Wireless Healthcare Monitoring System - Fall Detection

James Rhodes

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Electrical (Computer or Telecommunications) Engineering at The University of Newcastle, Australia.

THE UNIVERSITY OF
NEWCASTLE
AUSTRALIA

Abstract

A wireless healthcare monitoring system that can detect falls will be of a great use to the elderly and people with disabilities. This paper explores the creation of a device that can detect if a fall has occurred through the use of an accelerometer, gyroscope and barometric pressure sensor, as well as, a website that acts as a user interface for exploring these fall detection algorithms. The algorithms include a threshold method, support vector machine and neural networks. The algorithm that performed the best was the neural network approach which had a specificity of 99.76%, a sensitivity of 99.5% and an accuracy of 99.63%. This result performed very well in comparison to the baseline approach on the data set collected within this thesis.

Key Contributions

- The creation of a data acquisition unit purpose built for the detection of falls including an accelerometer, gyroscope and barometer. The data acquisition unit was built to interface with a website via Bluetooth low energy.
- The exploration of various fall detection methods including linear threshold methods (enhanced with the use of a genetic algorithm), support vector machines and artificial neural network.
- The formation of a fall detection data base consisting of activities of daily living, falls and various exercises. The exercises were included to allow for the creation of fall detection methods that can discern between exercise and falls for use with an active person as well as the elderly.

Acknowledgements

I would like to thank all the people in my life who have had to endure the endless "thinking out loud" that occurred during this project and for their support and help in surviving my final year of university.

Contents

Chapter 1: Introduction

1.1 History and Motivation

The monitoring of patients has in the past been challenging, especially in chronic illnesses, the aging population or during sports activities where wires may inhibit movement. The improvement of the internet of things (IOT) has led to advancements in monitoring people through the use of a body area network (BAN) [1]. This class of device has been coined a wireless healthcare monitoring system (WHMS).

WHMS's are devices that can monitor, through the use of sensors, the current state of a patient and wirelessly log the results. These devices have found numerous uses ranging from the tracking of sporting outcomes, falls in the ageing, heart health, stress management and diabetes management. The key sensors that allow for the monitoring of these health outcomes are primarily inertial measurement units (IMU), pressure sensors, heart rate monitors, electrocardiogram and blood glucose monitors [1].

An application that a WHMS suits perfectly is the monitoring of the elderly. According to the Australian Institute of Health and Welfare (AIHW) 2.8 million people aged 65 and over experienced same-day hospitalisations from 2016-2017 accounting for 42% of the entirety of all same-day hospitalisations [2]. This number is significantly correlated with the frailty of the ageing population and accentuates the impact even minor events can have on this cohort. Contributing to the rate of hospitalisations is the number of elderly people that live alone. According to a study performed in 2015 by the Australian Bureau of Statistics, over 26.8% of elderly people live alone [3]. These factors contribute to the need and subsequent benefits from constant monitoring of the elderly within their own home to improve independent living and increase the speed and quality to which an elderly patient can be treated if a fall or accident occurs.

In Australia during the years 2014-2015 falls accounted for 37% of injury related deaths [4] . Another study found that from the years 2016-2017 125,061 people over the age of 65 years were admitted to the hospital due to fall related injury which accumulated to over 1.2 million days of patient care over the year [5]. Many elderly people struggle with falls not just because of the initial impact but also the length of time it took to receive treatment after the time of impact. A fall that occurs without immediate help can lead to many adverse effects for the patient such as rhabdomyolysis, pressure sores, pneumonia, dehydration, hypothermia or adverse effects from not taking medication on time [6]. If you couple these issues with a reduced cognitive ability due to dementia or memory loss from increasing age and it will also be very difficult to determine when the incident occurred. This is vital in ensuring that the patient gets the best possible medical care for their fall.

It is for these reasons that the use of WHMS's in detecting falls within the elderly population is vital. The faster a fall is detected the less of an impact it has on the patient, as well as the hospital system freeing up resources for other illnesses and reducing the financial impact on the public health system.

1.2 Thesis Objectives and Research Contributions

The purpose of this thesis is to validate previous efforts in regard to fall detection and to expand upon these ideas to create a more robust system that could potentially detect falls not only in the elderly but even in a young, healthy patient. People who are able bodied may have an existing disability that prohibits them from calling for help when a fall occurs. It is for this reason that this thesis would like a robust system to account for this, especially considering the majority of the research into this field has been case studies on the elderly. This thesis explores the creation of a device purpose-built for fall detection and an accompanying website/server that provides the exploration and comparison of numerous fall detection methods with an intuitive user interface. This thesis is intended to also explore how these methods perform against a data set developed with activities of daily living including exercise to decrease false alarm rate during physical activities.

Data sets within the field of fall detection are sparse and hard to come by. The data set created within this paper contains various activities of daily living (ADL) and also data sets for falls in different directions and scenarios. The data sets were generated by an 85kg, 21-year-old male and so it is noted that this database is not necessarily representative of all fall scenarios. A lot of the falls were recorded by attaching the data acquisition unit to a plank and letting it fall in various directions as well as being worn by the 21-year-old male and having him fall onto mats in various different scenarios. Overall, 480 scenarios were created. These scenarios are discussed in more detail in Chapter 4.

The data was acquired using a purpose-built device (data acquisition unit) that wirelessly transmits via Bluetooth Low Energy (BLE), accelerometer, gyroscope and barometer data every ten milliseconds. The BLE functionality is provided by a library written by Neil Kolban [7]. The data acquisition unit connects to the custom website (with the help of a Bluetooth Application Programming Interface (API) [8]) which in turn connects to a server for running machine learning methods. The baseline for the validity of each fall detection method is compared to a threshold method that utilises only the accelerometer and gyroscope for fall detection [9]. To build on this previous work a barometric pressure sensor was added to provide an extra insight into the occurrence of a fall. The barometric pressure sensor was used to detect the height of the patient before a suspected fall and measure it afterwards to get a more accurate prediction. Additionally, the thresholds for all methods were then run through a genetic algorithm written by the author in Python with varying fitness functions to provide an optimal value of these thresholds based on the data collected. The genetic algorithm was converted to Python from a Java example written by Daniel Shiffman

in his book The Nature of Code [10]. His example explored, generating a sequence of random characters which eventually matched a quote from Shakespeare.

Machine learning methods have also been explored such as support vector machines created with the aid of the Scikit-Learn library and artificial neural networks created with the help of the TensorFlow library [11][12]. Various data preparation techniques for these methods were experimented with such as using multiple time-steps worth of data as an input, oversampling techniques and principal component analysis. The optimization of various hyper-parameters involved with neural networks was explored and achieved with the aid of the library Optuna [13]. All of the results are compared and implemented on the website.

The website allows the user to explore each of the discussed fall detection methods and alter various parameters associated with them to allow for the experimentation, comparison and understanding of how each method functions. The entire system explored within this thesis can be seen in Fig. 1.

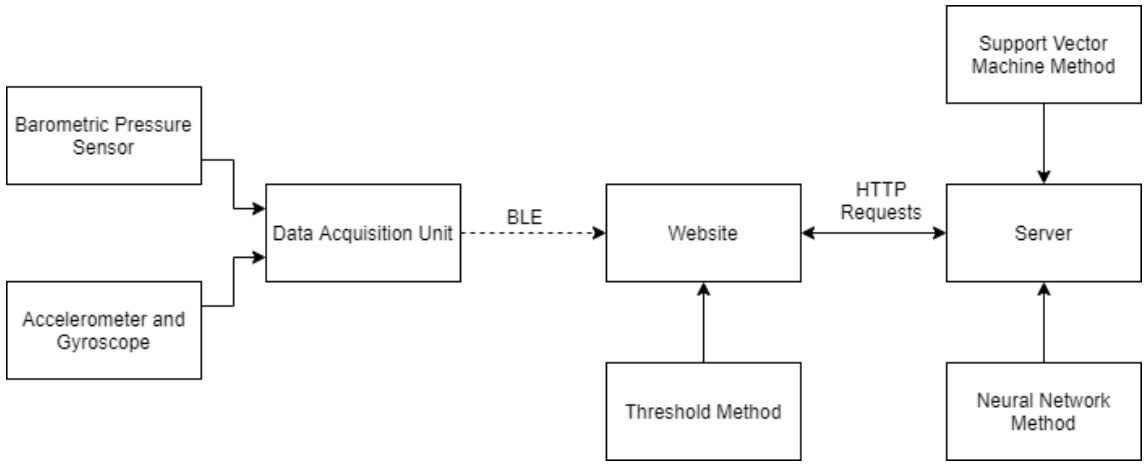


Figure 1: Overall System Design

1.3 Thesis Outline

1.3.1 Chapter 1: Introduction

This chapter provides an introduction to the paper, its motivations and research contributions.

1.3.2 Chapter 2: Literature Review

This chapter provides an insight into the previously researched fall detection methods as well as other theoretical components necessary for the remainder of the paper.

1.3.3 Chapter 3: Hardware Design

This chapter outlines the creation of the fall detection device known as the data acquisition unit.

1.3.4 Chapter 4: Data Collection

This chapter describes the creation of the fall data set which contains falls, activities of daily living and physical exercise.

1.3.5 Chapter 5: Software Design

This chapter outlines the software required to implement the fall detection algorithms as well as the creation of the website and server.

1.3.6 Chapter 6: Results and Discussion

This chapter outlines the results of the algorithms.

1.3.7 Chapter 7: Conclusion

This chapter concludes the paper and outlines the future works.

Chapter 2: Literature Review

2.1 Current Efforts in Fall Detection

2.1.1 Threshold Methods

Thresholding methods in fall detection are by far the most common algorithm that doesn't involve some form of machine intelligence. These algorithms generally revolve around the idea of trying to linearly separate all of the measurement values into a number of thresholds that if surpassed would indicate a fall. Common measurements that have this threshold method applied to them are the accelerometer and gyroscope magnitudes. These magnitudes are calculated as show in Equation (1) and Equation (2).

$$|A| = \sqrt{A_x^2 + A_y^2 + A_z^2} \tag{1}$$

$$|\omega| = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \tag{2}$$

These values are used as indications of a free fall and impact taking place. During an impact a free fall (the initial fall), a loss in acceleration is experienced in all of the axes as the sensor falls towards the ground. Conversely, during the free fall there is a steady increase in angular velocity as the person rotates in any direction towards the ground. Upon impact a vast spike in the accelerometer values is noted. The reason the magnitude of the accelerometer and gyroscope is so good at conveying this data is because the direction or axes in which these impacts occur does not matter in a fall, only how the length of the vectors are important.

This information is the fundamental idea behind the baseline approach explored by this paper which has been adapted from a paper written by Quoc T. Huynh et al. [9]. The baseline approach is the constant monitoring of the accelerometer and gyroscope magnitudes. If the accelerometer magnitude falls below a set threshold, then wait 0.5 seconds and if during that time the magnitude of the accelerometer and the gyroscope spike beyond their respective upper thresholds then a fall has been detected [9]. This algorithm can be described using the flow chart provided in Fig. 2.

The thresholds for this method were chosen by first calculating the sensitivity and specificity which are given in Equation (3) and Equation (4) and then minimising the distance from the measured results to the maximum sensitivity and specificity (100% and 100%). The distance that must be minimised is given by Equation (5).

$$Sensitivity = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3}$$

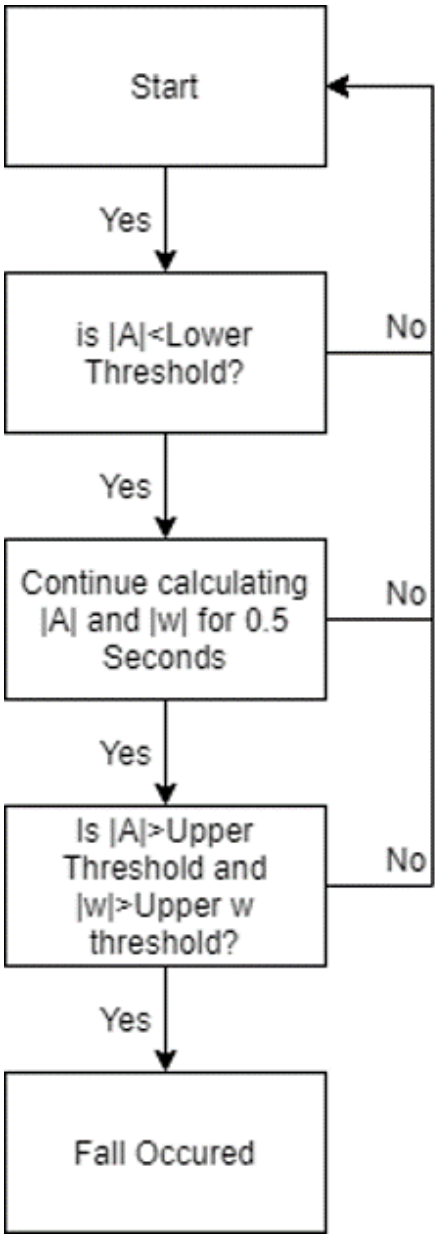


Figure 2: Baseline Threshold Method

$$Specificity = \frac{true\ negatives}{true\ negatives + false\ positives} \tag{4}$$

$$d = \sqrt{(1 - Sensitivity)^2 + Specificity^2} \tag{5}$$

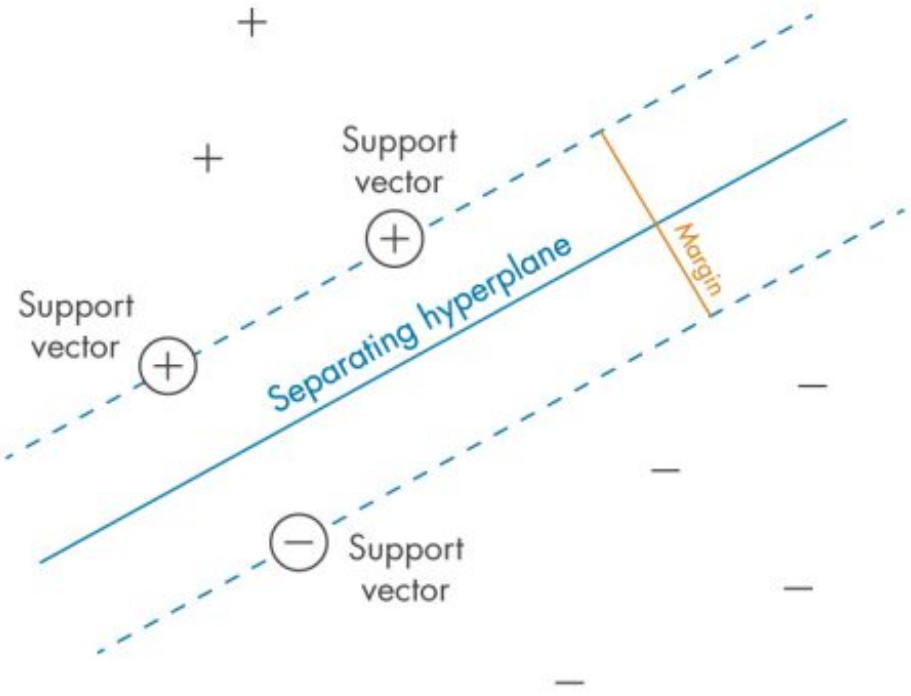


Figure 3: A Two Dimensional Feature Space Illustrating SVM [15]

2.1.2 Support Vector Machines

Support Vector Machines (SVM) are a machine learning method used commonly in classification and regression problems. SVM's are a supervised machine learning method that is well known for its aversion to over-fitting. It is this property that makes SVM's well suited to fall detection efforts.

SVM's can be used for both linear and non-linear tasks. Linear SVM's function by taking in a data set of labelled features and finding a hyperplane that separates the different classes. What is unique about SVM (and what allows it to resist over-fitting) is that the hyperplane that is calculated to separate the classes is calculated by maximising the distance between the plane and a margin created by nearby data points. These data points that are used to calculate the margins are known as support vectors. The hyperplane is defined as the midpoint between the two margins [14]. This naturally creates a buffer between nearby data points reducing the possibility of over fitting. This idea can be illustrated with a two dimensional feature space in Fig. 3.

Non-linear SVM is very similar to linear SVM however with one extra step before the hyperplane is calculated. The data sets feature space is projected onto a new higher dimensional feature space that contorts the positions of each data sample such that a hyperplane may be drawn between each class. This process converts a data set with classes that were not linearly separable into a higher dimensional space that allows for linear separation [16]. In practice these transformations into a higher dimensional space are very computationally expensive (having to calculate the coordinates of every point in n dimensions through the use of mapping functions and dot products). The time complexity of this calculation is $O(n^2)$ [17]. This is also incredibly difficult because it is impossible to tell (without information about the data) how many dimensions will be required to obtain a linear separation of the data points [18]. This transformation can be made incredibly efficient (O(n) time complexity) through the use of Kernels [17]. The Kernel 'trick' is an incredibly useful mathematical property that states that the transformation to a higher dimensional space is mathematically equivalent to the application of a Kernel function on to the original input data [18].

