Zachary Murtishi
ELE 547

# Bluetooth-enabled Smart Pedometer System

## ELE 547 Final Project Report

### What is a pedometer?

A pedometer is a device used for tracking a user's step count. They are worn on the body and were some of the first wearable electronic devices available to consumers. Pedometers are often used in order to track exercise progress with quantitative data, as a motivational aid for some users, and as research tools for those conducting public health and exercise science studies. They may be a discrete, standalone device or may be integrated into a smart device or fitness tracker. Figure 1 shows the diversity of modern pedometer models. Today, electronic-based pedometers are integrated into many common devices, such as smartphones and smart watches.


*Figure 1: Example of three types of commonly-available pedometers, ranging from basic step counters to advanced fitness trackers*

*Image credit:Runner's World https://www.runnersworld.com/uk/gear/tech/a28335625/best-pedometers/*

Early pedometers were mechanical devices that relied on the rhythmic motion of the user's gait to actuate a mechanism to count steps; for example, many early devices used a mechanical pendulum to increment a counter. This type of pedometer has largely disappeared in favor of models making use of solid-state technology to detect physical activity. The use of integrated circuit-based MEMS (micro electro-mechanical system) sensors is a constant in virtually all modern pedometers, along with software to implement step-counting algorithms using data polled from these sensors.

There are many different implementations and models of pedometers available on the market today. They can range from basic, low-cost devices with a single sensor and simple algorithm to "smart" devices with multiple sensors and advanced algorithms to make sense of a user's motion patterns. The latter devices

are most often integrated into “smart” systems, while basic devices are usually available as standalone units. Depending on their design, pedometers may be worn at the waist, on the wrist, or even left in the pocket. [1]

## Concept and Inspiration

It is generally the case that modern pedometers are “smart” devices with advanced sensor suites and complex software to determine not just step count, but several other activity metrics such as the user’s distance traveled and heart rate. Due to the fact that pedometer algorithms are often proprietary information, not much is known about their exact implementation. What is known is that these algorithms have been designed in a way to be less susceptible to the common flaws of basic pedometer implementations, such as detecting incidental motion (such as shuffling the feet) as legitimate steps. These algorithms have been developed by analyzing step data to make sense of patterns and observations observed for different types of gait and walking speed.

As of today, algorithms developed using heuristic processes are commonly used for step counting. Despite its use in many other modern electronic systems, machine learning is not a component of most commercially-available “smart” pedometers. This can be attributed to the high computational complexity of artificial neural networks and expensive hardware required for their physical implementation. Typically, pedometers tend to be lightweight embedded systems designed for power efficiency and lack the necessary computing power to run a neural network. On the other hand, academic researchers and independent makers have explored machine learning’s potential for application in step counting algorithms with varying levels of success.

However, many of these systems are not intended for real-time applications. These machine learning algorithms are not run in real-time and instead run on step data that is pre-collected. An example of this is Basil Lin’s sliding-window neural network; while effective, it makes use of a computationally-expensive algorithm that is apparently impractical to run in real time. [2] Many existing evaluations into integration of machine learning into step counting systems have this limitation due to the inherent high computational cost of neural networks.

One example of an effective real-time device is a project by independent researcher, Viktor Malyi [3]. The goal of his project was to develop a real-time motion classifier using a TensorFlow-trained neural network to run on an Apple iPhone and indicate if the user was walking or running based on activity data collected by multi-axis data from the iPhone’s IMU. In the end, Malyi’s efforts resulted in a classifier capable of classifying motion with over 99% accuracy in real time. While technically not a pedometer due to the fact it was incapable of determining step count, it was an example of a real-time motion classification using a neural network. The device used by Malyi is depicted in Figure 2.

*Figure 2: Viktor Malyi's iPhone-based implementation of a neural network-powered motion classifier*

A common complaint about commercial "smart" pedometers (Fitbit, iPhone, Apple Watch, etc.) is their apparent inaccuracy when used in non-ideal conditions. In ideal use, these "smart" pedometers tend to be reasonably accurate, though their algorithms may be fooled by incidental motion registering as legitimate walking motion. This is one potential area for machine learning to excel; as evidenced by the Malyi project, it has potential to be a valuable tool for classifying motion.

The intent of this project was to explore the application of an artificial neural network classifier as a component of a highly-accurate real-time step counting algorithm running on an embedded pedometer system. The resulting algorithm uses machine learning in conjunction with a traditional step counting algorithm to perform real-time step counting based on data polled from a 3-axis inertial measurement unit. Bluetooth capabilities were added to the system in order to mimic the functionality of commercial pedometers and allow for offloading of computationally-expensive neural networks to a more powerful client host. However, as this is a proof-of-concept system, it will be unlike Malyi's system in that it is not intended to be used while a user is running, nor will it be tested to verify if this capability is present.

The resulting system is to be baselined against a commonly-available "smart" pedometer to determine the success of the project's efforts. Due to its ubiquity, the iPhone's embedded pedometer system was selected to be the unit compared to the resulting pedometer system to measure success. If the resulting system has an error within +/- 10% of that of the iPhone pedometer, then this project can be considered successful. In addition, the resulting pedometer shall aim for a step counting error of no more than +/- 25% from true measured steps, if the original goal cannot be achieved.

## High-Level System Architecture and Components

The complete system can be divided into two components: the pedometer and the client. Figure 3 shows a simplified block diagram of the system architecture.

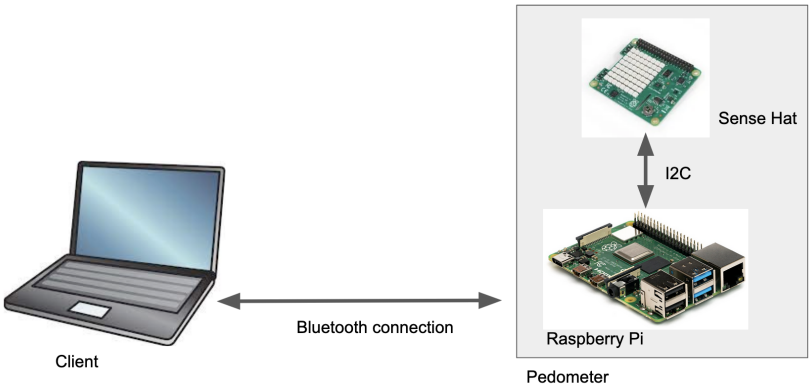


*Figure 3: Simplified system block diagram*

The pedometer is a wearable assembly equipped with a Raspberry Pi 4 single-board computer and the Pi Sense Hat's 3-axis inertial measurement unit (IMU) with accelerometer and gyroscope devices. The client is a laptop computer used to remotely connect to the pedometer. The pedometer is primarily responsible for collection of activity data through the Sense Hat, though it does have the ability to perform its own motion classification. The client is primarily responsible for processing the activity data through its own motion classifier, as well as allowing a graphical user interface to the pedometer. To this end, each component runs a set of Python software scripts developed to implement their part of the system.

The two components are linked through a Bluetooth connection using RFCOMM sockets. This connection is purely optional, as the client is not required for base system functionality. The pedometer is able to operate as a headless system to perform step counting and motion classification without a client connection, albeit with lesser performance and without a user interface.

As a headless unit, motion data will be polled from the Sense Hat and sent to a neural network classifier. When a client is connected, the pedometer will instead route all IMU data through the Bluetooth connection to a classifier running on the client system. This data is input to a neural network and zero-crossing detection algorithm to determine step count. The new step count is then sent back to the pedometer from the client so that they remain synchronized. A reset command may also be sent through this Bluetooth connection to synchronize a zero step count between the client and pedometer. The ability of the client to perform classification is largely to take advantage of the client's superior Intel processor to perform faster classification than the embedded ARM processor.

It is known that many commercially-available pedometers make use of an accelerometer to perform step counting. In contrast to these commercially-available pedometers, this pedometer makes use of only gyroscope data to perform step counting. This is due to the gyroscope's fixation to the leg providing accurate information about leg movement, particularly the angle and rotation of the leg during movement.

Figure 4 shows the pedometer system's hardware components. The pedometer's hardware components are rather simple, using only a Raspberry Pi 4B single-board computer and Pi Sense Hat sensor suite for its core system. While any single-board computer platform and IMU combination can be used in theory, the

software developed for this system is optimized for the Pi ecosystem. Additional hardware includes a leg pouch and USB power bank to allow it to be used as a wearable device.


*Figure 4: The pedometer system's hardware, with Sense Hat, Raspberry Pi, and battery visible in pouch*

In addition to the hardware requirements, there are software dependencies that must be satisfied for implementation of the pedometer system. These include the Raspberry Pi OS (Debian Linux for the Pi system), Python 3.9 (including built-in libraries such as Time, Pickle, Tkinter, Math, and Socket), the Pybluez Bluetooth API, TensorFlow, Keras, and the Sense Hat API. Note that some modifications are required for the Sense Hat API to be compatible with the pedometer system's software.

The client system's hardware requirements are more generic: any laptop computer with a Bluetooth interface and graphical user interface is suitable for use as the client. However, the client software has been developed to be executed on Ubuntu-based Linux hosts and so a Ubuntu-based operating system is a soft requirement. In theory, any Linux distribution or Windows version should be compatible with the client software, but it has only been tested with Ubuntu 22.04 and Linux Mint 21. Note that the client has software requirements identical to that of the pedometer, except that the Sense Hat API is not required. A Lenovo ThinkPad X270 running Linux Mint 21 as its operating system was used for development and testing.

The client was equipped with software to allow the developer the ability to remotely login to the headless pedometer for debugging and development purposes; SSH connections and a VNC interface were the two

types of remote connections used, enabled by the built-in Linux SSH command and RealVNC software application respectively.

## Design and Implementation

### Physical Implementation

The first action to be completed for the project was to determine how exactly the pedometer was to be mounted on a user. Most commercially-available pedometers are either wrist-mounted or attached at the waist or belt, relying on the user's incidental motions while walking to count steps; for example, incidental wrist movements while walking are used by wrist-mounted pedometers for step counting. It is uncommon for pedometers to be leg-mounted.

While they may be uncomfortable to some users, the leg would be in theory an optimal place to mount a pedometer, as it can directly measure the movements of the wearer's leg and translate the motion data into steps, instead of relying on incidental motion of other body parts and assuming those motions to correlate with steps. As the leg could provide more useful information to the Sense Hat's gyroscope and accelerometer, it was decided to mount the pedometer on the leg. Figure 5 shows the pedometer worn as designed.


*Figure 5: The pedometer system as worn by a user*

To make the pedometer a wearable device, two additional requirements needed to be met: a battery to power the Raspberry Pi while in use and a pouch to place both the Raspberry Pi and battery into. A 10000mAh Anker-brand power bank was selected as the battery solution. A sport band with two compartments was selected to mount the battery and Raspberry Pi; a Dibeister-brand example was purchased and found to be able to accommodate both devices while mounted to the leg.

