A NEXT GENERATION NEURAL PROSTHESIS TO IMPROVE GAIT IN PEOPLE WITH MUSCLE WEAKNESS

A thesis presented 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
Premkumar Subbukutti

Advisor: Dr. Martin L. Tanaka
School of Engineering and Technology

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

April 2020

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A NEXT GENERATION NEURAL PROSTHESIS TO IMPROVE GAIT IN PEOPLE WITH MUSCLE WEAKNESS

Premkumar Subbukutti, M.S.T.

Western Carolina University (April 2020)

Director: Dr. Martin L. Tanaka

Some of the 5.3 million people in the US who are living with some form of paralysis may be assisted by a neural prosthesis that employs Functional Electrical Stimulation (FES). FES produces muscular contractions by applying an electrical stimulation to nerves that supply a muscle. The specific goal of this research was to develop a neural prosthesis capable of accurately detecting human gait characteristics to determine proper timing for artificial muscle stimulation.

This third-generation neural prosthesis uses four force sensitive resistors, four inertial measurement units (IMUs), a Raspberry Pi microcontroller, and has improve data collection and storage software, real time data filtering and add wireless communication. Tests on a healthy individual were performed to evaluate the device's ability to measure and record gait data. Collected data was compared to the data collected from the camera motion capture system to determine the device's accuracy.

Testing showed that the neural prosthesis was able to capture the general shape of the joint angle curves when compared to the camera motion capture system. However, the joint angles obtained from the neural prosthesis device lagged the actual joint angles found using the camera system. This is likely due to a slow response time in the gyroscope. In the future, measures will

be taken to reduce lag in the gyroscope and reduce jitter in the accelerometer so that data from both sensors can be combination to obtain more accurate readings.

CHAPTER1: INTRODUCTION

Neurological diseases in America is estimated to impact about 100 million people every year [2]. It is predicted that by 2030 dementia and stroke alone will cost $600 billion annually [2]. Among the most common neurological diseases, paralysis is dramatically more widespread than previously thought. The number of people reported to be living with some form of paralysis [6] has reached approximately 1.7 percent of the U.S. population, about 5,357,970 people. Paralysis may be defined as a central nervous system disorder resulting in difficulty or inability to move the upper or lower extremities. The leading cause of paralysis is stroke (33.7 percent), followed by spinal cord injury (27.3 percent) and multiple sclerosis (18.6 percent) [7]. More than 50 million people are getting treatment every year, which is estimated to be $306 billion annually, twice the $158 billion spent on home care and nursing home services combined. Considering the cost of paralysis, developing effective solution can be beneficial to society. So, the research team is developing a device to augment the body's natural function of muscle contraction with the use of artificial electrical stimulation.

Our bodies naturally use electrical signals as part of the nervous system. When we move, the brain generates and sends electrical impulses along the spinal cord and nerves to initiate the muscles contractions. Functional Electrical Stimulation (FES) is a technique used to produce contractions in paralyzed muscles by the application of small pulses of electrical stimulation to nerves that supply the paralyzed muscle. The stimulation is controlled in such a way that the movement produced provides useful function. FES is usually applied through electrodes that are placed on the surface of the skin [26], although electrodes can also be implanted into the muscles [27]. Electrodes are placed over nerves or part of muscle that needs artificial stimulation to work. The electrodes are then attached to a device that generates the stimulation. The electrical

stimulation level is then gradually turned up until the muscles begin to tense or contract. An intact peripheral nerve and healthy muscle tissue is required to enable the external source of electricity to facilitate the muscle contraction.

The overall goal of this line of research is to develop a neural prosthesis using FES technology to improve gait in people with muscle weakness. There are several muscles that contribute to gait, but the research team chose to focus initially on the muscles associated with plantarflexion. Thus, the specific goal of this research was to develop a neural prosthesis capable of accurately detecting human gait characteristics in order to determine proper timing for muscle stimulation.

CHAPTER 2: LITERATURE REVIEW

2.1 Human Gait

Human gait refers to locomotion achieved through the movement of human limbs. Different gait patterns are characterized by differences in limb movement patterns. The movement patterns include, differences in overall velocity, forces, and kinetic and potential energy cycles. Human gait describes the various ways in which a human move, either naturally or as a result of specialized training. There are different gaits in human locomotion, such as walking, running and hoping [8]. This project will focus only on walking because it is the most frequently used gait. In general, a gait analysis method consists of data acquisition, modelling and assessment. The raw gait data are used to calculate features in a specific gait model. Various sensors are used to measure the parameters associated with the person's gait [7]. Human gait data measured for gait analysis mainly include lower limb kinematics usually collected with motion capture camera systems [8] [10] and ground reaction forces (GRFs) measured with force places implanted into the floor, however, full body motion capture is also used in gait analysis[8]. Human gait can also be detected with wearable sensors. Inertia Measurement Units (IMUs), containing accelerometers, gyroscopes, and magnetometers, are the most widely used wearable sensors in clinical studies.

2.2 Gait Cycle

The gait cycle consists of stance phase and swing phase from heel strike to heel strike on the same foot. Normally 60% of one gait cycle is spend in stance and 40% spend in swing. During the gait cycle, when the both feet are in contact with the ground, it is considered as a stance phase. When one foot is in contact with the ground other is not in contact with the ground, it is considered as a swing phase [19]. The stance phase can be divided in to five main parts [15]: Heel strike, foot flat, mid stance, terminal stance and toe off. The heel strike is the moment when the

heel strikes the ground. After the heel strike, the rest of the foot begins to contact the ground, finishing with the toes. The foot is now flat on the ground and the body weight is shifted to the stance foot. This part of the gait cycle is known as the foot flat. After the weight is shifted, the body balances upon the stance foot while the contralateral limb is swung through. This part of the gait cycle is called the mid stance. When the contralateral limb makes heel strike, the heel of the ipsilateral foot starts rising from the ground, and the foot enters the terminal stance. This is the part of the gait cycle where the heel is in the air and the toe is still in contact with the ground. The rising of the foot continues until the toe off. The toe off is the moment that the toe rises in air which is the end of the stance phase and beginning of the swing phase. The swing phase is divided in to three main parts: initial swing, mid swing, and terminal swing. The initial swing begins with elevation of the limb from the ground and ends with the knee at maximal flexion. During mid swing, the knee is extended to keep the shank generally vertical. During the terminal swing part of the gait cycle, the knee continues to extend, raising the shank out of vertical alignment and ends just prior to initial contact of the heel to the ground.

2.3 Muscles and Joints involved in Human Locomotion

Human movement is achieved by a complex and highly coordinated mechanical interaction between bones, muscles, ligaments and joints within the musculoskeletal system under the control of nervous system. Each leg consists of 3 segments, the thigh, the shank, and the foot, and segments are pivot jointed. During gait, each foot periodically interacts with the ground [10]. Three main joints and their corresponding muscles contribute to the human locomotion, the hip, knee, and ankle. During walking, the ankle provides a great portion of the required energy, so this project will focus on enhancing ankle joint torque [11].

Using a simple model, the ankle joint can be considered as a hinge type joint, with movement permitted in one plane, the sagittal. The sagittal plane is a plane which divides the body in to two parts, the right and left sides. The sagittal plane is also called the longitudinal plane.

Plantarflexion and dorsiflexion are the main movements that occur at the ankle joint. Plantarflexion is a movement which extends the top of your foot points away from the shank, like the movement used to push a seed into the ground with the foot (i.e. planter). Plantarflexion is used whenever a person stands on his or her toes or points the toes. Dorsiflexion is the movement in the opposite direction. With dorsiflexion, the foot moves upwards, so that the top of the foot is closer to the shin. The important muscle associated with ankle movement are described below [12].

Gastrocnemius Muscle: The gastrocnemius muscle is a muscle located on the back portion of the lower leg. It is a major component of the calf muscle. It connects to the femur just above the knee and attaches to the Achilles tendon, connecting it to the heel. Because the muscle spans two joints, contracting the gastrocnemius muscle leads to plantar flexion of the foot at ankle joint and flexing the leg at knee joint. It is involved in running, jumping and other fast movement of the leg.

Soleus Muscle: The soleus is a powerful muscle in the back part of the lower leg. It is another muscle comprising the calf muscle. It attaches to the tibia and fibula just below the knee and to the heel by passing forces through the Achilles tendon. Because this muscle spans only one joint, contracting the soleus muscle produces only plantarflexion. This muscle is involved in standing and walking.

Anterior Tibialis Muscle: The anterior tibialis muscle enables the ankle and foot to turn upward. It starts from upper lateral surface of the tibia and ends to the base of the first metatarsal bone in the foot. The tibialis anterior is needed for dorsiflexion.

2.4 Natural and Artificial Muscle Stimulation

Natural muscle contraction in human body occurs when the nervous system generates an electrical signal called an action potential [29]. This action potential travels through a type of nerve cell called motor neuron. The location where the motor neuron interacts with a muscle cell is called as neuromuscular junction. When the nervous system signal reaches the neuromuscular junction, chemicals are released at the synaptic junction. This causes chemicals to be released in the muscle fibers causing the microscopic filaments within the muscle to reorganize themselves in the way that shortens the muscle. This shortening of microscopic elements causes overall muscle contraction. When the nervous system stops generating electrical signal, the chemical process reverses, and the muscle fibers rearrange again leading to muscle relaxation [30].

Artificial Electrical Muscle Stimulation (EMS) works by delivering an electrical pulse that activates nerves in the body, causing muscles to contract [5]. Medical applications for this technology include slowing muscle wasting, making muscles stronger and increasing flexibility (range of motion) [28]. It can be used to re-train a muscle and to build strength after a surgery or a period of disuse.

2.5 Second Generation Neural Prosthesis Device

The previous version of the neural prosthesis, the second-generation device, was designed to stimulate the gastrocnemius (GN) muscle during the push off phase using a manual switch [33][34]. This research used artificial EMS to induce muscular contractions during gait (Figure2.1). EMS was used to contract and relax the muscle at the appropriate time. The EMS were connected to the muscle via two electrode pads. Larger electrode pads enable a stronger contraction to take place in the muscle, but there is less accuracy in contracting the target muscle.

The 1.75-inch x 3.75-inch electrode pads were utilized, because the size was large enough to have an enough contraction on the GN muscle.

The slow twitch muscle fibers respond to frequencies around 30 Hz and the fast twitch muscle fiber respond to 80-150 Hz frequencies [13]. EMS 5000 has three pulse frequency range 5HZ, 30 HZ and 100HZ. Pulse frequency can be chose based on the intensity required to contract the muscle. The EMS 5000 was selected for this research because of its low cost and it is approved by the FDA [33]. The features of the EMS 5000 are shown in table 2.1 below.

Table 2.1 Specifications of EMS

Specification	**Value**
Pulse Amplitude	0-80 mA
Pulse Frequency	5, 30, 100 Hz
Contraction Time	1-30 Seconds
Relaxation Time	1-45 Seconds
Power Source	9-Volt Battery

The device was tested on a healthy individual who walked in a straight line. Test were performed with and without the neural prosthesis activated. The results showed that muscle stimulation effectively changed the gait of the person walking.

Figure 2.1 EMS-5000 (photograph by Premkumar Subbukutti)

2.6 Inertial Measurement Unit

Today gait analysis is usually performed with optical systems [10]. These systems can deliver highly accurate data, but these camera systems can only see patients within a limited viewing area. As an alternative, inertial measurement units (IMU) can be used for some human movement detection applications and they have advantages in certain situations. An IMU is an electronic device that measures and reports a body's three-dimensional orientation and angular velocity using a combination of accelerometers and gyroscopes. These sensors are low-cost devices and can be worn all day, without disturbing the patient's motion [14]. A considerable amount of literature has been published on gait analysis using IMUs. IMUs have been used to provide data for pedestrian tracking, to reconstruct walking routes, and to analyze the gait of patients [15]. Usually IMUs have three types of sensors, accelerometers, gyroscopes and magnetometers embedded within them. An IMU that uses a 3-axis accelerometer and a 3-axis

gyroscope is considered to be a 6-Degrees of Freedom (DOF) IMU. If the IMU uses a 3-axis magnetometer along with 3-axis accelerometer and gyroscope is considered to be a 9-DOF IMU.

2.6.1 Accelerometer

An accelerometer is an electromechanical device that measures acceleration forces. These forces may be static, like the constant force of gravity, or they could be dynamic caused by moving or vibrating the accelerometer. Because the acceleration of gravity has a fixed direction and magnitude (9.8 m/s^2), it can be used to determine the orientation of an IMU in 3D space. IMUs output raw tri-axial accelerometer data, a_x, a_y, and a_z. The tilt angles relative to the x and y axis can be calculated using the following equations [20]:

$$\theta_x = \tan^{-1}\left(\frac{a_x}{\sqrt{a_x^2 + a_z^2}}\right) \quad (2\text{-}1)$$

$$\theta_y = \tan^{-1}\left(\frac{a_y}{\sqrt{a_y^2 + a_z^2}}\right) \quad (2\text{-}2)$$

In the above equations, θ_x and θ_y are the tilt angles relative to the x and y axis, respectively. a_x, a_y and a_z are the accelerations in x, y and z directions. The most accelerometers will have a selectable range of forces that they can measure. These ranges can vary from ±1g up to ±250g.

An accelerometer is the most accurate sensor to determine the position when the sensor is not moving because it measures position directly and it responds almost instantly. But when the sensors are accelerating during walking, it can cause errors in positional detection. Accelerometers are also vulnerable to high frequency noise caused by jarring of the accelerometer that occurs during heel strike. These glitches can be reduced somewhat by averaging the acceleration data over time to get a better estimate of the actual position from the accelerometer.

2.6.2 Gyroscope

Gyroscopes are devices that measure angular velocity around a fixed axis with respect to an inertial space. The triaxial gyroscope measures the angular velocity in three directions. Gyroscopes output a voltage proportional to the angular velocity. It is determined by its sensitivity, measured in millivolt's per degree per second (mV/ ° /s). The gyroscope gives the rate of change of the angular position over time (angular velocity) with a unit of [deg./s]. Thus, the angular position can be calculated using equation (2-3) below.

$$\theta(t) = \int_0^t \dot{\theta}(t)dt \approx \sum_0^t \dot{\theta}(t)T_s \qquad (2\text{-}3).$$

In this equation, $\dot{\theta}$ is the measured angular velocity, θ is the estimated angle, T_s is the sampling time and t is time. When gyroscope data changes faster than the sampling frequency, we will not detect it, and the summation approximation will be incorrect. Thus, it is important that we choose a good sampling period.

Because gyroscopes do not detect position directly and sensors are susceptible to errors resulting from angular random walk. This cause drift and it increases over time. A high pass filter can be used to reduce the long-term (low frequency) errors while not affecting the short-term (high frequency) measurements [18]. To reduce the noises in the IMUs we need to design the suitable filter to get the precise value.

2.6.3 Complementary Filter

By combining the information obtained from both the accelerometer and the gyroscope, a more accurate estimation of the angle can be obtained. This sensor fusion algorithm is known as a complementary filter [14], where,

$$\theta_{filtered} = \alpha * \theta_{gyroscope} + (1 - \alpha) * \theta_{accelerometer} \quad (2\text{-}4)$$

In this equation (2-4), $\theta_{gyroscope}$ is the angle measured by the gyroscope, $\theta_{accelerometer}$ is the angle measured by the accelerometer, $\theta_{filtered}$ is the filtered angle and α is a tuning parameter between 0 and 1, showing the contribution of each sensor measurement to the final estimation. The α in the equation (2-4) is called filter coefficient because it determines how much weighting to put on the accelerometer and the gyroscope. It can be seen from the equation (2-4) that if the value of α is high, more weighting is put on the gyroscope value and less weighting on the accelerometer value. Correspondingly, lower values of α indicate a higher weighting is put on the accelerometer data and a lower weighting on the gyroscope.

2.*6.4* Kalman Filter

The Kaman filter was designed to predict the actual value, when the measured value contains random error. A Kalman filter uses an iterative mathematical process that uses past data to quickly estimate the actual value of a parameter. The filter is named after Rudolf E. Kalman one of the primary developers of its theory. The Kalman filter has numerous applications in technology. The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurement and the previously calculated state and its uncertainty matrix. No additional past information is required [22]. When using the Kalman filter, the three important equations to calculate angle from raw sensor data are,

$$KG = \frac{E_{Est}}{E_{Est} + E_{MEA}} \quad (2\text{-}5)$$

$$EST_t = EST_{t-1} + KG[MEA - EST_{t-1}] \quad (2\text{-}6)$$

$$E_{EST_t} = [1 - KG][E_{EST_{t-1}}] \quad (2\text{-}7)$$

KG is the gain, E_{EST} is the error in the estimate, E_{MEA} is the error in the measured value, EST_t is the current estimate, EST_{t-1} is the previous estimate and MEA is the measured value. The figure 2.2 shows a flow chart diagram for the Kalman filter. It shows that the calculation of the current estimation is based on previous estimation and the measured value. The Kalman gain will decide the weight factor for the previous estimation and the measured value. The Kalman gain is calculated from the error in estimation and the error in measurement. If the error in the estimated value is higher than the error in the measured value, the Kalman gain will put more weight on measured value. If the error in measured value is higher than the error in the estimated value, the Kalman gain will put more weight on estimated value.

