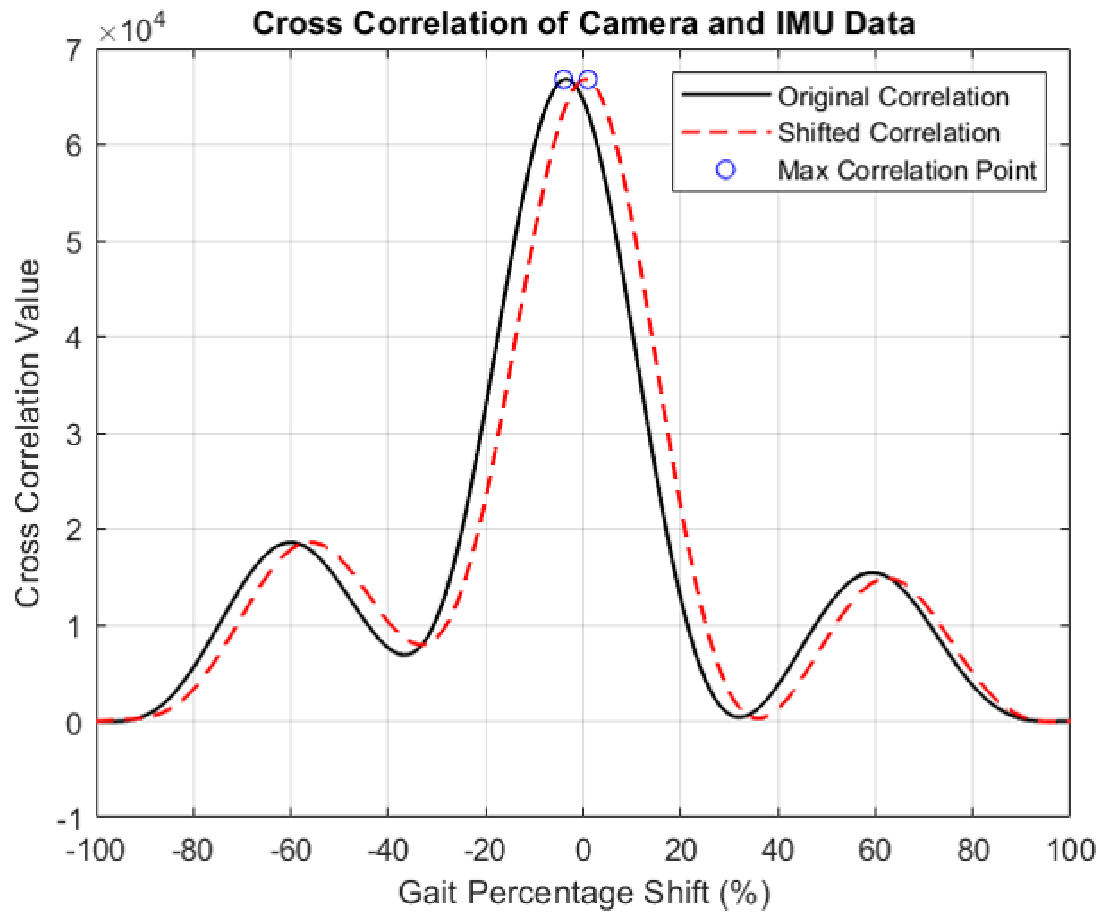


Figure 4.3: Knee joint angle comparison



```
The lag delay is estimated to be: 4 percent to the left
The expected time delay is: 0.044514 seconds.
```

Figure 4.4: Knee cross-correlation test

Figure 4.5 and Figure 4.6 show the cross-correlation of the hip joint. The hip joint measurements were calculated by subtracting the thigh IMU angles from the pelvis IMU angles. Unlike the ankle and knee joints, the hip joint curve did not have any features besides the large dip at the bottom of the curve, this made it difficult to analyze and therefore there is no visible delay. Figure 4.6 shows how the cross-correlation algorithm could not notice any delays and instead has the correlation center at 0% which means there is no delay. Figure 4.5 shows the angle differences between the thigh IMU and the pelvis IMU. Looking at the camera curve, one can see that as the body reaches 60% of the gait cycle (toe-off) the angle difference is decreasing to a 0 degree difference and then increases the other way, this occurs because as the person goes through the first 30% of the gait cycle the camera sensors align with each other but as the gait cycle moves towards toe-off the thigh sensors move backwards in comparison to the pelvis sensors creating the negative value. This movement was picked up well by the camera but the IMUs had a harder time capturing this movements which is why the curve seems wider and not as accurate.

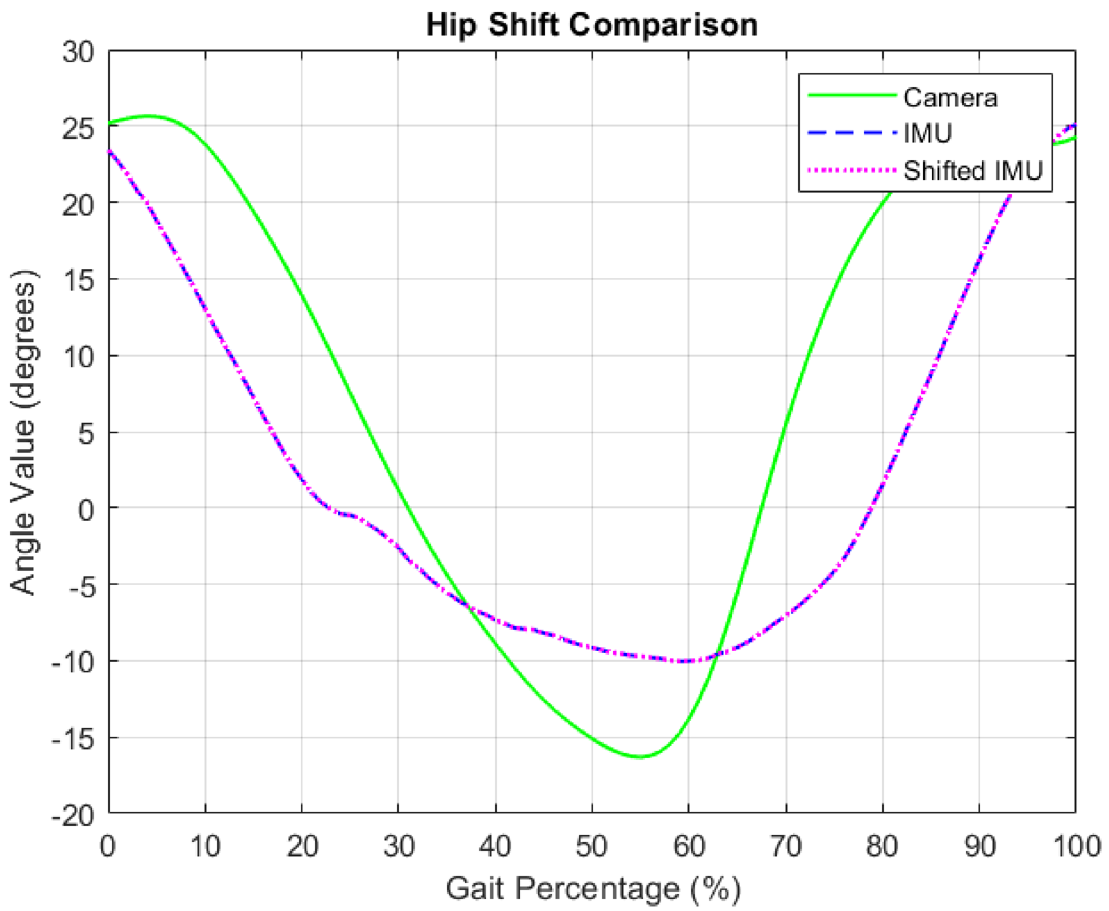
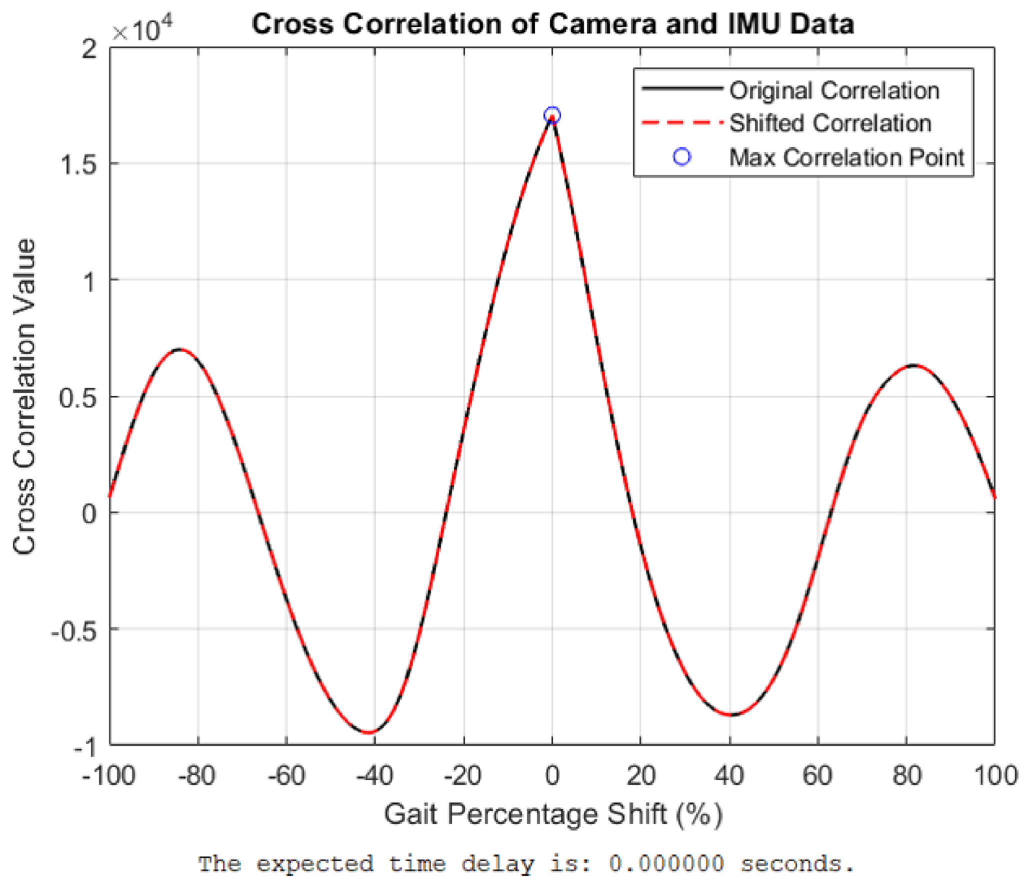


Figure 4.5: Hip joint angle comparison



```
The expected time delay is: 0.000000 seconds.
```

Figure 4.6: Hip cross-correlation test

4.2 Microcontroller Control of EMS Device Results

The function of the EMS device was changed to be controlled by the microcontroller. At first the EMS device was controlled by a push button, but after some electrical modification and code implementations the push button was switched to a solid-state switch that allowed the microcontroller to activate the contraction function of the EMS device. Two types of tests were used to see if the device worked. The first required that the electrodes and the electrode pads be connected to an oscilloscope to see the behavior when the device was in the contraction state. The device was turned on the fifth setting at 30 Hz and 100 Hz. Figure 4.7 shows a representation of what was observed on the oscilloscope. The blue line shows the max voltage that was pulled at 30 Hz at the fifth setting while the red dotted line shows the max voltage that was pulled at 100 Hz at the fifth setting.