There are a wide variety of kernels that can be used in a support vector machine. These include linear, polynomial, Sigmoid and Gaussian Radial Basis Function (RBF). The kernel that is most relevant to this paper is the RBF kernel. The RBF kernel is a general purpose kernel function that is widely used in machine learning techniques, especially with regards to SVM. The RBF kernel's popularity is due to it's similarity to the Gaussian distribution. The kernel is also widely used due to its built in notion of similarity. The RBF function is at its maximum when the points are the same and decreases as the points become more dissimilar [19]. The RBF kernel can be seen in Equation (6).

$$K(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x}-\mathbf{x}'\|} \tag{6}$$

There are two main parameters that can be altered when utilising the RBF kernel and SVM. The γ term in the RBF function controls the width of the Gaussian curve this is because $\gamma = 1/2\sigma^2$. This parameter in turn controls how far away (in terms of similarity) two data points must be to be considered similar or dissimilar. This in turn is used to decide how far away points must be before they affect the placement of the hyperplane [20]. A good choice for this term is calculated as shown in Equation (7). The data points that affect the placement of the hyperplane are known as support vectors. Where N is the number of features in the data sample and σ_x^2 is the variance of the data set. This formula is used (by default) in the popular Python package Scikit Learn [11].

$$\gamma = \frac{1}{N \cdot \sigma_x^2} \tag{7}$$

Another parameter within SVM is the C or regularization parameter. This parameter controls how important it is if an incorrect classification occurs in the data set. This is directly reflected in the width of the margin separating the hyperplane and the data

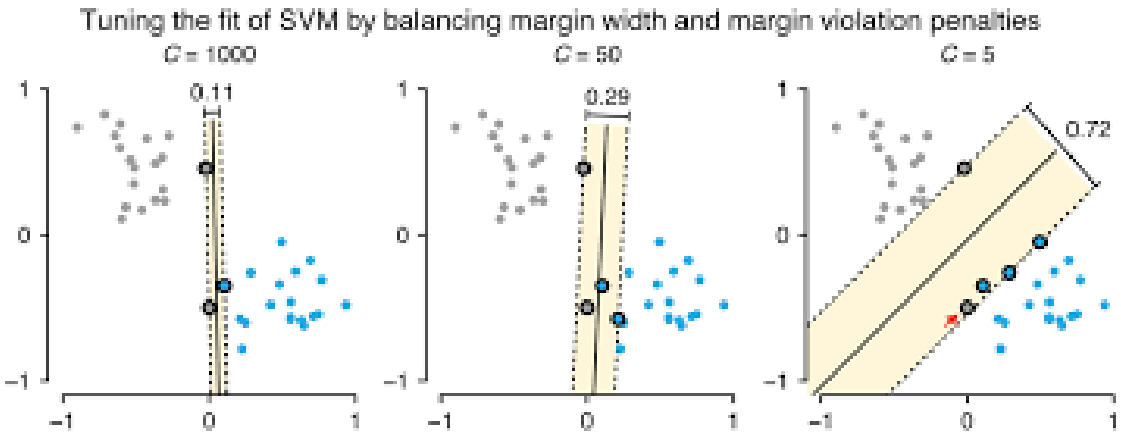


Figure 4: A Demonstration of the Influence of the Regularization Parameter [21]

classes. A large C value results in a smaller margin and a smaller C value results in a larger margin. This C parameter is often a trade off as a smaller margin can give the appearance of higher accuracy in the training set but can also cause some over-fitting as slight differences in a data point (in the test set for example) may cause it to be on the opposite side of the margin leading to an incorrect result. Conversely a large margin can cause under-fitting and lead to high proportion of incorrect classifications. This concept is displayed in Fig. 4.

Support vector machines are highly desirable in situations such as fall detection. This is because of it's ability to reduce over fitting (especially on small data sets such as most fall detection data sets). SVM's are also fast to train thanks to relying only on the support vectors for training. The only disadvantage is that SVM's are not easily able to supply a confidence value or probability of whether its prediction is correct. There are methods to obtain this, however they are computationally expensive algorithms.

2.1.3 Neural Networks

Neural Networks are powerful algorithms inspired by the human brain and the interconnection of neurons within it. There are many types of neural networks, the most relevant here being the Artificial Neural Network (ANN). Every type of neural network is comprised of an input layer, hidden-layer(s) and an output layer. If multiple hidden layers are utilised then the model is called a deep neural network. Neural networks implement supervised learning. The exact function of a neural network is to tune its parameters (lying within each layer) so that the input gives a correct output on a known data set which in turn will generalise to work on all data that is input into the algorithm whether the output is known or not. Simply put, if a machine learning model is trained on known data, it can then use the 'knowledge' it has gained to infer a solution to unknown data.

Achieving this 'learning' requires an understanding of the most basic unit within a neural network, the perceptron. The job of each perceptron is to take in all the inputs it receives and sum them together. The perceptron then applies an activation function

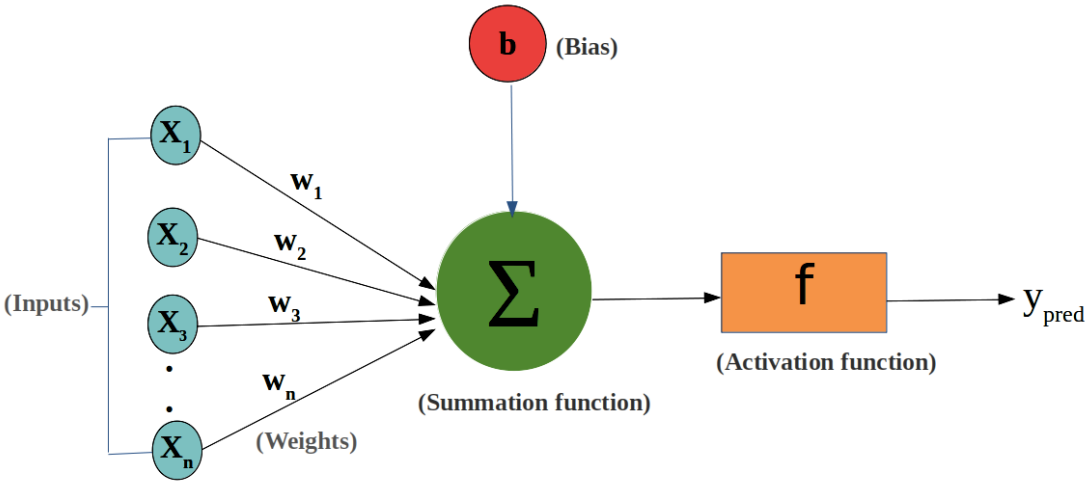


Figure 5: The Function of a Single Perceptron [23]

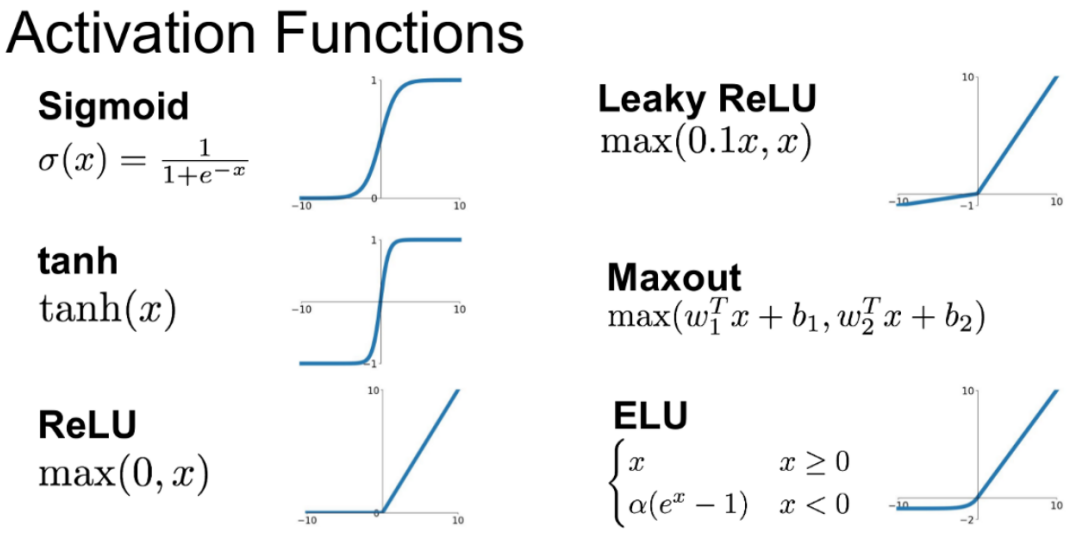


Figure 6: Different Activation Functions for Neural Networks [24]

to the output of this sum. The output of the activation function is the output of the overall perceptron. This is summarised in Fig. 5. The activation function can be any function however, to achieve a non-linear input-output relationship within the overall neural network, a non-linear function is used. Introducing this non-linearity allows for more complex relationships to be learned. The activation function is usually chosen such that it scales the parameter into a specific range such as 0 to 1 or -1 to 1 and is also chosen such that it is differentiable to aid in the 'tuning' process [22]. There are many different choices for activation functions. The ones most relevant to this paper are shown in Fig. 6.

Each layer consists of any number of perceptrons which have inputs from every perceptron in the layer before it and output to every perceptron in the layer after it. The connections between subsequent layers are weighted and a bias (or offset) is ap-

plied to each perceptron (as shown in Fig. 5). It is the weightings and biases that are 'tuned' throughout the learning process as certain weights are increased or decreased to culminate in an overall correct input-output relationship [25].

During the training of the model, the weights and biases are trained through the use of a cost function and an optimization algorithm. The cost function can be many different things, however, in general the cost for neural networks is the mean-squared error. This error is a measure of how far off the output of the neural network was from the correct output. The objective of the optimization algorithm is to minimise this error, which in turn, will increase the accuracy of the model [25]. The optimization is achieved by using the chain rule to determine the affect each weight and bias had on the output and then to alter them in the direction (found from the gradients) that is minimising the cost function. The most well known algorithm is stochastic gradient descent (SGD), which minimises the cost function as described previously [25]. Another common optimization algorithm is the adaptive moment estimation (ADAM) algorithm. This algorithm is very popular thanks to its versatility. The ADAM algorithm is less sensitive to noisy problems and problems with sparse gradients. ADAM also generally works well with its default settings for most use case [26]. SGD has two parameters that can be tuned to provide different convergence rates that are called the learning rate and momentum. ADAM only has one parameter that can be specified and that is the learning rate. The learning rate is a measure of how quickly a model can learn. A large learning rate allows for larger shifts in the solutions however can result in only jumping around local solutions and never actually converging to the most optimal answer. A small learning rate can cause the model to very slowly converge sometimes not converging at all [27]. The momentum for SGD is a term that can be tweaked in order to decrease the convergence time and also to increase the range of learning rates that convergence occurs over[28].

Two important parameters that are involved in neural networks are the batch size and the number of epochs. The number of epochs is how many times the model will be iterated over the entire data set for training. The batch size is the number of data samples the model will iterate over before the parameters of the model are updated via one of the aforementioned optimization algorithms.

Neural networks are well suited to fall detection problems thanks to their ability to classify very intricate non-linear relationships between data classes. They have been used previously for this task in not only ANN's but also other forms such as convolutional neural networks and recurrent neural networks which makes them versatile in their different configurations and hyper parameters that can be altered [29]. Hyper parameters are the different options available for a NN model such as layer size, number of layers, activation functions and optimization algorithm. Neural Networks also inherently output a "confidence value" based on how activated the output neuron is (in its range 0-1). This is extremely useful as it gives the confidence that the current data input is a fall or not in the task of fall detection and allows for the tuning of how sensitive the model is by changing how high the confidence must be before considering the current input a fall. The disadvantage of neural networks for fall detection is the amount of training time. Neural networks have to iterate over the entire data set to train. Neural networks are also susceptible to over-fitting which makes them less desirable on small

data sets such as the case with fall detection. Over-fitting can be counteracted through the use of dropout, which is the act of not using certain perceptrons (at random) on each different epoch of training. Efforts can be made to increase the effectiveness of neural networks in the case of fall detection through the use of hyper parameter tuning as described in Chapter 5.3.

2.2 Genetic Algorithm

A genetic algorithm is a type of evolutionary algorithm designed to maximise a cost function. A genetic algorithm is inspired by Charles Darwin's theory of evolution and survival of the fittest. The general idea is that out of a population, the members deemed 'most fit' have a higher probability of reproducing for the next generation. This process is performed continuously with each iteration having a probability of mutation known as the mutation rate. This is continued until either the system has reached a steady state solution or until a set of criteria is met [10].

Using this algorithm on data is performed in the same steps. First a population is initialised with some random values. This population is passed through a task and based on a fitness function (a function that determines how well each member of the population did at the specific task) a score or fitness is given to each member of the population.

Two members of the population are then chosen with members of a higher fitness score being more likely to be chosen (it is important to note that 'unfit' members can still be chosen it is just less likely). The two randomly selected members are then combined by a technique called crossover, where some attributes of one member is spliced with some attributes of another member. This splicing creates a member of the next generation and this process is repeated with 2 randomly chosen members of the population until an entirely new generation is created. After the crossover is achieved there is a random probability (known as a mutation rate) that a random parameter of each member of the population will be reinitialised with another random value. This new generation is then fed back through the system and the process begins again [30].

The main parameters of a genetic algorithm that can be altered to affect performance are the crossover technique, the fitness function and the mutation rate. The crossover technique can give different results based on the scenario. The crossover function used in this paper is to take a property from one of the parents and then the next property from the other parent alternating back and forth until the new member has alternating properties from both 'parent' members. The fitness functions chosen is discussed further in Section 5.1.

The genetic algorithm is a powerful tool and often used for situations where a Monte Carlo simulation would be too cumbersome due to the high number of possible combinations. A good genetic algorithm with a well calibrated set of parameters can evolve/-converge very quickly to a solution to the optimal possible value of the fitness function [10].

2.3 Alpha-Beta Filter

Measurements from sensors in the real world are always corrupted in one form or another by noise. Noise is inescapable in a real system and so it must be reduced in order to gain a better measurement allowing for more accurate representations of the real world states of the system. Commonly, simple low-pass, high-pass or band-pass filters are used to remove noise that has a consistent frequency response (or a frequency response sufficiently different to the measurement signal). However in situations where the noise is often random and does not have a consistent frequency response, tactics such as linear state observers, optimal state observers (Kalman filters) or alpha-beta filters may be implemented.

The alpha-beta filter is a much less computationally expensive algorithm to perform than other options such as Kalman filters however is considered sub-optimal. This does not mean that the alpha-beta is not sufficient for many applications however [31]. The alpha-beta filter has many advantages in that it does not require a robust model of the system but instead only requires that it has two states one that is termed the position and the other the velocity. These two states are derivatives of each other. These states do not necessarily have to be based on position they can be any measurements in which the two states are related by a derivative [31].

The alpha-beta filter works by initialising a starting value for position and a starting value for velocity. The alpha-beta filter works based on 5 key formulas. The first equation is the update equation which updates new estimated position based on the previous position, previous velocity and the change in time as shown in Equation (8). The second equation updates the current velocity to be equal to the previous velocity (this is because the alpha-beta filter assumes constant velocity) as shown in Equation (9). The third equation calculates the residual which is the difference between the measured position and the current position as shown in Equation (10). The fourth equation adds a proportion (where the proportion is denoted by α) of the residual to the position estimate as shown in Equation (11). The final equation adds a proportion of the residual (where the proportion is denoted by β) divided by the change in time to the velocity as shown in Equation (12)[31].

$$x_k = x_{k-1} + (v_{k-1} \cdot dt) \tag{8}$$

$$v_k = v_{k-1} \tag{9}$$

$$r_k = x_m - x_k \tag{10}$$

$$x_k = x_k + \alpha * r_k \tag{11}$$

$$v_k = v_k + \frac{\beta * r_k}{dt} \tag{12}$$

**Where x is position, v is velocity, r is residual, x_m is the measured value for position and k is the current time step*

This simple formula is what makes the alpha-beta filter so computationally efficient. The alpha-beta filter has the two parameters that can be altered which are α and β. Large α and β components will result in a faster response with less noise suppression while smaller values will result in more noise suppression with a slower response.

The alpha-beta filter can also generalise to a steady state Kalman filter if the formulas shown in Equation (13) - Equation (16) are used [32]. This relationship allows for a calculation of α and β based on the process variance σ_w^2 and the noise variance σ_v^2 which increases the performance of the filter. The process and noise variances can be calculated, measured or experimentally found by trial and error until a desired response is found [32].

$$\lambda = \frac{\sigma_w \cdot T^2}{\sigma_v} \tag{13}$$

$$r = \frac{4 + \lambda - \sqrt{8\lambda + \lambda^2}}{4} \tag{14}$$

$$\alpha = 1 - r^2 \tag{15}$$

$$\beta = 2(2 - \alpha) - 4\sqrt{1 - \alpha} \tag{16}$$

The alpha-beta filer can be extended to an alpha-beta-gamma filter which instead of assuming constant velocity assumes constant acceleration. This extended filter can also generalise into a steady state Kalman filter.

2.4 Principal Component Analysis

Many real engineering problems require large data sets with high dimensionality to get accurate results. Large data sets are often computationally expensive to perform operations on due to their sheer size. Principal Component Analysis (PCA) is a dimensionality reduction technique that can transform a high dimensional data set into a much smaller number of dimensions while still maintaining as much of the information as possible from the high dimensional data set [33].