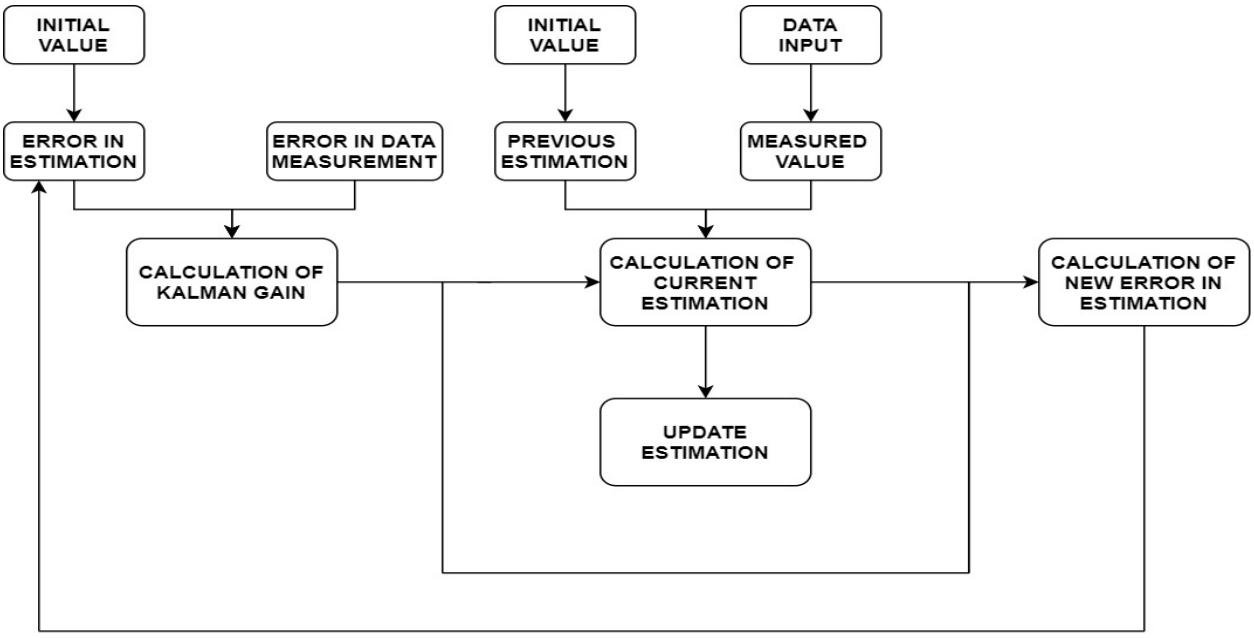

Figure 2.2 Block diagram of Kalman filter

2.7 Raspberry Pi 3 Model B

The Raspberry Pi 3 (RPi3) Model B is the third-generation Raspberry Pi (Raspberry Pi foundation, Cambridge, United Kingdom) [31][35]. This is the small computer can be used for many applications. The wireless LAN uses Bluetooth connectivity. Because of its 10x faster processing when compared to Raspberry Pi 1(RPi1), it was selected for this research. The availability of features such as the general-purpose input output (GPIO) pins make the computer amenable to programming hardware, as well as driving electronic circuitry and collect data through various means [16].

Figure 2.3 Raspberry Pi 3(RPi3) [35].

2.8 Camera Motion Capture System

Gait analysis requires knowledge of parameters such as walking speed, ankle, knee, and hip angles, stride length and width, etc. To obtain this information, a 3D human motion capture system (Figure 2.4) can be used. Marker based systems [19] are widely used for biometrics application. In these systems, several markers are attached to key points of test subjects' body. These key points are captured by the infrared cameras fixed at known positions. The marker positions are

transformed into 3D positions using feedback from several cameras [21]. After obtaining the marker position from the several cameras and combining the position data with a model of human body, one can estimate the joint angles of the test subject. The data can then be used to produce a 3D skeletal structure representing human movement.

Figure 2.4 3-D Camera motion capture system (Photograph by Martin Tanaka).

CHAPTER3: NEURAL PROSTHESIS DEVELOPEMENT

In this thesis, the methods are divided into two sections. The first section describes the design and development of the third-generation neural prosthesis. Like the second-generation device, the third-generation neural prosthesis uses four force sensitive resistors (FSRs) placed in the shoe to measure pressure on the heel, the first metatarsal, the fifth metatarsal, and the toes. It also uses four inertial measurement units (IMUs) [3] to measure the angle of the foot, shank, thigh, and pelvis. From these segment angles, joint angles at the ankle, knee and hip [2] can be calculated. The major changes with the third-generation device were to change the microcontroller from a Teensy microcontroller to a Raspberry Pi microcontroller, improve data collection and storage software, use real time filtering instead of post processing and add wireless communication between the device and the supporting computer.

The second section of the methods (Chapter 4) describes testing performed to evaluate the accuracy of the neural prosthesis. Data collected from the IMUs was compared to data collected using a professional camera system to determine the accuracy of the IMUs.

3.1 Foot Pressure Detection

The first step in estimating gait cycle is to detect the starting point of the gait cycle. The beginning of the gait cycle i.e. 0% is defined by the heel strike. It can be determined by using Force Sensitive Resistor (FSR) placed on heel of the foot. The FSR 402 model was chosen because it is cost efficient, compact and easy to use. The cost of a single FSR is approximately $7.00 US. The specification of FSR 402 are shown below in Table 3.1.

Table 3.1 Specifications of FSRs (Interlink Electronics Inc, California, USA) [30].

Specification	Value
Force Range	0 to 20 lb.
Resistance Change	0 to 200 Ω (maximum pressure)

An FSR is a device that changes its resistance when a force is applied. In other words, it is a sensor that allows you to detect physical pressure, squeezing and weight. FSR's are usually composed of two substrates layers with conductive film and a plastic spacer. When external force is applied to the sensor, the conductive film is deformed against the substrate, air in the spacer opening is pushed through the air vent, and the conductive film comes in to contact with the conductive print on the substrate. The more of the conductive ink area that gets touched by the conductive film, the lower the resistance.

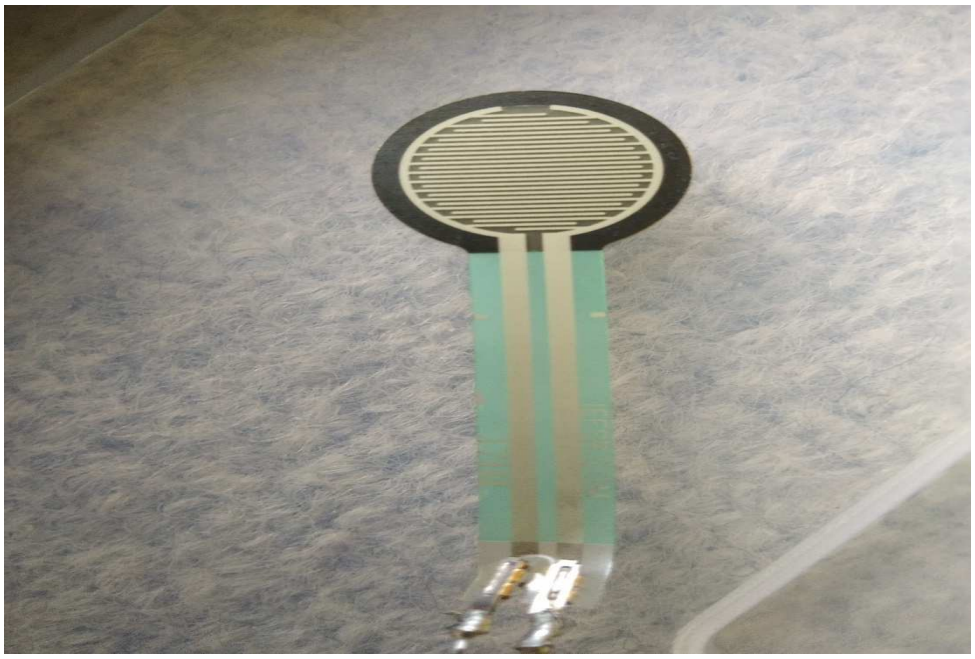

Figure 3.1 FSR (photograph by Premkumar Subbukutti).

To make the experiment convenient for the participants, all FSRs were attached to a shoe insole as shown below (Figure 3.2).

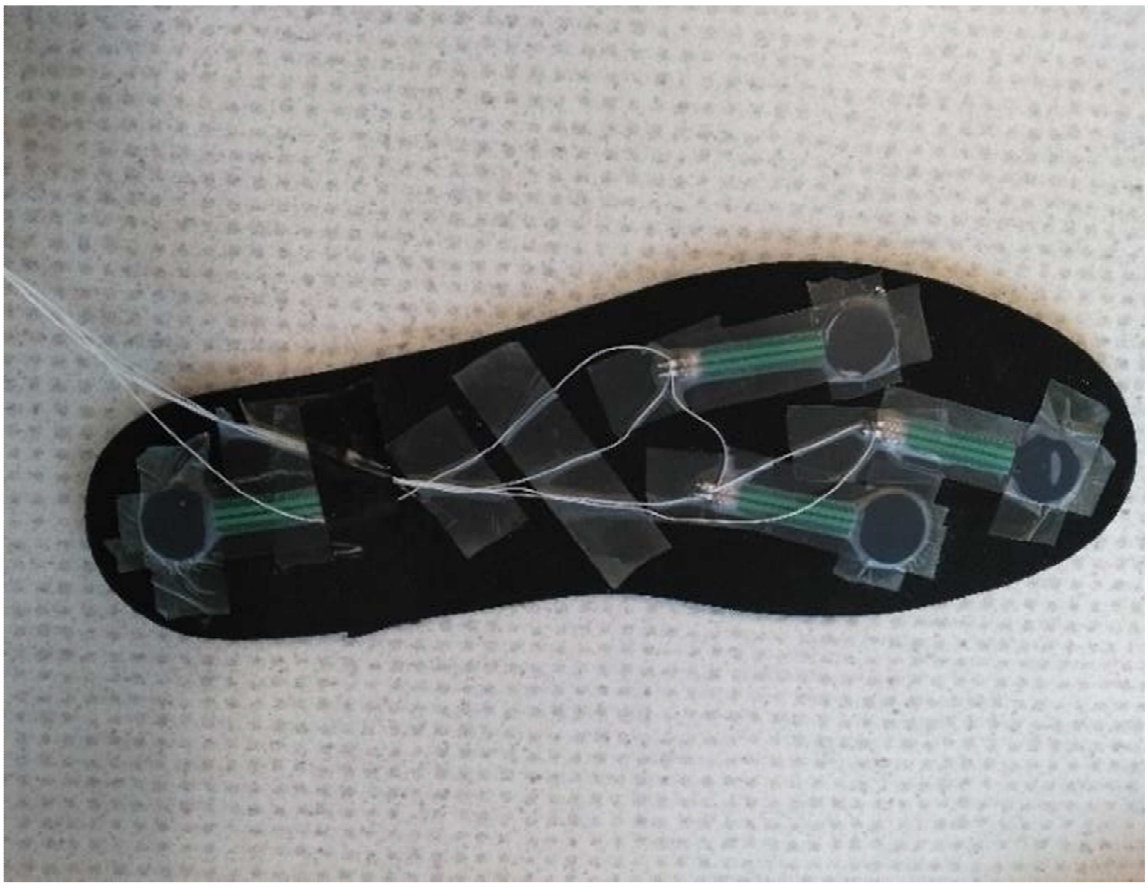

Figure 3.2 FSRs attached to foot pad (photograph by Premkumar Subbukutti).

The FSR sensor data were collected by a RPi3 microcontroller. Since RPi3 does not have an analog to digital conversion (ADC) system, the MCP3008, an 8-channel 10-Bit ADC (Microchip Technology, Arizona, USA) with SPI Interface was used. It was able to convert data from all four analog FSR's into digital values. The MCP3008 IC is shown in Figure 3.3.

Figure 3.3 MCP3008 IC (photograph by Premkumar Subbukutti).

The Figure 3.4 shows how the FSR are connected to the RPi3 using the MCP3008 (ADC IC). The pull-down resistor was used to predict the state of high or low. It stops the output from floating randomly when there is no input condition.

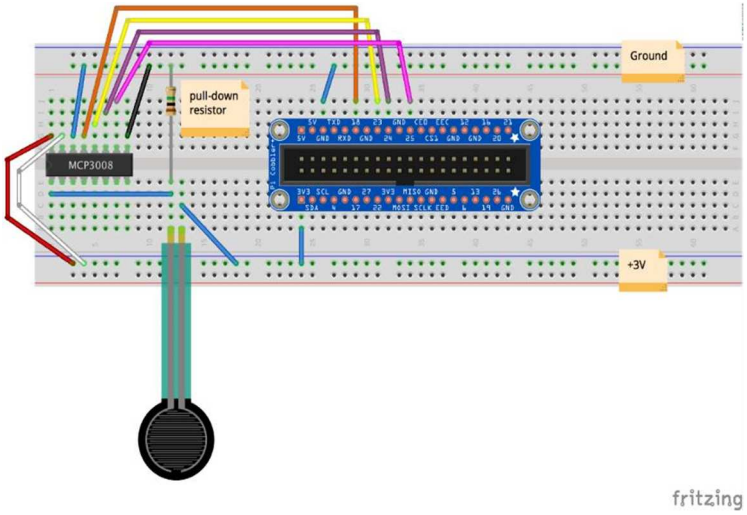

Figure 3.4 FSRs and MCP3008 connected with RPI3.

The Figure 3.5 shows the PCB development board designed to connect the FSR with the RPi3 using MCP3008 analog to digital converter IC. Since we have four FSR'S we designed four pull down resistors along with the MCP3008 IC.

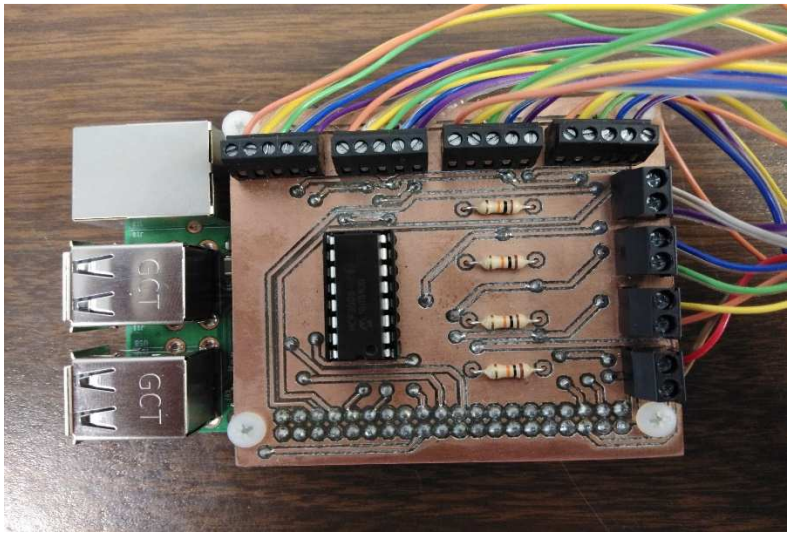

Figure 3.5 PCB Development board for MCP3008 (photograph by Premkumar Subbukutti).

The Figure 3.6 is the schematic diagram showing how the MCP3008 connected with RPi3. Since MCP3008 is working under SPI communication it used a four-pin communication, Master-out-slave-in (MOSI), master-in-slave-out (MISO), clock (SCK) and slave-select (CE0).