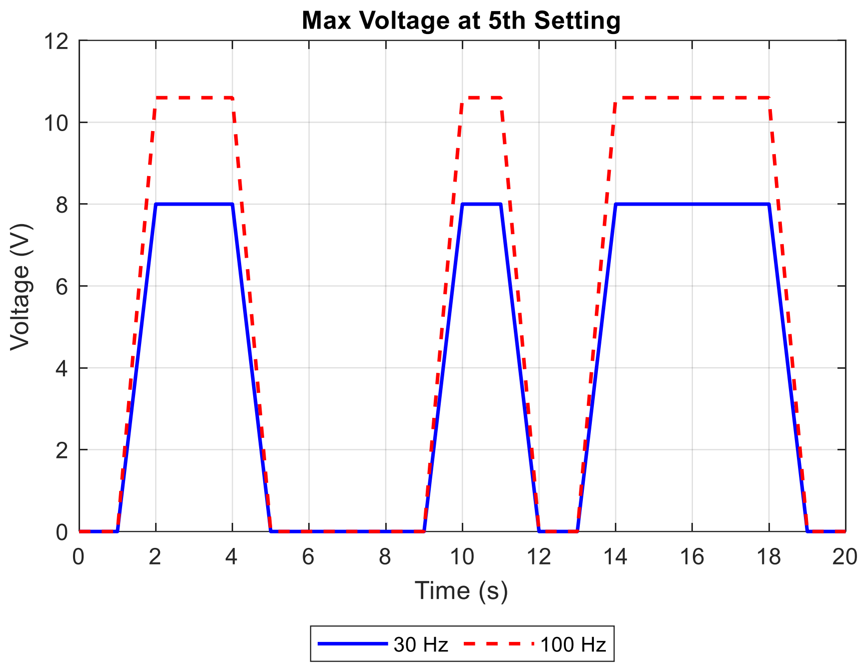


Figure 4.7: Recorded voltage output of EMS device after modifications

The time parameters used for both frequencies were the same. Once the timer reached one second, the user input "1" into the script. This raises the voltage that is received on the electrode pads. The same input is used at 9 seconds and 13 seconds. Once the timer reached 4 seconds then a "0" was entered into the script which lowered the voltage output on the electrode pads. The same input was used at 12 seconds and 19 seconds. At the 20 second mark the input was "5" which effectively shut down the program. The second test occurred informally since there was no real way to obtain a number. The second test consisted of putting the electrode pads on a person's arm and for them to input "1" or "0" whenever it was preferred. The person felt the contraction of the forearm muscles when the code input was 1 and felt the muscle relaxed once the input became 0.

4.3 Neural Network Model Classifications

This section shows the results for the different neural network models that were used to see which sensors were the most useful, and if a neural network was able to correctly classify when the artificial stimulation on the leg should occur. Figure 4.8 shows the graphs for the classifications of each model for the test data set. The first model used all the sensors (FSRs and IMUs) to train and classify data, the second model used only the IMUs, and the third model used only the FSRs. The actual classifications (controlled classifications) are shown by the orange lines in all the plots. The classifications obtained by using all the sensors is shown by the blue line, the classifications given by only IMUs is shown in grey in the second plot, and the classifications obtained from only using the FSRs is shown by the yellow curve.

The classifications ranged from 0 to 1 because this was a binary classification model. Using all the sensors show that the classifications accurately follow the same curve of the expected classifications, there is only one point in which it misclassified the data. Using all the sensors had an accuracy of 99%. Only using the IMUs show that the classifications are off, and the neural network was not able to classify a lot of the data points correctly, this model only reached a 90% accuracy. The model that used only FSR data for classification had a 98% accuracy, the model shown in the last graph of the classifications plot shows both the actual classifications curve and the FSR classification curve to be almost identical, there is some overlay which attributes to this model being 1% less accurate than using all the sensors, but all the gait cycles were predicted correctly.

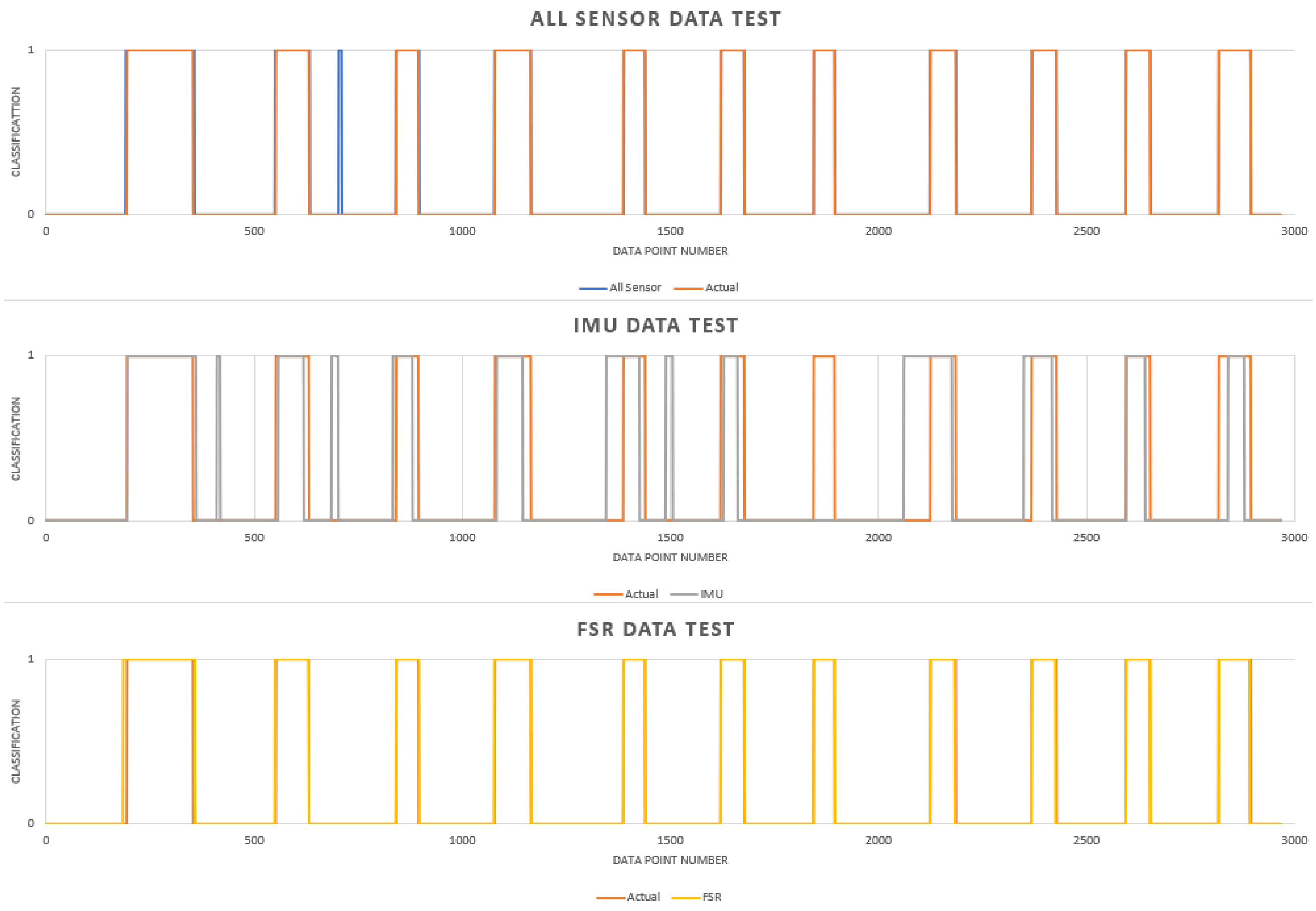


Figure 4.8: Knee Cross-Correlation Test

4.3.1 Final ANN Model Validation

The final neural network model included seven input nodes, 13 hidden layer nodes, and one output node. Once the model was trained and tested with the original training and data sets, it was observed that the network had an accuracy of 98.9%. However, further tests were needed in order to see how well the network responded to randomized data, and to see if the neural network was fit properly. Three more data sets were created for the purpose of testing the network. Each test set had a different data pattern since each set was a randomly sliced version of the original testing data set. Table 2 through Table 4 show the different results that were observed after the neural network tested the new data sets. Table 2 shows the confusion matrix for the first data set that was used to test the accuracy of the network. There were a total of 2276 data points in the data set. The actual class (expected) is shown on the left, and the class predicted by the ANN is located at the top. Most of the points fall in the green cells, this meant that those points were predicted correctly. The numbers in the red cells are those that were classified incorrectly. In the first model there were 19 points that were predicted as the ON (class "1") class even though their actual classification was OFF (class "0"). There were also five misclassified points for the ON class. The accuracy of the first model was calculated to be 99% which aligned with accuracy level of the original testing data set.

Total Data Points 2267		**Predicted**		
		Off	On	**Total**
Actual	OFF	2151	19	2170
	ON	5	92	97
			Error	1.0%

Table 2: First test data set confusion matrix

The second model, shown in Table 3, also had a total of 2267 points to classify. Out of all those points, 2243 were classified correctly and 24 points were classified incorrectly. This model also gave an accuracy of 99.0% similar to the accuracy levels of the other models. Table 4 shows the confusion matrix for the last test data sets. This set had the same number of points to classify, 25 of those points were classified incorrectly, and 2242 points were classified correctly. The last test set showed an accuracy of 98.9%.