### Motion Classifier I: Concept

The motion classifier is intended to be the part of the step counting algorithm that determines the current activity type of the user. It is implemented through the use of equally-weighted ensembled neural networks taking a 1-second window of gyroscope X and Z-axis data as their inputs. The output of these neural networks is a softmax output containing three possibilities: the user is walking, the user is idle, or the user is shaking the pedometer. This output is in the form of a tuple, which is read by the application to determine activity type. Note that the GUI will report a “shaking” operation as “idle” operation for simplicity, as it can be assumed that a shaking movement is not intentional and rather a byproduct of idle behavior. The motion classifier operates in real time.

**Motion Classifier II: Training**

Development of the pedometer software began with development of the motion classifier used to determine the user’s current activity state. The first action taken during this stage was to collect motion data in order to train a neural network to perform motion classification. This was accomplished by creating a training script to periodically poll the Sense Hat’s gyroscope and accelerometer data and write it to a timestamped .csv (comma separated value) file. This data was then separated into its different components and saved to separate .csv files using MATLAB to be used for training a model.

However, an issue arose early on in training the model. The Python dictionary operations required to parse the Sense Hat IMU data were rather computationally-costly and frustrating to work with; in order to simplify the algorithms used for parsing the data, the Sense Hat API was modified to return the raw IMU data values as a list rather than a dictionary; specifically, the _get_raw_data method of the SenseHat class was rewritten to return the data as a list in a format such that a return list was formatted [x, y, z].

The gyroscope was selected to be the primary device for step counting due to its rather accurate response to leg movement. Due to the pedometer’s location on the leg, the gyroscope is able to provide accurate information about the leg’s rotation and angle during a step motion. The gyroscope was selected due to heuristic analysis of IMU data recorded during steps taken; while the accelerometer showed similar trends, they were not as informative about the user’s motions as the gyroscope; that is, a high-amplitude, clear zero-crossing pattern representing steps was evident when observing gyroscope data. In contrast to the gyroscope data, the accelerometer data was noisier and less consistent. Data from the accelerometer and gyroscope Z-axis recorded during a short walk and pause is depicted in Figures 6 and 7, respectively.

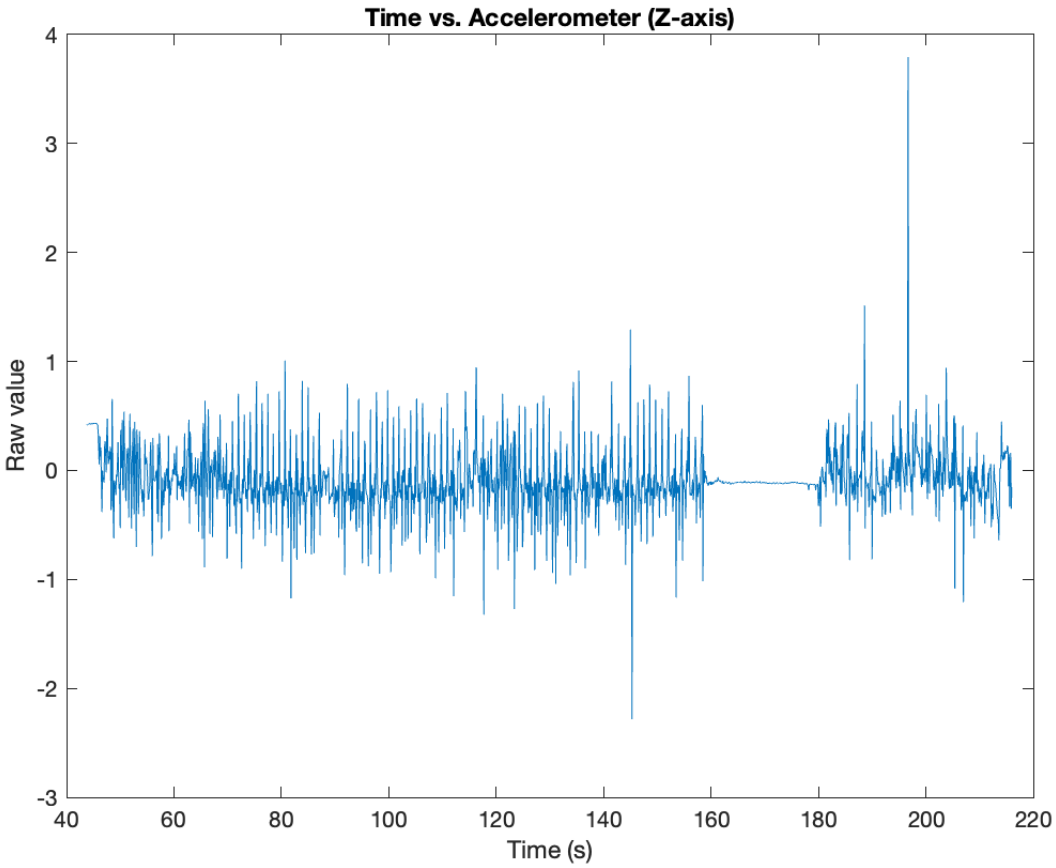


*Figure 6: Accelerometer Z-axis data recorded during periods of walk and rest*

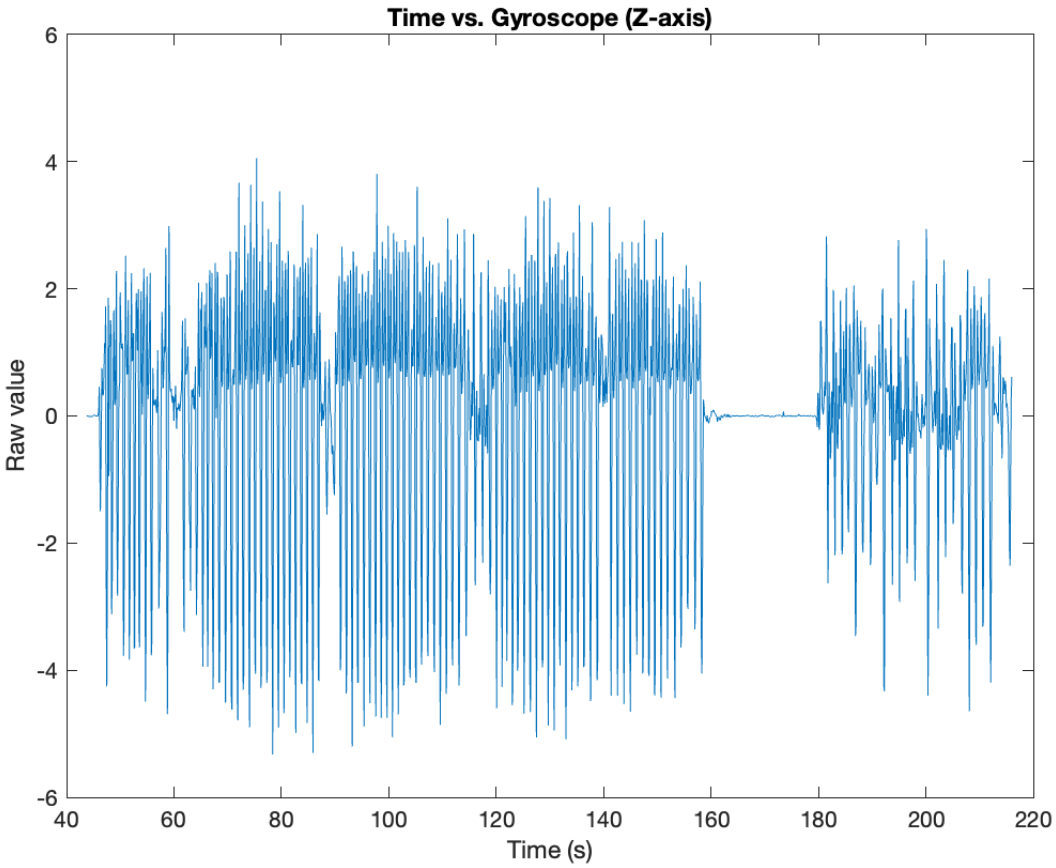


*Figure 7: Gyroscope Z-axis data recorded during periods of walk and rest*

In order to select a polling interval for the IMU, an evaluation procedure was performed. As part of the procedure, a program was created to record the Sense Hat gyroscope's Z axis. The developer would then walk two steps and record this data to a csv file. This procedure was performed for three polling intervals: 50ms, 100ms, and 200ms. The resulting data was then plotted in MATLAB for visual analysis. An example of this plotted data is shown in Figure 8.

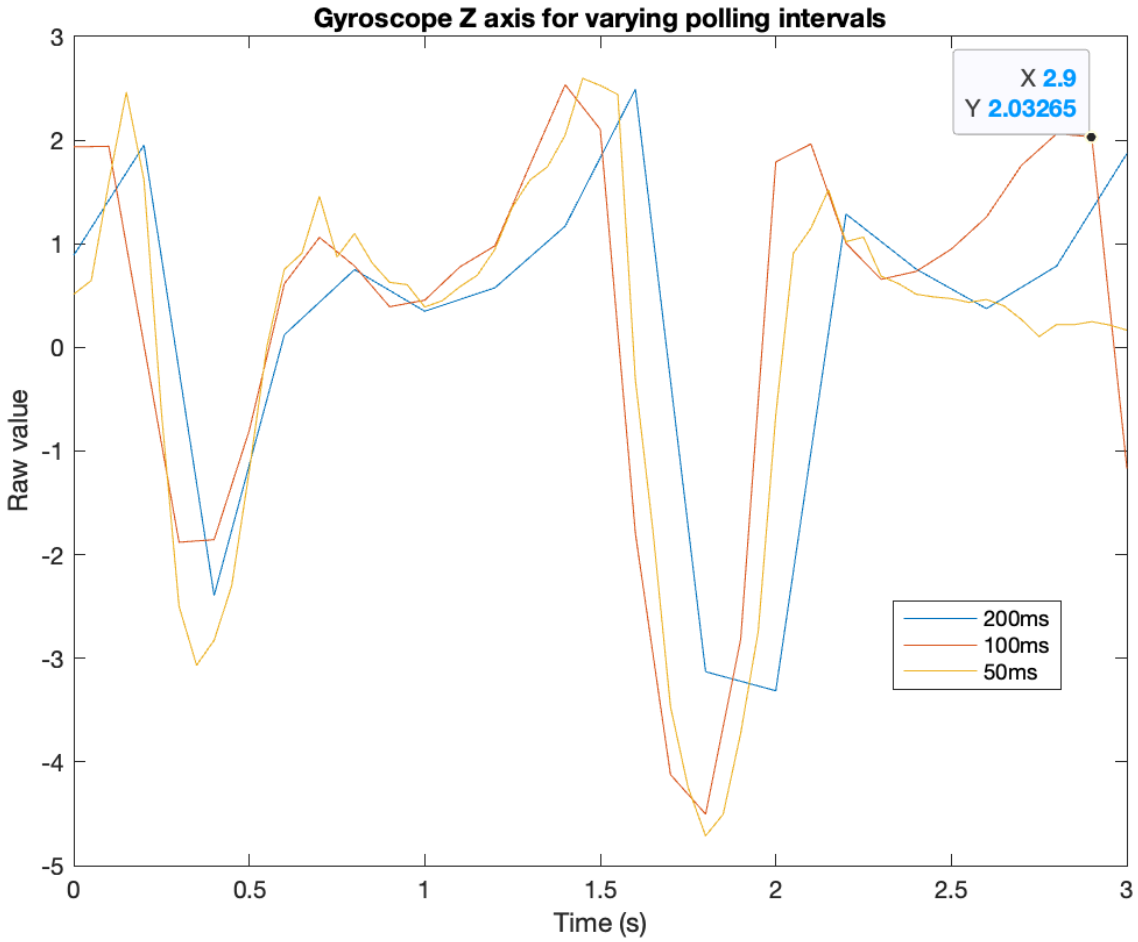


*Figure 8: Step data collected by polling the gyroscope Z-axis at three different intervals*

While 50ms and 100ms show data points that show approximately the same trends, it appears that a 200ms polling interval results in some loss of information, judging by the sharp peaks and valleys of the data. In order to minimize the number of inputs to the neural network and improve processing speed, the 100ms polling interval was selected, as it did not lose a great deal of information from 50ms and only requires half the amount of data points.