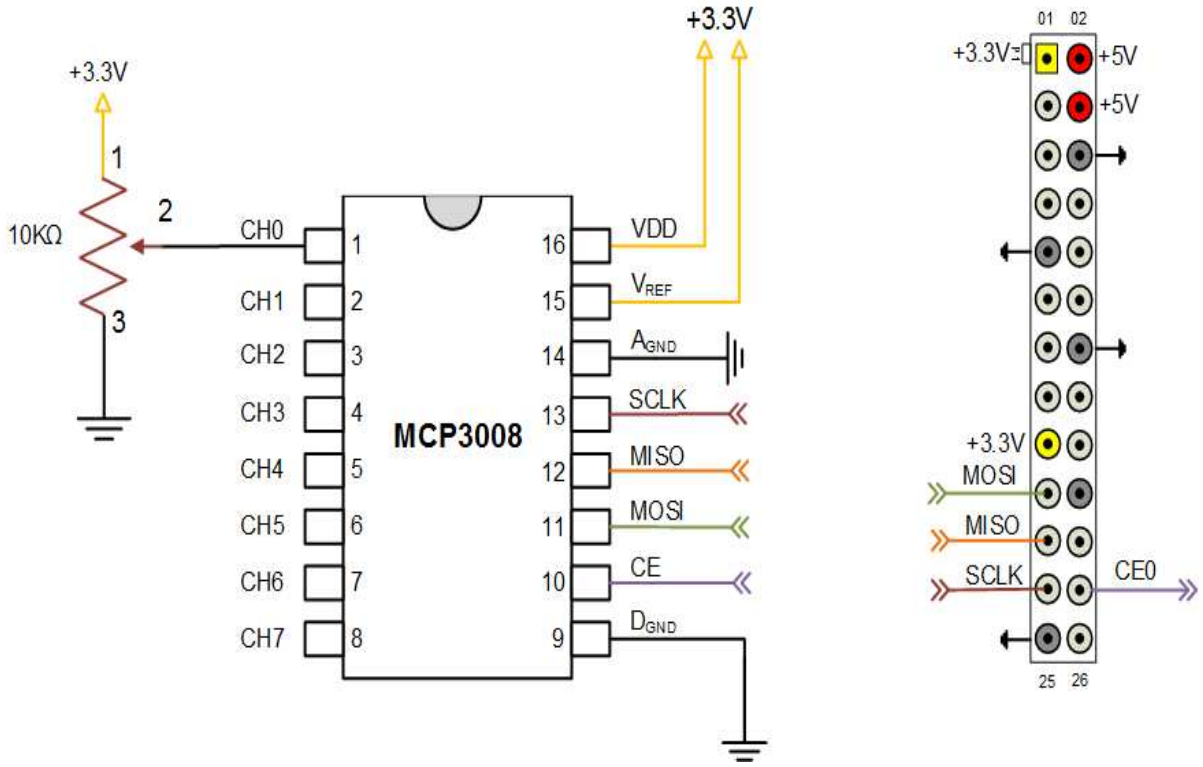

Figure 3.6 Schematic diagram of MCP3008 connected with RPi3 [32].

3.2 Detection of Human Gait Characteristics

Since the study participant was walking in a straight line, a decision was made not to use an IMU equipped with a magnetometer. This is because a magnetometer is used to measure the absolute angle in the transverse plane which does not vary much when the participant is walking in a straight line. The MPU-6050 (Intenseness, San Jose, California, USA) [21] IMU was selected because of its low cost and suitable performance characteristics. The cost of each MPU-6050 is about $3.00 US. The specifications for the MPU-6050 are summarized in Table 3.2:

Table 3.2 Specifications of MPU-6050 (InvenSense, California, USA) [21]

Parameter	**Accelerometer**	**Gyroscope**
Full-Scale Range	±2 g, ±4 g, ±8 g, ±16 g	±250˚/s, 500 ˚/s, ±1000 ˚/s, ±2000 ˚/s
Sensitivity Scale Factor	16384 LSB/g, 8192 LSB/g, 4096 LSB/g, 2048 LSB/g	131 LSB/(˚/s), 65.3 LSB/(˚/s), 32.8
		LSB/(˚/s), 16.4 LSB/(˚/s)
Zero offset	X and Y: ±50 mg, Z: ±80 mg	±20 ˚/s

All the four IMUs were connected to the Raspberry Pi 3 via I^2C protocol (Figure 3.7). This protocol is capable of transmitting data in series using only two busses. They are the data bus (SDA) and clock bus (SCL) as shown in the figure below. The python code for reading the sensor's output via this protocol will be included in Appendix A. Each sensor's output is a two-bytes (16-bit) signed integer. For having symmetric data around zero, the most significant bit was used for determining the sign of the output, thus the range of the received integer was -2^{15} to $2^{15}-1$ instead of 0 to $2^{16}-1$.

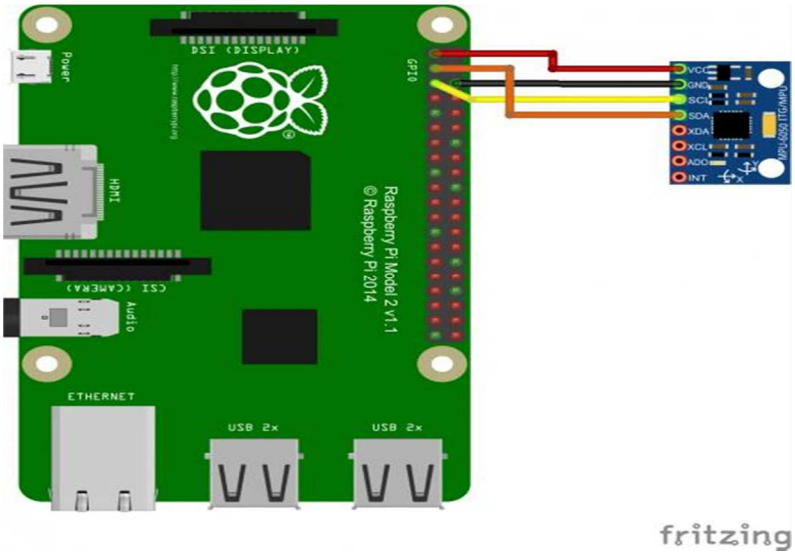

Figure 3.7 IMU with RPI3(image created by Premkumar Subbukutti).

This IMU can output tri-axial accelerometer and tri-axial gyroscope values. The computing orientation from an accelerometer relies on a constant gravitational pull of 1g (9.8 m/s^2). When the IMU is accelerated either by initiating motion or jarring, errors in measurement will occur. A gyroscope measures angular velocity (i.e. the change in orientation angle, not angular orientation itself). Angle data can be determined by integration of the gyroscope output. With integration an initial value must be provided, so the first step is to initialize the sensor with a known position value from the accelerometer, then measure the angular velocity (ω). Per the IMU's data sheet, using a scaling factor of 131 will convert the gyroscope output into degrees/sec. This scaling factor applies in all the three directions so:

Angular velocity = gyroscope output/131 (degrees/sec)

Filters can be used to account for the advantages and disadvantages associated with data collected using the accelerometer and gyroscope to better estimate the actual angle. The complementary filter and the Kalman filter were both evaluated to determine suitable for this application. After examining the results of several tests, the complementary filter was ultimately selected.

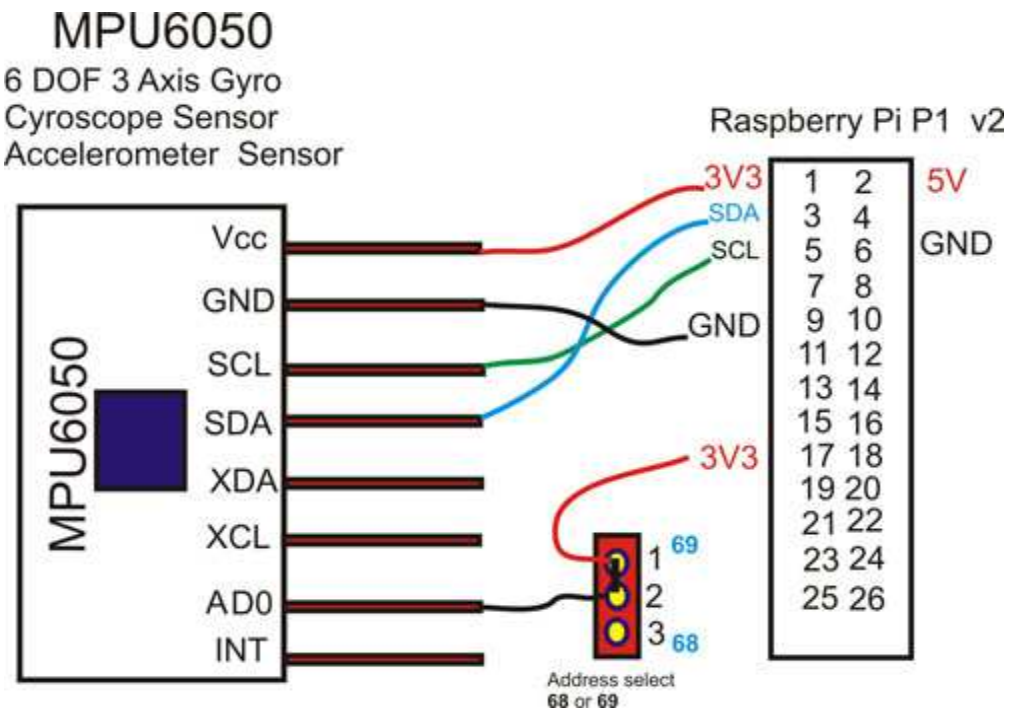

Figure 3.8 MPU6050 with RPI3 Wiring Diagram [32].

3.3 Microcontroller/Microprocessor

A Raspberry Pi 3(RPi3) microcontroller was utilized in this project because of its upgraded technology and popularity in embedded systems. This was an advancement over the second-generation neural prosthesis that used a Teensy microcontroller (PJRC, Oregon, USA). The RPi3 is powered by 3.3v lithium battery. It has two inbuilt advanced technologies, WIFI and Internet. It works on its own operating system called Raspbian, so it is easy to save the collected data directly in to the RPi3 memory as an excel file. The RP3 3 uses three communication protocols I^2C, SPI and UART. Thus, it can communicate with the IMUs using I^2C communication protocol and the FSRs using SPI communication protocol via the MCP3008.

RPi3 can be programmed using the python platform. The python platform is an effective platform for embedded system technology because of its versatile nature, its multitude of libraries

and support from the python community. All the coding for IMUs and FSRs was done using python. A complete list of code is attached in the appendix section A.

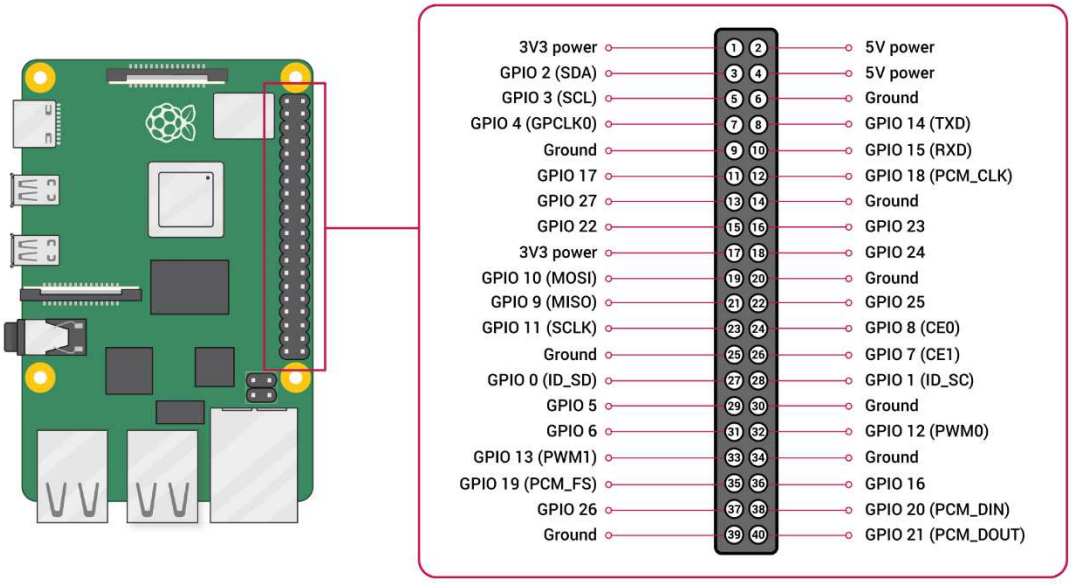

Figure 3.9 RPI3 Pin Diagram.

3.4 Complementary Filter

The complementary filter was used in the neural prosthesis to estimate the actual angle from the gyroscope and accelerometer data. The gyroscope gives precise values over moderate time duration but drifts for longer durations and has no positional reference. The accelerometer output does not drift over time, but significant jitter occurs on short time scales. We implemented the complementary filter to combine the data with the hope of getting better results than could be attained with a single sensor type.

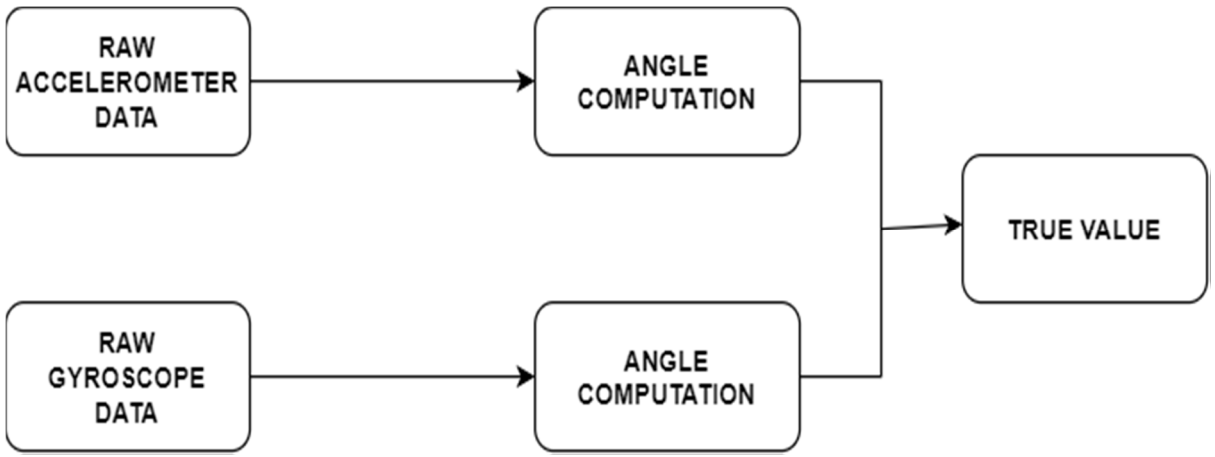

Figure 3.10 Block diagram of complementary filter.

The equation to estimate the actual angle by combining data from both the accelerometer and gyroscope is shown below

$$\theta_{filtered} = \alpha * \theta_{gyroscope} + (1 - \alpha) * \theta_{accelerometer} \qquad \text{(see eqn. 2-4)}$$

For our application, there are times when high frequency spikes occur. Whenever a test subject's heel strikes the ground, an immense vibration is generated. This can be observed as a high frequency spike in the accelerometer data.

3.4.1 Complementary Filter Code Explanation

The *import* command was used to import python libraries into main program. The *Import smbus* was used to get I^2C functions into main program. The *import math* function was used to perform mathematical calculations. The *import time* function was used to implement software clock. The *import RPi.GPIO as GPIO* library function was used to import to control the GPIO pins of RPi3 controller. *Import CSV* was included to store the sensor data's as a CSV file.

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

Listing 1.

The *with open* function was used to open CSV (Comma Separated Values) file named "pitch.ods" and to write the sensor data into it. CSV is the most common import and export format for spreadsheets and databases. The CSV module implements classes to read and write tabular data in CSV format. It allows the programmer to read or write the data's in spreadsheet format. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

```
with open('pitch.ods','a') as f:
    writer=csv.writer(f)
    writer.writerow(['Pitch(98-2)','pitch1(2-98)','pitch2(50-50)','pitch3(60-40)','pitch4(40-60)'])
a=[]
```

Listing 2.

The *def__init__* function was used to declare and initialize all the parameters. The MPU6050 has an embedded 3-axis MEMS gyroscope, a 3-axis MEMS accelerometer. So, we declared and initialized gyro x_axis, gyro y_axis, gyro z_axis, accel x_axis, accel y_axis, accel z_axis. To keep the timing count we initialized the timer to zero. The address of the MPU6050 (ox68) was initialized.

```
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
```

Listing 3.

The sensitivity functions were executed to choose the sensitivity of the IMU sensors. In MPU6050 the gyroscope and accelerometer have 4 types of sensitivity selection each, as shown in the figure above. So, a dictionary was created in python that has all the four options in it. The programmer can decide the value of sensitivity depending on the application's necessity.

```
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])
```

Listing 4.

The *comFilter* function was used to calculate the pitch and roll value from the acceleration data and gyroscope data to substitute those values into complementary filter formula to calculate the actual value. The acceleration and gyro pitch and roll were calculated using their respective formulas as shown in the figure above. Once the pitch value of accelerometer and gyroscope were calculated, the absolute value can be calculated by using the complementary filter equation (2-1).

```
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)
```

Listing 5.

The *try and except* technique was used to detect if an error occurred in current iteration it will not crash the whole program. Instead it will skip the error in current iteration and continue to process next step. In simple words the try block lets you test a block of code for errors. The except block lets you handle the error.

```
try:
    mpu.compFilter()
    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")
```

Listing 6.