Total Data Points 2267		**Predicted**		
		Off	On	**Total**
Actual	OFF	2152	18	2170
	ON	6	91	97
			Error	1.0%

Table 3: Second test data set confusion matrix

Total Data Points 2267		**Predicted**		
		Off	On	**Total**
Actual	OFF	2151	19	2170
	ON	6	91	97
			Error	1.1%

Table 4: Thirds test data set confusion matrix

The average accuracy of all the test trials was 98.97% which was close to the original accuracy from the first test of the ANN. The closeness of these values suggested that the network was properly fit, and it could be used to progress the development of the fourth-generation prosthesis.

4.4 Fourth Generation Neural Prosthesis Results

The 4th generation prosthesis results come from the data collected from the testing of the prosthesis during which the participant walked 10 meters on a flat surface. A total of 4 tests were conducted, and there was an average of 16 gait cycles that were recorded, Figure 4.9 through

Figure 4.15 show the curves of all the collected gait cycles along with their averages. Figure 4.9 shows the results for the 5th metatarsal FSR, the black line represents the average of all the gait cycles that were collected. The values show how the participant transfers their weight as they walk, the curve starts at a force of 0 N but it goes up as the gait cycle continues eventually going back to 0 N right after 60% of the gait cycle since that is when toe off occurs and no more weight is measured in the swing phase.

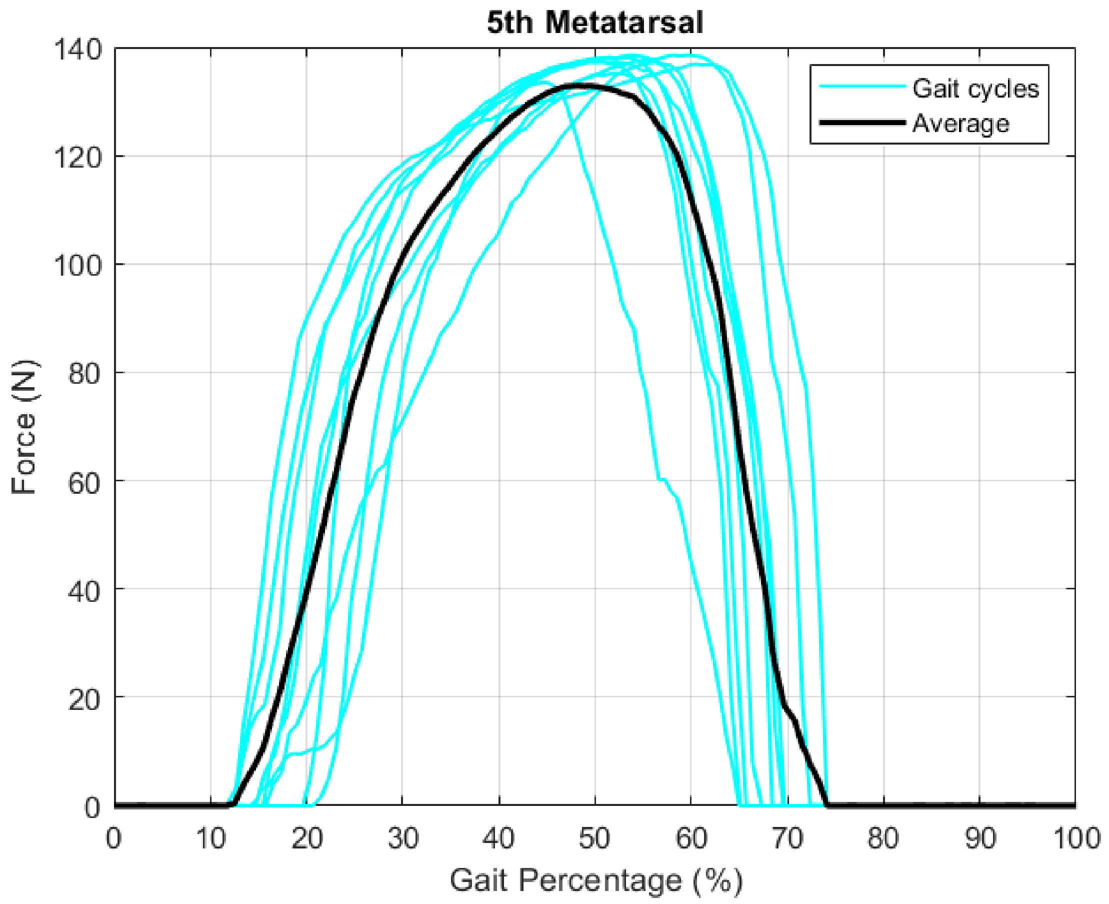


Figure 4.9: Fifth metatarsal FSR data

Figure 4.10 shows the 2nd metatarsal FSR data. This plot also shows the average of all the gait cycles with the black curve. This FSR follows a similar trend to the 5th metatarsal, the data starts at 0 newtons and then gets higher as the foot is from heel-strike to toe off. This curve shows that the force first flows to the 2nd metatarsal before the 5th metatarsal, this is due to the fact that the inside of the foot is the first one that has contact with the ground, but it still force does go down after the 60% because the foot is no longer making contact with the ground.

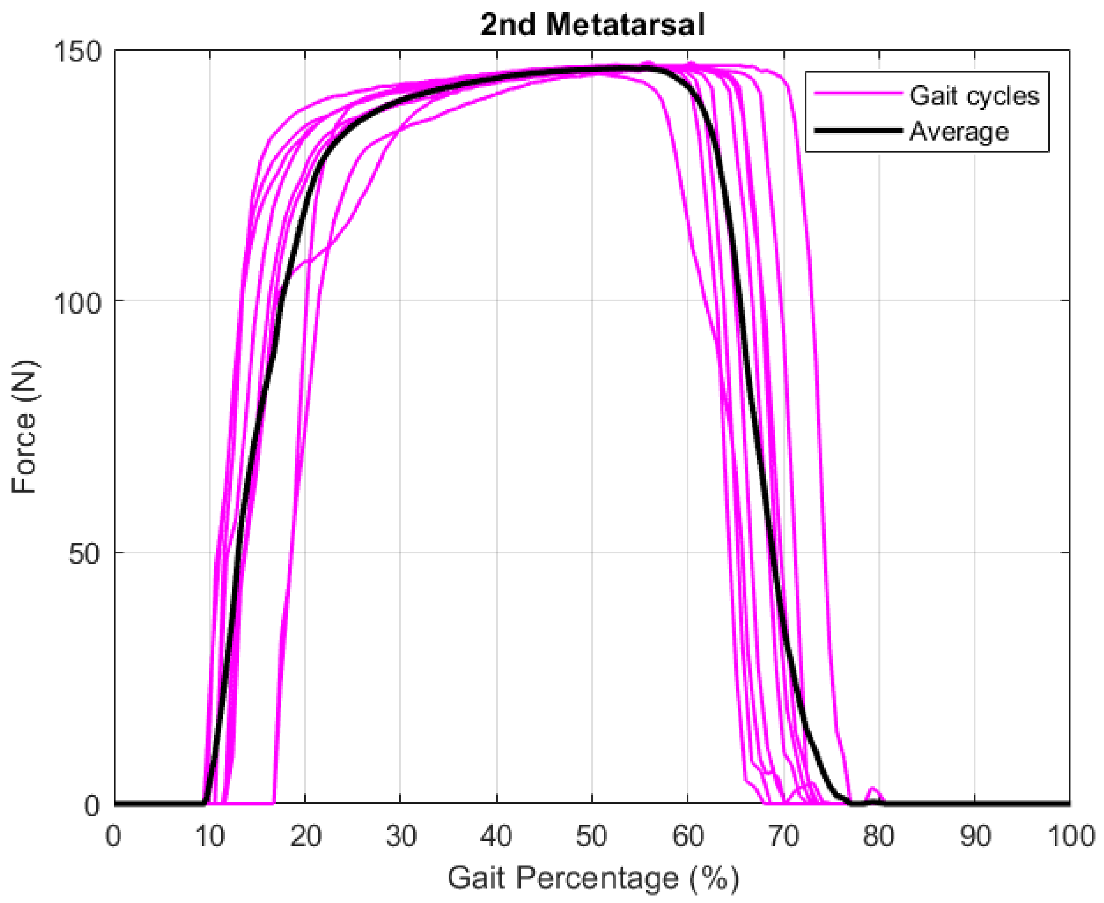


Figure 4.10: Second metatarsal FSR data

Figure 4.11 shows the results for the toe FSR. Like the 5^{th} metatarsal and 2^{nd} metatarsal, the toe FSR also starts at 0 N since there is no force there during heel strike. The force goes up as weight is transferred to the front of the foot eventually reaching the highest force at 60% of the gait cycle. The black curve again shows the representation of the average of all the gait cycles that were collected during this trial. There were some small curves near the end, but these may be residual forces that occurred due to the foot moving within the shoe.