After the polling interval was selected, it came time to collect the training data to be used to train and validate the classifier model. To this end, a training script was developed in order to log all IMU data (all three axes of both the accelerometer and gyroscope) with a 100ms polling interval and an appropriate label appended to each data point i.e. idle, walking, or shaking. This script was run on the Raspberry Pi while the user performed three actions: walking, sitting idly, and then intentionally shaking the device. As a result of these efforts, a total of 91758 data points were collected for training the neural network. Roughly 10% of these data points were reserved to be used as validation data. Note that all walking data was recorded with the pedometer fixed to the right leg. Examples of the training data are shown in Figures 9, 10, 11, 12, 13, and 14.

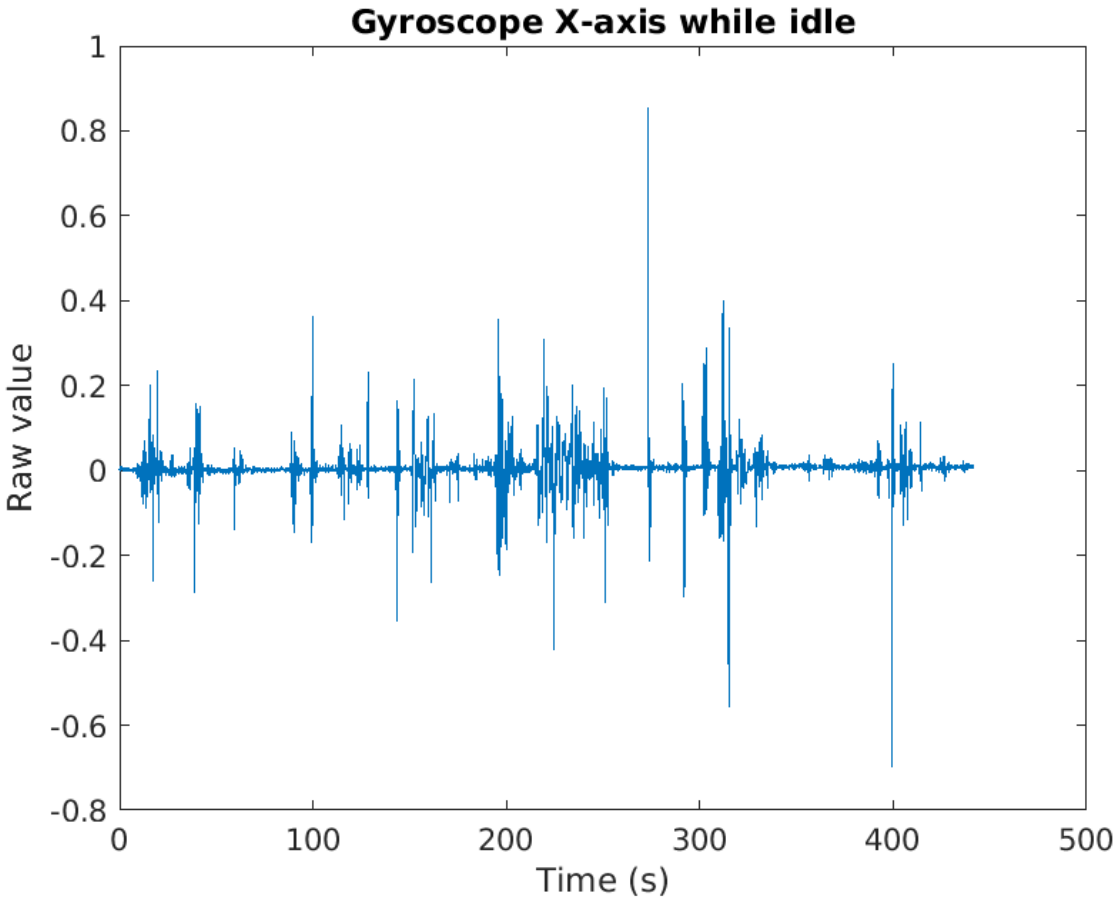


*Figure 9: Data from the X-axis of the gyroscope while a user is at idle*

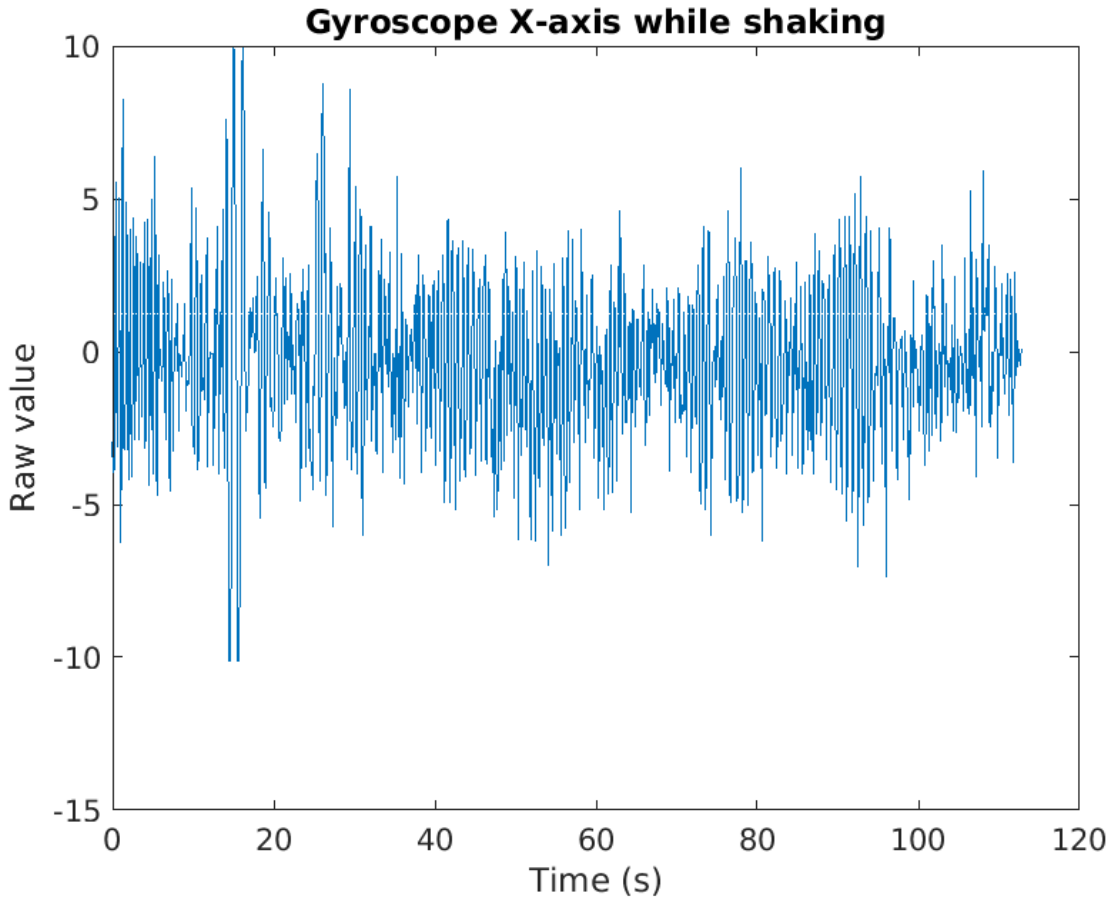


*Figure 10: Data from the X-axis of the gyroscope while shaken by a user*