3.5 Determining Complementary Filter Settings

Different values of α were evaluated to observe the effect of relying on the accelerometer and gyroscope data at different ratios. To suppress the high frequency noise, the research team initially tried using an alpha value of 98%, highly relying on data provided by the gyroscope value. Upon analysis, it was discovered that the gyroscope takes one second to settle back to zero before reliable data can be collected. This finding caused the team to implement a five second wait time prior to beginning each walking trial to allow the gyroscope to settle. When more emphasis was placed on the accelerometer data, the signal had too much noise for a reliable reading. The research team ultimately decided to use 98% on the gyroscope and 2% on the accelerometer to estimate the actual angle.

3.6 Gait Detection

As discussed in the literature review, the gait will be detected from heel strike to heel strike. To accurately determine that actual heel strike, data collected from the accelerometer (98% weight on the accelerometer), gyroscope (98% weight on the gyroscope) and the heel FSR were compared (Figure 3.11). The blue curve shows data collected from the IMU attached to the foot putting 98% weight on the accelerometer, the red curve is the foot angle collected from putting 98% weight on the gyroscope and the green curve is the heel strike data during walking. In the initial part of the curve, the first 100ms, the study participant is standing still. Notice after 100ms the person begins to walk at regular pace taking 3 steps. The heel strike has the maximum vibration in a single gait cycle. It shows the maximum vibration was happening at 250 ms, 500 ms , and 750 ms.

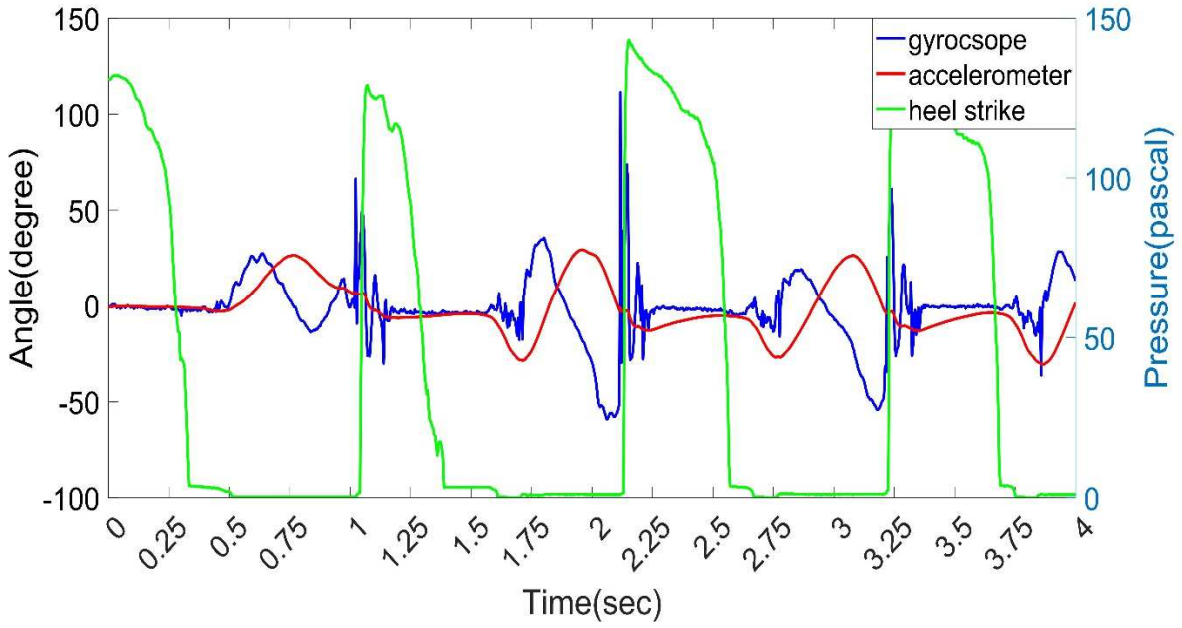

Figure 3.11 Gait detection using neural prosthesis device.

Since the accelerometers are vulnerable to vibrations, the accurate heel strikes can be predicted by comparing the accelerometer data with a heel FSR data. By adding the gyroscope with this comparison, the lagging nature of the gyroscope as well as invulnerability towards vibration was verified. In the Figure 3.11 the glitch in accelerometer happened when the heel strike occurred. Since the gyroscope is not vulnerable to vibration it did not have any glitch associated with it, but it has some time lag. From this verification the walking gait was calculated from the heel strike to heel strike.

3.7 Effect of Complementary Filter Weighting Factor

The complementary filter is used to estimate the actual angle from the gyroscope and accelerometer data. The Figure 3.12 shows the foot angle data with different weighting factor on the gyroscope. The blue curve shows 98 percent weight on the accelerometer and 2 percent weight on the gyroscope. The red curve shows 98 percent weight on the gyroscope and 2 percent weight

on the accelerometer. The green curve shows 50 percent weight on the accelerometer and 50 percent weight on the gyroscope. As shown in the Figure 3.12, increase in the weighting factor on the gyroscope leads to lag but no glitches. Increase in the weighting factor on the accelerometer did not show the lag but had glitches. Putting 50 percent on both the accelerometer and gyroscope has minimal amount of lag as well as glitches. Since the joint angle calculation is based on difference between two IMUs, the data with the glitches cannot be used to determine the shape. Even though the gyroscope data had time lag it can predict the shape precisely. Considering this result, the research team decided to go with the gyroscope data over accelerometer data to calculate joint angles.

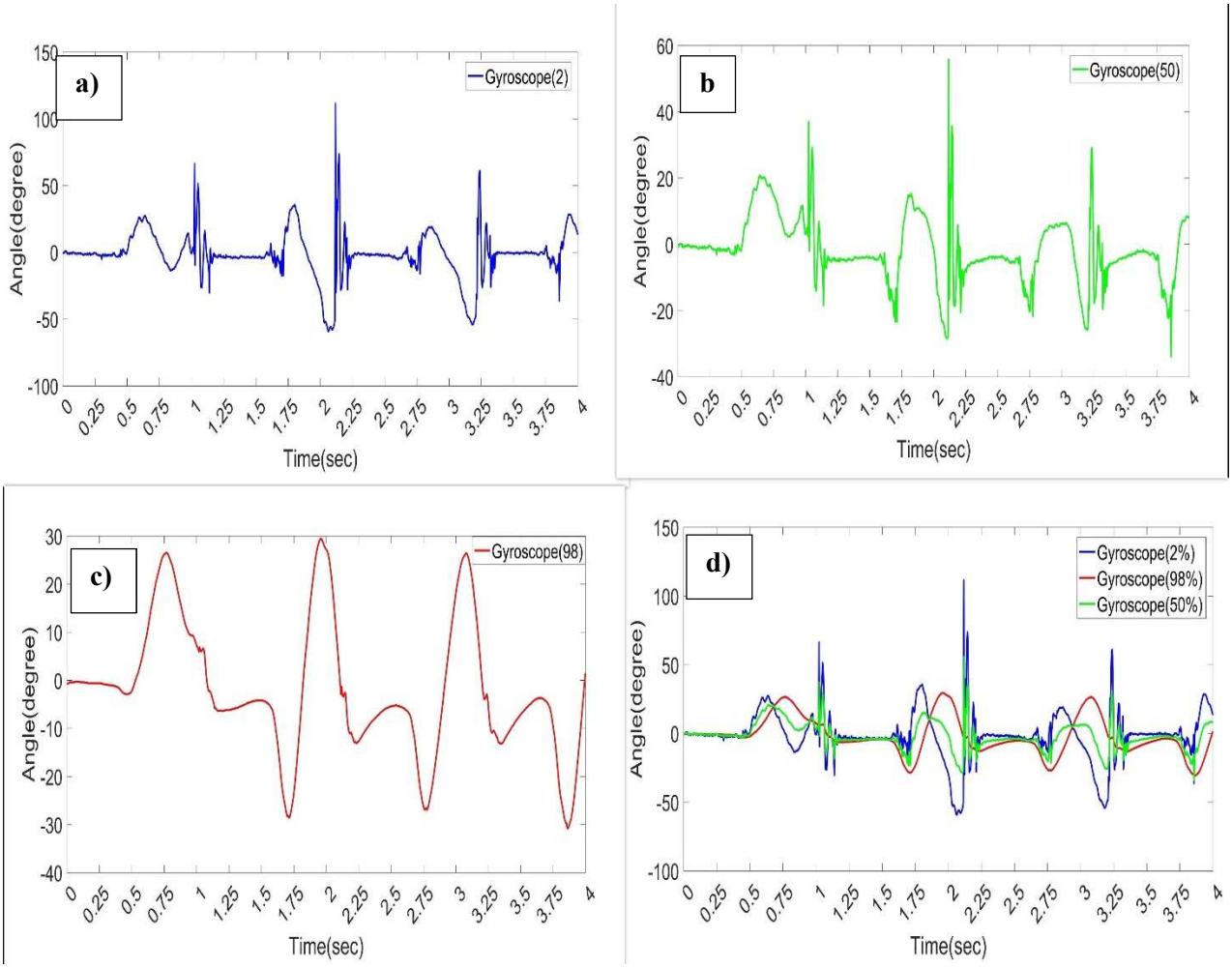

Figure 3.12 Foot angle measured by the complementary filter with different alpha values.

CHAPTER4: TESTING

The purpose of the testing was to determine the ability of the neural prosthesis device to measure and record gait data using IMUs. Tests on a healthy individual were performed in the Human Movement Laboratory located in the Health and Human Science building at Western Carolina University. To perform this test, the third-generation neural prosthesis device was used. The collected data by the neural prosthesis was compared to the data collected from the camera motion capture system to determine the accuracy of the IMUs. The camera motion capture system is an industry accepted standard against which other methods can be compared.

4.1 Equipment Preparation

The neural prosthesis device was tested the day before to the experiment to prevent any delay during the experiment due to technical issues. This test was performed by instructing the participant to walk while the research team observed the collected data. We observed mainly the quality of the data collected looking for hardware issues and the checking the wireless capability to monitor the data collection through the laptops. The data collection rate was also monitored.

4.2 Participant's Preparation

The test protocols were approved by the Institutional Review Board (IRB) at Western Carolina University and the participant signed a consent form prior to participating in the experiment. The participant was asked to wear shorts so that the camera system could record the movement of participant without any disturbance. The purpose of the test and the project were explained to the participant prior to the start of the test. The participant was given a trail walk to learn the protocol before starting the actual test.

4.3 Markers and Sensor Placements

The shoe insole equipped with the FSR sensors was placed in the right shoe of the participant. As shown in Figure 4.1.a., four IMUs were attached to the participant on the foot, shank, thigh and hip as shown in the figure 4.1.

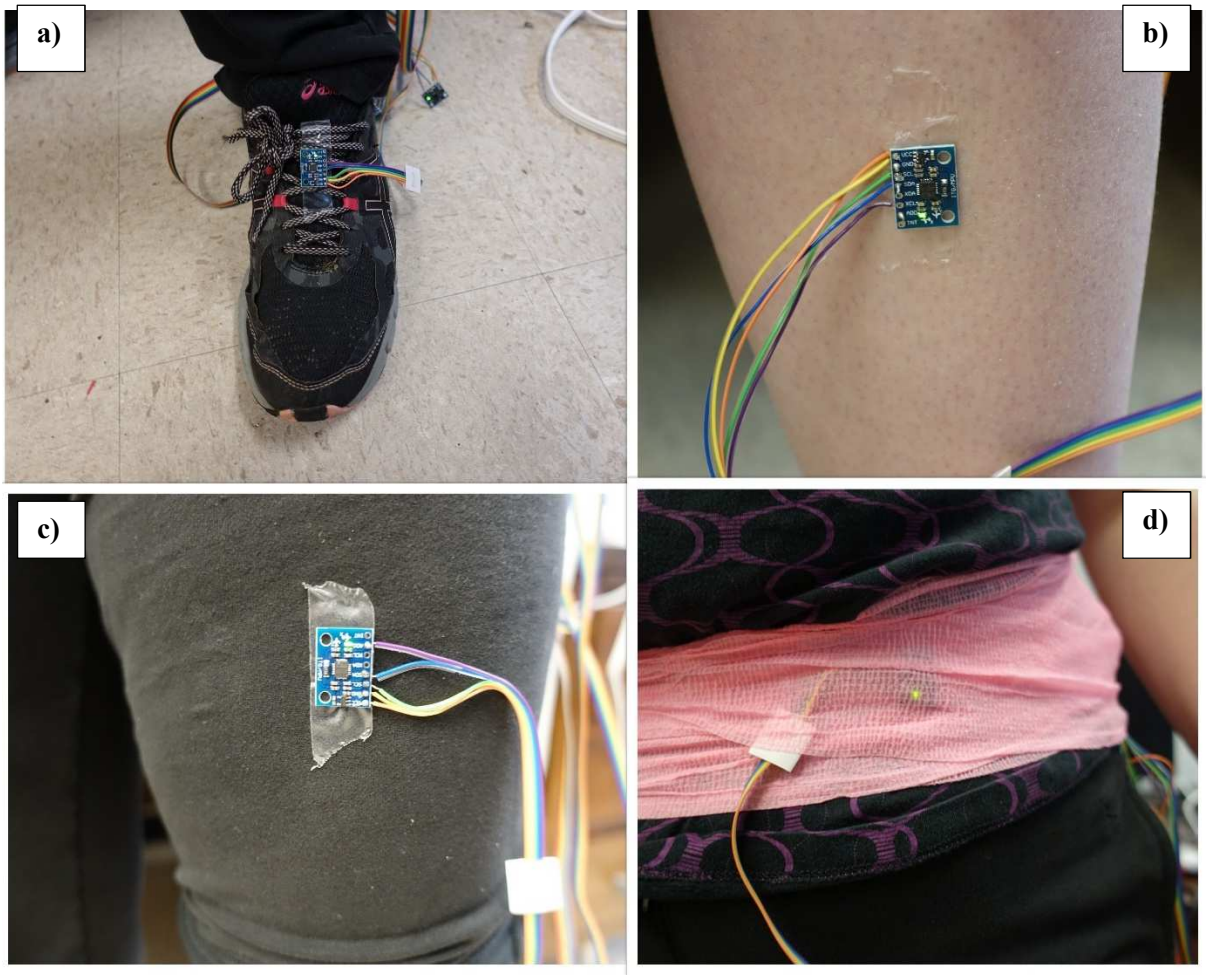

Figure 4. 1 Participant with IMUs attached to the a) foot, b) shank, c) thigh, and d) hip (Photographs by Martin Tanaka)

The difference between foot and shank IMUs can be used to calculate the ankle angle. The difference between shank and thigh can be used to calculate knee angle. The difference between thigh and pelvis can be used to calculate hip angle. The neural prosthesis device was carried in the hand by the participant. The Figure 4.2 shows the participant with markers and sensors attached.

Figure 4.2 Participant with markers and sensors attached (Photograph by Martin Tanaka).

4.4 Testing Procedure

Two different testing conditions were performed by the participant. One used complementary filter to collect IMU data and the other collected IMU data using the Kalman filter. Even though different methods were used to measure the movement data, for the participant, the walking trials were identical.

During testing, one researcher called out the trial number and the testing condition, to make sure the researcher collecting the IMU data was ready to record the data. After data recording began, the participant was asked to stand still for 5 seconds to calibrate gyroscope in the IMUs. Next, a signal was given to instruct the participant to start walking in a straight line. The participant walked normally while the sensors recorded the motion data.

4.5 Data Collection

The data from the IMUs and FSRs were collected using RPi3 microcontroller. The data was stored as a csv file into the microcontroller. Acceleration and angular velocities in three dimensions were captured using four IMUs. Simultaneously, data was collected using the camera motion capture system Qualisys Miqus M3 (Qualisys Americas, Chicago, IL, USA) [18]. The data was obtained, and post processed in the Qualisys Track Manager software (Qualisys Americas, Chicago, IL, USA) [31]. The figure 4.3 shows Qualisys Track Manager software.