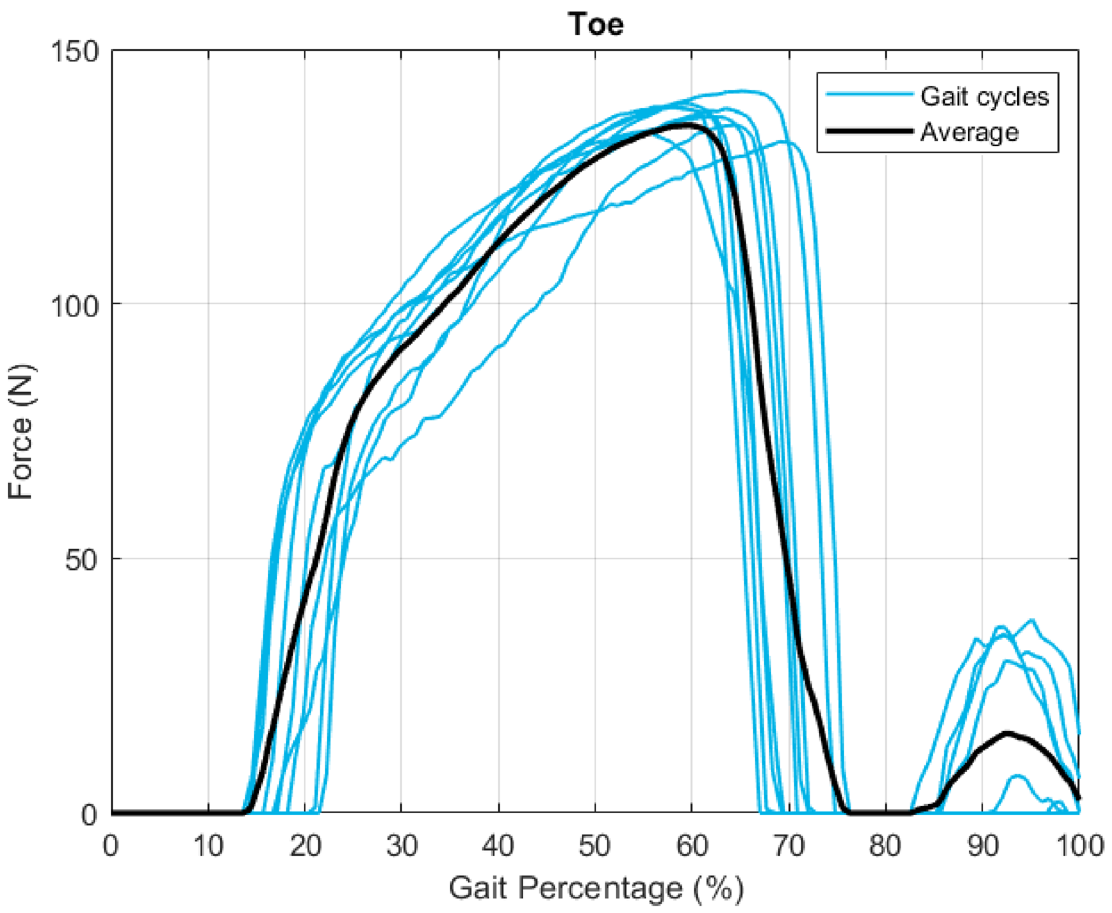


Figure 4.11 Toe FSR data

The last FSR that was looked at was the heel sensor. Figure 4.12 shows the heel FSR data that was collected during the testing. This is the one that differs from the rest since it begins with a maximum force, this shows when the heel strike is happening and the beginning of the gait cycle. As the gait cycle continues, the force goes down near 40% which is estimated to be when heel off occurs. This also lines well with the other plots since the other sensors start picking up the force at around 40% of the gait cycle.

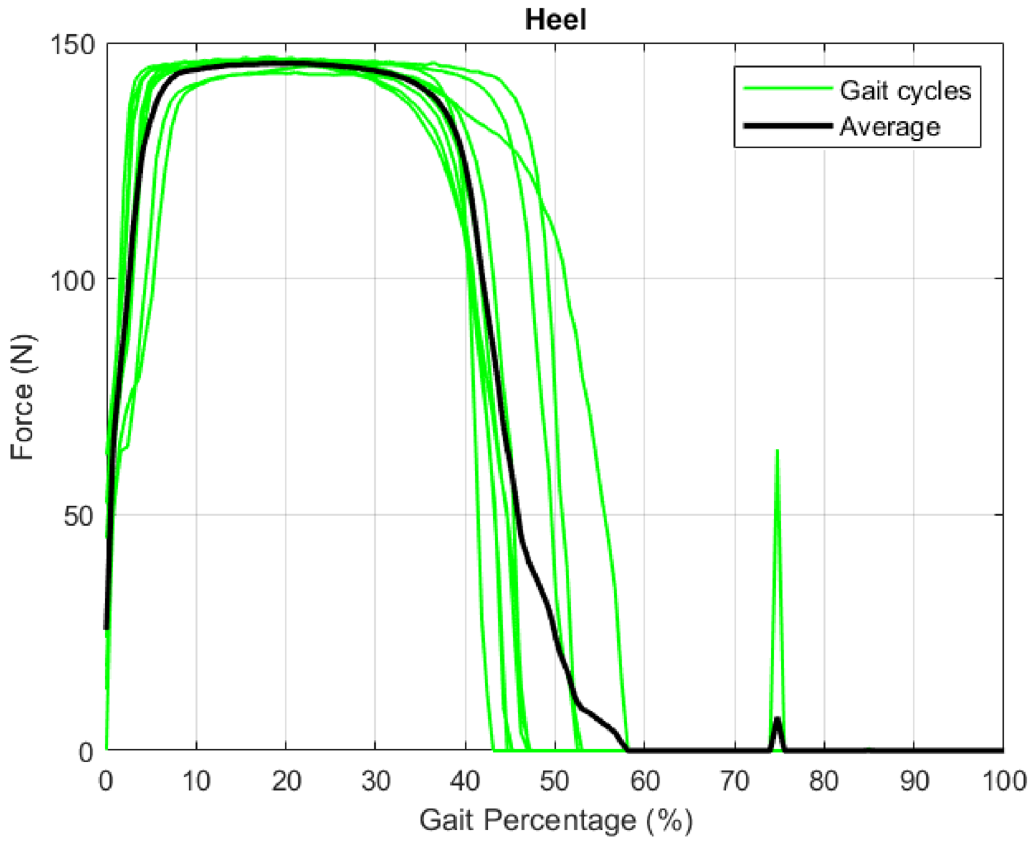


Figure 4.12: Heel FSR data

Figure 4.13 through Figure 4.15 show the IMU data measurements that were recorded during testing. In each graph the black line represents the average of all the gait cycles that were recorded. Figure 4.13 shows the measurements for the IMU located at the foot, Figure 4.14 shows the angle measurements of the IMU located in the shank, and Figure 4.15 shows the angle measurements of the IMU placed on the thigh. These graphs do not show the same pattern as previously seen in the cross-correlation plots because these are raw data scores. The cross-correlation results showed the plots of the ankle, knee, and hip joints. The raw data scores show the angle measurements recorded by each IMU at the foot, shank, and thigh. Before each trial, the IMUs were calibrated by asking the participant to stand still for 15 seconds which allowed for the gyroscope and accelerometer to settle down. It should also be noted that there is some adjustment to be done due to the delay associated with the current IMUs.

The foot angle graph shows the behavior of the foot IMU at different stages of the gait cycle. At the beginning the foot has a higher degree value because at heel strike the foot is pointing up, as the IMU moves to the foot-flat motion one can see that the angle measurement is closer to 0 degrees. After 45% of the gait cycle the IMU increases its angle values as the foot moves into the stages of heel-off and toe-off. After toe-off it can be observed that the angle increases on average to 12 degrees, at this stage the foot has the greatest difference since it will be close to vertical as it takes off during the swing phase and then returns to heel-strike.

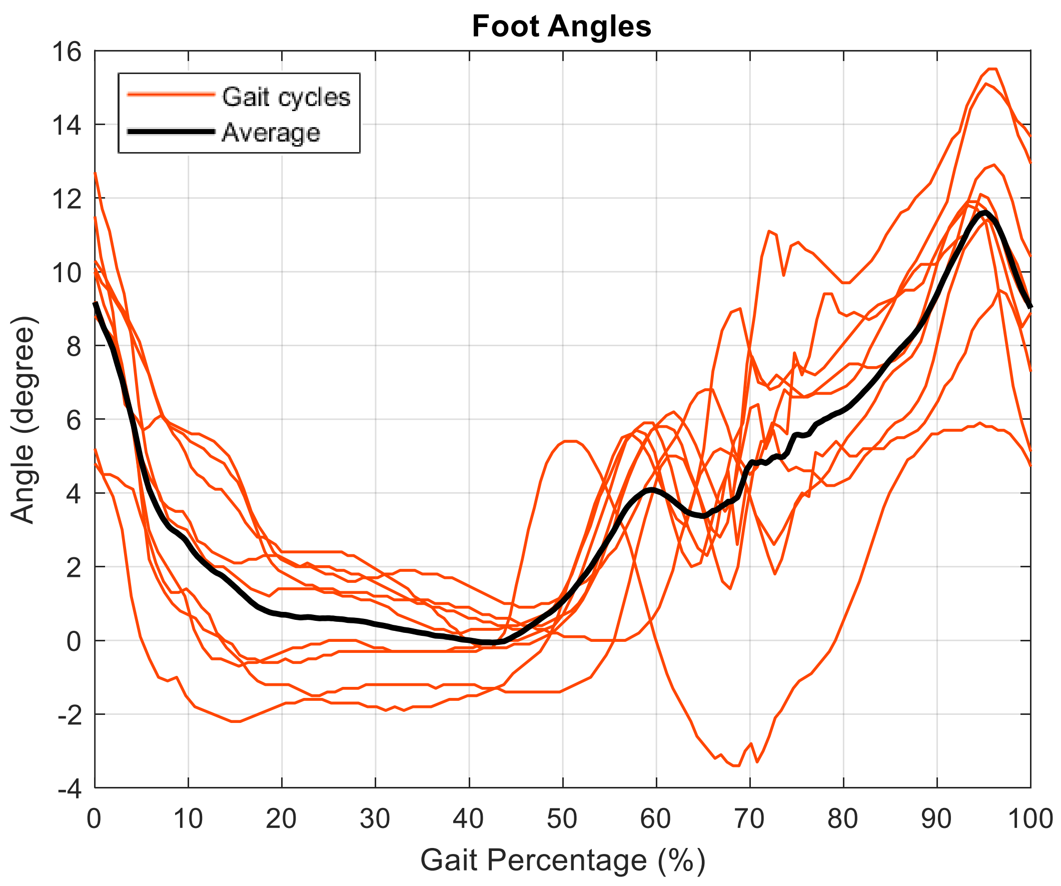


Figure 4.13: Foot IMU data

The shank angles show a different angular scale compared to the values of the foot IMU. This is due to the IMU already being placed near a 90-degree angle on the shank. During gait, the shank acts like an inverted pendulum that pivots at the ankle. The movement shown on the plot starts at the starting (neutral) position and as the body moves forward and the gait cycle progresses, the shank moves forward about the ankle. This movement increases the measured angle which is what is shown by the curves from the moment the gait cycle starts. It reaches a maximum near 70% of the gait cycle because that is when toe off occurs. After toe off the shank will have to go back to the neutral position to act as a support for the ankle which is shown by the curves decreasing in angle measurement from 70% to 100% of the gait cycle.