Statistical information about a dataset is often hidden within the variability of the data set. The goal of PCA is to lower the dimensionality of the problem while maintaining this variability. This in turn means finding a set of variables (principle components) which are linear combinations of the original variables in the data set and are also uncorrelated to each other [33]. The first step in calculating the principal components

is to normalize the input data along each of the dimensions. The way this is achieved in this paper is by using min max normalization which can be defined by Equation (17).

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{17}$$

(X_{max} is the maximum value found in X and X_{min} is the minimum value found in X)

After the data is normalised the covariance matrix of the data set is calculated. The covariance matrix consists of all the covariances between each dimension of the data set and is a square matrix with pxp size where p is the number of dimensions. each entry on the matrix is a measure of how dependent or independent the corresponding dimensions in the data set are. The eigenvalues and eigenvectors of the covariance matrix are then calculated. There are the same number of eigenvalues/eigenvectors as there are dimensions in the data set. The eigenvalues with the largest magnitude and their corresponding eigenvectors hold the most information (or variance) pertaining to the original data set. The largest eigenvalue's corresponding eigenvector gives the first principal component and so on in decreasing order, as the component number increases the amount of information that each principal component contains about the original data set decreases. If PCA is being used to minimise the data set into n dimensions then in decreasing order, n eigenvectors will be joined together into a matrix. The data set matrix and the eigenvector matrix multiplied together will give the reduced dimensionality data set consisting of the required number of principal components [33].

PCA is an incredibly powerful technique for dimensionality reduction however, because this operation takes place on the entire data set, when applying it to tasks such as machine learning PCA can cause issues if the training and test data sets have not yet been separated. If they have not yet been separated then the training set will contain information pertaining to the test data set and as such will give inaccurate results when the test data sets accuracy is calculated [34]. This accidental peak into the test data set is known as a data leak.

Chapter 3: Hardware Architecture

3.1 Overview

The data acquisition unit is pivotal in the operation of fall detection algorithms. The data acquisition unit provides all the necessary measurements for fall detection algorithms to operate. The unit is made up of several components all mounted on a printed circuit board (PCB) and enclosed inside a 3D printed case. The components that make up the data acquisition unit are a Li-ion battery, battery charger, battery level indicator, voltage regulator, 3-axis gyroscope, 3-axis accelerometer, barometric pressure sensor, indicator LED's, USB to UART converter and a micro-controller as shown in Fig. 7. The schematic and PCB design can be found in Appendix A. The final PCB can be seen in Fig. 8

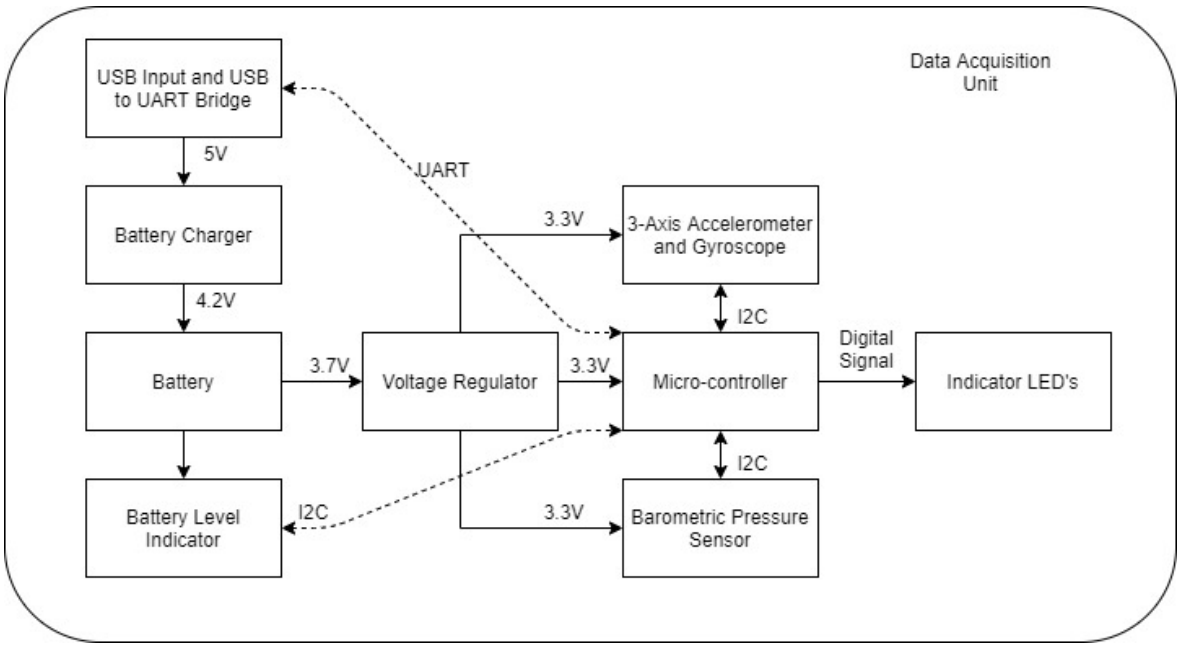


Figure 7: Hardware System Overview

3.2 Hardware Design

3.2.1 Micro-controller: ESP32-WROOM-32D

The ESP32-Wroom-32D developed by Espressif Systems was utilised within the data acquisition unit to facilitate computations and wireless transmission. The ESP32 has

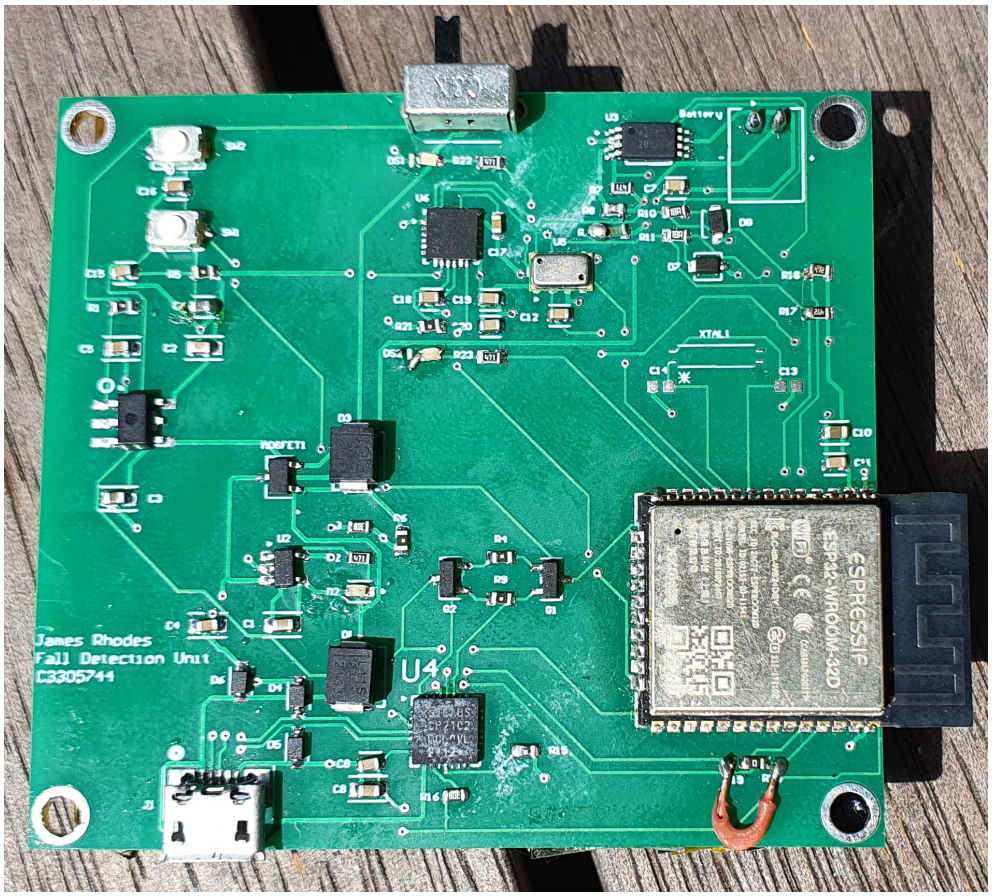


Figure 8: PCB Final Product

extensive use in the IOT industry with incredibly well documented libraries. This facilitates ease of use and allows for concentration on the algorithms related to fall detection. The ESP32 was also chosen for its multi-core processor and 240MHz clock which has an abundance of processing power to account for any computationally expensive algorithms that may be necessary for the accurate prediction of falls. The ESP32 also has inbuilt wireless communication systems, facilitating communication via WiFi and BLE which is utilised within the data acquisition unit. The ESP32 was programmed using the Arduino integrated development environment (IDE). The ESP32 supports I2C communication between peripherals and an API written by Espressif was used to read and write bits [35].

The schematic and PCB design was created using the documentation provided by Espressif for the ESP32 DevKit V4 which is a development kit produced by Espressif that utilises the ESP32-WROOM-32D [36]. All components used within Espressif's schematic were either used exactly or an equivalent component (such as the USB-UART bridge and the voltage regulator) was used. The PCB was also designed to allow free space below the inbuilt PCB antenna on the ESP32-WROOM-32D as per the requirements of the data sheet.

The power consumption of the board was reduced from the Espressif design by using a more advanced low drop-out voltage regulator with a much lower quiescent current.

This is discussed further in Section 3.2.4.

3.2.2 Accelerometer/Gyroscope: MPU6050

The MPU6050 is a 3-axis accelerometer and 3 axis gyroscope. It can make its measurements in various accuracy modes. The gyroscope has programmable full-scale ranges of ±250, ±500, ±1000 and ±2000 degrees per second. The accelerometer has programmable full-scale ranges of ±2, ±4, ±8 and ±16 g's [37]. The chosen ranges were ±250 degrees per second for the gyroscope and ±2g's for the accelerometer. This is because these ranges are the most sensitive allowing for more precise measurements which is especially important for fall detection algorithms where slight changes in these values could be the difference between a fall or not. The most accurate settings lead to the best and least noise polluted readings. The MPU6050 was also chosen due to its ability to calculate yaw, pitch, roll and linear accelerations without gravity with the use of the onboard MCU. The onboard MCU uses an unsented Kalman filter to accurately estimate the extra values. This feature is not utilised within the algorithms presented in this paper however they are displayed graphically on the website as described in Section 5.4.2.

The schematic for the MPU6050 was designed using the recommended layout provided by the datasheet and was connected to the ESP32 for I2C communication [37].

3.2.3 Barometric Pressure Sensor: MS5607-02BA03

A barometric pressure sensor can be used to calculate the altitude of the data acquisition unit. This can provide incredibly useful information about the current position of the user. The barometric pressure sensor chosen for this task was the MS5607-02BA03. This sensor has a high operating range of 10-1200mBar and a high resolution with its 24 bit ADC. The sensor has incredibly accurate relative measurements but not as accurate absolute measurements. This is an acceptable trade off in the data acquisition unit as the exact height above sea level is not helpful but rather how the height changes from its initial position. The sensor must be operated at high speeds to operate effectively with the other sensors on the data acquisition unit and so this barometric pressure sensor was chosen due to its ability to have a conversion rate down to 1ms. Since it will only be queried every 10ms this is more than adequate. This barometric pressure sensor is very small and also has an operating voltage of 3.3V and a very low power consumption with only 1μA of current draw making it ideal for a healthcare device where power and space are a limited resource. The sensor also operates via I2C making it a convenient choice for incorporation with the other existing sensors. The sensor also has a built in temperature sensor which allows for the compensation of temperature effects on the barometric pressure sensor output.

The circuit design for the MS5607-02BA03 was entirely based off the data sheets recommended layout for I2C communication. This layout ensures the effective integration into the data acquisition unit and the correct operating conditions for the sensor [38].

3.2.4 Voltage Regulator: S-1172

The device has a battery that can range from 3.4-4.2V and so a voltage regulator must be implemented to step this voltage down to the required 3.3V for all of the components on the data acquisition unit. The S-1172 series low dropout voltage regulator was chosen for its extremely low current consumption of 70μA during operation and 0.1μA when turned off. This will greatly increase the power efficiency of the data acquisition unit while in its off state. The S-1172 was also chosen for its low dropout voltage of 70mV which is desirable due to the requirements of the sensors on the board and the minimum voltage of the battery. The S-1172 also has an enable and disable pin present which allows for the implementation of an on/off switch allowing for power savings when the data acquisition unit is not in operation.

The circuit design was based on the recommended design provided by the datasheet with an added switch and low pass filter on the on/off pin to allow for the turning on and off of the data acquisition unit. The capacitance of the input and output capacitors was also chosen to be in the correct range required by the regulator, keeping in mind the decoupling capacitors required by the other components on the board [39].

3.2.5 Battery: Lithium Polymer

The battery chosen for the data acquisition unit is a 3.7V 2000mAh Lipo battery. The batteries voltage range is 3.4V-4.2V at an empty and full state respectively. The battery is overkill for the application as the device itself only draws at a maximum 200mA, however the size of the PCB allowed for a larger battery to be used without making the entire device significantly larger. This allows for an extended battery life thanks to the larger battery. The PCB has been designed to function with smaller battery sizes as well to allow for the adaptation into a smaller unit in the future. Lipo batteries were the obvious choice for this project thanks to their high energy density and small size. The specific lipo battery used for the data acquisition unit has an inbuilt over current and under voltage protection circuit prohibiting circuit faults from affecting the battery.

3.2.6 Battery Charger: MCP73832

The battery charger implemented on the data acquisition unit is the MCP73832. This charger is specifically built for lithium polymer and lithium-ion batteries making it a perfect fit for the battery chosen. The battery charger implements a constant current and constant voltage to charge the battery. Both the current and voltage levels are configurable based on the circuit surrounding the chip. The charger on the data acquisition unit is configured to charge up to 4.2V as per the specification for the lipo battery used. The charge current is also configured to 500mA, this value is smaller than the chosen battery can handle which may increase charge times however, leaving the charge current at 500mA allows for the use of smaller batteries in the future without

risk of charging the battery too quickly. The MCP73832 was also chosen for it's built in charge status pin which, when an LED and resistor is attached, will turn the light on while charging and turn it off when fully charged. The charger has an input voltage range that can accept 5V making it perfect for charging the battery via a USB port on the data acquisition unit.

The circuit for the charging circuit is designed using the recommendations from the data sheet making alterations for the 500mA charging current and 4.2V charging voltage [40]. A secondary sub-circuit designed by the author is in series with the output of the charger and the voltage regulator that consists of a MOSFET, diode and resistor in order for the data acquisition unit to be powered (if the switch is in the on state) by the USB port if it is plugged in while the battery charges to ensure that the battery is not constantly drained while its charging if the switch is in its on state. It also disables the battery charging circuitry when only the battery is plugged in without the USB port.

3.2.7 Battery Monitor: DS2782E+

The battery monitor maintains an accurate measure of the batteries current state of charge allowing for the display of a battery percentage. The battery monitor chosen for the data acquisition unit is the DS2782E+ for its highly accurate measurement characteristics, compatibility with the battery used and the configurable sense resistor used for different battery capacities. The battery monitor is able to communicate via I2C making it highly compatible with the data acquisition unit.

The circuit used for the battery monitor is replicated from the data sheet, with the change of the current sense resistor to 10mΩ which configures the monitor to be able to count the current used up to 2000mA as per the maximum battery requirements [41].

3.2.8 USB to UART Bridge: CP2102-GM

A USB to UART communication chip is necessary for the ease of programming the data acquisition unit and also for serial communication via a computer and the unit for debugging purposes. The chip used for this communication was the CP2102-GM. This is a very similar chip to the chip used on other ESP32 development boards and so the necessary drivers for communication are available within the Arduino software. The chip is also compatible with a 5V voltage source allowing for powering via the USB port.

The circuit design was performed in accordance to the data sheet and the ESP32 development board reference schematic to ensure the correct operation of the chip [36][42].

3.2.9 Indicator LEDS

The data acquisition unit has three LED's placed on the PCB to allowing for indication of certain processes. A red LED will illuminate if the device is not connected via Bluetooth and will remain on until connected. A blue LED will illuminate upon connection via Bluetooth and remain on for 5 seconds after which it will shutoff to save battery. A green LED is included to indicate whether the device is charging or not. The green LED will light up while charging and shut off once charging is complete.

3.2.10 Enclosure Design

The case for the data acquisition unit was designed using Fusion 360 a CAD program that is commonly used for 3D printing projects. The case is 3D printed out of a black PLA because it is a cost effective and light material. The case is designed to enclose the PCB and battery in a secure way with a removable backing making it easy for debugging in the early stages of the project. The case has a hole for the USB port and switch allowing for the use of these components while using the unit. The on/off switch has some basic labelling to signal which orientation results in on vs off. The case itself also has a light bar along the top which is also 3D printed out of a clear PLA to allow the indicator lights built into the data acquisition unit to shine through. The PCB layout of the lights orients them along the center to line up with the light bar.