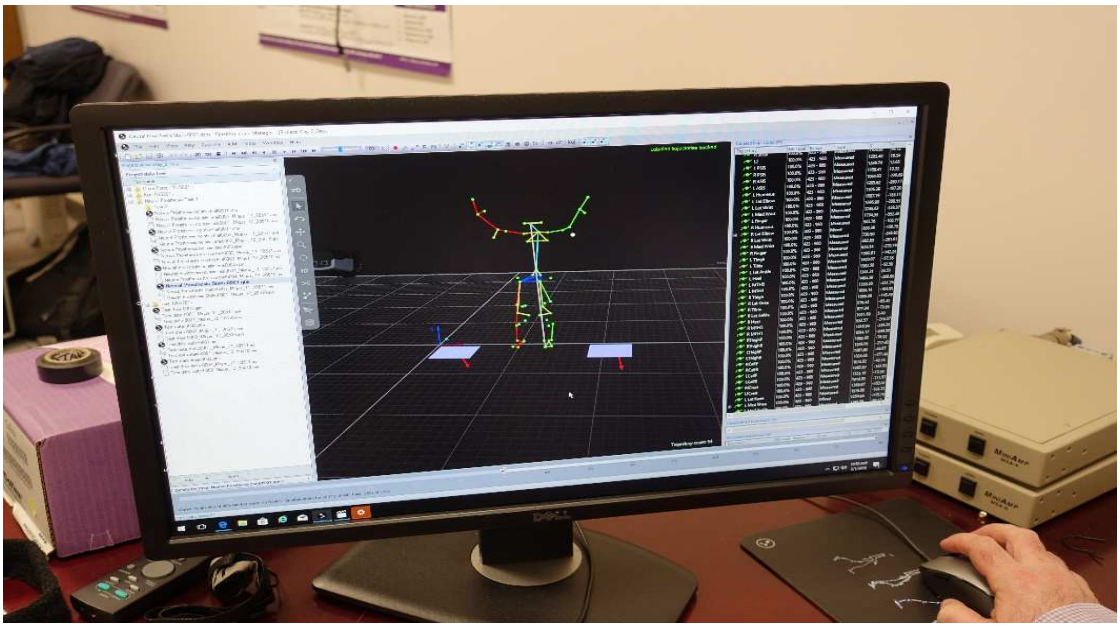

Figure 4.3 Qualisys Track Manager software (Photograph by Martin Tanaka)

CHAPTER 5: RESULTS

In this section, the results obtained from the neural prosthetic device were discussed. The ankle, knee and hip angle were measured using the IMUs and camera motion capture system. The comparison of the two measurement systems was used to determine the accuracy of the IMUs.

5.1 Gait Detection using the Neural Prosthesis

The data collected by the neural prosthesis during gait analysis is discussed in this section. Figure 5.1 shows the ankle angle data. These data were calculated by subtracting the data collected using the IMU on the foot from the IMU on the shank. The green curves are plots for the seven individual trials that were tested. The average of all seven trails is plotted in red.

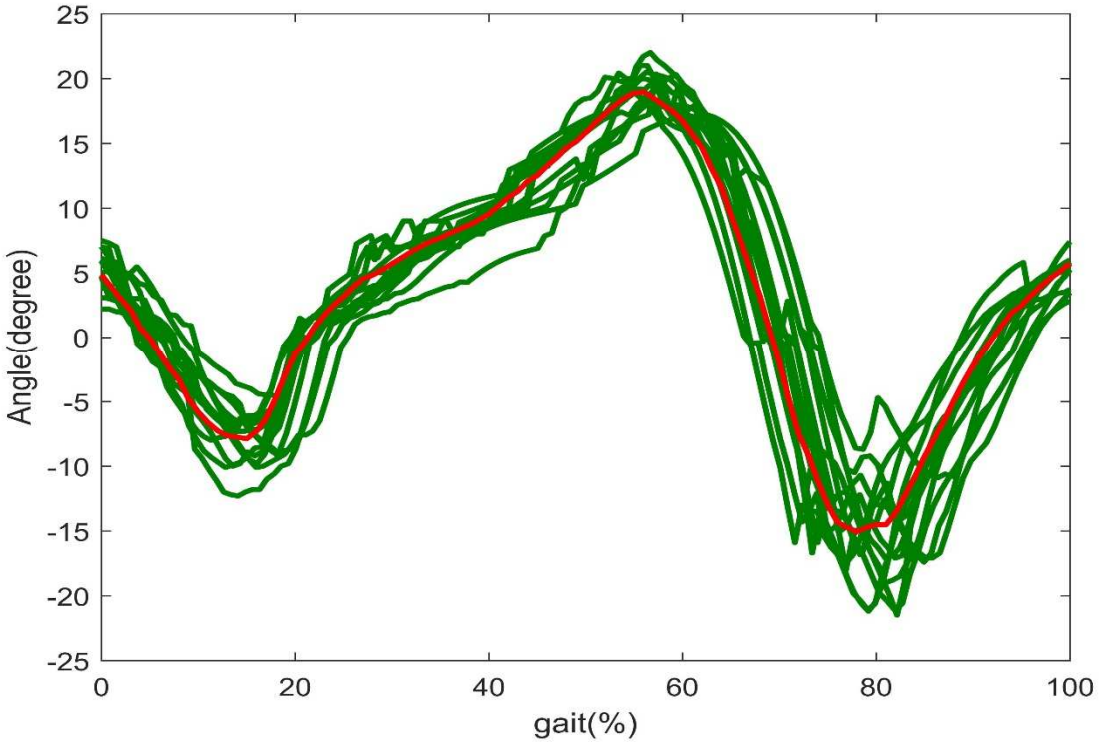

Figure 5.1 Ankle angle measured by the neural prosthesis

0% of the gait cycle represents heel strike. At that time the ankle angle was 5 degrees with the toe pointing upward. Over the next 18% of the gait cycle the foot drops to flat on the ground putting the ankle into about 8 degree of plantarflexion. Weight is applied to the foot and the shank begins to roll over the ankle decreasing the ankle angle about 15 degree of dorsiflexion before the heel raises from the ground at about 55% of the gait cycle. The ankle angle drops sharply as the GN and soleus muscles contract propelling the body forward. The ankle is at approximately 15 degrees of plantar flexion, just prior to toe off at 75% of the gait cycle. At this point, the foot lifts off the ground and the toes are lifted (dorsiflexion) to avoid tripping during the swing phase. At the end of the gait cycle the ankle is back to 5 degrees dorsiflexion in preparation for the next step.

To calculate the knee joint angle, the data from shank and thigh IMUs were used. Figure 5.2 shows the knee angle data. These data were calculated by subtracting the data collected from the IMU on the shank from the IMU on the thigh. The green curves are plots for the seven individual trials that were tested. The average of all seven trails is plotted in red.

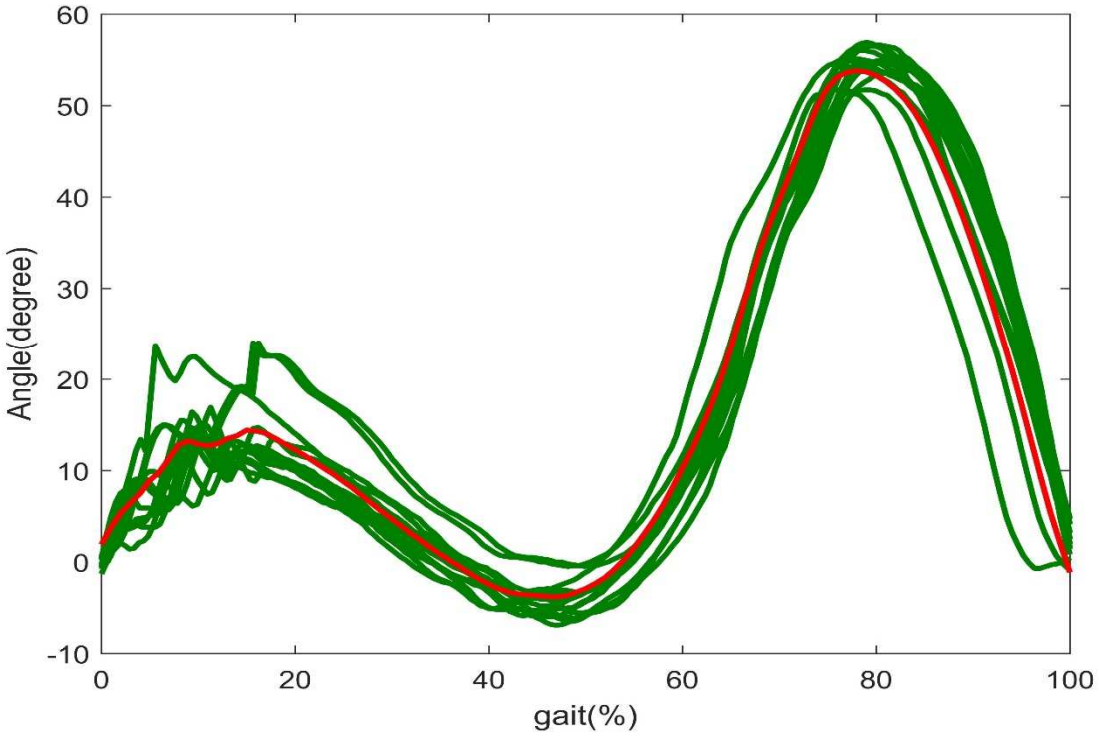

Figure 5.2 Knee angle measured by the neural prosthesis

At heel strike, the knee angle was close to 0 degree indicating that the shank and thigh are in straight line, so there is no flexion in the knee. Over the next 18% of the gait cycle the knee bends as weight is applied achieving a maximum deflection of about 10 degree of flexion, then returning to 0 degree by about 50% of the gait cycle. The knee begins to bend in preparation for the forced is applied through the GN and soleus muscles, and it continues to bend trough toe off and into the swing phase reaching a maximum value of almost 50 degree of flexion at 80% of the gait cycle. In the remaining 20% of the gait cycle, the knee straightens, returning the 0 degrees before the next heel strike.

To calculate the hip joint angle, the data from thigh and pelvis IMUs were used. Figure 5.3 shows the hip angle data. These data were calculated by subtracting the data collected from

the IMU on the thigh from the IMU on the pelvis. The green curves are plots for the seven individual trials that were tested. The average of all seven trails is plotted in red.

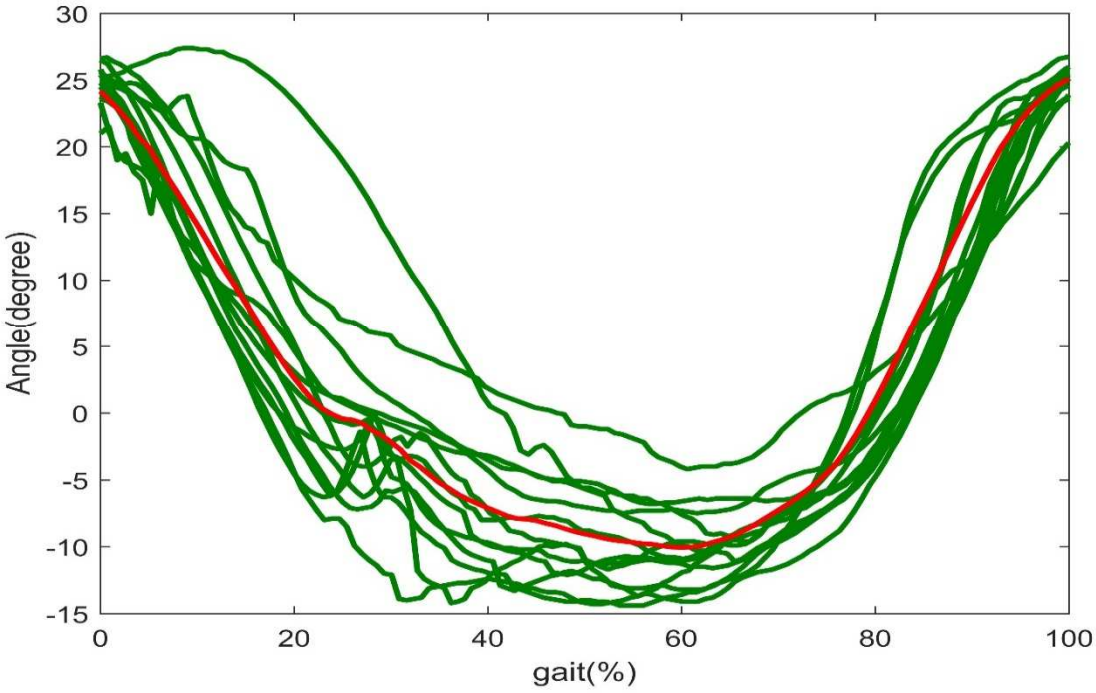

Figure 5.3 Hip angle measured by the neural prosthesis

At heel strike, the thigh is angled out in front of the body at a hip angle close to 25 degree of flexion. Over the next 60% of the gait cycle the hip extends as the body moves over the limb putting the hip into 8 degree of extension. Then the weight is transferred to the contralateral limb (the forward limb without the sensor). The hip flexes again as the ipsilateral limb (the limb for which data is being collected) swings through. At the end of the gait cycle the hip is back to 25 degrees of flexion in preparation for the next step.

The Figure 5.4 shows the average plot of ankle, knee and hip. The average plot of ankle is shown in blue color, the average plot of knee is shown in magenta and the average plot of hip is shown in cyan. The curves generated by data collected form the neural prosthesis device shows

that the device can detect the gait movement. Now we have angles obtained by neural prosthesis these can be compared to data collected by the camera motion capture system to see how accurately the neural prosthesis device can calculate human gait.

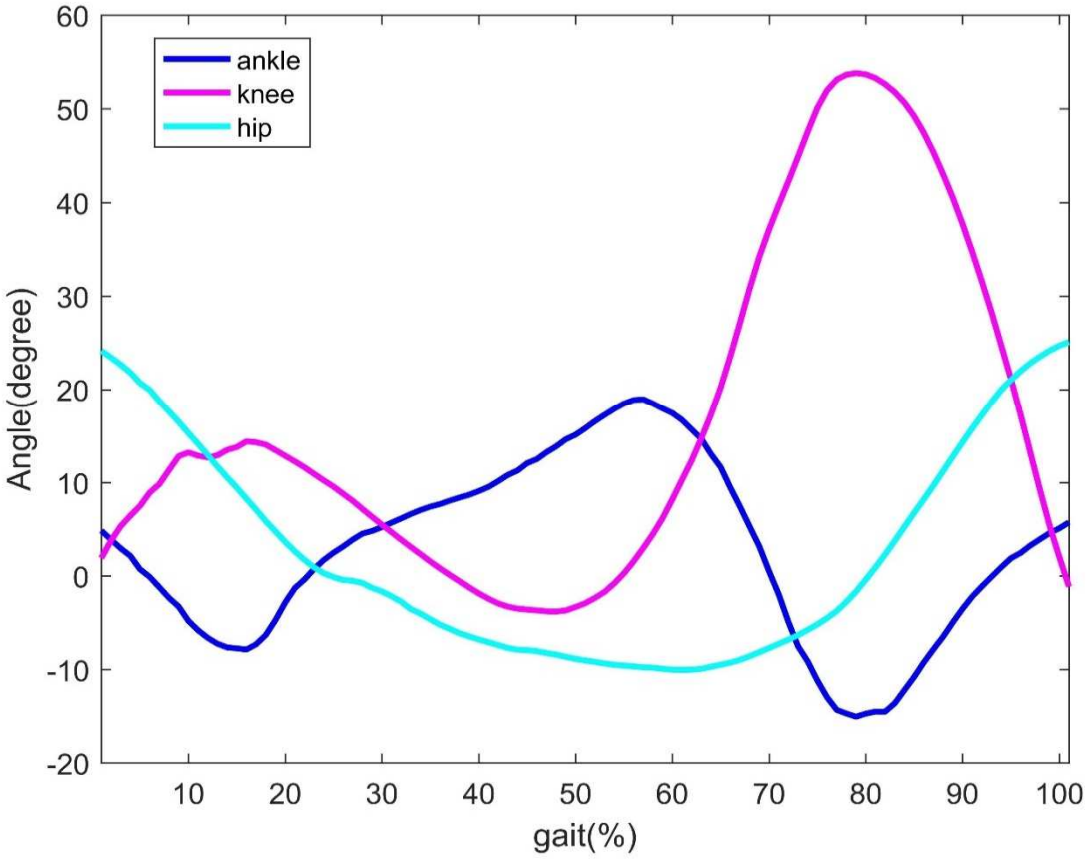

Figure 5.4 Average angles measured by the IMUs

5.2 Camera Motion Capture System's Data

The data of the joint angles collected using the camera motion capture system is discussed in this section. Figure 5.5 shows the ankle angle collected using camera motion capture system. It shows an over plot of ankle angle and the average of all the seven trails. The individual seven trails were plotted in sky blue color and their average was plotted in red.