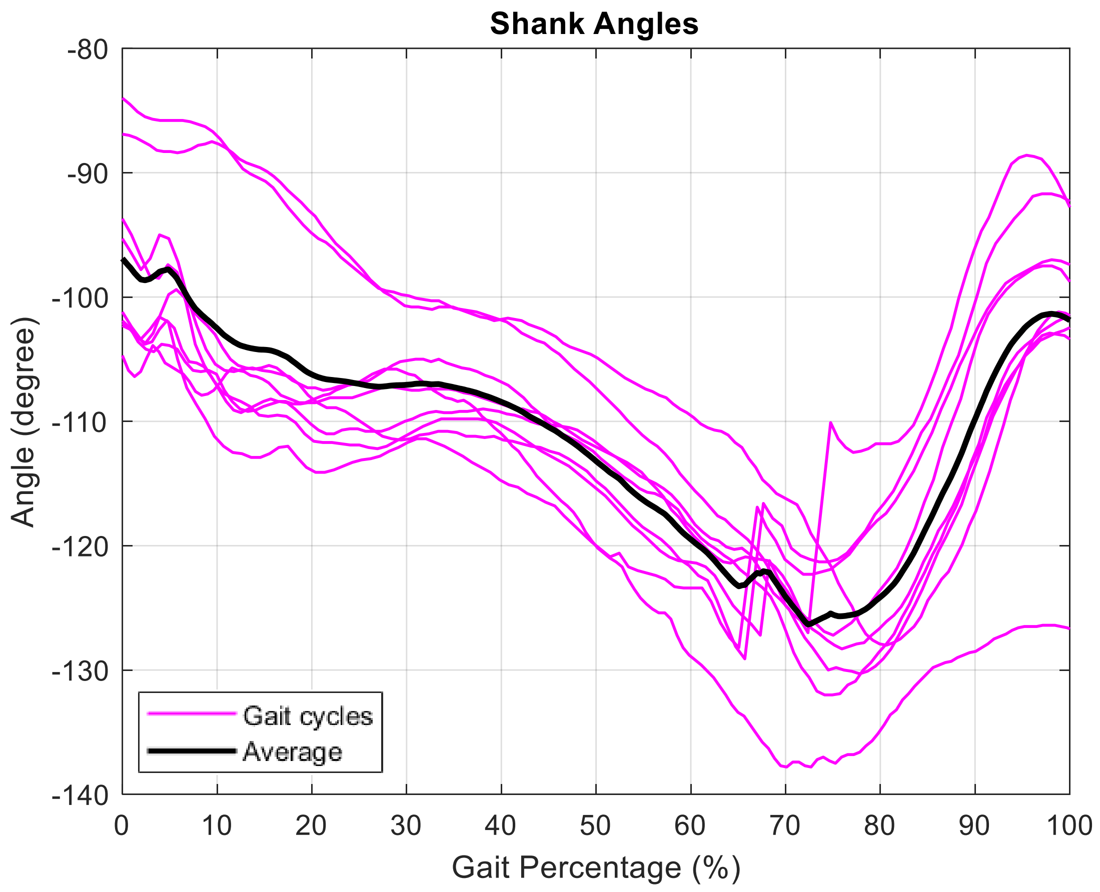


Figure 4.14: Shank IMU data

Figure 4.15 shows the angle values collected by the thigh IMU. The curve starts near the 75-degree value, this occurs because the thigh is located diagonally forward as heel strike occurs. In a similar behavior to the shank IMU the angle measurement keeps increasing in magnitude during the stance phase. This occurs because as the person approaches toe off the angle at the thigh keeps increasing as the foot maintains contact with the ground behind the body. After 60% of the gait cycle the thigh goes back to its original position which is why the degrees decrease in magnitude. These curves seemed to have the most dispersion which could have been caused by relative movement between the sensor and the thigh. After testing it was noticed in Figure 3.8 that the sensor was loosely attached to the clothing rather than being attached directly to the skin and wrapped with tape.

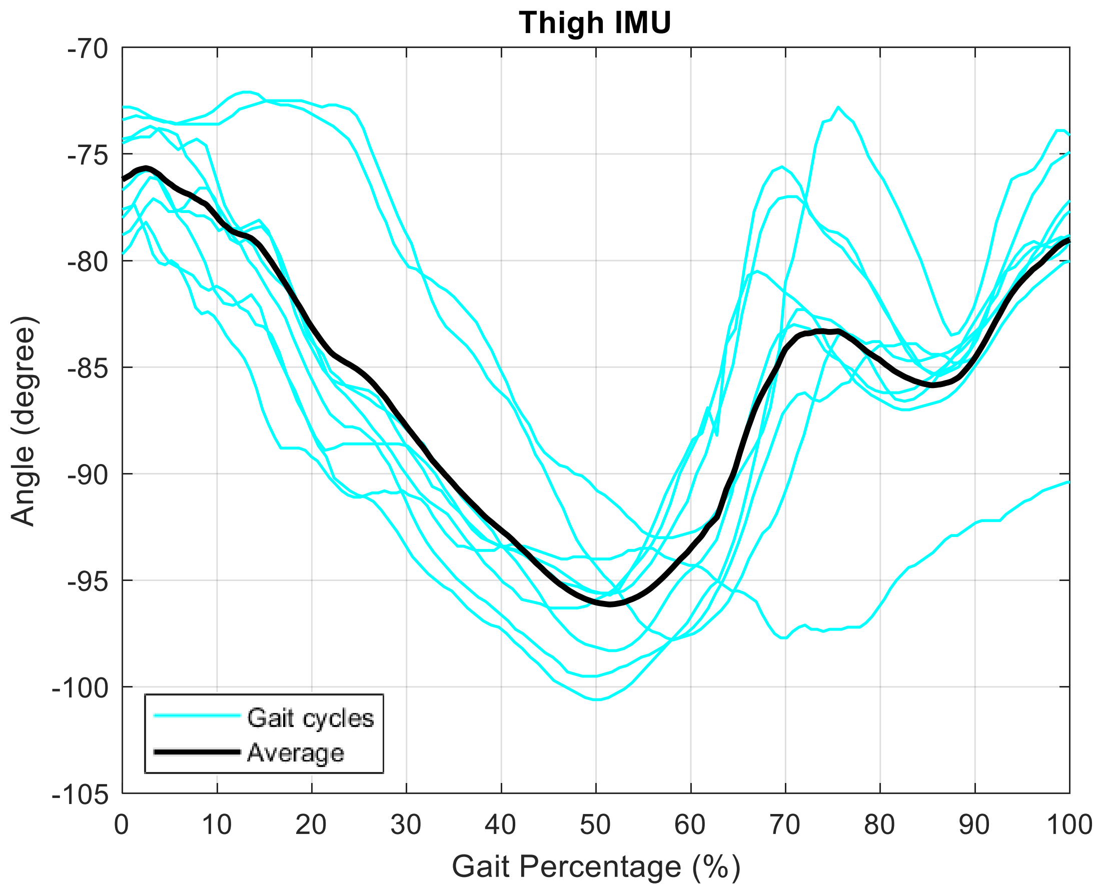


Figure 4.15: Thigh IMU data

Figure 4.16 shows the final classifications for the gait cycles. The classifications are either a 0 or a 1, if the data point is at 0 then the device is off, and if the data point is at one then the stimulation signal is being sent. Like in previous plots the black curve represents the average of all the curves in that plot, in this case the classifications plots for each gait cycle. The classification starts at 0 because that is when heel strike is occurring at 0% of the gait cycle and remains the same until after 40% of the gait cycle, that is when a change in class is visible. This occurred because during some gait cycles the participant may have had a different stride that made the stimulation occur faster. However, the stimulation stays on until 60% of the gait is completed, on average it is seen that the middle of the stimulation curve is during the 60% of the gait cycle, which is when the stimulation is needed the most.

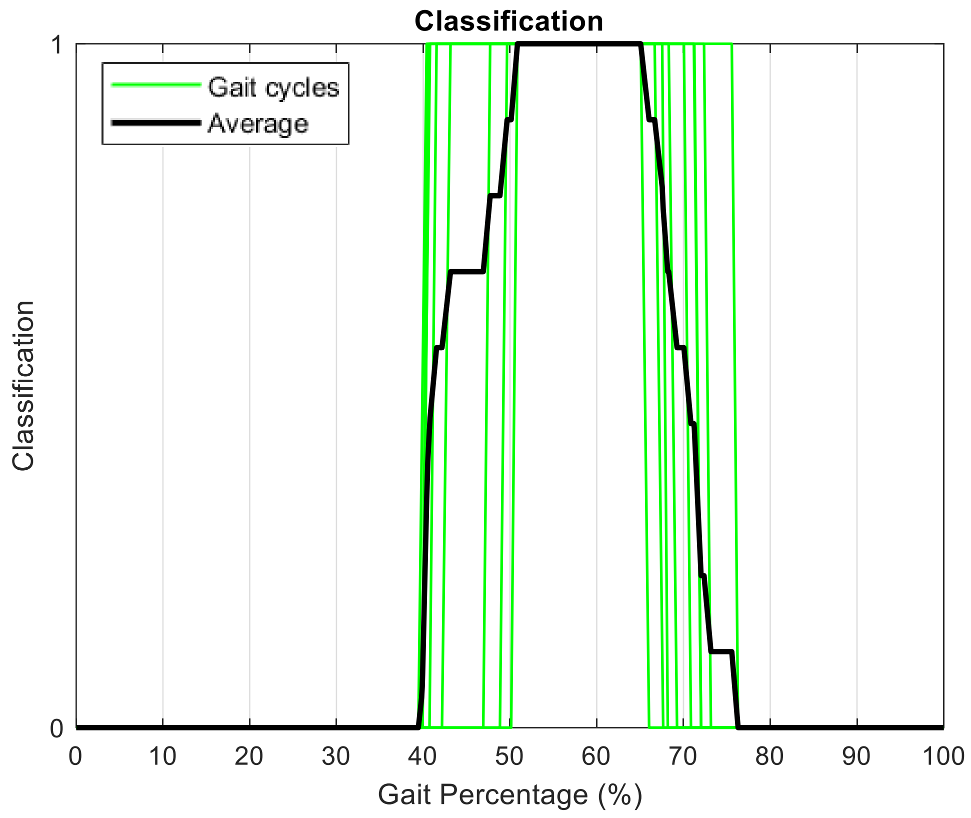


Figure 4.16: Classifications of test trials

CHAPTER V: DISCUSSION

When considering the performance of the IMUs the hip joint was the most “accurate” with respect to time delay but that was mainly due to not having recognizable features that the cross-correlation algorithm could quantify, in short, the magnitudes of the curves were relatively similar because they only had one dip, and this caused the code to misread it as no delay. In contrast, the ankle and knee joints had recognizable features that allowed the correlation algorithm to be able to calculate a delay. This occurred because the magnitudes of the points were not the same, so it was easier to establish the difference between the IMU curves and the camera curves. Overall, this analysis verified that the measurements provided by the IMUs were accurate and while there was some delay, the percentage was small enough to be accounted for by the ANN. Since the accuracy of the IMUs was at least 92%, the device was able to be used to collect data to train the network. This prosthesis also uses the FSRs to classify when the stimulation should occur. Having the FSRs adds more dimension to the data sets which help with the accuracy of the model. This analysis also quantified the delay, and this is something that could be improved in the next prosthesis. The analysis of delay is similar to what Vargas-Valencia et al [36] experienced in their research. They stated that there was some shift associated to the data that they collected. They did not quantify the amount of data shifting but they mentioned that one of the possible reasons was the use of the gyroscope and the technology limitations that exist within the IMUs. This cross-correlation analysis also gave insight that was useful later in the developmental process when implementing the EMS device.