The unit is intended to be worn on the hip attached to a belt with the on/off switch facing upwards. This orientation matches the same position the unit was in during the building of the fall data set which will lead to the most accurate results for fall detection as it more closely replicates the training data. The casing can be seen in Fig. 9.

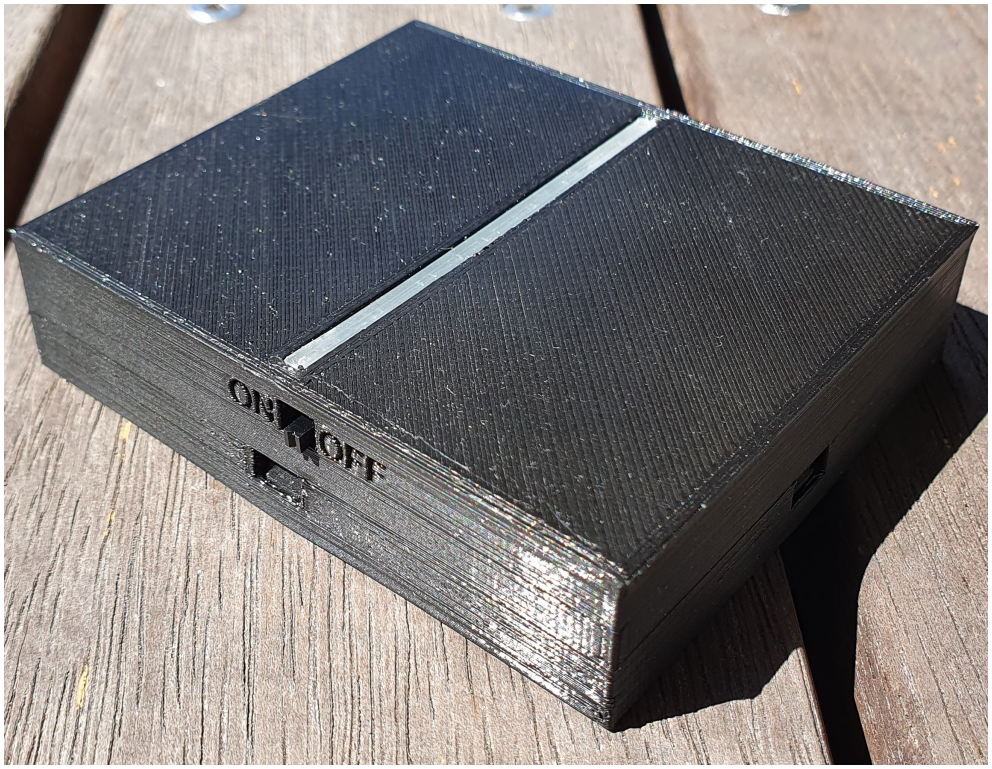


Figure 9: Data Acquisition Unit Casing

Chapter 4: Data Acquisition

4.1 Overview

The database used in this project was created using the data acquisition unit described in Chapter 3. This database was generated by an 85kg, 21-year-old male who wore the unit in various situations and falls. It is important to note that the test subject fell onto mats for safety reasons. Tests that were deemed too dangerous to be performed by a human were instead collected by attaching the data acquisition unit to a rod (at waist height for the test subject) and letting it topple over. This was to simulate the same dynamics without any safety concerns

The data transmitted by the data acquisition unit was collected every 10ms and included the acceleration and angular velocity in the x, y and z directions as well as the acceleration magnitude, angular velocity magnitude, pressure, temperature (for altitude correction), yaw, pitch, roll, a 3 component unit vector pointing in the direction of gravity and the time it was transmitted (in ms after the device was switched on). An overview of the data acquisition and storage is shown in Fig. 10.

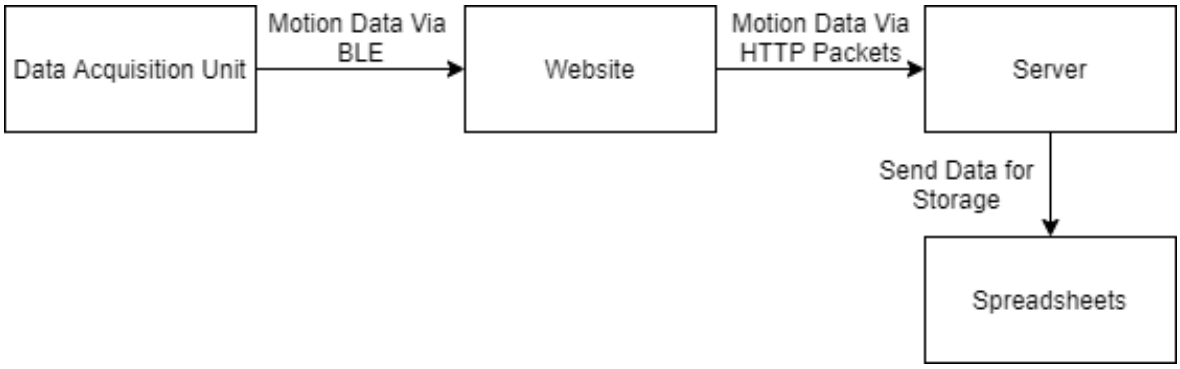


Figure 10: Data Acquisition and Storage Overview

4.2 Data Acquisition Unit

The data acquisition unit has 2 primary tasks and that is to collect the data and to transmit it via BLE to the website. The data acquisition unit could also be configured to run each individual fall detection scheme, however as the website is an exploration/comparison of all the different methods it is more efficient to have these schemes running on a server. The data collected comes from the on board sensors each having their own difficulties in acquiring the data. The entirety of the data acquisition unit's code can be seen in Appendix. B.

4.2.1 Bluetooth Functionality

The Bluetooth functionality is achieved utilizing a BLE library written specifically for the ESP32 by Neil Kolban [7]. The library allows for the advertisement of the ESP32 for Bluetooth connection as well as the basic functionality for giving the device an ID and transmitting/receiving data. The library in conjunction with some added logic allows for the basic functionality of detecting when a connection is made in order to toggle the blue LED to indicate to the user a connection has been successful. The disconnection of a device is also detected and used to turn the red LED on to indicate that the device is not currently transmitting any data as no devices are connected. When a device is disconnected the data acquisition unit immediately begins the advertising process again.

All of the data is converted into a JavaScript Object Notation (JSON) before transmission for ease of use on the website end of the fall detection calculations.

4.2.2 Accelerometer and Gyroscope Data Collection

The accelerometer and gyroscope requires calibration before they can be used effectively. This is because the accelerometer has an offset in each direction based on how the device is sitting in the case. The offsets can be calculated by first placing the device flat on a table and calculating the average value of each axis of the accelerometer (x,y and z) and the same is repeated for the gyroscope. The average is then applied to each axis and the process is repeated until the values in each direction are close to their expected values which for the orientation of the sensor in the case is 0 in the x and y directions and approximately 8192 in the z direction for the accelerometer and 0 in all directions for the gyroscope. The reason the accelerometer is 8192 instead of the $9.8ms^{-2}$ as expected for the acceleration due to gravity is for two reasons. Firstly, the accelerometer measures the acceleration in millimetres per second squared meaning that the acceleration due to gravity would be measured to be $9800mms^{-2}$ giving a very high accuracy. The value is also 8192 because the MPU6050 uses a 16 bit ADC (15 bits for the value and 1 bit for the sign) to calculate the acceleration and so for the range of $\pm 2g's$ gives $2g = \frac{2^{15}}{2} = 16384$ which in turn means that $1g = 9.8ms^{-2} = 8192$. This allows for the conversion from bit value to the range the accelerometer is set to. A similar calculation is repeated for the gyroscope.

The accelerometer, gyroscope, yaw, pitch, roll and a gravity vector pointing in the direction of gravity are all collected/calculated using a library written by Jeff Rowberg [43]. This library allows for the ease of data collection for the device with a simple to use API. The library also handles collecting the more advanced data such as yaw, pitch, roll and the gravity vector which is calculated on the MPU6050 chip using a Kalman filter. These more advanced features are only utilized for graphing purposes and are not included in the fall detection methods.

4.2.3 Barometer Data Collection

The barometer used can obtain two pieces of data, the temperature of the barometer and the atmospheric pressure. These values can be used to calculate the current altitude above sea level, using a constant value as the reference for sea level atmospheric pressure. The reason a constant can be used for the sea level calculation is because only the relative change in altitude is important and the exact height above sea level is irrelevant in determining if a fall has taken place. Retrieving the temperature and pressure values from the barometer was achieved using an external library that handles the i2c communication for requesting the data [44].

The altitude calculations are sensitive to changes in temperature and so the temperature of the barometer can be used to compensate for these fluctuations. Two methods of altitude calculation are used, they are the temperature compensated altitude and the non-temperature compensated values which are displayed in Equation (18) and Equation (19) respectively. Each calculation relies on a number of constant values shown in Table. 1.

$$h = \frac{T}{L}(\frac{P}{P_0}^{-\frac{LR}{g}} - 1) \tag{18}$$

$$h = \frac{T_0}{L}(\frac{P}{P_0}^{-\frac{LR}{g}} - 1) \tag{19}$$

A full list of the constants and variables used in these equations can be seen in Table. 1.

These formulas worked very well for the calculation of altitude however they were subject to enough noise to cause a ±0.5m change in altitude even while stationary (which of course stemmed from the noise in the temperature and pressure measurements) which in many situations could be enough to indicate a fall. It was apparent that the noise must be filtered out in order to get accurate altitude measurements. A number of test data samples were collected where the change in height had been measured to use as a reference against the filtered values. This data was used to indicate the performance of each filter. A few filtering methods were explored beginning with the alpha-beta filter described in Section 2.3.

The alpha-beta filter was very simple to implement boiling down to simple multiplications, additions and subtractions. The alpha and beta tuning values were calculated using the equations found in Equation (15) and Equation (16). The noise and process variances were experimentally tuned until a desired performance was met. The results of the alpha beta filter being applied to both the temperature and the pressure values and the resulting altitude calculations vs the measured altitude can be seen in Fig. 11. The results of the filter on the temperature and pressure values can be seen in Fig. 12. The process variance σ_w was found to be experimentally determined to be 1.5 for both the temperature and pressure and the noise variance $sigma_v$ was experimentally determined to be 0.04 and 0.8 for the pressure and temperature filters respectively.

Symbol	Value	Units	Description
P_0	1013.25	kPa	Pressure at sea level
T_0	288.15	K	Temperature at sea level
T	-	K	Current temperature
g	9.8	ms^{-2}	Acceleration due to gravity
L	-6.5×10^{-3}	Km^{-1}	Lapse rate
R	287.053	$J(kgK)^{-1}$	gas constant for air
h	-	m	calculated height above sea level

Table 1: All Constants and Variables Needed for Altitude Calculation [45]

It can be seen that each of the filters performs very well, however the temperature compensated filter was found to be the most accurate to the real life altitude drop of 0.8m. There were still some inherent spikes in the pressure data and so a simple saturation filter that ignores any changes in pressure greater than 0.2 from the previous time step was also implemented.

The extended version of the alpha-beta filter, the alpha-beta-gamma filter was also tested, however the results were very poor often resulting in a large amount of overshoot and undershoot [46]. This transient behaviour was very difficult to control even with very fine tuning of the process and noise variance parameters. This behaviour can be seen in Fig. 13. The alpha-beta-gamma filter was implemented in the exact same way on the same tests as the alpha-beta filter in order to maintain a fair comparison. The code for achieving this was adapted from a site written by Microstar laboratories and is simply an extension on to the alpha-beta filter assuming constant acceleration rather than constant velocity [46].

The last filtering method tested was the Kalman filter. The Kalman filter is an optimal state observer that is often used in control systems. A method involving the gravity vector (described in Section 4.2.2), the accelerometer values and the measured altitude from the barometer to calculate the true altitude was investigated. The states of the Kalman filter are described to be the vertical height, velocity and acceleration (h,$\dot{h}, \ddot{h}$)

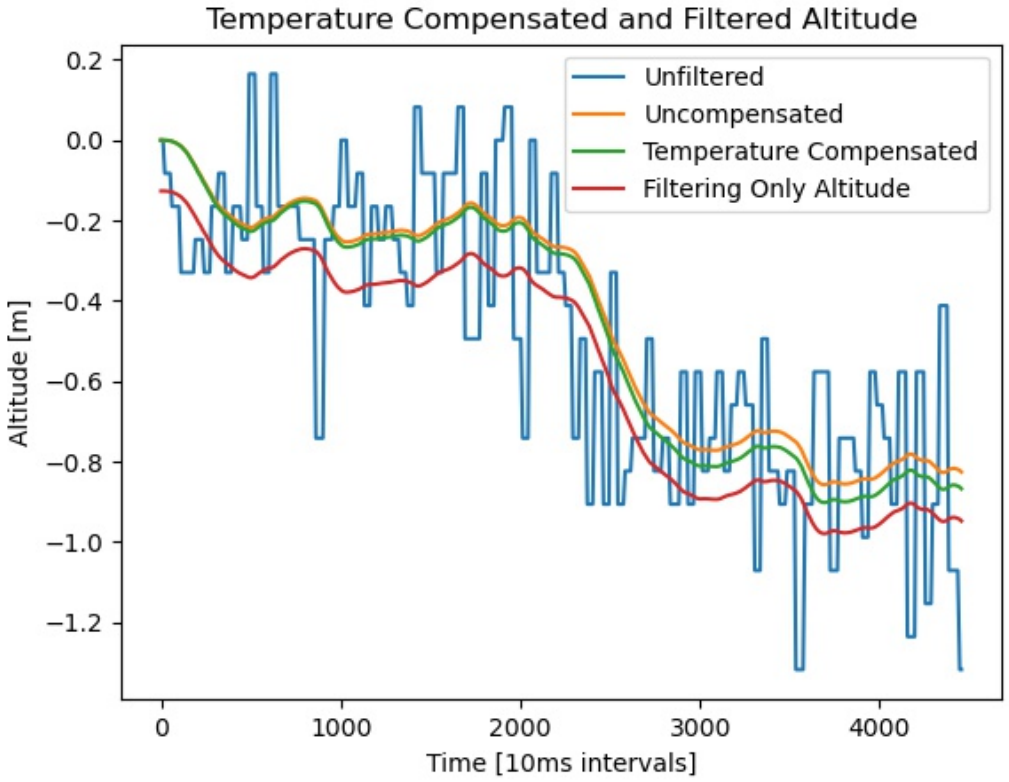


Figure 11: The Unfiltered Altitude, The Altitude Without Temperature Compensation, Temperature Compensated Altitude (Filtering Pressure and Temperature) and Filtering Only the Altitude

as shown in Equation (20). This method first calculated the linear acceleration towards the ground of the data acquisition unit by taking the dot product of the gravity vector and the acceleration vectors and subtracting off the gravity component. This results in the acceleration being felt by the device towards the ground during a fall. The unfiltered altitude is then used as the height measurement and a state matrix A can be formed. The state matrix can be seen in Equation (21) where dt is the time step (in this case 10ms). The output matrix C can be found in Equation (22) which is based off of the 2 states that are known from measurements (the acceleration and the altitude).

$$x = \begin{bmatrix} h \\ \dot{h} \\ \ddot{h} \end{bmatrix} \tag{20}$$

$$A = \begin{bmatrix} 1 & dt & \frac{dt^2}{2} \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \tag{21}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{22}$$