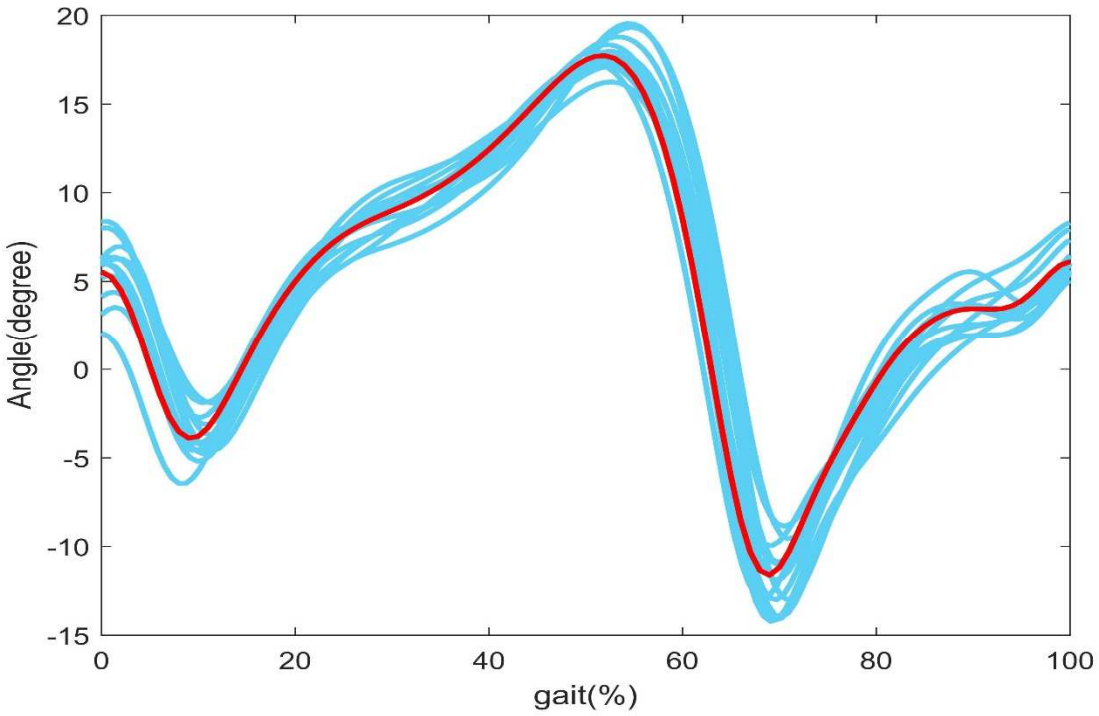

Figure 5.5 Ankle angle measured by the camera motion system.

Figure 5.6 shows the knee angle and the average of all the seven trails. The individual seven trails were plotted in sky blue color and their average was plotted in red.

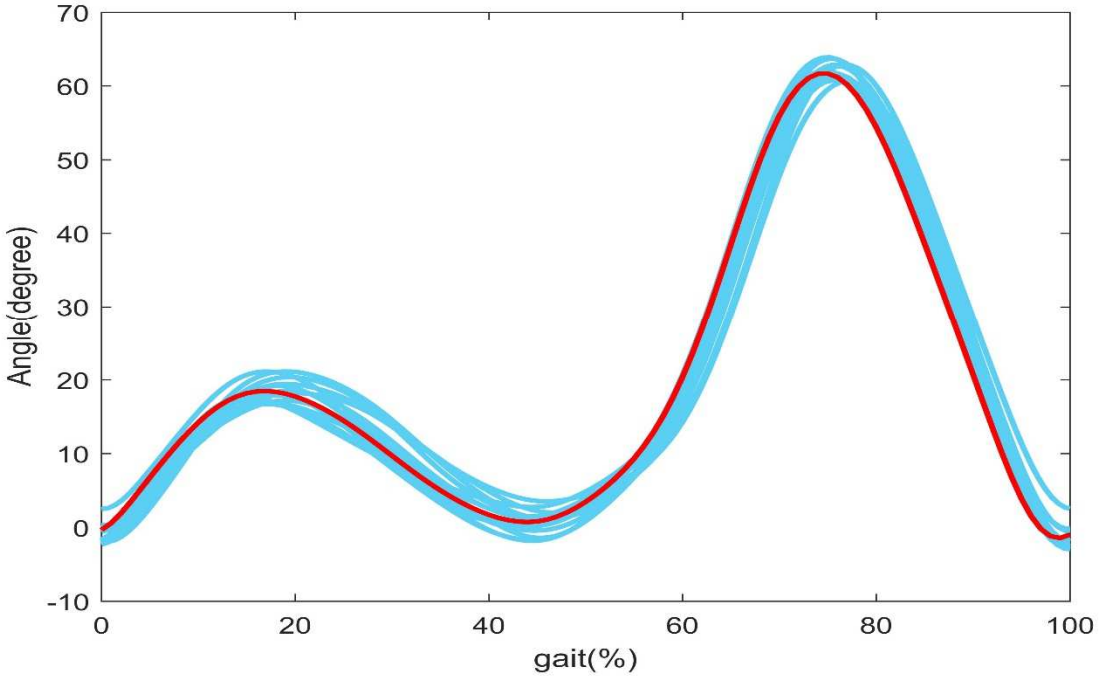

Figure 5.6 Knee angle measured by the camera motion system

The Figure 5.7 shows an over plot of hip angle and the average of all the seven trails. The individual seven trails were plotted in sky blue color and their average was plotted in red.

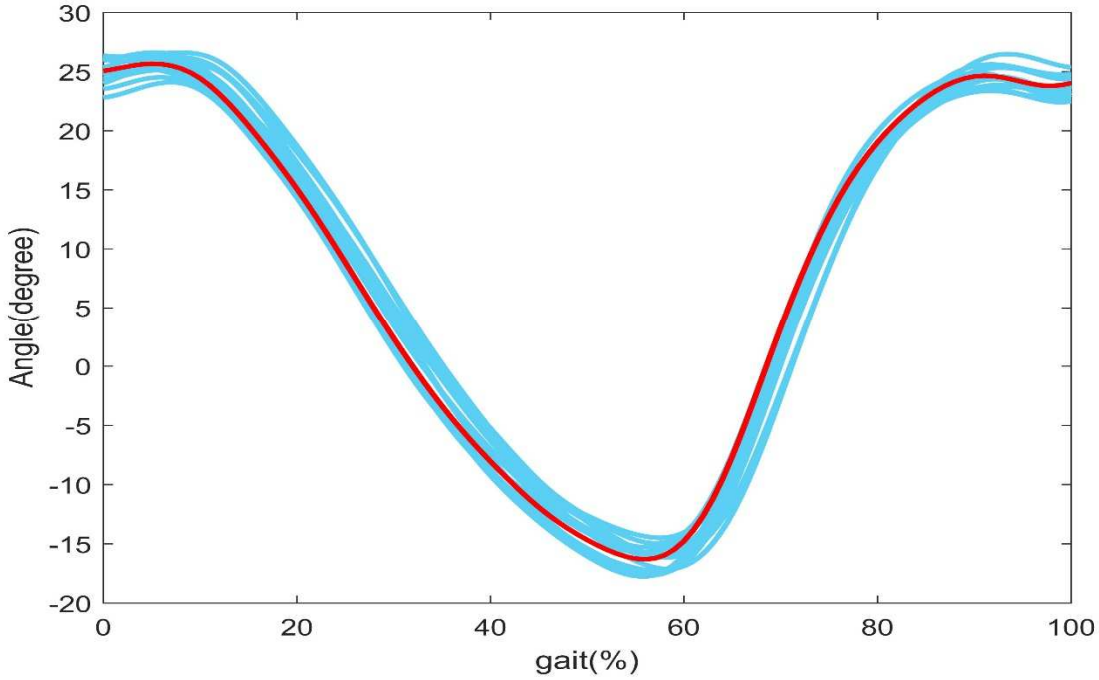

Figure 5.7 Hip angle measured by the camera motion system

The Figure 5.8 shows the average of ankle, knee and hip on the same plot. The average of ankle is shown in blue color, the average of knee is shown in magenta and the average of hip is shown in cyan.

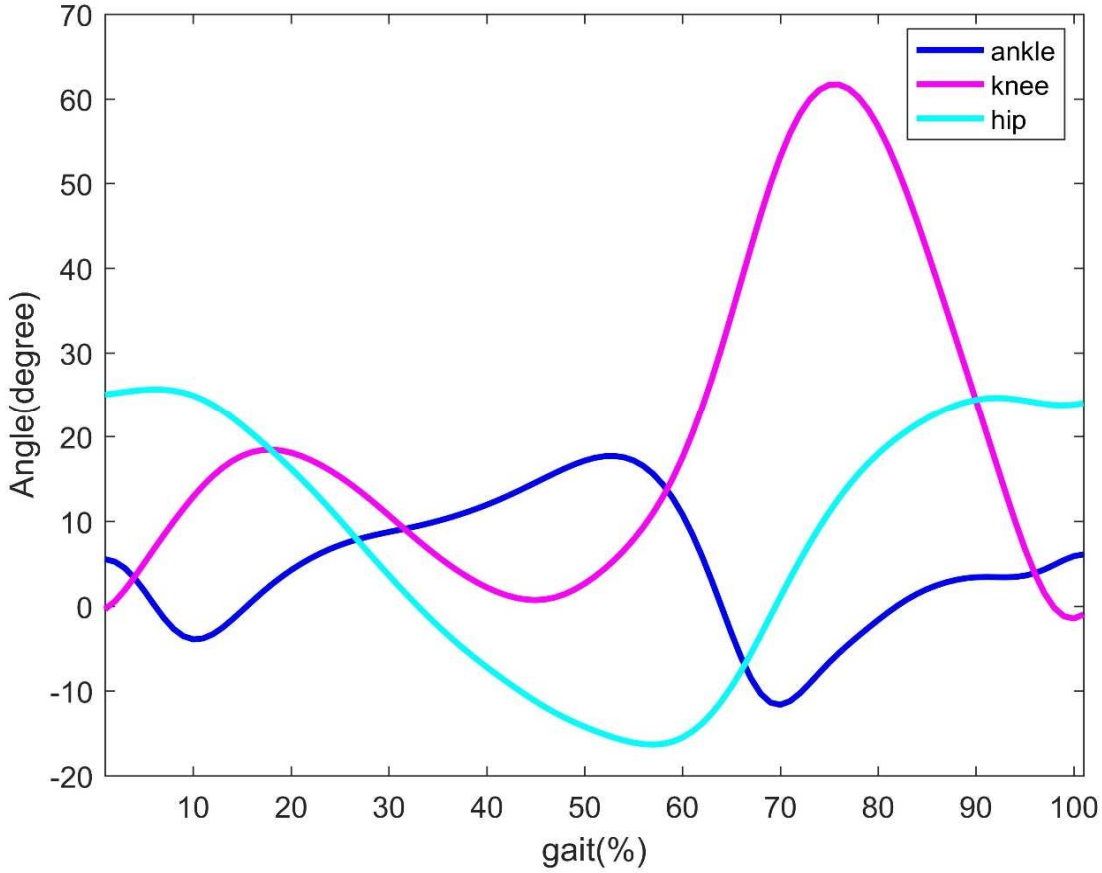

Figure 5.8 Average angles measured by the camera system.

5.3 IMUs and Camera Motion Capture System's Data Comparison

The Figure 5.9 shows the error curve of ankle data collected from camera system versus ankle data collected from neural prosthesis. It shows clearly that the peaks of the neural prosthesis data lags 10 percentage in gait cycle when compared to peaks of the camera data. Because the angle was calculated using gyroscope data, it is expected to have some lag associated with it because of the slow changing nature of the gyroscope. The average error of ankle angle calculated between camera system and neural prosthesis is about 6 degrees.

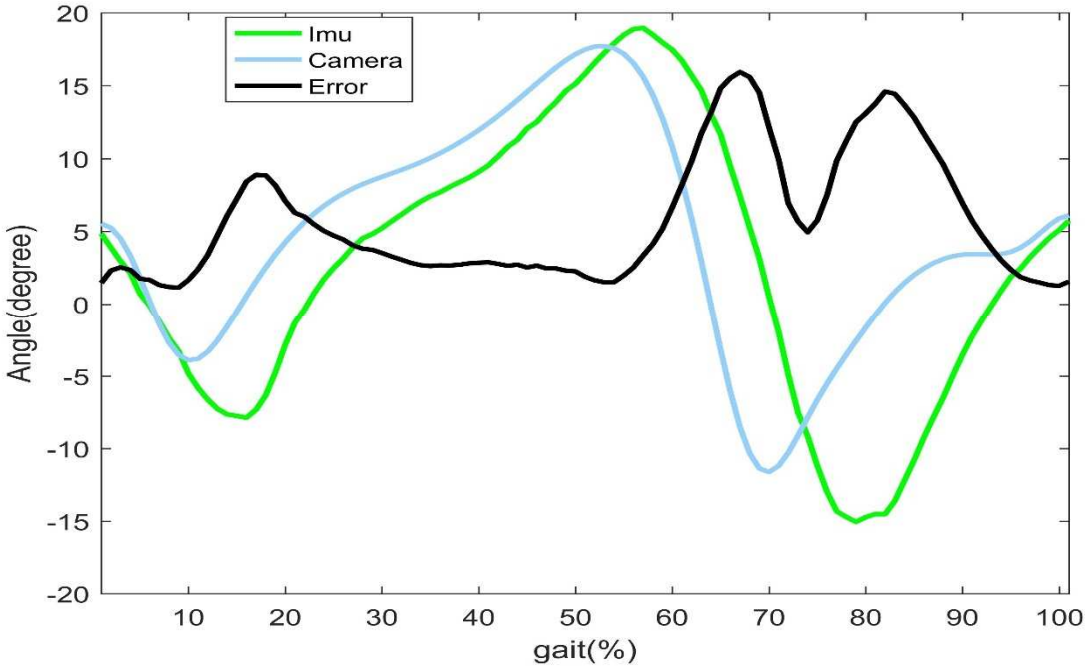

Figure 5.9 Error Estimation of ankle angle.

Figure 5.10 shows the error curve of knee angle data collected from camera system versus knee data collected from neural prosthesis. In the knee angle measurement error is minimal except for between 70 to 90 percentage of the gait cycle. The average error of the knee angle data is about 8 degrees.

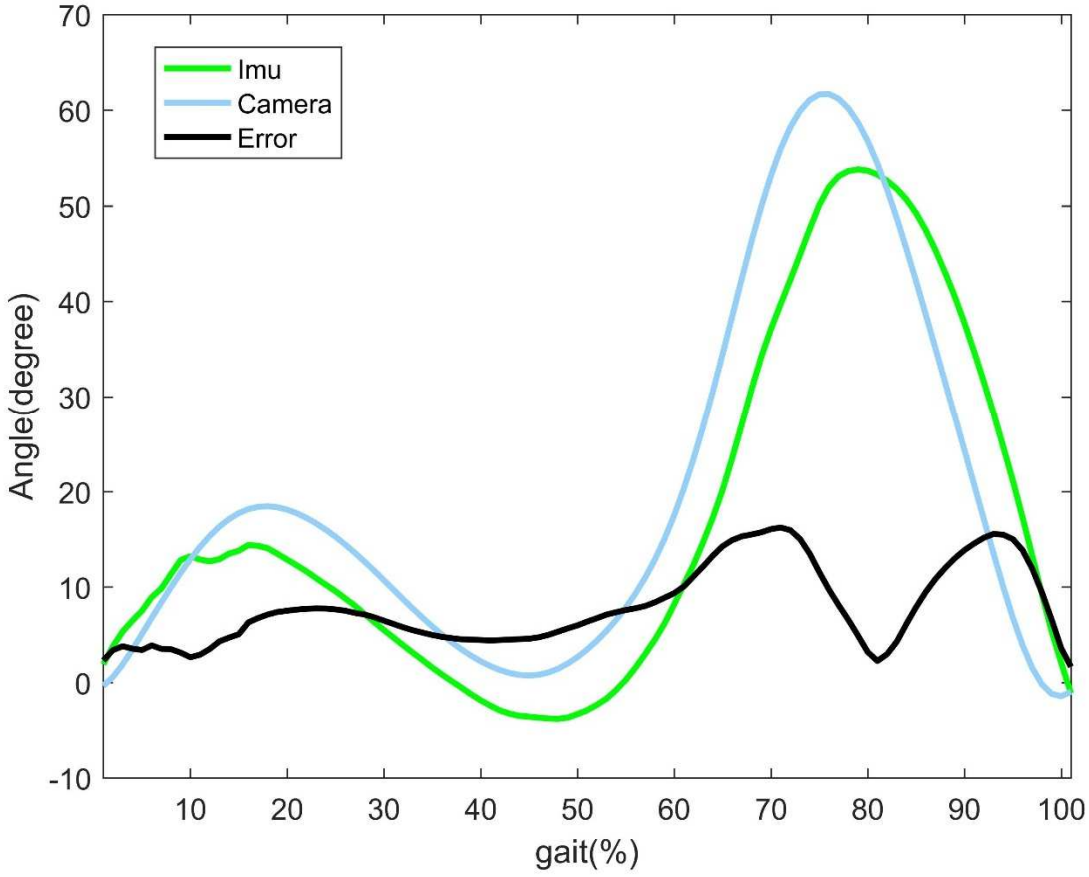

Figure 5.10 Error Estimation of knee angle

The Figure 5.11 shows the error curve of hip angle data collected from camera system versus hip data collected from neural prosthesis. Other than the starting and ending point it has a lot of error associated with it. It unknown why this error exists. The average error of hip angle calculated between camera system and neural prosthesis is about 9 degrees.