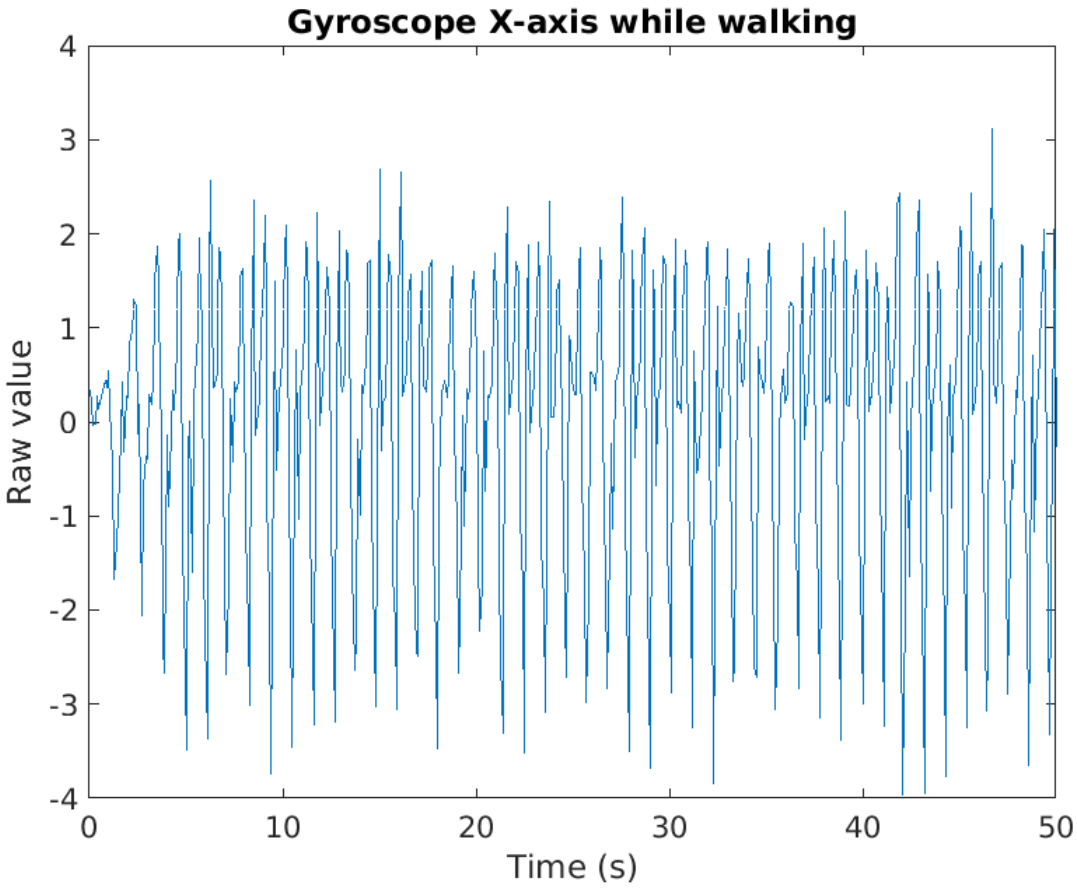


*Figure 11: Data from the X-axis of the gyroscope while a user is walking*

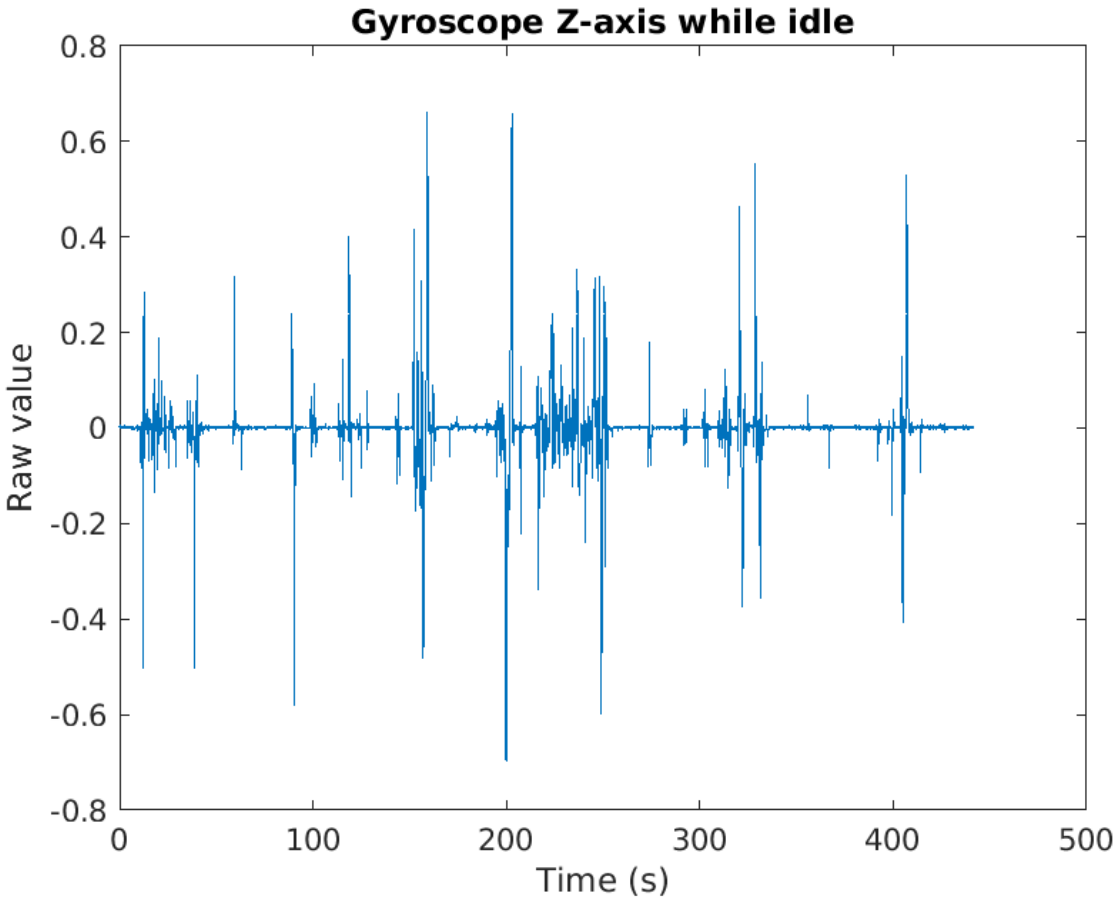


*Figure 12: Data from the Z-axis of the gyroscope while a user is idle*

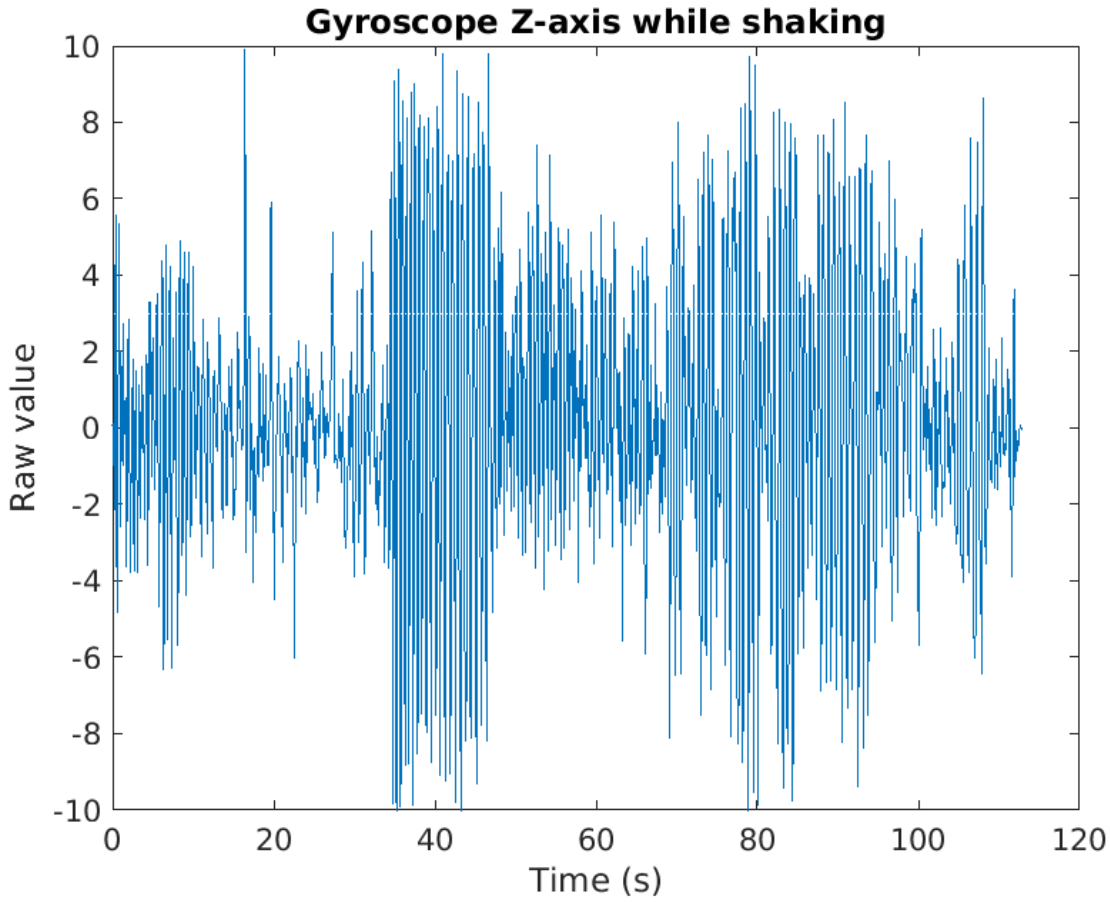


*Figure 13: Data from the Z-axis of the gyroscope while shaken by a user*

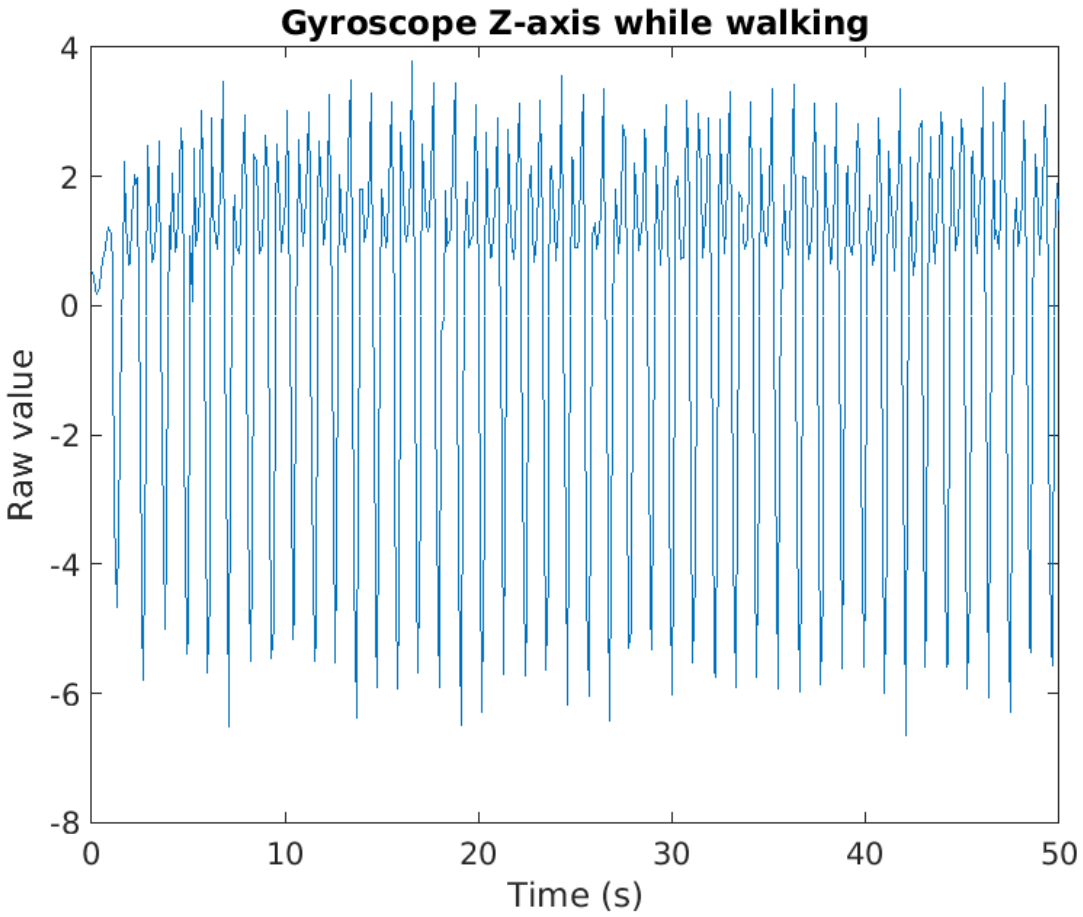


*Figure 14: Data from the Z-axis of the gyroscope while a user is walking*

Originally, the only data points collected for training included data collecting while the user was walking and idle. Early testing of the model with a classifier trained with these two data sets revealed that the resulting system was not resistant against incidental motion and that it could be easily set off by irregular, non-step-related foot motions. Introducing another classification to the neural network consisting of "shaking" helped to reduce sensitivity of the system to noise, as it classified this "shaking" noise as a separate group than walking. This data was collected by running the training script while vigorously shaking and moving the pedometer.