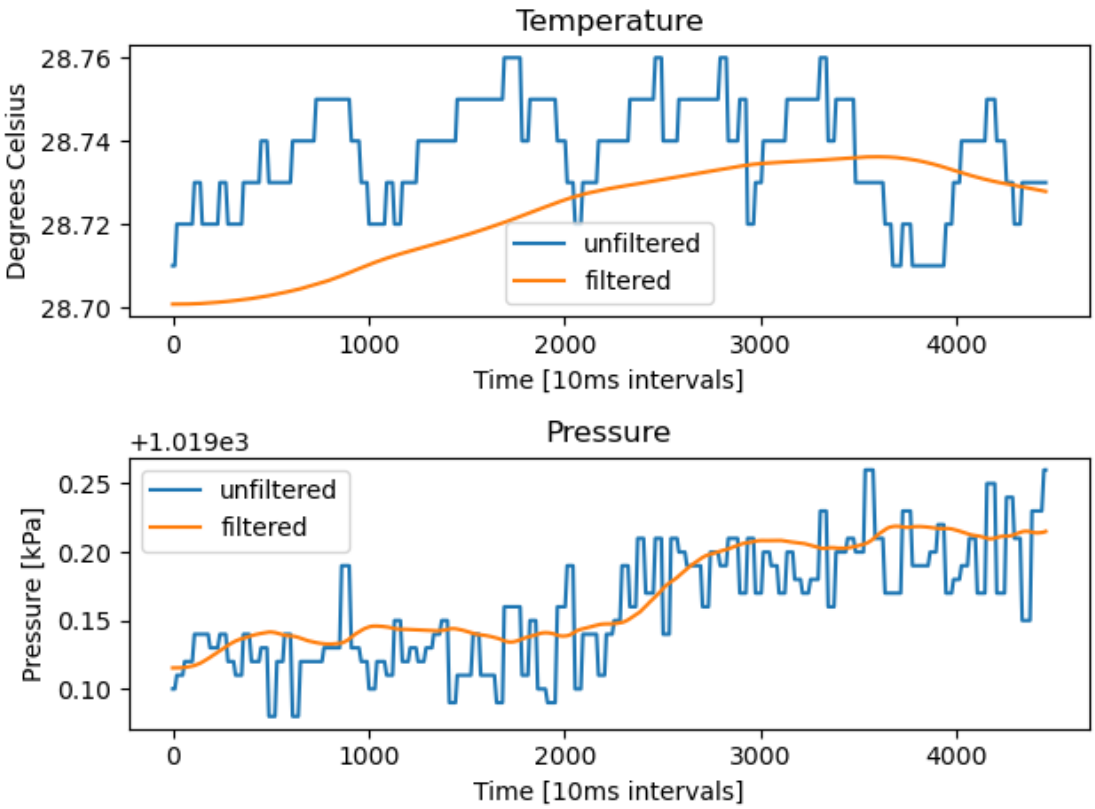


Figure 12: Results of Alpha-Beta Filter on Pressure and Temperature

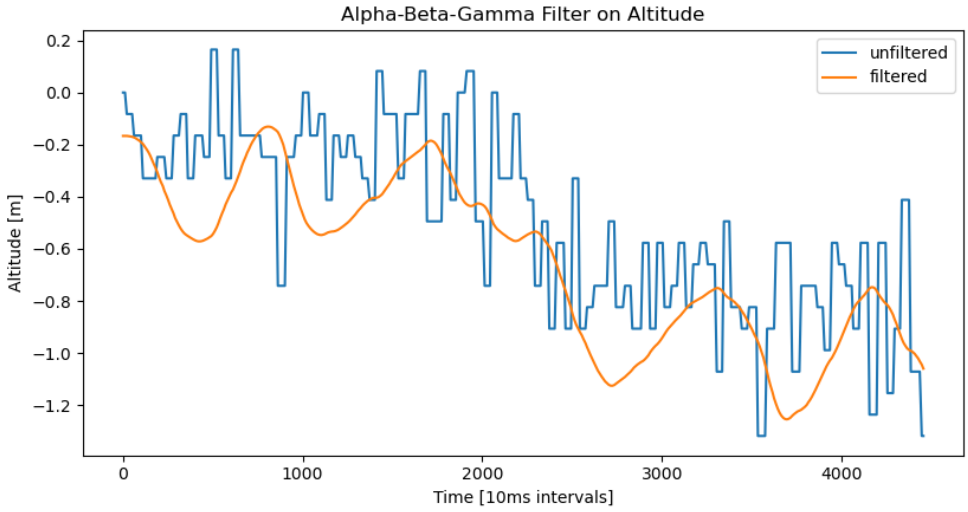


Figure 13: Results of Alpha-Beta-Gamma Filter on Altitude

All initial values for the states and for the state error covariance matrix (P) was set to zero. The process noise covariance and the state covariance can be seen in Equation (23) and Equation (24) respectively and were experimentally determined.

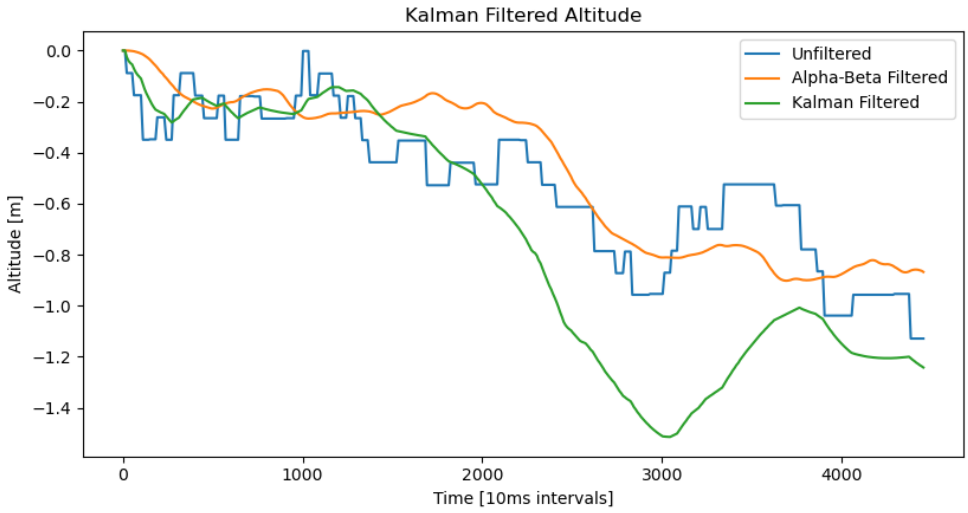


Figure 14: Kalman Filter for Altitude Estimation

$$Q = \begin{bmatrix} 0 & 0 & 0.0005 \\ 0 & 0.0005 & 0 \\ 0.0005 & 0 & 0 \end{bmatrix} \tag{23}$$

$$R = \begin{bmatrix} 0 & 0.5 \\ 0.5 & 0 \end{bmatrix} \tag{24}$$

The standard predict and update Kalman steps were then implemented in the Python programming language using the library Numpy for all matrix operations [47][48]. This resulted in a filtered altitude as shown in Fig. 14. It can be seen that the Kalman filter although theoretically should perform better than the alpha-beta filter, it resulted in large over and undershoot even with fine tuning of the covariance matrices.

It is for these reasons that the alpha-beta filter was implemented to give an accurate depiction of the altitude at any given moment in time.

4.3 Types of Data Collected

Obtaining a wide range of different fall scenarios as well as activities of daily living (ADL) is extremely important in obtaining valid results for fall detection. The ADL tasks included many exercise based measurements in order to allow for better indications of what exercises may be mistakenly detected as a fall allowing and to adapt the fall detection methods to account for this. A total of 480 measurements were taken. A complete list of the activities of daily living and the falls measured can be seen in Table. 2.

Data Collected			
Falls		**Activities of Daily Living**	
Action	**Number of Tests**	**Action**	**Number of Tests**
Backwards Fall	20	Laying Down	20
Roll Out of Bed	20	Laying Still	20
Downstairs Fall	20	Lifting Objects from Ground	20
Frontwards Fall	20	Push-Ups	20
Leftwards Fall	20	Running Up/-Down Stairs	10
Rightwards Fall	20	Sitting Down Fast	20
Running Dive	10	Sitting Down Slow	20
Running Fall	20	Squats	20
Straight Drop	20	Standing Still	20
Staggered Fall	20	Star Jumps	20
Upstairs Fall	20	Staggered Walk	20
Walking Fall	20	Walking	20
		Walking Up/-Down Stairs	20

Table 2: All Types of Data Collected Using the Data Acquisition Unit

4.4 Data Storage

In order to obtain and store the data acquisition unit data, a basic website and back-end server was created. The website was simply written using HTML, CSS and JavaScript utilising the charting library Chartjs for displaying the received data [49]. The website was able to connect to the data acquisition unit using BLE utilising a web Bluetooth API which allows for communication to BLE devices through the browser [8]. The website appends a Boolean value indicating whether a fall is occurring or not to the data acquisition unit data. The website collects this data for 5 seconds and then transmits it using a HTTP post request to the back-end server which is a Flask server written in the coding language Python [50]. The server takes the 5 seconds worth of data and adds it to a CSV file for data storage. The saving to a CSV is accomplished using the Python library Pandas which has a very easy to use API for dealing with CSV files [51].

The website includes a connect button which connected the site to the data acquistion unit, a text input allowing for the naming of the CSV file where the data will be stored and buttons to begin and stop measurements. The website also allowed the user to hit space bar to toggle whether a fall was occurring or not which directly influenced the Boolean fall indicator stored in the CSV file. This allowed the user to watch the test

Figure 15: Data Collection Website

taking place and toggle the fall indicator to true while a fall was occurring and toggle it back once the fall had finished. The magnitude of the acceleration data could also be viewed in a graph on the website to ensure the data was transmitting correctly. This could also be toggled to stop updating the graph when not needed. The basic data collection website can be seen in Fig. 15.

The Boolean fall indicator is to allow for the data to have an accurate depiction of when a fall occurred. The user of the website would toggle the fall indicator to true while a fall is occurring (the moment they noticed the fall had started) and was toggled back to false after the impact had stopped and all motion had ceased.

Chapter 5: Software Architecture

The software within this paper integrates directly to the hardware with various communication protocols. The overall software design can be seen in Fig. 16. Each component described in Fig. 16 will be discussed in the following sections.

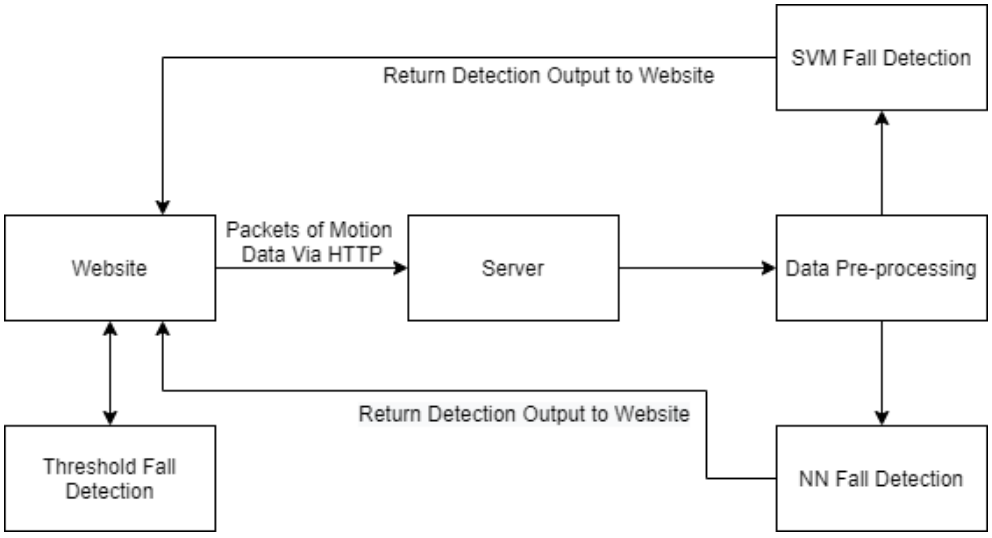


Figure 16: Software Overview

5.1 Threshold Method Fall Detection

5.1.1 Data Preparation

The data collected with the data acquisition unit must be prepared before the threshold methods can be applied. Firstly the magnitude of the acceleration and angular velocities were normalised to make choosing the thresholds for the model easier. The data was then collated into two folders, one with all the falls and one with all the non-fall CSV files. The Scikit learn function "train_test_split" was then used to split each of these folders into a 70:30 split of training data vs test data [11]. The reason this was done with the two folders separately is to ensure that the training and test data had the same number of falls as non-falls. All of the test data was then combined and all of the training data was then combined resulting in 70% of the data being used for training and 30% being used for testing with both having equal numbers of falls and non-falls. It is important that the data is split into training and test data to ensure that the threshold methods haven't just been tuned specifically for the data set it was trained on but can generalised to data it has never seen before.

5.1.2 Extending Baseline Algorithm with Height Information

The baseline algorithm presented in Section 2.1.1 is a very logical approach to fall detection however there is a key piece of information missing and that is the height. The height of the user is a key piece of information in determining if a fall has taken place. It will not be helpful in determining if a fall is currently occurring due to the relatively slow update time of the barometric pressure. However, it can be a very good indication of if a fall has already taken place. The way this algorithm works is to measure perform the same baseline algorithm; Check if the magnitude of the acceleration dips below a set threshold and if so start a timer. However, where this algorithm differs is that as soon as the algorithm detects the dip in acceleration the current height is also recorded. The same baseline algorithm continues.

The accelerometer and gyroscope magnitudes are continually monitored constantly remembering the highest/peak value obtained for each. As soon as the timer is complete the current height is recorded and the change in height over the time period is calculated from the initial height – current height. The three values, acceleration magnitude, gyroscope magnitude and the change in height are now all compared against their thresholds. If the gyroscope magnitude surpasses its threshold and the accelerometer surpassed and the height surpassed then a fall occurred.

The height threshold gives an extra element of accuracy detecting the outlier cases that the upper threshold of the accelerometer cannot. An overview of the logic structure can be found in the flow chart in Fig. 17.

Some metrics for the accuracy of each method are calculated such as the number of true positives, true negatives, false positives, false negatives, sensitivity, specificity and percent correct in training and test sets. The method is tested against each data sample (each CSV file) and it is decided whether that test was a true positive, false positive, true negative or false negative. The way this is decided is, if a fall was detected but there was no fall within the data sample then a false positive was detected. If the entire data sample was iterated over and a fall wasn't detected then two conditions are checked. If there was a fall in the data sample then a false negative was detected, if there wasn't a fall then a true negative was detected. Finally, if a fall was detected, there was a fall in the data sample and the fall didn't occur after it was detected (meaning the fall was prematurely detected) then a true positive was detected however if the fall was detected before it happened then a false positive was detected. Percent correct was calculated by dividing the number of true positives plus true negatives by the number of data sets in either the training or test set (depending on which set is being tested). The sensitivity and specificity was calculated as shown in Equation (3) and Equation (4) respectively.

5.1.3 Applying Genetic Algorithm

A genetic algorithm, being an optimisation algorithm is the ideal algorithm to use in the selection of thresholds. The genetic algorithm technique was used in conjunction

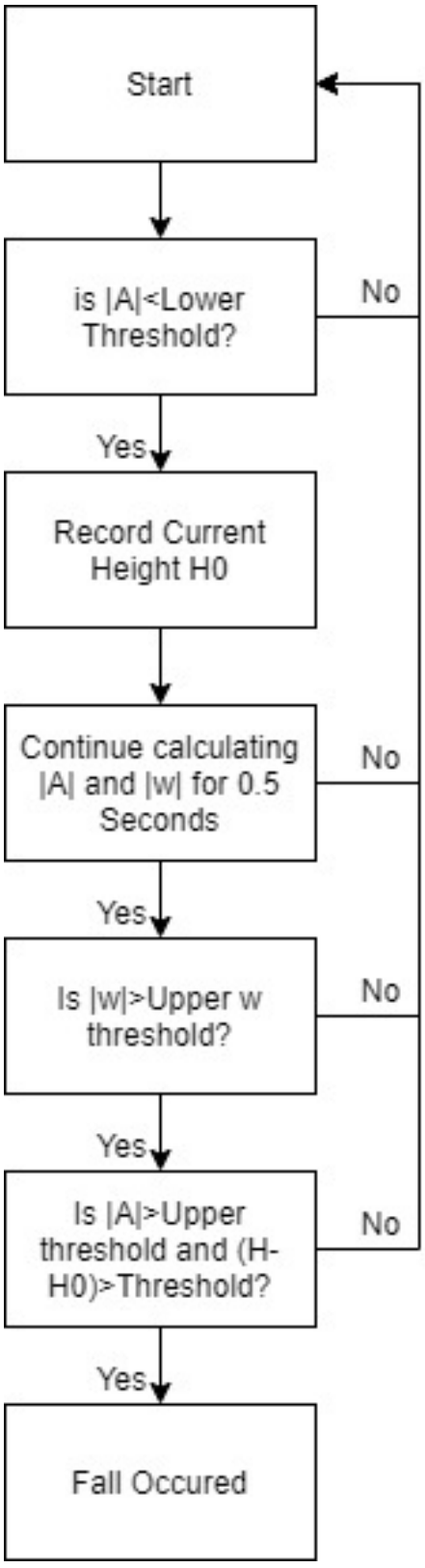


Figure 17: Extending the Logic of the Baseline Method with the Addition of Height

with the threshold methods to more effectively decide the thresholds for each set of data. A genetic algorithm has 4 tuning parameters, the fitness function, the mutation

rate, the population and the number of generations to perform.

The fitness function was varied from test to test first beginning with linear fitness functions that rewarded true positive results and true negative results. With a higher weighting on the true positive results because it is preferred that all falls get caught with some false alarms than to miss a fall all together. The linear fitness function can be seen in equation Equation (25).

$$F = 3tp + 2tn \tag{25}$$

(F = fitness score, tp=number of true positives, tn= number of true negatives)

Upon further research an exponential fitness function was found to converge much quicker than a simple linear function. This is because that even small changes in the result for true positives and true negatives will yield a much greater difference in fitness score which will give that member of the population a much greater chance of being chosen to reproduce for the next generation of thresholds even if the total results of that member is not vastly greater than the other members [52]. One exponential fitness function used is shown in equation Equation (26).

$$F = e^{\frac{tp+tn}{2fn+fp}} \tag{26}$$

(fn= number of false negatives and fp = number of false positives)

Equation Equation (26) is designed to increase the fitness score for true positive and true negative results and penalise the fitness score for false positives and false negatives. The false negative values are scaled to be more important than false positives because as previously mentioned it is more desirable to correctly identify every fall with some false alarms than it is to let a fall go unnoticed.