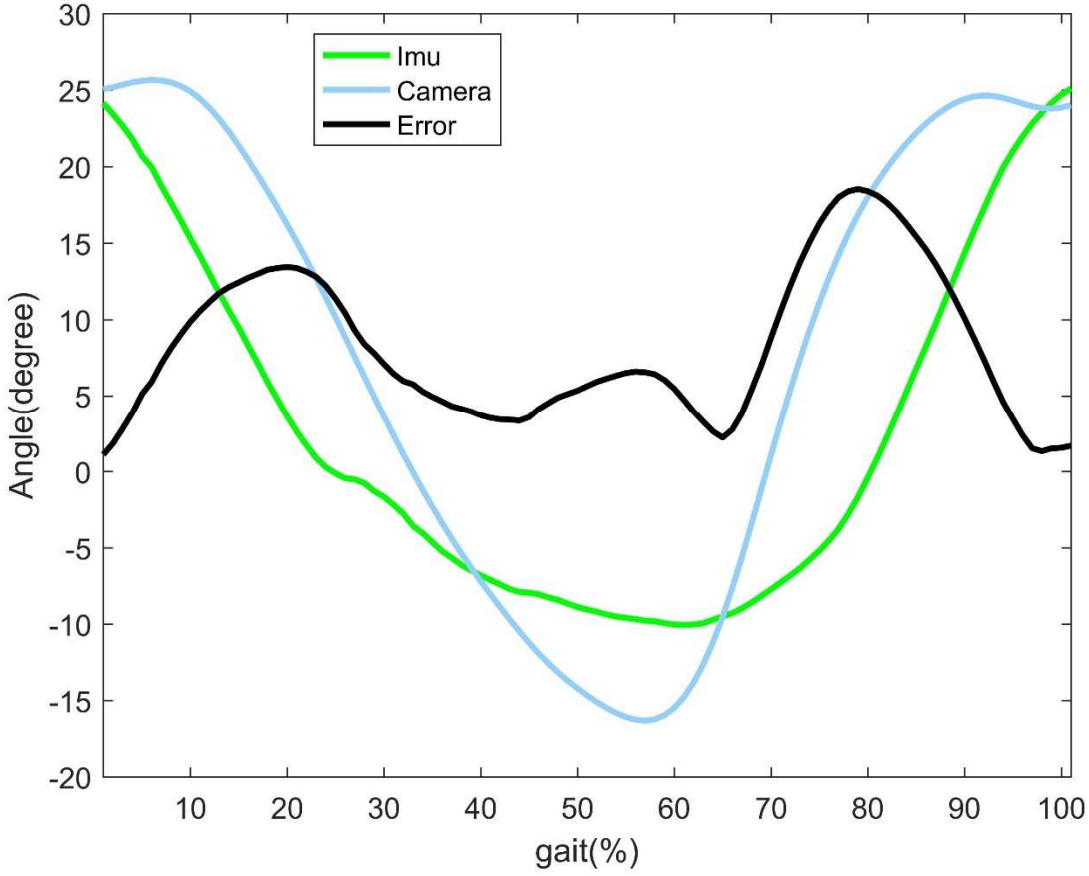

Figure 5.11 Error Estimation of hip angle

CHAPTER 6: DISCUSSION

The specific goal of this research was to develop a neural prosthesis capable of accurately detecting human gait characteristics in order to determine proper timing for artificial muscle stimulation. In this section the strength and weakness of neural prosthesis will be discussed.

The neural prosthesis device was developed using a RPi3 microcontroller. Advanced and powerful CPU core, faster clock speed, high Random-Access Memory (RAM) and advanced technologies like WIFI and python platform in RPi3 convinced our research team to go with this microcontroller over teensy microcontroller which was used in second generation of the neural prosthesis device. The FSR which was used to find the pressure of various parts of foot is an analog sensor which means it will give only the analog output (voltage). It needs a conversion from analog to digital value. The RPi3 does not have a built-in ADC to convert the analog value produced by the FSR to a usable digital input. Absence of inbuilt ADC is the major disadvantage verses the Arduino or the Teensy microcontrollers used in previous generations. As a result, a separate ADC processor (MCP3008 IC) was used. It led to the addition of extra component in the device.

Upon analysis of the collected data, the ankle angle calculated by the neural prosthesis clearly showed that the trails were not tight when compared to the camera motion capture system. Especially around 80 percent of the gait cycle, it seemed that the neural prosthesis had some trouble measuring the lower peak. However, near 0% and 100% of the gait cycle the curves look tight when compared to the peaks of the trail. This could be because of the gyroscope's slow nature in processing the change of values. All the seven trails of the knee angle measured by the neural prosthesis were tight when compared to ankle angle trails. It had a problem of detecting the peak precisely as well. The knee angle was also vulnerable to sudden peak changes. The hip angle

measured using neural prosthesis seems it had lot of errors between the trails when compared to ankle and knee.

On other hand the ankle angle, knee angle, hip angle measured by the camera motion capture system replicates all the seven trails were close and tight to each other. There is not much of the peak amplitude difference between camera system and neural prosthesis. This says we are not facing any problem on predicting the shape or magnitude, but the time lag is the problem causing the error.

The error curves of joint angle data collected with camera system versus neural prosthesis shows clearly that the peaks of the neural prosthesis data lags in gait cycle when compared to peaks of the camera system. This lagging nature could be caused by the gyroscope as we know that the gyroscopes predicts the change very slowly. The error in start and end of the gait cycle is almost zero. If we notice clearly whenever the peak happens the error tend to increase. It could be because of the gyroscope vulnerability towards sudden changes. However, the neural prosthesis successfully captures the shape of ankle joint and the knee joint. But it had trouble on capturing hip joint. This may be the problem associated with pelvis IMU. Because the pelvis had very minimal movement on IMU sensor.

CHAPTER 7: CONCLUSION

The neural prosthesis device utilizing integrated IMUs was able to estimate the gait characteristics while walking. Usage of RPi3 allowed us to utilize WIFI to monitor the data collection at real time. The PCB design for the hardware reduced the wiring complications. The design of the complementary filter using python software allowed us to do real time filtering instead of post processing which was in the case of second-generation neural prosthesis device. This improvement opens the way to utilize the filtered real time data for the future upgrades. The results showed that the neural prosthesis was able to capture the general shape of the joint angle curves when compared to the camera motion capture system. However, the joint angles obtained from the neural prosthesis device lagged that actual joint angles found using the camera system. This is likely due to a slow response time in the gyroscope.

Future work will include measures taken to suppress lag and the drift in the gyroscope data. This can be done by including a high pass filter in the design. Since our MPU6050 IMU is a digital sensor, implementing a digital high pass filter will be easy to build and test. Digital filters also do not drift with temperature or humidity and it does not require precision components like analog filters. In addition, a digital filter does not suffer from aging. The main drawback is that the digital filter will require additional processing power in the microcontroller processor and must be implemented in real time. Even though we implement high pass filter to the gyroscope data it will only reduce the drift not the lag. There are also high performance IMUs available in the market which has faster processing speed and response time than the MPU6050 IMU which could be good option to reduce lag in the gyroscope.

On the other hand, we can also utilize the accelerometer data to calculate joint angles once we suppress the high frequency noise associated with the accelerometer data. This can also be done by designing digital low pass filter or buying the advanced IMU with a low pass filter built into the hardware. The inbuilt hardware low pass filter can reduce the accelerometer noise without placing extra stress on microcontroller processor in the neural prosthesis.

RPi3 microcontroller used in the third-generation neural prosthesis can also be replaced with the latest version of Raspberry Pi or with some other improved microcontroller. This change will improve the overall device performance. The latest version of RPi family is RPi4 which has advanced CPU with fastest clock speed of 1.5GHZ and 4GB RAM. Which will be literally twice as fast as the RPi3.

There is one other idea that could further improve the performance of the neural prosthesis. This study quantified that accuracy of joint angle data collected from the neural prosthesis using data collected from camera system as a reference. Instead of post processing the neural prosthesis data to calculate joint angles, we can directly use the data from foot, shank, thigh and pelvis IMU to predict the gait on real time. For our future goal is to design an artificial neural network to stimulate the gastrocnemius muscle on the particular percent of the gait. Real time data is needed for this application, so using the data directly from foot, shank, thigh and pelvis will be most helpful.

REFERENCE

[1] K. Tong and M. H. Granat, "A practical gait analysis system using gyroscopes," *Medical Engineering & Physics*, vol. 21, no. 2, pp. 87–94, 1999.

[2] Clifton L. Gooch, Etienne Pracht, Amy R. Borenstein. "The Burden of Neurological Disease in the United States: A Summary Report and Call to Action". Annals of Neurology 2017; DOI: 10.1002/ana.24897

[3] G. P. Panebianco, R. Stagni, and S. Fantozzi, "Comparative analysis of 12 methods using wearable inertial sensors for gait parameters estimation during walking," *Gait & Posture*, vol. 57, p. 21, 2017.

[4] Y. Li and J. J. Wang, "A robust pedestrian navigation algorithm with low cost IMU," *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2012.

[5] L. Meng, B. Porr, C. A. Macleod, and H. Gollee, "A functional electrical stimulation system for human walking inspired by reflexive control principles," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 231, no. 4, pp. 315–325, 2017.

[6] Ate and M. Abdelrahim, "Controlling the temperature reactor based on Raspberry Pi system control," *2018 5th International Conference on Electrical and Electronic Engineering (ICEEE)*, 2018.

[7] Christopher & Dana Reeve Foundation https://www.christopherreeve.org/living-with-paralysis/stats-about-paralysis.

[8] Human Gait Modeling and Analysis Using a Semi-Markov Process with Ground Reaction Forces Hao Ma, Student Member, IEEE, and Wei-Hsin Liao, Senior Member, IEEE.

[9] J. J. Kavanagh and H. B. Menz, “Accelerometry: A technique for quantifying movement patterns during walking,” Gait Posture, vol. 28, no. 1, pp. 1–15, Jul. 2008.

[10] N. C. Bejarano et al., “A novel adaptive, real-time algorithm to detect gait events from wearable sensors,” IEEE Trans. Neural Syst. Rehabil. Eng., vol. 23, no. 3, pp. 413–422, May 2015.

[11] Simulation of Human Locomotion Using A Musculoskeletal Model Taesoo Kim and Sungho Jo Department of Electrical Engineering and Computer Science, KAIST, Daejeon, Korea

(Tel: +82-42-869-3540; E-mail: {tsgates, shjo}@kaist.ac.kr)

[12] Farris, Dominic James, and Gregory S. Sawicki. "The mechanics and energetics of human walking and running: a joint level perspective." Journal of The Royal Society Interface (2011): rsif20110182.

[13] Moore, K. L., Dalley, A. F., & Agur, A. M. (2013). Clinically oriented anatomy. Lippincott Williams & Wilkins

[14] Lazar, Eric, and Juan Nicolás Cuenca. "Functional electrical simulation (FES) in stroke." (2008).

[15] Gait Analysis with IMU Gaining New Orientation Information of the Lower Leg Steffen Hacker, Christoph Kalkbrenner, Maria-Elena Algorri and Ronald

Blechschmidt-Trapp Institute of Medical Engineering, University of Applied Science Ulm, Albert-Einstein-Allee 55, 89075 Ulm, Germany.

[16] An Open-source Multi Inertial Measurement Unit (MIMU) Platform Isaac Skog, JohnOlof Nilsson, and Peter Handel ¨ Department of Signal Processing, ACCESS Linnaeus Centre KTH Royal Institute of Technology Oscula's vat 10, SE- ¨ 100 44 Stockholm, Sweden.

[17] Raspberry Pi Foundation. "About Us". (2017 Jun 25). [Online]. Available: https://www.raspberrypi.org/about/.

[18] Wang Mei, Prediction and location system of three-phase cable fault based on neural network, Journal of Xi'an University of Science and Technology, China,2004, 24(2), PP225-229.

[19] Shetty, Yadira K. Robust Human Motion Tracking Using Low-cost Inertial Sensors. Diss. Arizona State University, 2016.

[20] Deluca, P. A., & Renshaw, T. S. (1995). Gait analysis: principles and applications. Emphasis on its use in cerebral palsy. JBJS, 77(10), 1607-1623.

[21] Shetty, Yatiraj K. Robust Human Motion Tracking Using Low-cost Inertial Sensors. Diss. Arizona State University, 2016.

[22] Guerra-Filho, G. (2005). Optical Motion Capture: Theory and Implementation. RITA, 12(2), 61-90.

[23] An Improved Adaptive Kalman Filtering Algorithm for balancing vehicle.

[24] Shetty, Yatiraj K. Robust Human Motion Tracking Using Low-cost Inertial Sensors. Diss. Arizona State University, 2016.

[25] Abhayasinghe, Kahala Nimsiri. Human gait modelling with step estimation and phase classification utilising a single thigh mounted IMU for vision impaired indoor navigation. Diss. Curtin University, 2016.

[26] Lazar, Eric, and Juan Nicolás Cuenca. "Functional electrical simulation (FES) in stroke." (2008).

[27] Guy, J. E., FUNCTIONAL ELECTRICAL STIMULATION RECUMBENT BICYCLE FOR STROKE REHABILITATION, MST Thesis, Western Carolina University, 2013.

[28] Haibin Wang And Qing He. “An electrical muscle simulator based on function electrical stimulation”.2012 IEEE International Conference on Robotics and Biomimetics (ROBIO).

[29] Matthew C. Gash; Matthew Varacallo. “Physiology, Muscle Contraction”.

[30] FSR 402, https://www.interlinkelectronics.com/fsr-402, retrieved 2018-07-26.

[31] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[32] https://components101.com/ics/mcp3008-adc-pinout-equivalent-datasheet

[33] Reza Farsad Asadi, “A Neural Prosthesis to Improve Gait in People with Muscle Weakness”, MST Thesis, Western Carolina University, 2013.

[34] Tanaka ML, Hudson D, Farsad R,”Development of Electrical Stimulation Devices for Fall Prevention and Stroke Rehabilition”, ASME International Mechanical Engineering Congress and Exposition(IMECE), November 3-9, Tampa, Florida(2017).

[35] https://www.digikey.com/en/maker/blogs/raspberry-pi-wi-fi-bluetooth-setup-how-to-configure-your-pi-4-model-b-3-model-b

APPENDIX A: COMPLEMENTARY FILTER

```
import smbus
import math
import time
import RPi.GPIO as GPIO
import csv
with open('pitch.ods','a') as f:
    writer=csv.writer(f)
    writer.writerow(['Pitch(98-2)','pitch1(2-98)','pitch2(50-50)','pitch3(60-40)','pitch4(40-60)'])
a=[]


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:  [65.3,  0x08],
```

```
            1000: [32.8,  0x10],
            2000: [16.4,  0x18]
        }.get(x,  [65.3,  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(self.address, 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!")

    # Take N readings for each coordinate and add to itself
    '''for ii in range(N):
        self.getRawData()
        self.gyroXcal += self.gx
        self.gyroYcal += self.gy
        self.gyroZcal += self.gz

    # Find average offset value
    self.gyroXcal /= N
    self.gyroYcal /= N
    self.gyroZcal /= N'''

    # Display message and restart timer for comp filter
    '''print("Calibration complete")
    print("\tX axis offset: " + str(round(self.gyroXcal,1)))
    print("\tY axis offset: " + str(round(self.gyroYcal,1)))
    print("\tZ axis offset: " + str(round(self.gyroZcal,1)) + "\n")'''
    #time.sleep(2)
    self.dtTimer = time.time()

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

    # Subtract the offset calibration values
    '''self.gx -= self.gyroXcal
    self.gy -= self.gyroYcal
    self.gz -= self.gyroZcal'''

    # 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(str(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)))"'

'"GPIO.setmode(GPIO.BOARD)
GPIO.setup(7,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(7,GPIO.HIGH)'''
#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)
while Actual:
    '''#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)'''

    '''# Run for 20 secounds
    startTime = time.time()
    while(time.time() < (startTime + 20)):'''
    try:
        mpu.compFilter()
        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")
```