Multiple attempts were made to collect training data while walking, but failed due to improper development of the software script and contamination of the training data with incidental motion that could be falsely trained to be recognized as legitimate steps. The final attempt was successful, with the data polished and cleaned up using MATLAB to remove the presence of idle motion.

**Motion Classifier II: Model Design**

After collecting all necessary training data, design of the neural network architecture was next on the agenda. In order to maximize the effectiveness of the classifier, it was designed to operate on multiple

inputs. That is, the classifier was to be composed of multiple neural networks with each of their outputs connected to a single output layer with fixed weights. This was inspired by Malyi using ensembling methods to combine the outputs of multiple neural networks to make use of multiple sensor inputs in their motion classifier project.

Each sensor input for this project was intended to use the same neural network architecture. This is due to the fact that each sensor output has similar data trends in response to walking and that re-using the architecture for multiple inputs would hasten development time. A 1D convolutional neural network was selected, as it would be able to make better sense of the data trends with little supervision required.

To this end, the model was specified to have two convolutional layers. The first layer features 64 filters, with the second layer having 32. A 1D max pooling layer followed each of these convolutional layers. A dropout layer set to a frequency of 0.5 followed in order to reduce the possibility of overfitting. The data is then flattened. After this, two hidden layers with ReLu activation functions follow. These two layers feature 80 and 30 hidden units respectively. Their outputs are then fed to a softmax layer configured to perform categorical classification. This output is then brought to an output layer, where it is summed with a fixed weight with the outputs of the other neural networks processing other sensor inputs.

Originally, this model was designed with a single convolutional layer and hidden layer; however, with trial and error, it was found that two convolutional layers and two hidden layers resulted in higher accuracy with regard to the validation data. The number of hidden units per layer was set similarly, through trial and error and the variables' effects on the validation accuracy.

Per Figure, it was decided that 1 second of motion data would be sufficient to perform motion classification. There is approximately one second per zero-crossing, each of which can be considered to be steps; therefore reading 1 second of motion data would be ideal for determining if the wearer is currently walking. Therefore, the neural network was designed to accept a 1D array of 10 data points (each were polled at an interval of 100ms) as its input, or 1 second of motion data. A

10% of the training data was reserved for use as validation data. Due to errors in the development notes, it was originally reported in the presentation that the model was only around 97% accurate based on the training data. However, this was only for the accelerometer models. The gyroscope X-axis, Y-axis, and Z-axis models were all determined to be >99% accurate based on validation data. Note that the algorithm used for training is nondeterministic due to the 0.5 dropout layer.

While the accelerometer model accuracy was very good, this number could not be improved further despite efforts to do so. With limited time at hand, it was decided that this was very good accuracy for the task at hand and that no more time should be allocated to this component of the project. Figure 15 shows the Keras training output for accelerometer X-axis data.

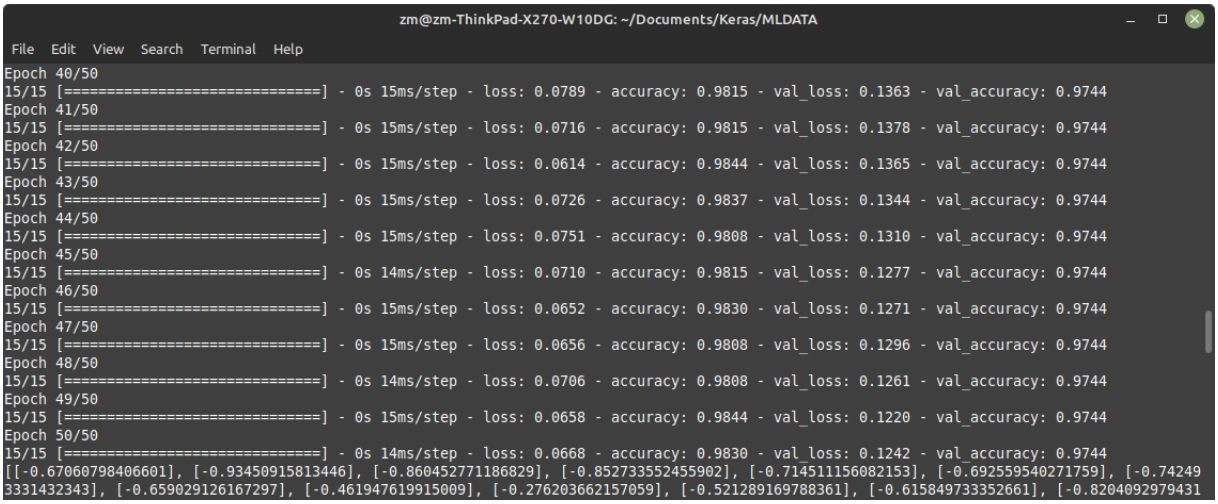


*Figure 15: An excerpt from the output of the training program. The model in training is for accelerometer X-axis data.*

It was determined that a combination of the gyroscope X axis and the gyroscope Z axis provided the most accurate results. The final configuration for the classifier uses two neural networks, one using inputs from the gyroscope Z-axis and the other using inputs from the gyroscope X-axis, with both outputs summed at a single output layer with fixed weights of 0.5. Accelerometer data was deemed to be superfluous, as gyroscope data alone was sufficient to determine activity; when added to the model, accelerometer data did not contribute to accurate classification and instead hindered the effectiveness of the model by introducing noisy data. It was further determined that a model based on both the X-axis and Z-axis of the gyroscope performed better than a single neural network using either of those inputs independently.

All neural network models were trained using Keras running on a ThinkPad X270 running Linux Mint 21 as its operating system. Neural networks were trained for all three axes of the gyroscope and accelerometer using relevant training data; however, only the gyroscope's X and Z axes were used in the final product. They were trained with 50 epochs and a batch size of 200. Interestingly, this is unlike many commercial pedometers that make use of accelerometer data rather than gyroscope data.

Early real-world testing showed promising results, with short steps registering as motion with near-perfect accuracy. However, step counting was not functioning properly when the classifier was allowed to increment the step-counter by assuming 1s of activity was always equivalent to a single step; a separate step-counting algorithm was required to add this functionality to the system.

**Step Counting Algorithm**

Originally, it was an assumption that a successful detection of walking motion would result in a single step added to the step count. However, this was not an accurate means for step counting, as it would not account for faster walking speeds or steps that occurred near the beginning or end of the window, which could register as multiple steps.

A solution to this problem was rather simple and takes a page out of the book of classic pedometer design. It is a heuristic approach that was formulated by observing graphed training data for walking motion.

Each step triggers a zero-crossing on the gyroscope's Z-axis; counting the number of steps is as simple as counting each low-to-high zero crossing. However, this algorithm has to be performed in conjunction with the motion classifier; counting zero crossings without motion detected will result in noise and incidental motion being registered as steps. These zero-crossings are visible in Figure 16, with the zero line clearly demarcated.

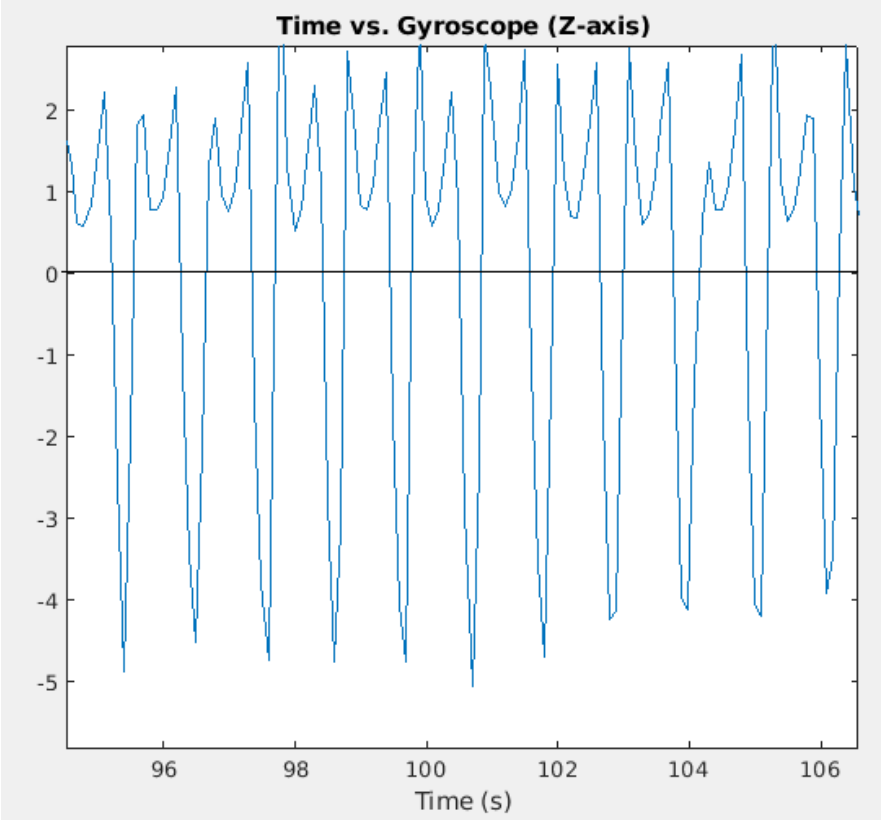


*Figure 16: Steps recorded by the Z-axis of the gyroscope. Zero is marked by a line. Zero-crossings are indicative of steps taken.*

This solution takes a two-tiered approach. First, the presence of walking motion is to be detected using the motion classifier. Using the softmax output of the final layer, the threshold for a detected walking motion should be determined by checking if the probability of walking is greater than 40% and the probability of shaking is less than 50%. If walking motion is detected by the classifier, the input waveform is input to a zero-crossing detection algorithm that counts the number of low-to-high zero crossings, which will be equal to the number of steps. Despite its relative simplicity, it has proven to be a highly-effective means of step counting.

The Z-axis of the gyroscope was selected as an input to this algorithm over the X-axis due to its more obvious zero crossings when observed heuristically; though it can be observed that counting zero-crossings on the X axis would also been a suitable means of step counting. The complete algorithm is depicted in Figure 17.