Overall, the neural networks were found to be effective. The most accurate models were the first model that used data from all the sensors and the one that used only the FSR data. The least accurate was the second model which only took data from the IMU sensors. The first and third models both follow the assigned classifications, but the second model had a lot of misclassifications. The reason for the many misclassifications is because of the data features that are associated with the IMUs. Unlike the FSR data, the IMUs follow the curve of different leg sections which could have the same value multiple times throughout one gait cycle. When neural network read the data if it recognized it as a different part of the gait cycle it could lead to classify that data point incorrectly. The FSR data has a recognizable pattern because as the gait cycle proceeds the person's weight moves from heel to toe. Based on the weight differential between each FSR the neural network has an easier time recognizing the pattern when stimulation is needed. It should be noted that the reason the "FSR" ANN had less accuracy than the "All Sensor" ANN was because of the tightness of the classification spread. The "FSR" ANN had a wider spread, which means that the misclassifications occurred around the values where the actual classifications changed, during this shift in classes the "All Sensor" ANN had a tighter spread, this meant that as the data got closer to the switch in classifications the ANN only misclassified one data point. This suggested that both type of sensors should be kept for accurate readings. The analysis of the three ANN models also showed that there was a need to improve the data collection abilities of the IMUs. The noise that was created around the data did not allow the NN to perform well when only using IMU data, this results in the need to rely on the FSR data as well to have a functional and accurate classifier. A way to improve this could be by obtaining a newer model of IMU or adding filters to reduce the delay and the noise when collecting data.

The third-generation prosthesis was built using all the pins of the RPi3, even though some pins were not used they were occupied by a block of connectors that connected to the PCB board. In order to control the electrical stimulator one of the ground pin connections and an output pin were needed. Since there were no other pins available, we considered removing the pelvis IMU connections to free up some pins on the RPi3. The pelvis IMU was chosen because of observations that were made from the ANN model classifications and the cross-correlation results. Looking at the classification results for the first three models of the ANN, it was observed that the IMUs provided the least accurate results. These inaccuracies could have occurred due to the lag associated with the sensors or because the angle values were similar throughout certain points which made the ANN misclassify data points. Compared to the FSR data curves, the IMU curves show a greater variance. The IMU plots are not as compacted together as the FSR plots, this makes the values not as clear to the neural network which causes misclassifications. After other observations, mainly from the cross-correlation plots, it was decided that the pelvis IMU should be removed. The pelvis IMU did not provide much information since it was relatively still during gait. This occurred because it was anchored at a point of balancing for the body. While the pelvis does move during gait, it is mainly rotational about the plane of motion that we are measuring unlike the foot, shank, and thigh which rotate in plane.

The final model includes only seven sensors, four FSRs and three IMUs. The classification results provided by this last model were fairly accurate. The accuracy was recorded to be 98.5% which was affected by the fact that the network was trained more biased towards the FSR sensors than the IMUs. As was seen in the results, the neural network that used just the FSR sensors had a similar accuracy value, and the classifications curve showed that there were no random misclassifications of the data. The issue with using only FSR sensors was that it created a wider misclassification zone for the arrays that were close to each other in value. Using the

IMUs helped close this zone but there could be one or two data points that were still misclassified which caused the accuracy to be lower than when all eight sensors were used. The misclassified data points also did not affect the final model much because it was calculated that the time lost was on average about 5 ms.

The final test results proved the accuracy of the final ANN model. Gait research shows that the there are two phases in gait the stance phase and the swing phase, and that the stance phase accounts for 60% of the gait cycle [8]. Toe-off occurs near that 60% value since that is when the leg pushes off and starts to swing. While testing the final model it was expected that the stimulation (class 1) occur at around 60%. The results showed that the stimulations for all the gait cycles happened near that value. Even though the square waves show some spread, the average line shows that the stimulation occurs anywhere from 50% to 65% of the cycle. The reason for the 15% gap, is that the neural network was trained to predict the stimulation period earlier to account for the ramp up time of the EMS device and delay that was associated with the model, this allowed the user to receive the contraction during toe-off as opposed to after. The number is also not exact because the gait literature refers to an average gait cycle percentage which could vary for everyone. In this case the reason for the extra five percent after the desired 60% is because in this case the participant had a slower time completing the motion of toe-off.

CHAPTER VI: CONCLUSION

In this research study the neural prosthesis was upgraded from manual stimulation control to automatic stimulation control using sensor feedback. An artificial neural network was used to identify the proper time for muscle stimulation using input from inertial measurement units and force sensitive resistors. When stimulation was required, the microcontroller closed a solid-state relay that completed the circuit between the muscle and the continuously “ON” electrical stimulator which induced muscular contractions.

This research showed that it was possible to obtain an accurately fitted neural network based on sensor readings. The network also showed a high accuracy percentage when testing other values that were not part of the training data sets. Following implementation, it was observed that the neural network correctly classified when muscle contraction was required and was able to send the stimulation signal.

For future work, it is recommended that further research and experimentation be done with other models of inertial measurement units. The current model of IMUs does not include any low or high pass filters. A complimentary filter was used to determine the percent reliance on data provided by each sensor type in the IMU. A complementary filter setting of 2% reliance on the accelerometer and a 98% reliance on the gyroscope was selected based on empirical methods. Although the gyroscope data was more reliable than the accelerometer data, it had an associated time lag. It may be possible to reduce the lag in the IMU data using different IMUs. Furthermore, identifying a new IMU with built in filters that better manages the accelerometer noise could improve the quality of data from these sensors. Thus, reliance on the accelerometer data could be increased by shifting the complementary filter closer to 50% reliance on each sensor type. Since the accelerometer data does not have an appreciable time delay, this advancement could improve the overall performance of the system.

Another suggestion is upgrading the force sensitive resistor sensors to enable higher force measurements. The current FSRs have a maximum force detection of 150 Newtons which translates to roughly 34 pounds force. Thus, the FSR behaves more like a switch than a variable value sensor. Having a higher maximum value would allow the actual force value to be measured during the gait cycle which could improve the accuracy of the neural network.

The last suggestion is to increase the ruggedness of the prosthesis. The current device has a few physical points that are fragile. An example of this is the wires that connect to the PCB board. During testing or handling of the device the wires became undone, and they had to be reconnected to the pin connectors on the PCB board. Another issue is that the EMS modifications are not directly a part of the PCB board but instead connected to it. Further work can make the device more compact and implement all the sensors and EMS modifications together to be connected to the microcontroller. This could help in making the next model better suited to be handled during experimentation and overall use.

REFERENCES

[1] World Health Organization & World Bank, “World Report on Disability.” .

[2] B. S. Armour, E. A. Courtney-Long, M. H. Fox, H. Fredine, and A. Cahill, “Prevalence and Causes of Paralysis—United States, 2013,” *Am. J. Public Health*, vol. 106, no. 10, pp. 1855–1857, Oct. 2016, doi: 10.2105/AJPH.2016.303270.

[3] L. Inverarity, “How Electricity Is Used in Physical TherapTitle,” 2019. https://www.verywellhealth.com/electrical-stimulation-2696122.

[4] Barrett, “Foot Drop,” 2018. https://www.mstrust.org.uk/a-z/foot-drop.

[5] J. Rueterbories, E. G. Spaich, and O. K. Andersen, “Gait event detection for use in FES rehabilitation by radial and tangential foot accelerations,” *Med. Eng. Phys.*, vol. 36, no. 4, pp. 502–508, 2014, doi: 10.1016/j.medengphy.2013.10.004.

[6] C. Barrett, G. Mann, P. Taylor, and P. Strike, “A randomized trial to investigate the effects of functional electrical stimulation and therapeutic exercise on walking performance for people with multiple sclerosis,” *Mult. Scler. J.*, vol. 15, no. 4, pp. 493–504, Apr. 2009, doi: 10.1177/1352458508101320.

[7] S. Il Choi, J. Moon, H. C. Park, and S. T. Choi, “User identification from gait analysis using multi-modal sensors in smart insole,” *Sensors (Switzerland)*, vol. 19, no. 17, pp. 1–14, 2019, doi: 10.3390/s19173785.

[8] A. Kharb, V. Saini, Y. Jain, and S. Dhiman, “A review of gait cycle and its parameters,” *IJCEM Int J Comput Eng Manag*, vol. 13, no. July, pp. 78–83, 2011.

[9] Tekscan, “The Gait Cycle: Phases, Parameters to Evaluate & Technology.” https://www.tekscan.com/blog/medical/gait-cycle-phases-parameters-evaluate-technology.

[10] T. T. Ngo, Y. Makihara, H. Nagahara, Y. Mukaigawa, and Y. Yagi, “Similar gait action recognition using an inertial sensor,” *Pattern Recognit.*, vol. 48, no. 4, pp. 1289–1301, 2015, doi: 10.1016/j.patcog.2014.10.012.

[11] S.-Y. Kim and G.-I. Kwon, “Gravity Removal and Vector Rotation Algorithm for Step counting using a 3-axis MEMS accelerometer,” *J. Korea Soc. Comput. Inf.*, vol. 19, no. 5, pp. 43–52, May 2014, doi: 10.9708/jksci.2014.19.5.043.

[12] Y. Qi, C. B. Soh, E. Gunawan, K.-S. Low, and R. Thomas, “Assessment of Foot Trajectory for Human Gait Phase Detection Using Wireless Ultrasonic Sensor Network,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 24, no. 1, pp. 88–97, Jan. 2016, doi: 10.1109/TNSRE.2015.2409123.