The final fitness function attempted is based on equation Equation (5) [9]. This fitness function is designed to perform the minimisation of the distance described in equation Equation (5) using the exponential function. This fitness function can be seen in equation Equation (27).

$$F = e^{\frac{1}{\sqrt{(1-Sn)^2+Sp^2}}} \tag{27}$$

(Sn = sensitivity Equation (3) and Sp = specificity Equation (4))

The mutation rate was set to a constant 0.2 with a total population of 100 members. These values were found experimentally to give the fastest convergence time to a solution. The baseline and extended baseline threshold methods were allowed to run for 250 iterations each. On each iteration the member of the population that had the highest percent correct from the training data was deemed the best member of the population for that iteration. If the current best member was also better than any previous best member then it was deemed the overall best and was validated against the test data set. Information about the overall best member of the population such as the thresholds chosen, training percent correct, test percent correct, specificity, sensitivity

and the time from the beginning of the fall that the member took to detect that a fall had occurred was then logged in a CSV file.

5.2 Support Vector Machine Fall Detection

5.2.1 Data Preparation

Data preparation is a necessary first step in any machine learning task. The data must be prepared in order to improve the results of the machine learning model but also to reduce the computational expenses involved with training a model. The fall detection data set described in Chapter 4 is no different.

The first step in data preparation is to determine what features of a data set should be included during the training of the machine learning model. A logical choice for the fall detection data set would be the acceleration and angular velocity in the x,y and z directions and their magnitudes as well as the atmospheric pressure. The atmospheric pressure is directly related to the altitude as described in Section 3.2.3 and so to save on computation of the altitude, the machine learning model can just take the pressure in as an indication of altitude. The labels for this data set is simply the Boolean fall parameter which is simply a 1 while a fall is occurring and a 0 when a fall is not occurring.

A number of methods for structuring the data after applying normalization was explored during the creation of this fall detection method. One approach was to simply pass in each 10ms data sample as an input to the machine learning model and use the corresponding value in the fall column as the label for training. This however did not yield good result (as described in Section 6.2.1) because the model is trying to predict if a fall occurs based on only a 10ms snapshot of data. Another approach included passing in multiple time steps and setting the label for training to simply be the last time steps fall value (0 or 1). This idea is further illustrated in Fig. 18. This means that instead of the model having to make a prediction based on only the current time step it can look at the previous x time-steps to predict whether a fall is currently occurring.

A time series class was written in the Python programming language to complete this data prepping process and for the training and testing of the SVM which is discussed in Section 5.2.2. The time series class uses the Python libraries Numpy and Pandas to aid in array/ CSV manipulation [51][48]. The class starts by taking in a folder with CSV files and goes through each file one at a time. Each time-step/ column of the CSV file is added to a queue which has a max sequence length. This sequence length is the total number of time steps the machine learning model will learn from. Once the queue is full the queue is added to an array of input data and the output label is added to an output array. As the next time step comes in, the first element of the queue is removed keeping the total queue length the same and the queue is again appended to an input data array and the output label is added to an output array, creating a sliding list of input/output data across the entire CSV file. This is then repeated for all of the CSV

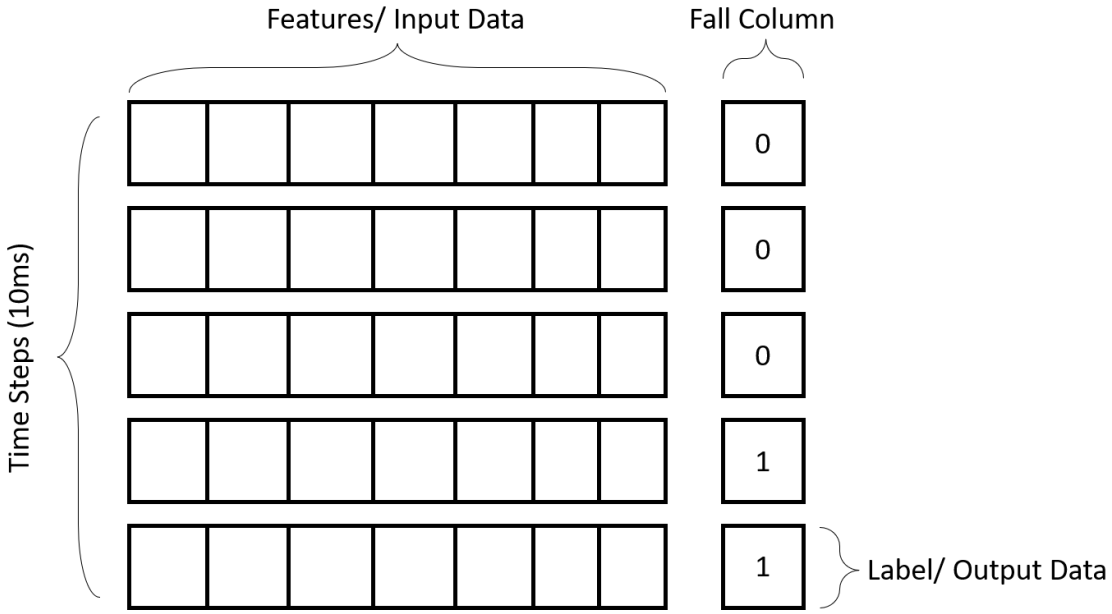


Figure 18: Using Multiple Previous Time Steps of Data to Predict the Current Output

files in the folder. This gives a multidimensional array that contains a set of inputs that each are a 2 dimensional array containing the measurement data for the amount of time-steps specified and each have their corresponding output label in an output array.

Certain features of the input data have inherent offsets associated with them such as the pressure feature. The offset at the beginning of each time interval (number of time steps) is removed by a simple subtraction.

The next step in pre-processing the data is normalizing all of the data to a range of 0 to 1 to allow for faster training by the machine learning models. The normalization method used in this paper is min-max normalization as described in Equation (17). This normalization was applied to all features of the data set.

The data is then shuffled (while keeping track of which input data corresponds to what output data) using the Sci-kit Learn function "shuffle" [11]. The data is then split into training and testing data gain using the Sci-kit Learn libraries function "train_test_split" [11]. The data was split into a 60:40 split of training to test data to allow for enough variance of data within the test set. This data can is then ready to be fed into the SVM model. The time series class written by the author can be found in Appendix. C.

A limitation of this approach is that on the authors current computer only 100ms of data samples can be used for training due to how large the input data set gets. A single input is the number of features multiplied by the number of time steps big and so loading all of that data into memory proved impossible. This is the motivation for principle component analysis in the fall detection data set.

Principle Component Analysis was achieved using the Sci-kit Learn API which allows

for ease of use by creating a PCA object and defining the number of principle components desired and fitting the PCA object on the training data. Once the object has been fit it can be used to transform a set of inputs to 25 principle components as described in Section 2.4. 25 principle components was found to be sufficient in describing the data set completely while still maintaining enough memory on the authors laptop. The input data sets which were very large are now only 25 features long allowing for much more efficient training and for the ability to train the model on much longer time periods. The model can be trained on this transformed input data. The test data must be transformed using the same PCA object that was fit with the training data for reasons described in Section 2.4.

5.2.2 Implementing Support Vector Machine

Training a SVM is achieved through the use of the Sci-kit Learn library and the svm.SVC class. The svm.SVC class is a support vector machine classifier and is used to generate the support vector machine for classification tasks. The svm.SVC class takes in the kernel type, C value and the γ value as described in Section 2.1.2.

After the data is prepared as described in Section 5.2.1 a SVM model can be created and fit against the training data using the fit method of the svm.SVC class [11]. This method takes in the training data features and labels as inputs and will train the SVM on them. Making a prediction with the SVM is achieved by calling the svm.SVC classes method "predict" which takes in an input data sample and returns the prediction (1 for a fall and 0 for a non-fall). The code that implements this can be seen in Appendix. D.

The time series class written by the author has built in methods for evaluating the SVM's accuracy and for evaluating the model on the test data. The SVM's accuracy is characterised by the number of true positives, true negatives, false positives, false negatives, total percent correct, sensitivity and specificity.

The SVM is implemented for live use by saving the pre-trained model for the SVM and for the PCA and using those models to convert the current streamed input data into the 25 principle components. The prediction can be made on this data and returned as a result. The prediction step is very fast once the model has been trained which makes this method viable to be running in a live situation.

5.3 Neural Network Fall Detection

5.3.1 Data Preparation

The data preparation for neural networks is almost identical to what is described in Section 5.2.1, multiple time steps were used (2.25 seconds) and PCA with 25 principal components, with one added step. The fall detection data set is heavily imbalanced

with a ratio of approximately 10:1 of non-fall data to fall data. This means that in machine learning methods such as NN the learning is heavily skewed/biased towards the majority set which in this case are the non-falls [53]. SVM does not face this issue thanks to the way it functions as described in Section 2.1.2.

A method of overcoming this is through under-sampling. Under-sampling is discarding data from the majority class such that both classes have the same number of data points. This would not work in a fall detection system as the non-fall data contains important information about how to avoid false alarms (detecting a fall when one did not occur).

The method used in this paper is oversampling. Oversampling is synthetically creating more data samples of the minority class such that both classes have the same number of data points. There are many methods of oversampling, however the one implemented in this paper is simply to randomly duplicate members of the minority class until both classes are the same size. This was achieved using the Imbalanced Learn library [54].

The oversampling was performed on the data after PCA had taken place, this is simply because the authors computer could not store enough data in memory to perform oversampling on the original data set and also because the minority class is already present in the PCA data and so duplicating the PCA results would have the same effect. With the data set successfully balanced it is now ready to be used to train the NN.

5.3.2 Implementing Neural Network

The implementation of the neural network was achieved with the use of the Python library TensorFlow [12]. TensorFlow is a professional API for machine learning used by many companies and funded by Google. It is open source and empowers its users to perform numerous machine learning tasks using its API and learning resources.

TensorFlow has a model based API where a neural network model is defined and can then be trained and used for prediction. The architecture of the model and what hyper parameters (as described in Section 2.1.3) are chosen. The model can be trained on the training data and validated using the test data with help from the TensorFlow API.

Many different models were experimentally determined with varying hyper parameters such as different activation functions, layer sizes, dropout per layer, optimisation algorithms and the various options available for each optimisation algorithm. The epoch size was kept at a constant 1000 (to give each model the same number of iterations through the data set) and the batch size was tweaked experimentally as well. This naive approach can be used to obtain decent results however the tuning of hyper parameters is a very time consuming task and so an optimisation method such as genetic algorithms can be applied to better determine these hyper parameters.

The optimisation of the hyper parameters of a neural network can be achieved with the use of a genetic algorithm to repeat iterations of different model architectures. After

Parameter	Range/Options
Optimisation Algorithm	"Adam", SGD, AdaMax, Adagrad, RMSProp (and all associated parameters ie. learning rate and momentum)
Activation Function (per layer)	relu, sigmoid, tanh, leaky relu
Number of Layers	Range of 1-5
Number of Perceptrons per Hidden Layer	Range of 4-150
Dropout Rate	Range of 0-0.5 (This is a percentage)
Batch Size	Range 10-2000

Table 3: All Parameters Tuned by Optuna

each iteration the architectures accuracy can be evaluated and treated as its fitness score allowing for the optimisation of arbitrary architectures. A purpose built library Optuna can be utilised to optimise the hyper parameters [13]. Optuna allows for the definition of what parameters should be included as variables and use them as the parameters to be optimised. The parameters that will be optimised are given a range (or in the case of a categorical value, a list of options) that they can exist in to narrow down the search space. All of the tests were given 300 iterations to converge into an optimal solution for the model. The parameters and the ranges/options can be found in Table. 3.

After an optimised model was determined and trained the model is ready for making predictions. A prediction can be made using the TensoFlow API call "predict". This predict method returns a value ranging from 0-1 with 1 corresponding to a fall and 0 corresponding to a non-fall. The prediction output can be thought of as a confidence score as to the current set of input data being a fall. The closer the number is to a 1 the more confident the model is that the data set was a fall. The same is true for the non-fall case. This allows for the tuning of the fall detectors sensitivity as an input data set could be considered a fall if the model's confidence score is greater than 0.9 to make it less sensitive to potential falls and greater than 0.2 to make it more sensitive (these numbers are examples and can be tuned manually on the website as described in Section 5.4).

5.4 Website Design

5.4.1 Overview

All of the fall detection methods previously discussed require a user interface to allow for the operation by anyone. The user interface is incorporated into a website written using the Angular framework [55]. The Angular framework is a component based web-development framework that treats each object on the screen as its own component with its own set of styles and code that allows it to function. Angular also allows for the functionality of something known as a service. A service is a piece of code that may be reused by any component within the website.

The website is constructed with a home page which displays info about the website as well as provides information about the tabs that can be moved to. Upon clicking on another tab a window will pop up asking if the user would like to connect to the data acquisition unit. This is achieved through the use of an Angular service. The service is what controls the connection, disconnection and retrieval of data from the data acquisition unit. Once a connection is established the data is available on that tab of the website (note upon moving to a new tab the connection must be authorised again as part of the requirements of the web Bluetooth API [8]). The Bluetooth connection is achieved in the same way as described in Section 4.4 however the code was ported from JavaScript to TypeScript (the native language of Angular) and written as an Angular service such that every tab on the website can access the data.

If entering the support vector machine or neural network tabs, the device will transmit a HTTP Post request to the back-end server that informs the server which model to load for incoming predictions. This is simply done by transmitting either "SVM" or "NN" as a string to the back-end.

5.4.2 Data Acquisition

The data acquisition tab is provided to give a view of all of the data being streamed by the data acquisition unit and graph it in real time using the graphing JavaScript library ChartJs [49].

This tab of the website allows for the selection of which piece of data from the data acquisition unit should be graphed with the use of a drop down selection menu. The piece of data that was selected will be graphed on the y-axis with time on the x-axis. It is important to note that the time on the x-axis is time in milliseconds since the data acquisition unit was turned on.

A maximum of 150 data points are displayed at any one time and will act as a queue from then on, removing the first data point to make room for the latest incoming data point. This is simply to save memory on the device running the website and ensures that there is no lag as the website consumes more and more memory. The exact design

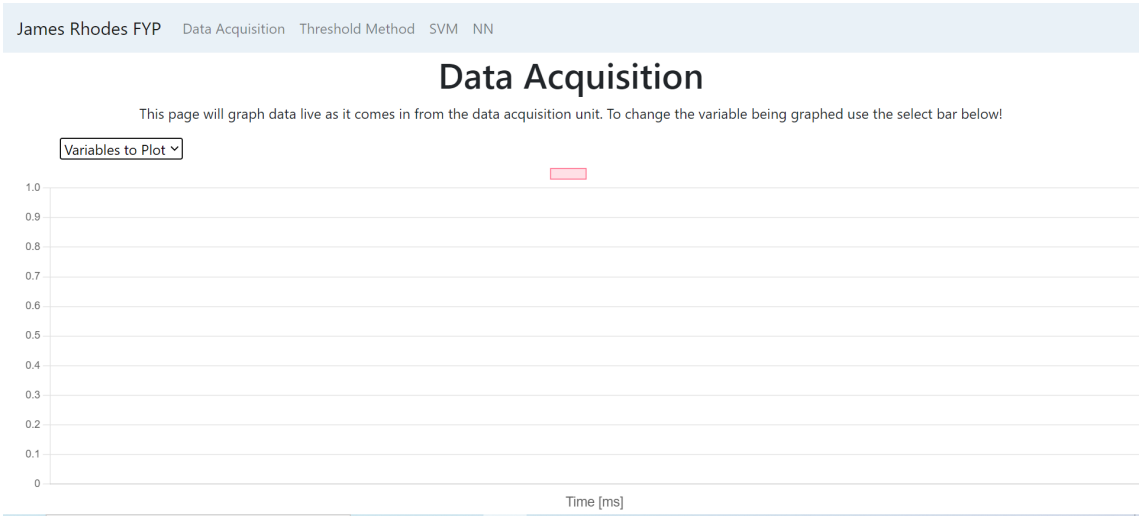


Figure 19: Data Acquisition Tab

of the data acquisition page can be seen in Fig. 19.

5.4.3 Threshold Method

The threshold method tab contains a brief overview of the threshold method, some sliders to alter the thresholds manually (for experimentation by the user), a button to reset the thresholds back to their original values and a block diagram of how the algorithm works.

The button resets the thresholds back to the thresholds determined by the genetic algorithm. The user can manually adjust these thresholds to allow for the experimentation of each parameter and to explore what parameters increase and decrease the sensitivity.

The threshold method algorithm is the same as described in Section 5.1.2 and is run locally in the browser. This is to allow for the information to be more easily changed by the user and also to take the computational load off the data acquisition unit, especially seeing as the unit will not always be required to implement the threshold method (such as when exploring other tabs of the website).

When the algorithm detects a fall a warning sign will completely cover the screen and remain covering the screen for 5 seconds after the fall was detected. This is to demonstrate to the user that a fall has occurred and also to allow further experimentation once the 5 seconds has finished. This tab of the website can be seen in Fig. 20 and the warning page can be seen in Fig. 21.