APPENDIX B: KALMAN FILTER

```
#Connections
#MPU6050 - Raspberry pi
#VCC - 5V (2 or 4 Board)
#GND - GND (6 - Board)
#SCL - SCL (5 - Board)
#SDA - SDA (3 - Board)

import RPi.GPIO as GPIO
from Kalman import KalmanAngle
import smbus                #import SMBus module of I2C
import time
import math
#SPI
import Adafruit_GPIO.SPI as SPI
import Adafruit_MCP3008

import csv
with open('pitch.ods','a') as f:
    writer=csv.writer(f)
    writer.writerow(['fp1','fp2','fp3','fp4','p1','p2','p3','p4'])
a=[]

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


kalmanX = KalmanAngle()
kalmanY = KalmanAngle()

RestrictPitch = Actual
radToDeg = 57.2957786
kalAngleX = 0
kalAngleY = 0
#some MPU6050 Registers and their Address
PWR_MGMT_1   = 0x6B
SMPLRT_DIV   = 0x19
CONFIG       = 0x1A
GYRO_CONFIG  = 0x1B
INT_ENABLE   = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
```

```
GYRO_XOUT_H  = 0x43
GYRO_YOUT_H  = 0x45
GYRO_ZOUT_H  = 0x47


#Read the gyro and acceleromater values from MPU6050
def MPU_Init():
        #write to sample rate register
        bus.write_byte_data(DeviceAddress, SMPLRT_DIV, 7)

        #Write to power management register
        bus.write_byte_data(DeviceAddress, PWR_MGMT_1, 1)

        #Write to Configuration register
        bus.write_byte_data(DeviceAddress, CONFIG, 0)

        #Write to Gyro configuration register
        bus.write_byte_data(DeviceAddress, GYRO_CONFIG, 24)

        #Write to interrupt enable register
        bus.write_byte_data(DeviceAddress, INT_ENABLE, 1)


def read_raw_data(addr):
        #Accelero and Gyro value are 16-bit
        high = bus.read_byte_data(DeviceAddress, addr)
        low = bus.read_byte_data(DeviceAddress, addr+1)

        #concatenate higher and lower value
        value = ((high << 8) | low)

        #to get signed value from mpu6050
        if(value > 32768):
                value = value - 65536
        return value


bus = smbus.SMBus(1)    # or bus = smbus.SMBus(0) for older version boards
DeviceAddress = 0x68   # MPU6050 device address


#GPIO pin setup
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)
while Actual:
        try:
            # Read all the ADC channel values in a list.
            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} | {7:>4}
|'.format(*values))
            a.append(values[0])
            a.append(values[1])
            a.append(values[2])
            a.append(values[3])

            GPIO.output(29,GPIO.LOW)
            MPU_Init()

            #time.sleep(1)
            #Read Accelerometer raw value
            accX = read_raw_data(ACCEL_XOUT_H)
            accY = read_raw_data(ACCEL_YOUT_H)
            accZ = read_raw_data(ACCEL_ZOUT_H)

            #print(accX,accY,accZ)
            #print(math.sqrt((accY**2)+(accZ**2)))
            if (RestrictPitch):
                roll = math.atan2(accY,accZ) * radToDeg
                pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
            else:
                roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
                pitch = math.atan2(-accX,accZ) * radToDeg
            print(roll)
            kalmanX.setAngle(roll)
            kalmanY.setAngle(pitch)
            gyroXAngle = roll;
            gyroYAngle = pitch;
            compAngleX = roll;
            compAngleY = pitch;
```

```
timer = time.time()
flag = 0
if(flag >100):
    #Problem with the connection
    print("There is a problem with the connection")
    flag=0
    continue
try:
    #Read Accelerometer raw value
    accX = read_raw_data(ACCEL_XOUT_H)
    accY = read_raw_data(ACCEL_YOUT_H)
    accZ = read_raw_data(ACCEL_ZOUT_H)

    #Read Gyroscope raw value
    gyroX = read_raw_data(GYRO_XOUT_H)
    gyroY = read_raw_data(GYRO_YOUT_H)
    gyroZ = read_raw_data(GYRO_ZOUT_H)

    dt = time.time() - timer
    timer = time.time()

    if (RestrictPitch):
        roll = math.atan2(accY,accZ) * radToDeg
        pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
    else:
        roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
        pitch = math.atan2(-accX,accZ) * radToDeg

    gyroXRate = gyroX/131
    gyroYRate = gyroY/131

    if (RestrictPitch):

        if((roll < -90 and kalAngleX >90) or (roll > 90 and kalAngleX < -90)):
            kalmanX.setAngle(roll)
            complAngleX = roll
            kalAngleX   = roll
            gyroXAngle  = roll
        else:
            kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)

        if(abs(kalAngleX)>90):
            gyroYRate  = -gyroYRate
            kalAngleY  = kalmanY.getAngle(pitch,gyroYRate,dt)
    else:
```

```
            if((pitch < -90 and kalAngleY >90) or (pitch > 90 and kalAngleY < -90)):
               kalmanY.setAngle(pitch)
               complAngleY = pitch
               kalAngleY   = pitch
               gyroYAngle  = pitch
            else:
               kalAngleY = kalmanY.getAngle(pitch,gyroYRate,dt)

            if(abs(kalAngleY)>90):
               gyroXRate  = -gyroXRate
               kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)

         #angle = (rate of change of angle) * change in time
         gyroXAngle = gyroXRate * dt
         gyroYAngle = gyroYAngle * dt

         #compAngle = constant * (old_compAngle + angle_obtained_from_gyro) + constant *
angle_obtained from accelerometer
         compAngleX = 0.93 * (compAngleX + gyroXRate * dt) + 0.07 * roll
         compAngleY = 0.93 * (compAngleY + gyroYRate * dt) + 0.07 * pitch

         if ((gyroXAngle < -180) or (gyroXAngle > 180)):
            gyroXAngle = kalAngleX
         if ((gyroYAngle < -180) or (gyroYAngle > 180)):
            gyroYAngle = kalAngleY

         print("Angle X 1: " + str(kalAngleX)+"   " +"Angle Y: " + str(kalAngleY))
         a.append(str(kalAngleX))
         #print(str(roll)+"  "+str(gyroXAngle)+"  "+str(compAngleX)+"  "+str(kalAngleX)+"
"+str(pitch)+"  "+str(gyroYAngle)+"  "+str(compAngleY)+"  "+str(kalAngleY))
         #time.sleep(0.005)
         #a.append(str(pitch))
         #a.append(str(compAngleX))
         GPIO.output(29,GPIO.HIGH)
      except Exception as exc:
         flag += 1
         #time.sleep(1)
#SENSOR2

      GPIO.output(31,GPIO.LOW)
      MPU_Init()

      #time.sleep(1)
      #Read Accelerometer raw value
      accX = read_raw_data(ACCEL_XOUT_H)
```

```
accY = read_raw_data(ACCEL_YOUT_H)
accZ = read_raw_data(ACCEL_ZOUT_H)

#print(accX,accY,accZ)
#print(math.sqrt((accY**2)+(accZ**2)))
if (RestrictPitch):
   roll = math.atan2(accY,accZ) * radToDeg
   pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
else:
   roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
   pitch = math.atan2(-accX,accZ) * radToDeg
print(roll)
kalmanX.setAngle(roll)
kalmanY.setAngle(pitch)
gyroXAngle = roll;
gyroYAngle = pitch;
compAngleX = roll;
compAngleY = pitch;

timer = time.time()
flag = 0


if(flag >100): #Problem with the connection
   print("There is a problem with the connection")
   flag=0
   continue
try:
   #Read Accelerometer raw value
   accX = read_raw_data(ACCEL_XOUT_H)
   accY = read_raw_data(ACCEL_YOUT_H)
   accZ = read_raw_data(ACCEL_ZOUT_H)

   #Read Gyroscope raw value
   gyroX = read_raw_data(GYRO_XOUT_H)
   gyroY = read_raw_data(GYRO_YOUT_H)
   gyroZ = read_raw_data(GYRO_ZOUT_H)

   dt = time.time() - timer
   timer = time.time()

   if (RestrictPitch):
      roll = math.atan2(accY,accZ) * radToDeg
      pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
   else:
      roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
```

```
            pitch = math.atan2(-accX,accZ) * radToDeg

        gyroXRate = gyroX/131
        gyroYRate = gyroY/131

        if (RestrictPitch):

            if((roll < -90 and kalAngleX >90) or (roll > 90 and kalAngleX < -90)):
                kalmanX.setAngle(roll)
                complAngleX = roll
                kalAngleX   = roll
                gyroXAngle  = roll
            else:
                kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)

            if(abs(kalAngleX)>90):
                gyroYRate  = -gyroYRate
                kalAngleY  = kalmanY.getAngle(pitch,gyroYRate,dt)
        else:

            if((pitch < -90 and kalAngleY >90) or (pitch > 90 and kalAngleY < -90)):
                kalmanY.setAngle(pitch)
                complAngleY = pitch
                kalAngleY   = pitch
                gyroYAngle  = pitch
            else:
                kalAngleY = kalmanY.getAngle(pitch,gyroYRate,dt)

            if(abs(kalAngleY)>90):
                gyroXRate  = -gyroXRate
                kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)

        #angle = (rate of change of angle) * change in time
        gyroXAngle = gyroXRate * dt
        gyroYAngle = gyroYAngle * dt

        #compAngle = constant * (old_compAngle + angle_obtained_from_gyro) + constant *
angle_obtained from accelerometer
        compAngleX = 0.93 * (compAngleX + gyroXRate * dt) + 0.07 * roll
        compAngleY = 0.93 * (compAngleY + gyroYRate * dt) + 0.07 * pitch

        if ((gyroXAngle < -180) or (gyroXAngle > 180)):
            gyroXAngle = kalAngleX
        if ((gyroYAngle < -180) or (gyroYAngle > 180)):
            gyroYAngle = kalAngleY
```

```
            print("Angle X 2: " + str(kalAngleX)+"   " +"Angle Y: " + str(kalAngleY))
            a.append(str(kalAngleX))
            GPIO.output(31,GPIO.HIGH)
        except Exception as exc:
            flag += 1
        #time.sleep(1)


#SENSOR3
        GPIO.output(33,GPIO.LOW)
        MPU_Init()

        #time.sleep(1)
        #Read Accelerometer raw value
        accX = read_raw_data(ACCEL_XOUT_H)
        accY = read_raw_data(ACCEL_YOUT_H)
        accZ = read_raw_data(ACCEL_ZOUT_H)

        #print(accX,accY,accZ)
        #print(math.sqrt((accY**2)+(accZ**2)))
        if (RestrictPitch):
            roll = math.atan2(accY,accZ) * radToDeg
            pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
        else:
            roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
            pitch = math.atan2(-accX,accZ) * radToDeg
        print(roll)
        kalmanX.setAngle(roll)
        kalmanY.setAngle(pitch)
        gyroXAngle = roll;
        gyroYAngle = pitch;
        compAngleX = roll;
        compAngleY = pitch;

        timer = time.time()
        flag = 0
        if(flag >100):
            #Problem with the connection
            print("There is a problem with the connection")
            flag=0
            continue
        try:
            #Read Accelerometer raw value
            accX = read_raw_data(ACCEL_XOUT_H)
            accY = read_raw_data(ACCEL_YOUT_H)
            accZ = read_raw_data(ACCEL_ZOUT_H)
```

```
#Read Gyroscope raw value
gyroX = read_raw_data(GYRO_XOUT_H)
gyroY = read_raw_data(GYRO_YOUT_H)
gyroZ = read_raw_data(GYRO_ZOUT_H)

dt = time.time() - timer
timer = time.time()

if (RestrictPitch):
   roll = math.atan2(accY,accZ) * radToDeg
   pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
else:
   roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
   pitch = math.atan2(-accX,accZ) * radToDeg

gyroXRate = gyroX/131
gyroYRate = gyroY/131

if (RestrictPitch):

   if((roll < -90 and kalAngleX >90) or (roll > 90 and kalAngleX < -90)):
      kalmanX.setAngle(roll)
      complAngleX = roll
      kalAngleX   = roll
      gyroXAngle  = roll
   else:
      kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)

   if(abs(kalAngleX)>90):
      gyroYRate  = -gyroYRate
      kalAngleY  = kalmanY.getAngle(pitch,gyroYRate,dt)
else:

   if((pitch < -90 and kalAngleY >90) or (pitch > 90 and kalAngleY < -90)):
      kalmanY.setAngle(pitch)
      complAngleY = pitch
      kalAngleY   = pitch
      gyroYAngle  = pitch
   else:
      kalAngleY = kalmanY.getAngle(pitch,gyroYRate,dt)

   if(abs(kalAngleY)>90):
      gyroXRate  = -gyroXRate
      kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)
```

```
            #angle = (rate of change of angle) * change in time
            gyroXAngle = gyroXRate * dt
            gyroYAngle = gyroYAngle * dt

            #compAngle = constant * (old_compAngle + angle_obtained_from_gyro) + constant *
angle_obtained from accelerometer
            compAngleX = 0.93 * (compAngleX + gyroXRate * dt) + 0.07 * roll
            compAngleY = 0.93 * (compAngleY + gyroYRate * dt) + 0.07 * pitch

            if ((gyroXAngle < -180) or (gyroXAngle > 180)):
               gyroXAngle = kalAngleX
            if ((gyroYAngle < -180) or (gyroYAngle > 180)):
               gyroYAngle = kalAngleY

            print("Angle X 3: " + str(kalAngleX)+"    " +"Angle Y: " + str(kalAngleY))
            a.append(str(kalAngleX))
            GPIO.output(33,GPIO.HIGH)
         except Exception as exc:
            flag += 1
#SENSOR4

         GPIO.output(35,GPIO.LOW)
         MPU_Init()

         #time.sleep(1)
         #Read Accelerometer raw value
         accX = read_raw_data(ACCEL_XOUT_H)
         accY = read_raw_data(ACCEL_YOUT_H)
         accZ = read_raw_data(ACCEL_ZOUT_H)

         #print(accX,accY,accZ)
         #print(math.sqrt((accY**2)+(accZ**2)))
         if (RestrictPitch):
            roll = math.atan2(accY,accZ) * radToDeg
            pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
         else:
            roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
            pitch = math.atan2(-accX,accZ) * radToDeg
         print(roll)
         kalmanX.setAngle(roll)
         kalmanY.setAngle(pitch)
         gyroXAngle = roll;
         gyroYAngle = pitch;
         compAngleX = roll;
         compAngleY = pitch;
```

```
timer = time.time()
flag = 0
if(flag >100):
    #Problem with the connection
    print("There is a problem with the connection")
    flag=0
    continue
try:
    #Read Accelerometer raw value
    accX = read_raw_data(ACCEL_XOUT_H)
    accY = read_raw_data(ACCEL_YOUT_H)
    accZ = read_raw_data(ACCEL_ZOUT_H)

    #Read Gyroscope raw value
    gyroX = read_raw_data(GYRO_XOUT_H)
    gyroY = read_raw_data(GYRO_YOUT_H)
    gyroZ = read_raw_data(GYRO_ZOUT_H)

    dt = time.time() - timer
    timer = time.time()

    if (RestrictPitch):
        roll = math.atan2(accY,accZ) * radToDeg
        pitch = math.atan(-accX/math.sqrt((accY**2)+(accZ**2))) * radToDeg
    else:
        roll = math.atan(accY/math.sqrt((accX**2)+(accZ**2))) * radToDeg
        pitch = math.atan2(-accX,accZ) * radToDeg

    gyroXRate = gyroX/131
    gyroYRate = gyroY/131

    if (RestrictPitch):

        if((roll < -90 and kalAngleX >90) or (roll > 90 and kalAngleX < -90)):
            kalmanX.setAngle(roll)
            complAngleX = roll
            kalAngleX   = roll
            gyroXAngle  = roll
        else:
            kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)

        if(abs(kalAngleX)>90):
            gyroYRate  = -gyroYRate
            kalAngleY  = kalmanY.getAngle(pitch,gyroYRate,dt)
    else:
```

```
            if((pitch < -90 and kalAngleY >90) or (pitch > 90 and kalAngleY < -90)):
               kalmanY.setAngle(pitch)
               complAngleY = pitch
               kalAngleY   = pitch
               gyroYAngle  = pitch
            else:
               kalAngleY = kalmanY.getAngle(pitch,gyroYRate,dt)

            if(abs(kalAngleY)>90):
               gyroXRate  = -gyroXRate
               kalAngleX = kalmanX.getAngle(roll,gyroXRate,dt)

         #angle = (rate of change of angle) * change in time
         gyroXAngle = gyroXRate * dt
         gyroYAngle = gyroYAngle * dt

         #compAngle = constant * (old_compAngle + angle_obtained_from_gyro) + constant *
angle_obtained from accelerometer
         compAngleX = 0.93 * (compAngleX + gyroXRate * dt) + 0.07 * roll
         compAngleY = 0.93 * (compAngleY + gyroYRate * dt) + 0.07 * pitch

         if ((gyroXAngle < -180) or (gyroXAngle > 180)):
            gyroXAngle = kalAngleX
         if ((gyroYAngle < -180) or (gyroYAngle > 180)):
            gyroYAngle = kalAngleY

         print("Angle X 4: " + str(kalAngleX)+"   " +"Angle Y: " + str(kalAngleY))
         a.append(str(kalAngleX))
         GPIO.output(35,GPIO.HIGH)
         with open('pitch.ods','a') as f:
            writer=csv.writer(f)
            writer.writerow(a)
            a=[]
      except Exception as exc:
         flag += 1
   except (ZeroDivisionError,IOError) as e:
         print("program faced an interruption")
```