[13] Taesoo Kim and Sungho Jo, “Simulation of human locomotion using a musculoskeletal model,” in *2008 International Conference on Control, Automation and Systems*, Oct. 2008, pp. 1761–1764, doi: 10.1109/ICCAS.2008.4694514.

[14] J. Johnson, “Everything you need to know about plantar flexion,” 2017. https://www.medicalnewstoday.com/articles/318249.

[15] N. Hamilton and K. Luttgens, *Kinesiology : Scientific Basis of Human Motion*, 10th ed. McGraw-Hill Publishing Company, 2002.

[16] "Functional electrical stimulation (FES)," *Multiple Sclerosis Trust*, 2020. https://mstrust.org.uk/a-z/functional-electrical-stimulation-fes.

[17] J. Eraifej, W. Clark, B. France, S. Desando, and D. Moore, "Effectiveness of upper limb functional electrical stimulation after stroke for the improvement of activities of daily living and motor function: a systematic review and meta-analysis," *Syst. Rev.*, vol. 6, no. 1, p. 40, Dec. 2017, doi: 10.1186/s13643-017-0435-5.

[18] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural network design*, Print. Boston: PWS Pub, 1996.

[19] A. Santoro *et al.*, "A simple neural network module for relational reasoning," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, pp. 4968–4977, 2017.

[20] "Concolutional Neural Networks for Visial Recogniziton," *GitHub*, 2020. https://cs231n.github.io/neural-networks-1/.

[21] upGrad, "Neural Network: Architecture, Components & Top Algorithms," 2020. https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/#:~:text=Usually%2C a Neural Network consists,hence%2C they are all connected.

[22] J. Heaton, "The Number of Hidden Layers," *Heaton Research*, 2017. https://www.heatonresearch.com/2017/06/01/hidden-layers.html.

[23] R. Keim, "How Many Hidden Layers and Hidden Nodes Does a Neural Network Need?," *All About Circuits*, 2020. https://www.allaboutcircuits.com/technical-articles/how-many-hidden-layers-and-hidden-nodes-does-a-neural-network-need/.

[24] B. Szabtowski, "What is an artificial neuron and why does it need an activation function?," *Towards Data Science*, 2020. https://towardsdatascience.com/what-is-an-artificial-neuron-and-why-does-it-need-an-activation-function-5b4c1e971d80.

[25] R. M. S. de Oliveira *et al.*, "A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges," *J. Microwaves, Optoelectron. Electromagn. Appl.*, vol. 16, no. 3, pp. 628–645, Sep. 2017, doi: 10.1590/2179-10742017v16i3854.

[26] S. Sharma, "Activation Functions in Neural Networks," *Towards Data Science*, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[27] A. Al-Masri, "How Does Back-Propagation in Artificial Neural Networks Work?," *Towards Data Science*, 2019. https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7.

[28] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," Jan. 2012, [Online]. Available: http://arxiv.org/abs/1201.0490.

[29] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-

learn project," pp. 1–15, 2013, [Online]. Available: http://arxiv.org/abs/1309.0238.

[30] M. F. Sanner, "Python: a programming language for software integration and development.," *J. Mol. Graph. Model.*, vol. 17, no. 1, pp. 57–61, Feb. 1999, [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/10660911.

[31] K. R. Srinath, "Python – The Fastest Growing Programming Language," *Int. Res. J. Eng. Technol.*, pp. 354–357, 2017, [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/55458585/IRJET-V4I1266.pdf?1515226715=&response-content-disposition=inline%3B+filename%3DPython_The_Fastest_Growing_Programming_L.pdf&Expires=1593202307&Signature=HBD7oa85wDxqRzTWX01uVRBlMacGX5mkGk1b~SVVTTkENJ6cf5diKz.

[32] D. Amos, "Object-Oriented Programming (OOP) in Python 3," *Real Python*, 2020. https://realpython.com/python3-object-oriented-programming/.

[33] K. Parsa, T. A. Lasky, and B. Ravani, "Design and Implementation of a Mechatronic, All-Accelerometer Inertial Measurement Unit," *IEEE/ASME Trans. Mechatronics*, vol. 12, no. 6, pp. 640–650, Dec. 2007, doi: 10.1109/TMECH.2007.910080.

[34] P. Picerno, A. Cereatti, and A. Cappozzo, "Joint kinematics estimate using wearable inertial and magnetic sensing modules.," *Gait Posture*, vol. 28, no. 4, pp. 588–95, Nov. 2008, doi: 10.1016/j.gaitpost.2008.04.003.

[35] E. D. Ledoux, "Inertial Sensing for Gait Event Detection and Transfemoral Prosthesis Control Strategy," *IEEE Trans. Biomed. Eng.*, vol. 65, no. 12, pp. 2704–2712, 2018, doi: 10.1109/TBME.2018.2813999.

[36] L. S. Vargas-Valencia, A. Elias, E. Rocon, T. Bastos-Filho, and A. Frizera, "An IMU-to-body alignment method applied to human gait analysis," *Sensors (Switzerland)*, vol. 16, no. 12, pp. 1–18, 2016, doi: 10.3390/s16122090.

[37] A. G. Cutti, A. Ferrari, P. Garofalo, M. Raggi, A. Cappello, and A. Ferrari, "'Outwalk': a protocol for clinical gait analysis based on inertial and magnetic sensors.," *Med. Biol. Eng. Comput.*, vol. 48, no. 1, pp. 17–25, Jan. 2010, doi: 10.1007/s11517-009-0545-x.

[38] T. A. L. Wren, K. P. Do, S. A. Rethlefsen, and B. Healy, "Cross-correlation as a method for comparing dynamic electromyography signals during gait.," *J. Biomech.*, vol. 39, no. 14, pp. 2714–8, 2006, doi: 10.1016/j.jbiomech.2005.09.006.

[39] Wikipedia Contributors, "Cross-correlation," *Wikipedia: The Free Encyclopedia*, 2020. https://en.wikipedia.org/w/index.php?title=Cross-correlation&oldid=936791961.

[40] P. Subbukutti, "A Next Generation Neural Prosthesis to Improve Gait in People with Muscle Weakness," 2020.

[41] "Qualisys Miques M3." Qualisys Americas, Chicago, Illinois.

[42] "MATLAB." The MathWorks Inc., Natick, Massachusetts, 2018.

APPENDICES

Appendix A

```
clc, clear all

%% Rading Data %%
gait=xlsread('camera_gait.xlsx','Sheet1');
input_file=xlsread('average_curves.xlsx');

% Col-add is used to change the number of column that is used in the
data
col_add=1;

%% IMU data Input %%
select_imu=~isnan(input_file(:,1+col_add));
b=input_file(select_imu,1+col_add);
c=[0:100/(length(b)-1):100]';
e=[];

%% Camera Data Input %%
select_cam=~isnan(input_file(:,6+col_add));
b_1=input_file(select_cam,6+col_add);
c_1=[0:100/(length(b_1)-1):100]';
e_1=[];

%% Interpolation Calculations %%
% This allows us to use the cross correaltion function created by
MATLAB or
% the one designed since both of the vectors will be the same size

for j=1:101
    e(j) = interp1(c,b,j,'spline');
end
for j=1:101
    e_1(j) = interp1(c_1,b_1,j,'spline');
end

%% First Lag Calculation %%

[cc,lags]=xcorr(e_1,e);
[max_cc, index] = max(cc);
lag_max = lags(index);

if mod(col_add,2)==0
    time_delay=((gait(1,3)-gait(1,2))/100)*abs(lag_max);
elseif mod(col_add,2)==1
    time_delay=((gait(1,4)-gait(1,3))/100)*abs(lag_max);
end

if lag_max>0
```

```
    fprintf('The lag delay is estimated to be: %d percent to the right
\n',abs(lag_max))
elseif lag_max<0
    fprintf('The lag delay is estimated to be: %d percent to the left
\n',abs(lag_max))
end
fprintf('The expected time delay is: %f seconds.\n',time_delay)

%% Shift Calculation and Update Lag %%
shift=lag_max;

s=circshift(e,shift);
[cc_update,lags_update]=xcorr(e_1,s);
[max_cc_update, index_update] = max(cc_update);
lag_max_update= lags_update(index_update);

%% Plots %%
x_var=0:100;

figure (1)
plot(x_var,e_1,'g','linewidth',1)
hold on
plot(x_var,e,'b--','linewidth',1)
plot(x_var,s,'m:','linewidth',1.5)
grid on
legend('Camera','IMU','Shifted IMU')
title('Hip Shift Comparison')
xlabel('Gait Percentage (%)')
ylabel('Angle Value (degrees)')
xlim([0 100])

figure (2)
plot(lags,cc,'k','linewidth',1)
hold on
plot(lags_update,cc_update,'r--','linewidth',1)
plot(lag_max_update,max_cc_update,'bo')
plot(lag_max,max_cc,'bo')
grid on
xlabel('Gait Percentage Shift (%)')
ylabel('Cross Correlation Value')
legend('Original Correlation','Shifted Correlation','Max Correlation
Point')
title('Cross Correlation of Camera and IMU Data')
```

Appendix B

```
# Final neural network code
# Inputs = 7, hidden = 13, outer = 1

# This imports all the required programs that will help with using the
NN
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
import numpy as np
import matplotlib.pyplot as plt

# This imports all the files necessary as the data for the NN
trainData_df = pd.read_csv("Final_train.csv")
testData_df = pd.read_csv("Final_test.csv")
X_train = trainData_df.drop(columns=["Class_Train"])#train data inputs
Y_train = trainData_df["Class_Train"] #classification for train data
X_test = testData_df.drop(columns=["Class_Test"])#inputs for test data
Y_test = testData_df["Class_Test"] #classification for test data

# MLP Classifier Usage
my_NN = MLPClassifier(hidden_layer_sizes = 13, activation='logistic',
                      batch_size='auto', max_iter=10000,verbose=True,
                      random_state=5)

# Fitting the classifier
NN_fit = my_NN.fit(X_train, Y_train)

# Information for accuracy
NN_accuracy_train = round(my_NN.score(X_train,Y_train),4)*100
print('Accuracy for training is :', NN_accuracy_train,'%')
predictions = my_NN.predict(X_test)
NN_acccuracy_test = round(my_NN.score(X_test, Y_test),4)*100
conf_mat = confusion_matrix(Y_test, predictions)
print('The confussion matrix is:')
print(conf_mat)
print('The accuracy of the test data is:')
print(NN_acccuracy_test,'%')

# Plotting loss curve
loss_values=my_NN.loss_curve_
plt.plot(loss_values)
plt.grid()
plt.title('Loss Function')
plt.xlabel('Iterations')
plt.ylabel('Loss Value')
plt.show()
```

```
# Saving weights and biases of the network
weights = my_NN.coefs_
biases = my_NN.intercepts_
np.save('weight_1', weights[0])
np.save('weight_2', weights[1])
np.save('bias_1', biases[0])
np.save('bias_2', biases[1])
```