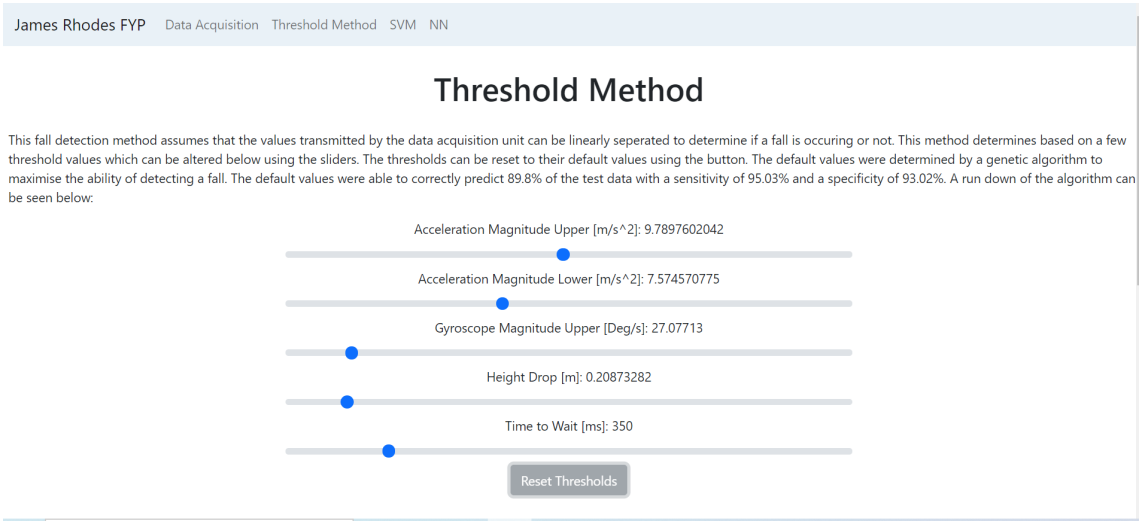


Figure 20: Threshold Method Tab

Figure 21: Fall Detected Signal

5.4.4 Support Vector Machine

The SVM tab is a relatively simple user interface which simply describes the function of the SVM and the results gained from it. The page behind the scenes is accumulating the data recieved from the data acquisition unit in a buffer and transmits that buffer (once full) via a post request to the back-end described in Section 5.5.

To minimise the number of HTTP requests made the device accumulates 200ms worth of data and then transmits it to the back-end which has its own queue of data that predictions can be made on as described in Section 5.5. This is preferable over immediately making a HTTP request every time a new data entry is received (every 10ms). The results of the SVM are determined by the back-end and returned to the front-end for display.

Once a fall is detected the same sign as shown in Fig. 21 is displayed for 5 seconds after the fall was detected and then the page returns to normal. The entire SVM tab

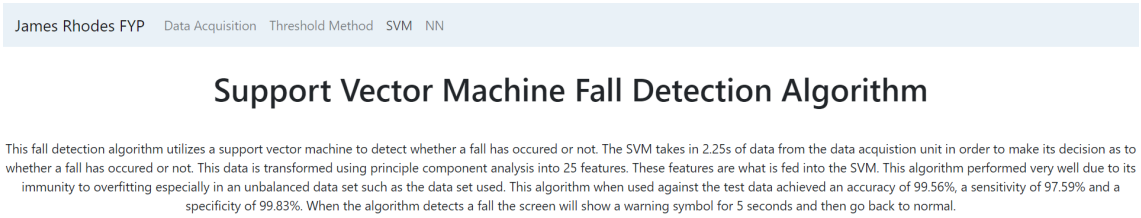


Figure 22: Support Vector Machine Tab

can be seen in Fig. 22.

5.4.5 Neural Network

The neural network tab contains a brief overview of the functionality of the fall detection method as well as a slider that controls the confidence threshold and a graph of the current confidence score as well as the confidence threshold.

The confidence threshold can be altered by the user in order to change the sensitivity of the device. The closer the confidence threshold is to 100% the less sensitive the device becomes and the closer to 0% the threshold becomes the more sensitive it is. This value is altered by the slider on the page and can be seen in the graph. The current returned value from the neural network (the confidence value) based on the current data is graphed live along side the confidence threshold.

The neural network fall detection algorithm is running on the back-end and data is transmitted via a HTTP Post request every 20 incoming data samples (200ms) similarly to the SVM tab as described in Section 5.4.4. The confidence value is returned by the back-end for display on the graph. When the confidence value is greater than the confidence threshold, a fall is triggered and the same warning sign shown in Fig. 21 is displayed for 5 seconds from the time the fall was detected and then the page will return to normal. The entire neural network tab can be seen in Fig. 23.

5.5 Server Design

The server that runs the machine learning methods (SVM and NN) is written in Python using the server framework Flask [50]. The server contains two available routes that can be called upon "/initModel" and "/Predict". The "/initModel" route will accept a Post request containing a string that specifies what model (SVM or NN) to initialise. The "/Predict" route accepts a Post request containing the data transmitted by the

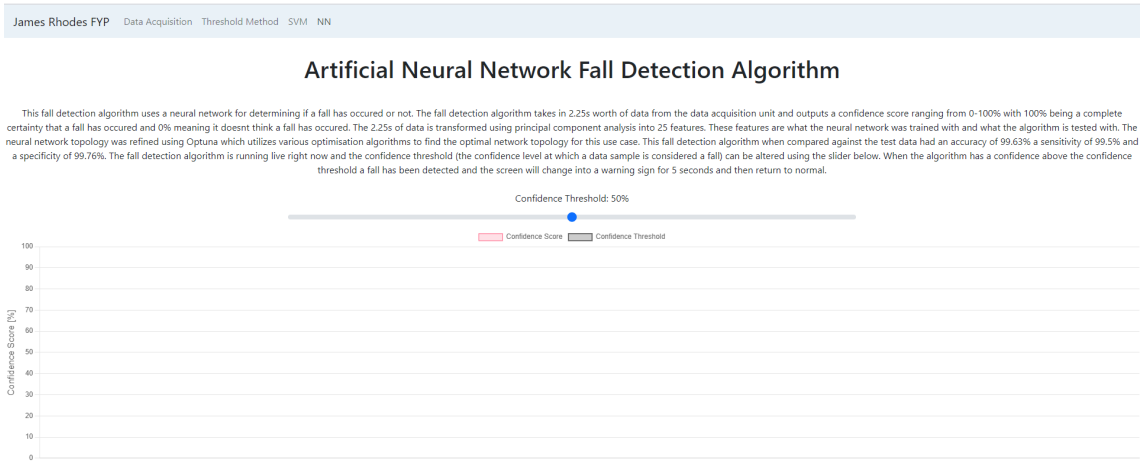


Figure 23: Neural Network Tab

front-end and returns the output of the corresponding machine learning model to the front end server.

The actual machine learning tasks are performed by a fall detection class written by the author. The class is an abstract base class that contains the relevant methods for pre-processing the data as described in Section 5.2.1 and Section 5.3.1. This means the abstract base class also contains the buffer (a queue) of data that each model requires for prediction. On every incoming data packet from the front-end the buffer/queue will release the data at the start of the buffer/queue and append the new data to the end such that the total length of the data matches the amount of time steps required by each model for prediction. An abstract base class is a class that contains methods relevant to classes that inherit from it (children of the base class) and methods that must be implemented by its children. This allows for a separate SVM and NN class that extends the functionality of the base class. This is very useful as it allows for less code duplication as the pre-processing for both SVM and NN are the same and the only methods that change are the predict and initialise methods.

The initialise method for the SVM child class loads in the PCA model required and also loads the pre-trained Scikit-Learn SVM model itself [11]. The predict method is called on the data in the buffer (after pre-processing as described in Section 5.2.1) upon each new data sample received. The predict method is the same as described in Section 5.2.

Similarly the initialise method for the NN child class loads in the PCA model required and loads the pre-trained TensorFlow neural network model. The predict method pre-processes the data in the buffer (the same as for the SVM as described in Section 5.3.1) and calls the TensorFlow predict method on the buffer every time a new data sample is received [12]. The predict method is further described in Section 5.3.

The children classes NN and SVM make it easy to initialise the model upon each access of the ”/initModel” route, this is because when the route is called a global model object can be set to the NN or SVM as required and the name of the method (initialise or

predict) are the same for both classes (thanks to the abstract base class). This means the initialise and predict methods can simply be called on whatever class the global model object is currently set to. The initialise and predict methods are called in the "/initModel" and "/Predict" routes respectively. This makes the code much shorter and succinct minimising code duplication.

The outcome of each call to predict is returned to the front-end by encoding the result in the received message of the route itself. This means less HTTP requests have to be made as the result is returned as a response to each call to the "/Predict" route.

Fitness Function	Specificity	Sensitivity	Accuracy
$F = 3tp + 2tn$	86.05%	95.03%	87.07%
$F = e^{\frac{tp+tn}{2fn+fp}}$	86.13%	96.88%	87.07%
$F = e^{\frac{1}{\sqrt{(1-Sn)^2+Sp^2}}}$	87.20%	90.68%	89.8%

Table 4: Baseline Threshold Method Results

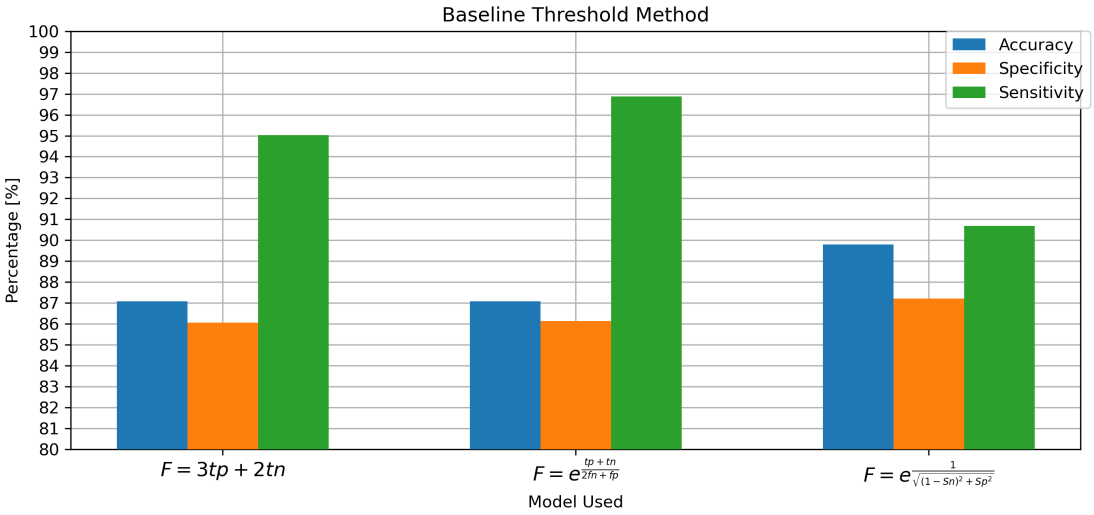


Figure 24: The Baseline Threshold Method Results for Each Fitness Function

Chapter 6: Results and Discussions

6.1 Threshold Method

The thresholds methods after being optimised with a genetic algorithm received much better results than tuning the thresholds by hand. The baseline method and the threshold method with height were both tested and their thresholds optimised with each of the three fitness functions Equation (25), Equation (26) and Equation (27). The results of the baseline method can be seen in Table. 4 and Fig. 24. The results of the threshold method with height can be seen in Table. 5 and graphically in Fig. 25.

Fitness Function	Specificity	Sensitivity	Accuracy
$F = 3tp + 2tn$	90.70%	92.55%	89.12%
$F = e^{\frac{tp+tn}{2fn+fp}}$	93.02%	95.03%	89.80%
$F = e^{\frac{1}{\sqrt{(1-Sn)^2+Sp^2}}}$	93.02%	91.30%	92.52%

Table 5: Threshold Method with Height Results

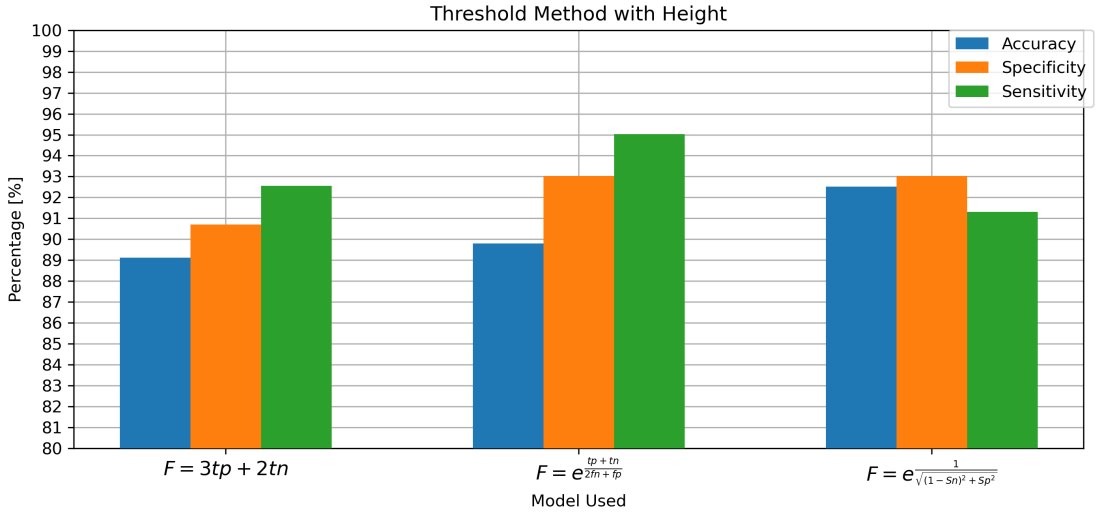


Figure 25: The Threshold Method with Height Results for Each Fitness Function

6.2 Support Vector Machine

6.2.1 Trained on Single Time Step

Support vector machines were initially examined using only one time step and varying the value of C. Training the SVM with altitude or pressure was also examined and served the basis of the remainder of the tests being performed with the pressure as input. The results of these tests can be found in Table. 6 and is graphically shown in Fig. 26.

These results have very low sensitivity and are deemed useless however the results with the most promise were achieved with the pressure as an input rather than the altitude which lead to using pressure in all preceding tests.

Model	Specificity	Sensitivity	Accuracy
Using pressure and C = 10	99.22%	54.35%	94.27%
Using pressure and C = 100	99.19%	57.16%	94.83%
Using pressure and C = 1000	99.16%	59.13%	95.14%
Using altitude and C = 10	99.37%	36.35%	92.39%
Using altitude and C = 100	99.36%	38.30%	92.80%
Using altitude and C = 1000	99.34%	40.36%	93.22%

Table 6: SVM Results After Being Trained on a Single Time step

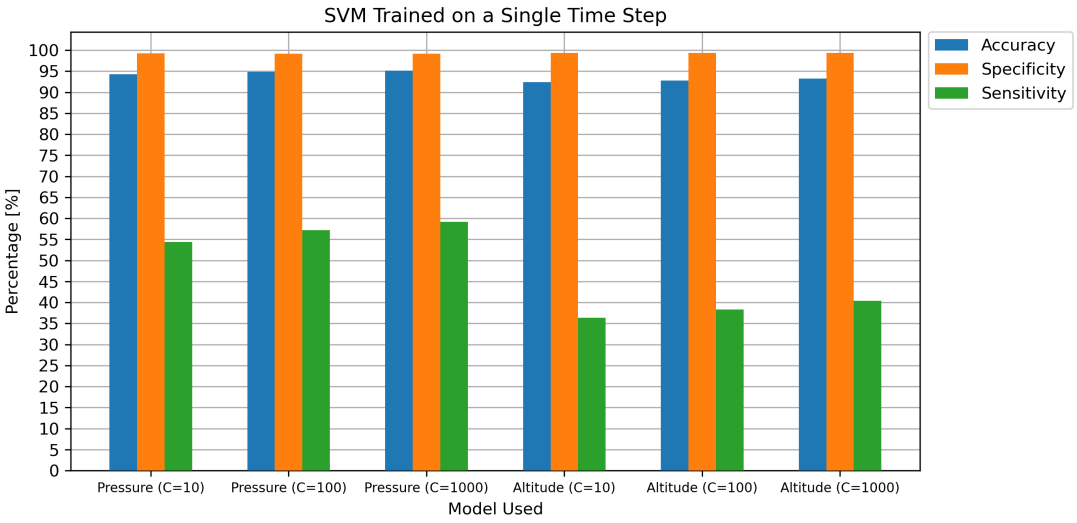


Figure 26: SVM Trained on a Single Time Step

6.2.2 Trained on Multiple Time Steps without PCA

Multiple values of C were experimented with while training the SVM on multiple time steps. Setting C equal to 500 gave the most promising results and so was maintained throughout all tests. During this stage of experimentation training the model with altitude and also with the atmospheric pressure separately. The maximum amount of

Model	Specificity	Sensitivity	Accuracy
With Altitude and 100ms of Data	95.02%	94.56%	94.99%
With Pressure and 100ms of Data	94.77%	93.02%	94.65%

Table 7: SVM Results After Being Trained on Multiple Time Steps without PCA

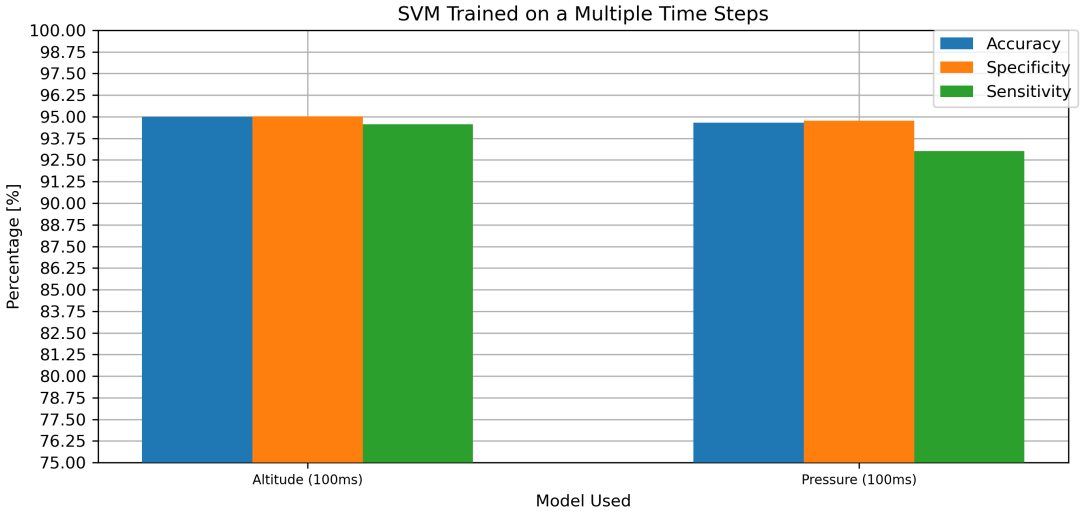


Figure 27: SVM Trained on Multiple Time Steps

time that the authors computer could train the SVM on without PCA was 100ms which also corresponded to the best results. A summary of these results can be seen in Table. 7 and shown graphically in Fig. 27.