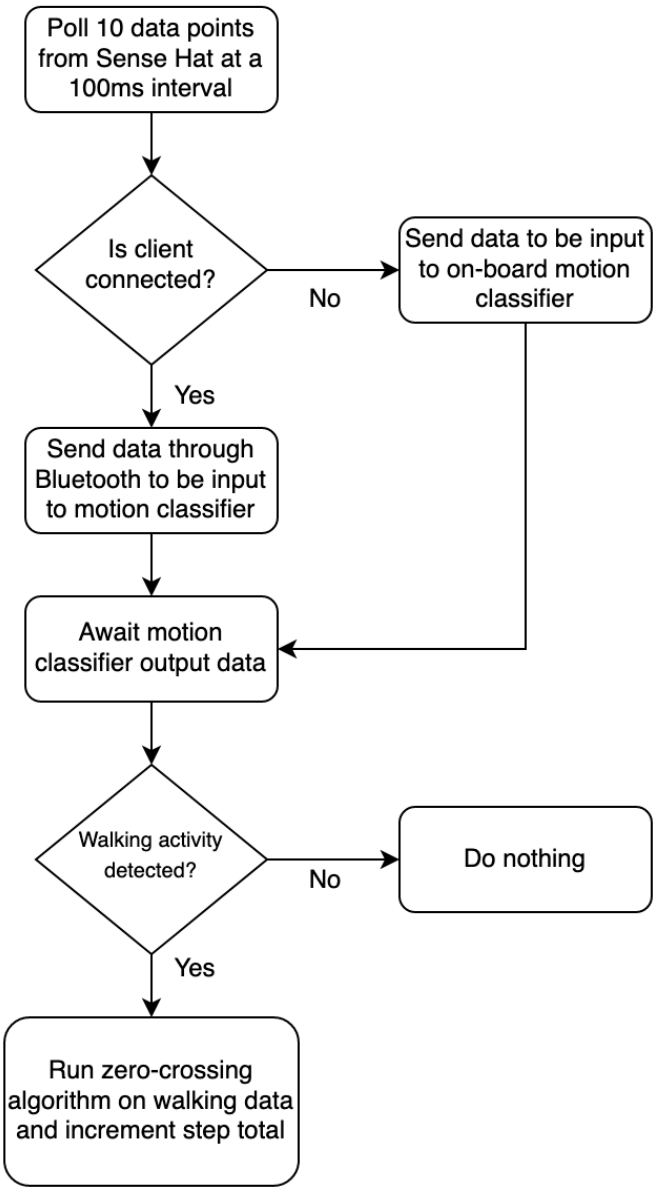


*Figure 17: Complete flowchart of step-counting algorithm implemented by system*

**Software Architecture**

After the model was trained and validated, it was time to develop the software suite to be designed around the classifier. Software systems were developed for both the pedometer and client platforms and were designed to be complimentary. Figure 18 shows a block diagram of each system's software processes and the manner in which they communicate with each other.

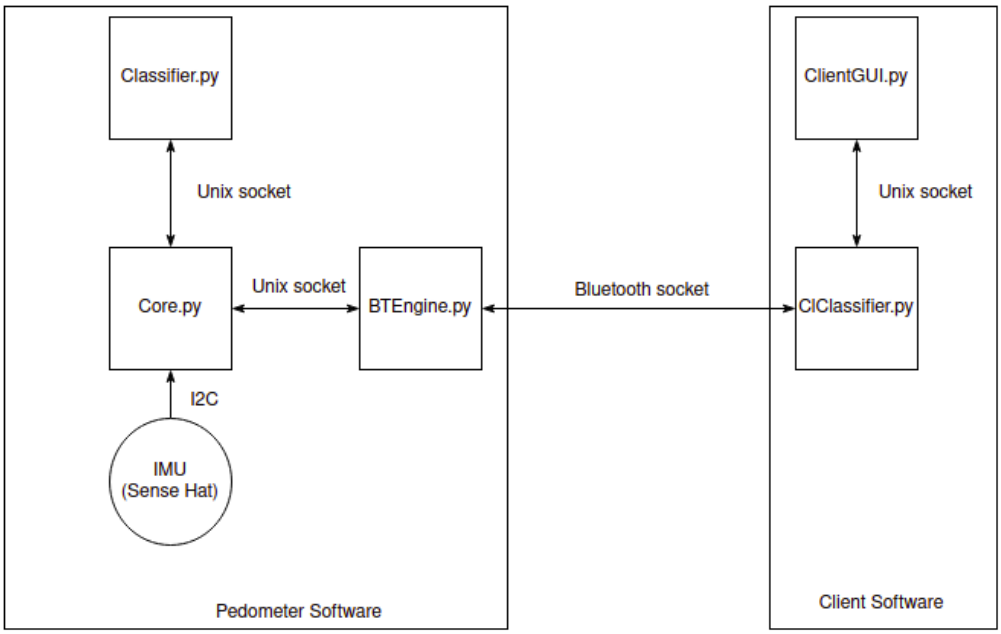


*Figure 18: Software architecture of both components of the system with connections between processes and resources indicated*

As can be seen in the above figure, the system features a large amount of I/O on the pedometer side, which translates to potential delays resulting from I/O overhead. Coupled with the high computational expense of the classifier, this presents a potential issue for performance. In order to minimize the effects of these potential delays and maximize performance, the software functions were split up into multiple discrete processes. This allows for use of multiple CPU cores and parallel processing, allowing every process to run separately despite the presence of processing or I/O delay on another process.

The entire system (both client and pedometer) was developed using Python 3.9 to be run on Linux devices. Python was selected due to its large variety of software libraries providing access to the Raspberry Pi's many hardware interfaces. The Sense Hat API is an example, providing a Python interface to the Sense Hat's sensor hardware. The TensorFlow and Keras libraries for machine learning are also native to Python and it would be advantageous to directly reference them in the code rather than requiring a wrapper to interface with their functions. Despite these advantages, Python is a relatively slow language and its extensive use presents performance challenges.

Due to the way Python is implemented in CPython, true multi-threading is not permitted by the Global Interpreter Lock. To circumvent this limitation, processes are used in lieu of threads to allow for multitasking. In this way, each process runs as its own thread and multiple CPU cores can be leveraged for executing the full system program.

Unix sockets are used to share data between processes in this architecture. Data shared includes raw sensor data and processed motion data. This data is serialized into a byte stream before transmission and reconstructed at receipt using the Pickle library built into Python. For example, the Core process on the pedometer will poll raw sensor data from the Sense Hat, serialize into a byte stream, and through a socket,

transmit it to either the Bluetooth data transceiver process or the Classifier process on the pedometer depending on whether there exists a connection to a client host. The Classifier process will then communicate its results back to the Core process through a socket which will then be reconstructed to a Python list object containing the data, or the Bluetooth socket will receive results from the client and transmit them to the Core program through its socket to be reconstructed.

**Pedometer Software**

The pedometer software was the first component of the system to begin development as it is responsible for implementing the core functionality of the system. A rough development of the client was ongoing during this time in order to test Bluetooth functionality, though serious development of the client and its GUI did not commence until the pedometer software was fully developed.

**Core**

The Core process contains the active step count and is responsible for polling IMU data from the Sense Hat. It is responsible for routing the IMU data to either the Classifier process or the Bluetooth socket, depending on the client's status. Similarly, it is responsible for retrieving the classifier's outputs from either the on-board process or client and updating the global step count. This process will always run, regardless of the presence of a client. Currently, it sends data for all three axes of the gyroscope and the X-axis of the accelerometer with a 100ms polling interval.

The Core process keeps track of all steps counted by the system. This step count is the sum of steps counted by both the pedometer and the client's classifier. They are kept track of separately so that an accurate total can be kept in the event of a disconnection event. As steps are counted and received by the Core process, they are added to a running total. This running total is used for multiple purposes, including updating the client in the event of a Bluetooth connection, saving the total to a text file to be recalled later upon closing the program, and to be saved in the event of a client-issued reset.

Both the running step total and the previous session's step total are periodically saved to a text file so that they can be retrieved by the BTEngine process to be sent to the client for synchronization. After every system reset, the new start time is written to a text file so it may be recalled. Similarly to the step count data, the start time data is recalled so that it may be sent to the client for synchronization.

A lock file is created upon a client connection; this file is deleted in the event of the client disconnecting or the program stopping. This file is used by the Core process to determine which process to send the IMU data to; its presence indicates that the data is to be sent to the Bluetooth interface, while its absence leads to the data being sent to the on-board Classifier process.

**Classifier**

The Classifier process performs motion classification. It contains the classifier and takes IMU data as an input. This data is input to the neural network to perform classification; if a successful walking classification is made, a zero-crossing detection algorithm is performed to count the number of steps in that window. While it does receive data for all three axes of the gyroscope and the X-axis of the accelerometer, only the gyroscope's X and Z-axis are factored into the classifier; i.e., their weights are set

to zero to exclude them. The Classifier process only receives information from the Core process when a client is not connected to the system; it is in an I/O blocking state when a client is connected.

**BTEngine**

The BTEngine process is responsible for Bluetooth communications and functions as the system's software interface to the Bluetooth hardware module. It is capable of both sending and receiving Bluetooth data. In its current configuration, it is set up as a server; i.e., it will wait in a blocking state for a client to connect. It is designed to send raw motion data to the client's classifier process, as well as receive step data from the classifier to be sent to the Core process. This is done through a Bluetooth RFCOMM socket opened on the pedometer side that interfaces with an equivalent socket on the client side.

Upon establishing a Bluetooth connection, BTEngine will update the client software with the current step count, as well as the previous session's step count and time elapsed since the beginning of the current session. A lock file will be created to allow the Core process to know that there is an active connection. The lock file will be deleted upon the cessation of the Bluetooth connection.

**Client Software**

The client software was not fully-developed until the client's software was fully-functional. A rough build of the client was in development concurrently with the pedometer, though with a prototype interface in the form of a very early GUI to test Bluetooth data transfer. This early build of the client did not include its own classifier and instead mirrored data sent from the pedometer over Bluetooth. This GUI is depicted by Figure 19. After completion of the pedometer, completion of the client software and GUI became a priority and were promptly completed.

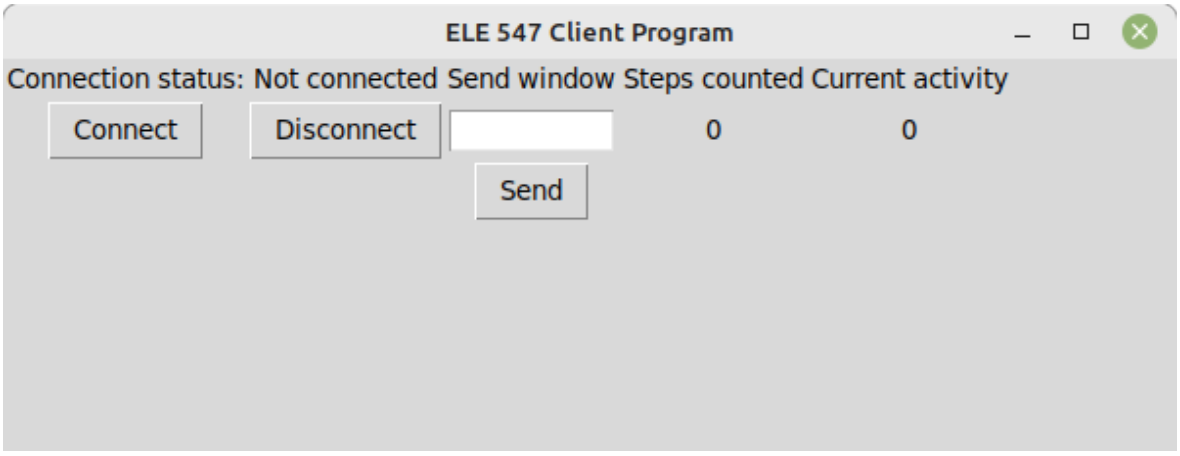


*Figure 19: Early graphical user interface used for testing Bluetooth features*

**Client Classifier**

The client software is similar to that of the pedometer, albeit without the ability to gather IMU data. It features the same motion classifier as the pedometer. The ClientClassifier process receives IMU over a Bluetooth socket opened by the client system and inputs it to the motion classifier. The algorithm performed is identical to that of the pedometer's classifier. The results are then sent back to the pedometer through the Bluetooth connection and sent to the ClientGUI process over a Unix socket.