Appendix C

```
import smbus
import math
import time
import RPi.GPIO as GPIO
import csv
import numpy as np

#SPI
import Adafruit_GPIO.SPI as SPI
import Adafruit_MCP3008

with open('pitch.ods','a') as f:
    writer=csv.writer(f)
    writer.writerow(['5th meta','2nd
meta','Toe','Heel','Thigh(98)','Shank(98)','Foot(98)','Class'])
a=[]

#Configuration of SPI ports
SPI_PORT   = 0
SPI_DEVICE = 0
mcp = Adafruit_MCP3008.MCP3008(spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE))


class MPU:
    def __init__(self, gyro, acc, tau):
        # Class / object / constructor setup
        self.gx = None; self.gy = None; self.gz = None;
        self.ax = None; self.ay = None; self.az = None;

        self.gyroXcal = 0
        self.gyroYcal = 0
        self.gyroZcal = 0

        self.gyroRoll = 0
        self.gyroPitch = 0
        self.gyroYaw = 0

        self.roll = 0
        self.pitch = 0
        self.yaw = 0

        self.dtTimer = 0
        self.tau = tau

        self.gyroScaleFactor, self.gyroHex =
self.gyroSensitivity(gyro)
        self.accScaleFactor, self.accHex =
self.accelerometerSensitivity(acc)
```

```
        self.bus = smbus.SMBus(1)
        self.address = 0x68

    def gyroSensitivity(self, x):
        # Create dictionary with standard value of 500 deg/s
        return {
            250:  [131.0, 0x00],
            500:  [65.5,  0x08],
            1000: [32.8,  0x10],
            2000: [16.4,  0x18]
        }.get(x,  [65.5,  0x08])

    def accelerometerSensitivity(self, x):
        # Create dictionary with standard value of 4 g
        return {
            2:  [16384.0, 0x00],
            4:  [8192.0,  0x08],
            8:  [4096.0,  0x10],
            16: [2048.0,  0x18]
        }.get(x,[8192.0,  0x08])

    def setUp(self):
        # Activate the MPU-6050
        self.bus.write_byte_data(0x68, 0x6B, 0x00)

        # Configure the accelerometer
        self.bus.write_byte_data(self.address, 0x1C, self.accHex)

        # Configure the gyro
        self.bus.write_byte_data(self.address, 0x1B, self.gyroHex)

        # Display message to user
        print("MPU set up:")
        print('\tAccelerometer: ' + str(self.accHex) + ' ' +
str(self.accScaleFactor))
        print('\tGyro: ' + str(self.gyroHex) + ' ' +
str(self.gyroScaleFactor) + "\n")
        #time.sleep(2)

    def eightBit2sixteenBit(self, reg):
        # Reads high and low 8 bit values and shifts them into 16 bit
        h = self.bus.read_byte_data(self.address, reg)
        l = self.bus.read_byte_data(self.address, reg+1)
        val = (h << 8) + l

        # Make 16 bit unsigned value to signed value (0 to 65535) to
(-32768 to +32767)
        if (val >= 0x8000):
            return -((65535 - val) + 1)
        else:
            return val
    def getRawData(self):
        self.gx = self.eightBit2sixteenBit(0x43)
        self.gy = self.eightBit2sixteenBit(0x45)
        self.gz = self.eightBit2sixteenBit(0x47)

        self.ax = self.eightBit2sixteenBit(0x3B)
        self.ay = self.eightBit2sixteenBit(0x3D)
        self.az = self.eightBit2sixteenBit(0x3F)

    def calibrateGyro(self, N):
        # Display message
        print("Calibrating gyro with " + str(N) + " points. Do not 
move!")
        self.dtTimer = time.time()

    def processIMUvalues(self):
        # Update the raw data
        self.getRawData()

        # Convert to instantaneous degrees per second
        self.gx /= self.gyroScaleFactor
        self.gy /= self.gyroScaleFactor
        self.gz /= self.gyroScaleFactor

        # Convert to g force
        self.ax /= self.accScaleFactor
        self.ay /= self.accScaleFactor
        self.az /= self.accScaleFactor

    def compFilter(self):
        # Get the processed values from IMU
        self.processIMUvalues()

        # Get delta time and record time for next call
        dt = time.time() - self.dtTimer
        self.dtTimer = time.time()

        # Acceleration vector angle
        accPitch = math.degrees(math.atan2(self.ay, self.az))
        accRoll = math.degrees(math.atan2(self.ax, self.az))

        # Gyro integration angle
        self.gyroRoll -= self.gy * dt
        self.gyroPitch += self.gx * dt
        self.gyroYaw += self.gz * dt
        self.yaw = self.gyroYaw

        # Comp filter
        self.roll = (self.tau)*(self.roll - self.gy*dt) + (1-
self.tau)*(accRoll)
        self.pitch = (self.tau)*(self.pitch + self.gx*dt) + (1-
self.tau)*(accPitch)
```

```
        self.pitch1 = (1-self.tau)*(self.pitch + self.gx*dt) +
(self.tau)*(accPitch)
        self.pitch2 = (0.5)*(self.pitch + self.gx*dt) + (1-
0.5)*(accPitch)
        self.pitch3= (0.6)*(self.pitch + self.gx*dt) + (1-
0.6)*(accPitch)
        self.pitch4= (0.4)*(self.pitch + self.gx*dt) + (1-
0.4)*(accPitch)

        # Print data
        print(" R: " + str(round(self.roll,1)) \
            + " P: " + str(round(self.pitch,1)) \
            + " Y: " + str(round(self.yaw,1)))
        a.append((round(self.pitch,1)))
        #a.append(str(round(self.pitch1,1)))
        #a.append(str(round(self.pitch2,1)))
        #a.append(str(round(self.pitch3,1)))
        #a.append(str(round(self.pitch4,1)))

def sigmoid(x):
    return  1 / (1 + np.exp(-x))

def neural_net(nn):
    w_1 = np.load('weight_1.npy')
    w_2 = np.load('weight_2.npy')
    b_1 = np.load('bias_1.npy')
    b_2 = np.load('bias_2.npy')

    z_1 = np.dot(nn, w_1) + b_1
    h_inputs = sigmoid(z_1)
    z_2 = np.dot(h_inputs, w_2) + b_2
    out_class = sigmoid(z_2)
    if out_class >= .500:
        out_class = 1
        GPIO.output(35,True)
        print ('Stimulate')
    elif out_class < .500:
        out_class = 0
        GPIO.output(35,False)
        print ('Device is off')
    a.append(out_class)


GPIO.setmode(GPIO.BOARD)
GPIO.setup(29,GPIO.OUT)
GPIO.setup(31,GPIO.OUT)
GPIO.setup(33,GPIO.OUT)
GPIO.setup(35,GPIO.OUT)

GPIO.output(29,GPIO.HIGH)
GPIO.output(31,GPIO.HIGH)
GPIO.output(33,GPIO.HIGH)
```

```
#GPIO.output(35,GPIO.HIGH)
#GPIO.output(29,GPIO.LOW)'''




GPIO.output(29,GPIO.LOW)
# Set up class
gyro = 250      # 250, 500, 1000, 2000 [deg/s]
acc = 2         # 2, 4, 7, 16 [g]
tau = 0.98
mpu = MPU(gyro, acc, tau)

# Set up sensor and calibrate gyro with N points
mpu.setUp()
mpu.calibrateGyro(500)
GPIO.output(29,GPIO.HIGH)



GPIO.output(31,GPIO.LOW)
# Set up class
gyro = 250      # 250, 500, 1000, 2000 [deg/s]
acc = 2         # 2, 4, 7, 16 [g]
tau = 0.98
mpu1 = MPU(gyro, acc, tau)

# Set up sensor and calibrate gyro with N points
mpu1.setUp()
mpu1.calibrateGyro(500)
GPIO.output(31,GPIO.HIGH)

GPIO.output(33,GPIO.LOW)
# Set up class
gyro = 250      # 250, 500, 1000, 2000 [deg/s]
acc = 2         # 2, 4, 7, 16 [g]
tau = 0.98
mpu2 = MPU(gyro, acc, tau)

# Set up sensor and calibrate gyro with N points
mpu2.setUp()
mpu2.calibrateGyro(500)
GPIO.output(33,GPIO.HIGH)

while True:

    try:
        #mpu.compFilter()
        values = [0]*8
        for i in range(8):
            # The read_adc function will get the value of the
specified channel (0-7).
            values[i] = mcp.read_adc(i)
            values[i]=(values[i]/(1024*3.3))*500
            # Print the ADC values.
        print('|{0:>4}|{1:>4}|{2:>4}|{3:>4}|{4:>4}|{5:>4}|{6:>4}|'
.format(*values))
        a.append(values[0])
        a.append(values[1])
        a.append(values[2])
        a.append(values[3])

        GPIO.output(29,GPIO.LOW)
        print('s1')
        #mpu = MPU(gyro, acc, tau)
        #mpu.setUp()
        mpu.compFilter()
        GPIO.output(29,GPIO.HIGH)

        GPIO.output(31,GPIO.LOW)
        #mpu = MPU(gyro, acc, tau)
        #mpu1.setUp()
        print('s2')
        mpu1.compFilter()
        GPIO.output(31,GPIO.HIGH)

        GPIO.output(33,GPIO.LOW)
        #mpu = MPU(gyro, acc, tau)
        #mpu2.setUp()
        print('s3')
        mpu2.compFilter()
        GPIO.output(33,GPIO.HIGH)

        neural_input = np.array(list(a[0:7]), dtype=np.float32)
        neural_net(neural_input)

        with open('pitch.ods','a') as f:
            writer=csv.writer(f)
            writer.writerow(a)
            a=[]
    except (ZeroDivisionError,IOError) as e:
        print("program faced an interruption")
```