**Graphical User Interface**

The ClientGUI process provides a user interface to the pedometer. It was developed using the Tkinter library for graphical user interfaces. It communicates with the pedometer through the ClientClassifier process using a Unix socket, which exchanges information with the pedometer through a Bluetooth connection. The GUI created by this process informs the user of the current step count, the step count of the previous session, the current activity type, and time elapsed since the beginning of the session. The user is also afforded the ability to start a new session and erase the current step count through a button. This GUI is shown in Figure 20.

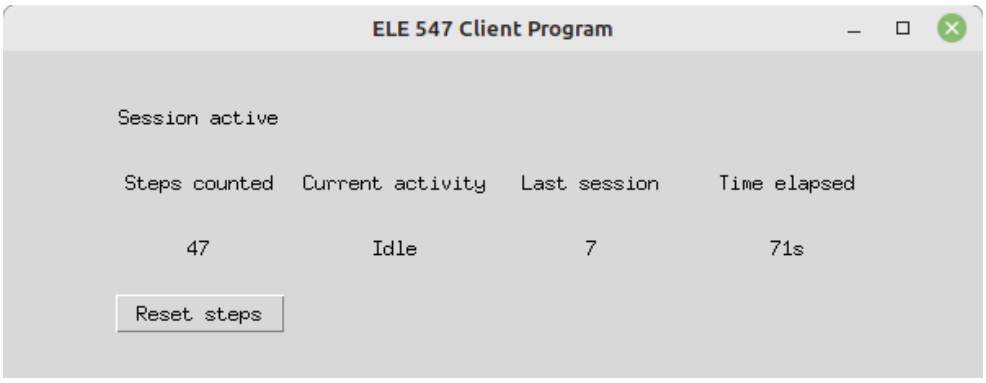


*Figure 20: The graphical user interface included in the final client software design*

Upon the client establishing a Bluetooth connection with the pedometer, all relevant data stored in the Core process is sent to the ClientGUI so that they are synchronized. This data includes the current step count, the step count of the previous session, and time elapsed since the beginning of the current session. When a new session is started by the user, the client generates a new start time and resets its own step count to zero, then issues a restart command to the Core process on the pedometer. This restart command sends a new start time and resets the current step count to zero on the Core process.

When in use, the client's current step count and current activity are derived from processing performed by the ClientClassifier process. The current activity is indicated to be either "walking" or "idle" depending on the results of the classifier. This information updates in real time and no behind-the-scenes processing is required, unlike many commercial pedometers.

**Bluetooth Communication**

Communication with the client system is performed over Bluetooth connection between the client and pedometer. Bluetooth is ideal for this use case for several reasons. One, it allows for short-ranged wireless communications without the presence of an access point or cell tower, which may not be available or within range while one is walking outdoors. The pedometer was originally envisioned as a Wi-Fi device; however, this would mean that this device would be largely useless when used outdoors or in a building without Wi-Fi. Two, it is sufficiently fast enough for data transfer such that data transmission and its related I/O overhead do not have a serious impact on system performance. Lastly, Bluetooth is a ubiquitous protocol that is present on nearly every mobile device available on the market and so it is

available on both the Raspberry Pi and client laptop; in fact, commercial pedometers make extensive use of Bluetooth technology to communicate with host platforms.

Using the Bluetooth user interface of Linux Mint and a test script to evaluate the potential of the Pybluez API, it can be determined that a reasonable average speed for Bluetooth transmission using Python sockets is around 50KB/s. This can be seen in Figure 21. The test was performed by executing a script to rapidly send 500 byte messages from the Raspberry Pi to the client machine and reading the Linux Mint Bluetooth interface to determine the effective speed calculated by the hardware module. This 500 byte message is intended to be an analog for the data structure containing step data; it is a 424 byte message and so a 500 byte message would give a worst-case scenario for transmission speed, due to the fact a longer message takes more time to send.

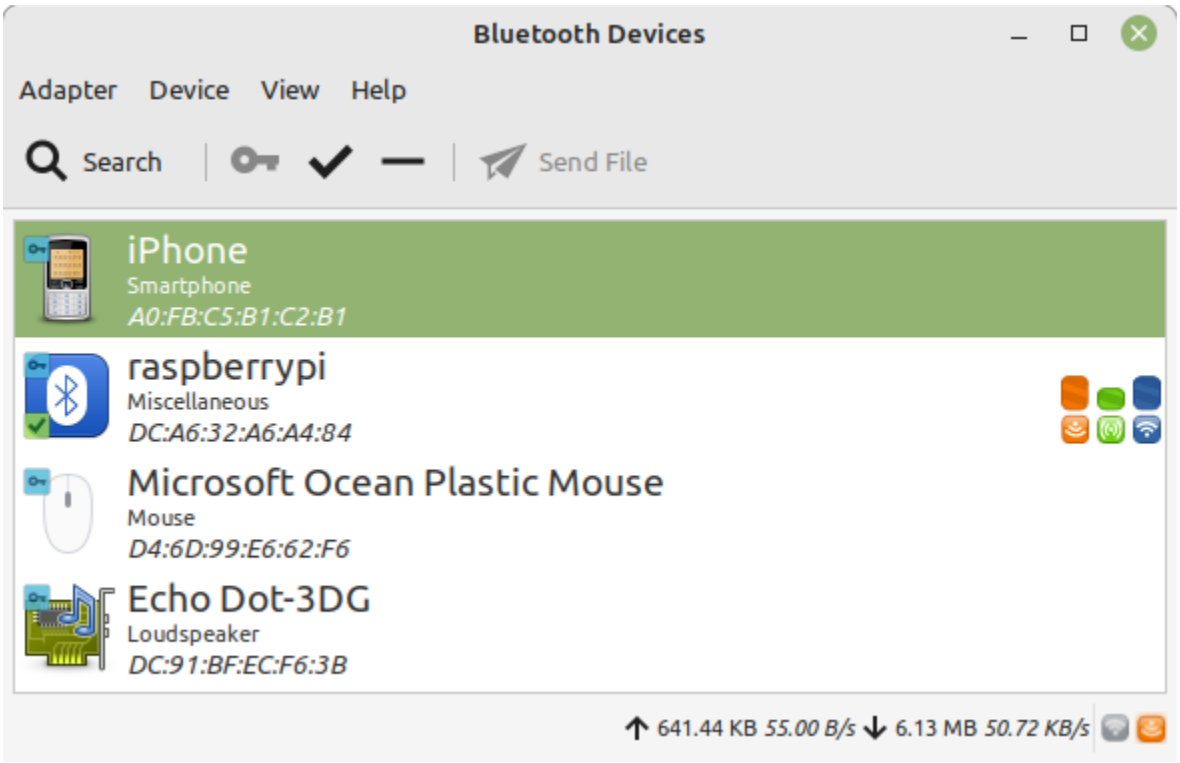


*Figure 21: The Linux Mint Bluetooth interface. On the bottom right, Bluetooth data transfer speed metrics can be observed; in this instance, a 50.72KB/s transfer rate is measured*

While Bluetooth is not known to be an extremely reliable data transfer protocol, no data transmission losses were observed during testing and it has been sufficiently reliable for implementation of this project. However, a disconnection event would sometimes occur if the client machine were to be removed from the immediate vicinity of the pedometer device, which is not out of the ordinary for a low-power protocol like Bluetooth. In order to maximize reliability, the Bluetooth address of the Raspberry Pi is hardcoded into the client system rather than being dynamically detected using a search function.

## Challenges and Solutions

Due to the relative complexity and high computational demands of this system, this project's development was met with several challenges relating to the limited computational resources of the Raspberry Pi. Luckily, solutions to these challenges were found and implemented to greatly improve the system's performance. These solutions take advantage of the Raspberry Pi's multi-core processor and I/O peripherals to maximize performance of the system in light of its relatively-low speed.

The artificial neural network used for motion classification is highly computationally-expensive and not well-suited for execution on a low-power processor. On the Raspberry Pi, an average processing delay for running the classifier is approximately 900ms. Running the system's I/O capabilities in the same process as this classifier would be disastrous, as there would be a noticeable delay while data is processed through the classifier. Unfortunately, Python is not well-suited for real-time performance and features such as interrupts and task scheduling are not available. Tasks such as regularly polling the gyroscope and listening for a Bluetooth connection would be impractical if the classifier were to be run in the same process as all other functions. Indeed, it would be impossible to perform meaningful step counting with the system in constant gridlock between polling the gyroscope and running the data through a classifier.

The solution to this problem involved splitting the program into multiple processes, as described in the Design and Implementation section. Using multiple processes allows the pedometer software to take full advantage of the processor's multiple cores, instead of being restricted to a single core if run as a single process. Unix sockets are used as a method to allow the separate processes to communicate with each other; for example, a Unix socket allows the Core process to send gyroscope data to the Classifier process despite running as separate processes.

Originally, UDP sockets were used due to their ease of use. However, as the project progressed, these were replaced with Unix sockets to remove potential I/O overhead relating to their implementation of the IP stack. This was a last-minute improvement made at the end of the project in order to provide a small improvement in I/O delay.

Additionally, the Bluetooth interface was used as a means for improving processing time for the classifier through access to the client's computational resources. The client device features a powerful Intel processor capable of executing instructions faster than the Raspberry Pi's ARM processor and can be used to offload the computational burden on the pedometer. The pedometer is capable of sending raw IMU data from the Sense Hat through a Bluetooth connection to be run through a classifier running on the client system; this classifier is identical to the classifier on the pedometer.

The Intel processor of the client takes an average of 300ms to run the motion classifier. This is about three times faster than the speed of the Raspberry Pi running the same classifier. The Bluetooth transmission time is fast enough such that its transmission delay is included in the 300ms figure; based on an average 50KB/s Bluetooth transmission speed reported by the Linux Mint Bluetooth interface, the delay is roughly 8ms for a 424-byte message containing IMU data. In short, running the motion classifier on the laptop results in a significant processing time improvement over running it on the Raspberry Pi despite requiring wireless data transmission.

## Test and Evaluation

After completion of the project, a period of test and evaluation was required in order to determine if the primary objective of the project was satisfied by its result. In order to create a test plan, it was necessary to take note of the validation methodology used by biomechanical studies employing pedometers as research tools. Many studies, including those funded by the National Institutes of Health, use pedometers as a research tool to collect activity data from test subjects and independent validation of pedometer accuracy is a paramount concern for researchers using them for studies. These third-party validation studies are much more valuable for testing of this device than manufacturer-provided data as it is impartial.

A public health study studying the accuracy of commercial pedometers provided a blueprint for the test and evaluation efforts for this project. This study was performed by the University of Pittsburgh and involved verifying the accuracy of pedometers by comparing the steps walked by participants to their recorded values to calculate an accuracy value for each of the pedometers evaluated. [4] Eight pedometer models were tested by 43 participants during this study, with each participant equipped with every pedometer model during a 100-step walk.

In this study, accuracy was calculated with the following formula.

$$Accuracy = 100 \times \frac{Steps\ counted}{Observed\ steps}$$
$$if\ Accuracy > 100,\ Accuracy = 100 - (Accuracy - 100)$$

It was decided that this test model would be adapted for testing the pedometer system, with one change: instead of a 100-step walk, it was decided to change the test to a 500-step walk. This would make for a better test, as the pedometer would be used for a longer duration of time and be exposed to more irregular step patterns during this time. Five trials were decided on, as this would be a reasonable number of trials to assess the device's accuracy given the cold climate and time constraints associated with testing; this would be equivalent to the steps walked by 25 participants in the University of Pittsburgh study.

The test undertaken to evaluate the accuracy of the pedometer involves the use of the pedometer device and its physical enclosure and power supply, the client laptop computer, and a mechanical counting device. The test plan below was followed by walking around the developer's neighborhood. However, due to the surrounding environment being in a low-lying area by the ocean, the entirety of the test was performed on flat land; this is identical to the testing procedure of the University of Pittsburgh study, which seems to have been performed on flat land.

The test plan consisted of the procedure performed by the user:

1. Place both the USB power bank and Raspberry Pi into the leg pouch.
2. Turn on the pedometer by connecting the Raspberry Pi to the USB power bank using a USB-C cable.
3. Place the leg pouch on the upper right leg, such that the pedometer is on the right side of the leg and is directly below the knee.
4. Start the pedometer program through a SSH connection with the Raspberry Pi.
5. Proceed to the start position. Do not make any steps or engage in any walking motion once this step is complete.

6. Connect the client laptop computer to the pedometer by executing the client program and establishing a Bluetooth connection.
7. Using the client program, start a new session to ensure that the step count is zero, if it is already not zero.
8. Clear the mechanical counting device to zero, if it is already not zeroed. The mechanical counting device used in the test is visible in Figure 22.
9. Walk 500 steps and record each step with the mechanical counting device.
10. When 500 steps have been counted by the mechanical counting device, stop walking. Record the value displayed on the client GUI.
11. Repeat steps 7 to 10 until five trials have been completed.


*Figure 22: Mechanical counting device used to record total steps walked. A step count of 50 is visible on this device.*

Figure 23 contains tabulated data from the test effort.

| Trial | Steps measured | Accuracy |
|---|---|---|
| 1 | 453 | 90.6% |
| 2 | 465 | 93.0% |
| 3 | 466 | 93.2% |
| 4 | 467 | 93.4% |
| 5 | 480 | 96.0% |

| | | |
|---|---|---|
| Average | 2329 | 93.1% |

*Figure 23: Step and accuracy data collected during test and evaluation effort*

If the Pittsburgh study results are compared to the experiment's findings, it can be observed that the test results indicate that the pedometer performs better than two pedometers tested in that study: the Fitbit Charge and Digiwalker. While it underperformed compared to the remaining pedometer models, a 93.1% accuracy is impressive given the development time of this device. Figure 24 shows the pedometers evaluated in the Pittsburgh study compared to the project pedometer's results. Note that in the below table, a standard deviation for a 95% confidence interval was calculated for the data in order to match the format of the data collected by the study.

Note that the results from the University of Pittsburgh study tabulated in Figure were associated with a measured gait speed greater than 1.0m/s. The gait speed observed in the test was 1.1m/s, reported by an iPhone accelerometer application.

| Pedometer type | Position Worn | Accuracy (%) |
|---|---|---|
| Fitbit Charge | Wrist | 80.32 ∓ 18.63 |
| Garmin Vivofit | Wrist | 97.20 ∓ 2.08 |
| Fitbit Zip | Waist | 98.81 ∓ 0.61 |
| Digiwalker | Waist | 89.81 ∓ 14.43 |
| Omron | Waist | 99.16 ∓ 0.84 |
| Yamax EX-510 | Waist | 98.87 ∓ 0.89 |
| Accusplit | Waist | 99.14 ∓ 1.21 |
| **This project** | **Leg** | **93.1 ∓ 1.675** |

*Figure 24: Accuracy data resulting from University of Pittsburgh study compared to project's test and evaluation finding. Data is in the format M +/- SD*

While there is not such a simple blueprint for testing the anti-tampering capabilities of the pedometer, it can be reported from informal testing that the anti-shake capabilities are rather robust. It is difficult to get a shaking motion to register as a step; while this may be possible by rhythmically shaking the pedometer up and down, it is not as easy to do so by shaking the pedometer at random intervals. This is sufficient for

testing purposes, as the goal for anti-tampering technology was to have a device capable of knowing that it is being physically-manipulated in a way that should not be registered as a step.

The original goal of this project was to be within 10% of the accuracy of the built-in iPhone pedometer. A Stanford study to evaluate smartphone-based activity monitoring in 114 peripheral artery disease patients used similar methodology to the University of Pittsburgh study the accuracy of the iPhone pedometer, though using the 6-minute walk test in lieu of a fixed 100 step walk. [5] The results of this study yielded an iPhone pedometer error of $-7.2\% \pm 13.8\%$, or a mean accuracy of 92.8%; comparable to the Raspberry Pi pedometer's mean error of -6.9%.

Given that the methodology of this study was very similar to the University of Pittsburgh study used to develop the original test plan, it can be assumed that this is a suitable figure to compare the test results against. Therefore, it can be concluded that the project succeeded in its goal to approach the accuracy of the iPhone pedometer, as well as the secondary goal of a step counting error less than +/- 25%.

## Future Improvements

The goal of this project was to develop a proof-of-concept device for integrating machine learning capability into a pedometer to evaluate its utility for step counting algorithms. The final product is rather rough around the edges and there is much room for improvement for several features that improve both its efficiency and user experience.

Much of the code running on the system can be optimized to move away from a Python-based implementation. While the current system is usable, its speed and efficiency can be improved using existing technology. A C-based implementation of the software would be achievable, though a C implementation of the neural network would be a challenge if Keras is not to be used. It is even conceivable that the entire system may be implemented as an FPGA design, with all features of the system including its neural networks defined in a hardware description language such as VHDL. These developments would make for a more computationally-efficient system requiring lesser computational resources.

A laptop computer is a rather impractical means of a client device. The decision to use a laptop as a client device was made purely to showcase the Bluetooth connection and GUI capabilities of the system. Linux is a much easier platform to develop for than iPhone or Android, considering that much of the client code was adapted from the pedometer's Linux-optimized Python code.

In a commercially-viable system, a smartphone or smartwatch-based client system would be ideal; they are light, easily-carried, and provide for a convenient user interface with the pedometer. It is certainly possible to port the client code to a smartphone or smartwatch platform. Ideally, no neural network implementation would be required for a commercial client, as a commercialized pedometer would be able to efficiently run a neural network and not need to offload these capabilities.

It is unknown how this device functions when tested with different users with varying gait patterns. The developer trained and tested the device using only their own steps. It would be informative to test this device with other users, as well as add their gait and walking patterns to the dataset used to train the

model. Another attempt to add accelerometer data to the motion classifier model should be pursued in the future, as this could be useful information to improve step counting accuracy.

More motion types such as running may be added to this dataset as well. It should be noted that the right leg was used to both train and test the model; it would also provide useful information to test the model on the left leg to observe its effectiveness, as well as train the model using data from both legs. The device should also be tested and trained on uneven and hilly terrain to further improve its functionality. It is conceivable that the device may be self-learning in the future, with step data recorded during walks being added to the training data set. In short, more testing is required before this device is usable for a mass audience.

Afuture iteration of this device should be able to allow for the calculation and tracking of several other motion-related metrics using IMU data. This data would include common metrics recorded by most commercially-available smart pedometers, including measurements for distance walked, stride length, and walking speed. GPS integration to indicate a user's current location would be another potential feature for a future device.

## Conclusion

This project was a successful exploration into the use of machine learning as part of a real-time step counting algorithm. The use of neural networks, particularly their training and validation, was a rewarding challenge that revealed their great utility and potential for use in systems to replace heuristic-based algorithms. As revealed by the test and evaluation effort for this project, neural networks are a powerful tool for identification of motion patterns, based on gyroscope sensor data.

The project met its primary objective to come within 10% of the accuracy of the iPhone pedometer. According to a Stanford study and a self test, this objective was met with rather ease; with the project's step count featuring 0.3% better accuracy than the iPhone pedometer, according to a Stanford study. Its secondary objective was also met, with the device featuring a step counting error of +/- 6.9% compared to the goal of +/- 25% or less.

This project served as a useful lesson in software systems design, particularly in systems with multiple sources of delay, in this case processing and I/O-related delay. Despite the Raspberry Pi's limitations, solutions were found to maximize system performance by making use of other resources provided by its hardware modules. The end result was rather satisfying: a functional pedometer system with Bluetooth capabilities making use of a neural network-powered motion classifier as part of its step-counting algorithm.

There is much room for improvement, however. On the technical side, many changes can be made to improve efficiency of the system for benefit to the user experience. These include an optimized software implementation and specialized hardware devices to result in improved system performance and efficiency. The methods to train the neural network may also be improved significantly, making use of a larger and more diverse training set to result in a more effective system implementation.

## References

1. J. Wise and N. Hongu, “Pedometer, Accelerometer, And mobile technology for Promoting PhysicAl Activity.” [Online]. Available: https://extension.arizona.edu/sites/extension.arizona.edu/files/pubs/az1491-2014.pdf
2. B. Lin, “Machine Learning and Pedometers: An Integration-Based Convolutional Neural Network for Step Counting and Detection,” *All Theses*, Dec. 2020 [Online]. Available: https://tigerprints.clemson.edu/all_theses/3462/
3. V. Malyi, “Run or Walk (Part 4): Using Keras Neural Network Classifier in iOS with Core ML,” *Medium*, Jan. 25, 2018. https://towardsdatascience.com/run-or-walk-part-4-using-keras-neural-network-classifier-in-ios-with-core-ml-a29723ab3235
4. R. Ata *et al.*, “Clinical validation of smartphone-based activity tracking in peripheral artery disease patients,” *npj Digital Medicine*, vol. 1, no. 1, Dec. 2018, doi: 10.1038/s41746-018-0073-x.
5. A. L. Hergenroeder, B. Barone Gibbs, M. P. Kotlarczyk, S. Perera, R. J. Kowalsky, and J. S. Brach, “Accuracy and Acceptability of Commercial-Grade Physical Activity Monitors in Older Adults,” *Journal of Aging and Physical Activity*, vol. 27, no. 2, pp. 222–229, Apr. 2019, doi: 10.1123/japa.2018-0036.