6.2.3 Trained on Multiple Time Steps with PCA

The value of C for the SVM trained on multiple time steps with the use of PCA for dimensionality reduction was experimentally determined to give the best results at 1000 and was kept constant across all tests. The variable that did change however was the amount of time used to determine if a fall had occurred or not. Multiple different tests were run and can be seen in Table. 8 and graphically in Fig. 28.

Amount of Time Used	Specificity	Sensitivity	Accuracy
1 second	98.99%	97.76%	98.84%
1.5 seconds	99.29%	98.58%	99.20%
2 seconds	99.60%	98.79%	99.50%
2.25 seconds	99.83%	97.59%	99.56%

Table 8: SVM Results After Being Trained on Multiple Time Steps with PCA

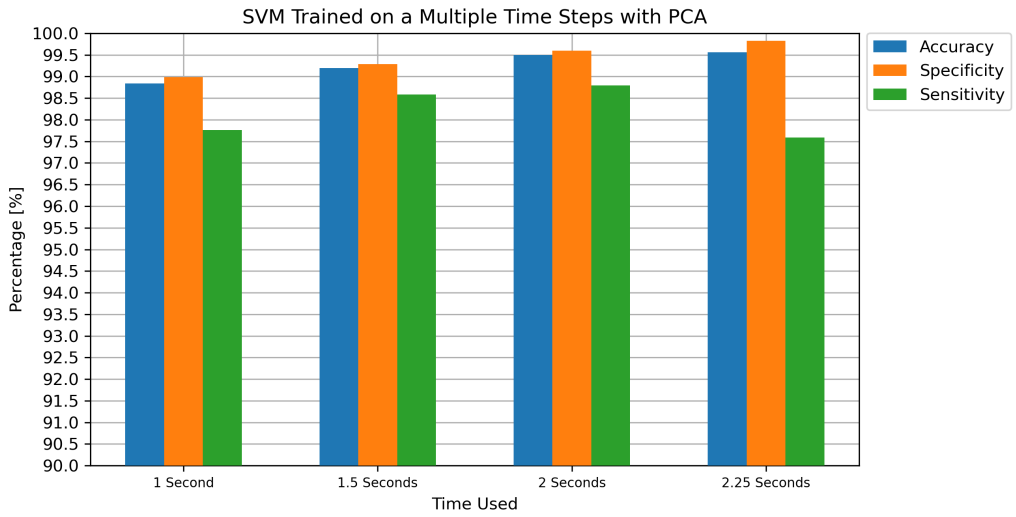


Figure 28: SVM Trained on Multiple Time Steps with PCA

6.3 Neural Network

6.3.1 Hyper-parameters Experimentally Determined

Many neural networks were tested with different hyper-parameters and number of layers. All models have the same input layer of 25 input nodes (corresponding to the 25 principle components of the input data) and 1 node in the output layer (for the binary classification). There were four models of note which are:

- **Model 1**
  - Optimizer: Adam with default parameters
  - Batch Size: 32
  - Epochs: 1000
  - Input layer
  - 4 x Dense layers with 50 nodes and Relu activation function
  - Output layer with Sigmoid activation function
- **Model 2**
  - Optimizer: Adam with default parameters
  - Batch Size: 32
  - Epochs: 1000
  - Input layer
  - 4 x Dense layers with 50 nodes, Relu activation function and dropout of 0.2
  - Output layer with Sigmoid activation function
- **Model 3**
  - Optimizer: Adam with default parameters
  - Batch Size: 32
  - Epochs: 1000
  - Input layer
  - 10 x Dense layers with 50 nodes, Relu activation function and dropout of 0.2
  - Output layer with Sigmoid activation function
- **Model 4**
  - Optimizer: Adam with default parameters
  - Batch Size: 32
  - Epochs: 1000
  - Input layer
  - 4 x Dense layers with 200 nodes, Relu activation function and dropout of 0.2
  - Output layer with Sigmoid activation function

The results of each of these tests can be found in Table. 9 and graphically in Fig. 29.

Model	Specificity	Sensitivity	Accuracy
Model 1	99.75%	98.68%	99.21%
Model 2	89.05%	98.28%	93.67%
Model 3	87.42%	97.10%	92.26%
Model 4	96.38%	99.57%	97.98%

Table 9: NN with Hyper Parameters Experimentally Determined

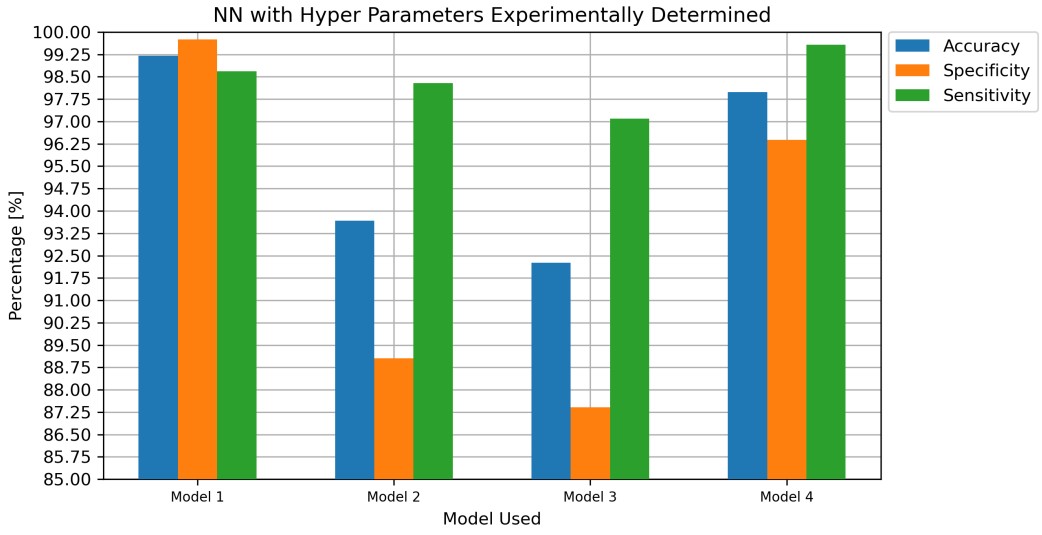


Figure 29: NN with Hyper Parameters Experimentally Determined and Trained on 2.25 Seconds of Data with PCA Applied

6.3.2 Hyper-Parameters Optimised with Optuna

Many different models were experimented with using Optuna to tune their hyper-parameters. As previously described, the input layer is the same for all models (25 to match the principal components of the data) and the output layer is always a single node with the sigmoid activation function. The most notable models can be seen below:

- **Model 1**
  - Optimizer: SGD with learning rate = 0.0725 and momentum = 0.0795
  - Batch Size: 152
  - Epochs: 1000
  - Input layer
  - 3 x Dense layers
    - Layer 1
      - Number of Nodes: 86
      - Activation Function: tanh
      - Dropout Rate: 0.0399
    - Layer 2
      - Number of Nodes: 99
      - Activation Function: relu
      - Dropout Rate: 0.0250
    - Layer 3
      - Number of Nodes: 76
      - Activation Function: relu
      - Dropout Rate: 0.0151
  - Output layer with Sigmoid activation function
- **Model 2**
  - Optimizer: Adam with learning rate = 0.0031
  - Batch Size: 152
  - Epochs: 1000
  - Input layer
  - 2 x Dense layers
    - Layer 1
      - Number of Nodes: 143
      - Activation Function: tanh
      - Dropout Rate: 0.1019
    - Layer 2
      - Number of Nodes: 127
      - Activation Function: relu
      - Dropout Rate: 0.0447
  - Output layer with Sigmoid activation function

- **Model 3**
  - Optimizer: SGD with learning rate = 0.0531 and momentum = 0.0939
  - Batch Size: 231
  - Epochs: 1000
  - Input layer
  - 5 x Dense layers
    - Layer 1
      - Number of Nodes: 147
      - Activation Function: relu
      - Dropout Rate: 0.0655
    - Layer 2
      - Number of Nodes: 139
      - Activation Function: relu
      - Dropout Rate: 0.2406
    - Layer 3
      - Number of Nodes: 10
      - Activation Function: tanh
      - Dropout Rate: 0.2428
    - Layer 4
      - Number of Nodes: 97
      - Activation Function: sigmoid
      - Dropout Rate: 0.1665
    - Layer 5
      - Number of Nodes: 86
      - Activation Function: leaky-relu
      - Dropout Rate: 0.0136
  - Output layer with Sigmoid activation function

The results of each of these models can be found in Table. 10 and graphically in Fig. 30.

6.4 Discussion

The fall detection methods explored within this paper are all viable methods of preventing long-lie and other fall related health complications.

The linear relationship defined between a fall and the data gathered from the data acquisition unit described by the threshold methods (even with complex approaches to determining the thresholds) showed promising results however were consistently

Model	Specificity	Sensitivity	Accuracy
Model 1	99.79%	99.42%	99.60%
Model 2	99.76%	99.50%	99.63%
Model 3	99.75%	99.38%	99.56%

Table 10: NN with Hyper-Parameters Optimized with Optuna

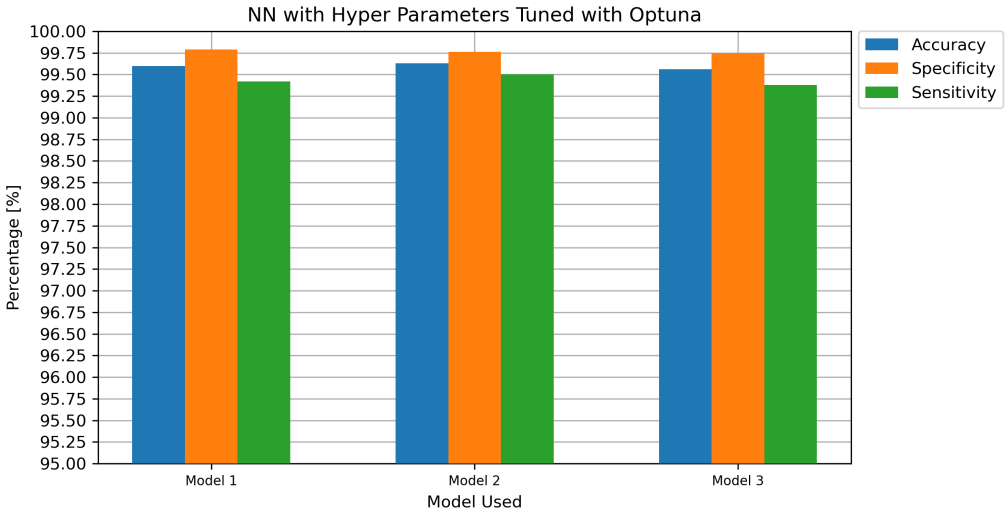


Figure 30: NN with Hyper Parameters Optimised with Optuna and Trained on 2.25 Seconds of Data with PCA Applied

less accurate than the machine learning methods. What the threshold method lacks in accuracy it makes up for in computational costs. The threshold method is an expensive algorithm to configure but once the thresholds have been determined the actual method can run at a very fast pace on almost any micro-controller due to it boiling down to simple if statements. The threshold method in general had a higher sensitivity than it did specificity which is indicative of the fitness functions used which rewarded true positives more heavily than true negatives. This is in contrast to the support vector machine method which in general was found to have a higher specificity than sensitivity. It is more desirable in a fall detection system to have a high sensitivity than it is to have a high specificity because a high sensitivity relates to correctly identifying when a fall occurs where as specificity relates to the number of false alarms. Although specificity is important, a false alarm is less of a concern than missing the detection of a fall event.

The model that performed the best within the threshold methods was the threshold method with height using fitness function Equation (27). This model performed the best in terms of accuracy however in terms of sensitivity the threshold method with height using fitness function Equation (26) performed better.

It was clear that all machine learning methods benefited from seeing larger amounts of time per classification with the accuracy trending upwards with this increase in time. This makes intuitive sense however does vastly increase the pre-processing and memory usage in comparison to the threshold methods.

The support vector machine results were very promising and well suited to a small data set such as the one presented within this paper. SVM is especially good at making complex binary classification decisions even with a small data set and less complex data preparation than with methods such as neural networks. The SVM performed exceptionally well showing very promising results especially in terms model complexity, training time and memory usage. The SVM stores only the support vectors of the model in order to operate, which are in this circumstance much fewer than the parameters needed to be stored for neural network operation. Making it more feasible to be running in an embedded application. The SVM method for fall detection was a nice medium in terms of accuracy, specificity and sensitivity while also keeping the computational complexity to a minimum. The SVM method has a disadvantage in that it does not give any kind of indication as to how confident the model is in its decision of a fall or non-fall. This means that the model can not be made more sensitive or less sensitive by varying the threshold that decides a fall. This means that for each individual user the SVM method may not be as well suited as it cannot be configured to be less sensitive for highly active clients and more sensitive for less mobile clients.

The model that performed the best in terms of accuracy for the SVM fall detection method was the model trained using 2.25 seconds worth of data with 25 principle components considered. The model that performed the best in terms of sensitivity for the SVM method was the model trained using 2 seconds worth of data with 25 principle components.

The neural network results were exceptional across the board in terms of specificity, sensitivity and percent correct, even surpassing the results of the SVM. This was a surprising result due to the ability for SVM to perform well on small data sets. The neural network performed very well in the task of detecting falls even amongst some of the most vigorous activities within the data set, indicating that the neural network is a good method for detecting falls amongst physically active clients. The neural networks have the advantage of returning a confidence value in its decision as to whether the current input is a fall or not. This makes the model more versatile as it can be configured as previously described by each individual client. The disadvantages of neural networks are the time complexities and memory consumption of the algorithm. Also, the complexity required to correctly tune the parameters is quite large, due to the large number of hyper-parameters that can be tuned during the building of a neural network model. The use of the Optuna optimisation library helped with this task however took a very long time to converge on a solution and quite a lot of code and configuring on start up.

The model that performed the best for the NN method in terms of accuracy was model 2 with the hyper-parameters optimised with Optuna. The model with the highest sensitivity for the NN method was model 4 with the hyper-parameters experimentally chosen however the specificity was much lower than the previously discussed model.

The neural network is the clear winner in terms of performance in the task of fall detection however the support vector machine shows promise in terms of speed of configuration and building of the model due to there only being one main parameter that must be tuned. The fall detection methods explored within this paper all exceed the original baseline approach metrics.

Chapter 7: Conclusion

7.1 Summary

Detecting falls will enable independent living of people at risk of long-lie and other adverse effects from falls. Many methods have been implemented in the past with an emphasis on the elderly. This paper has explored the creation of a data set consisting of falls, activities of daily living as well as physical exercise to aid in the correct classification of falls even amongst physically active clients.

This paper has also explored the creation of a fall detection device and website as a platform for experimenting with different fall detection methods. Three main parameters of the methods were calculated, sensitivity, specificity and accuracy which were used to compare their performance. The methods include two different threshold methods, one used as a baseline and the other an extension of the baseline that also included height. Machine learning methods were next explored, especially the most optimal selection of the various hyper parameters involved with each method. The machine learning methods included support vector machines and neural networks.

The website implements each of these methods with the use of a server and a front end user interface. The website allows the user to alter various parameters to explore how it will affect the performance of each of the models.

The fall detection method with the highest performance was implemented with a neural network with hyper-parameters tuned using the library Optuna and performed much better than the baseline approach in all performance metrics.

7.2 Future Works

The fall detection website created within this paper in the future should be extended to a mobile app that can function anywhere allowing for more usability and portability of the device. The website should also apply an online database that can store the users data for further training of the model and also to allow the user to view their own history.

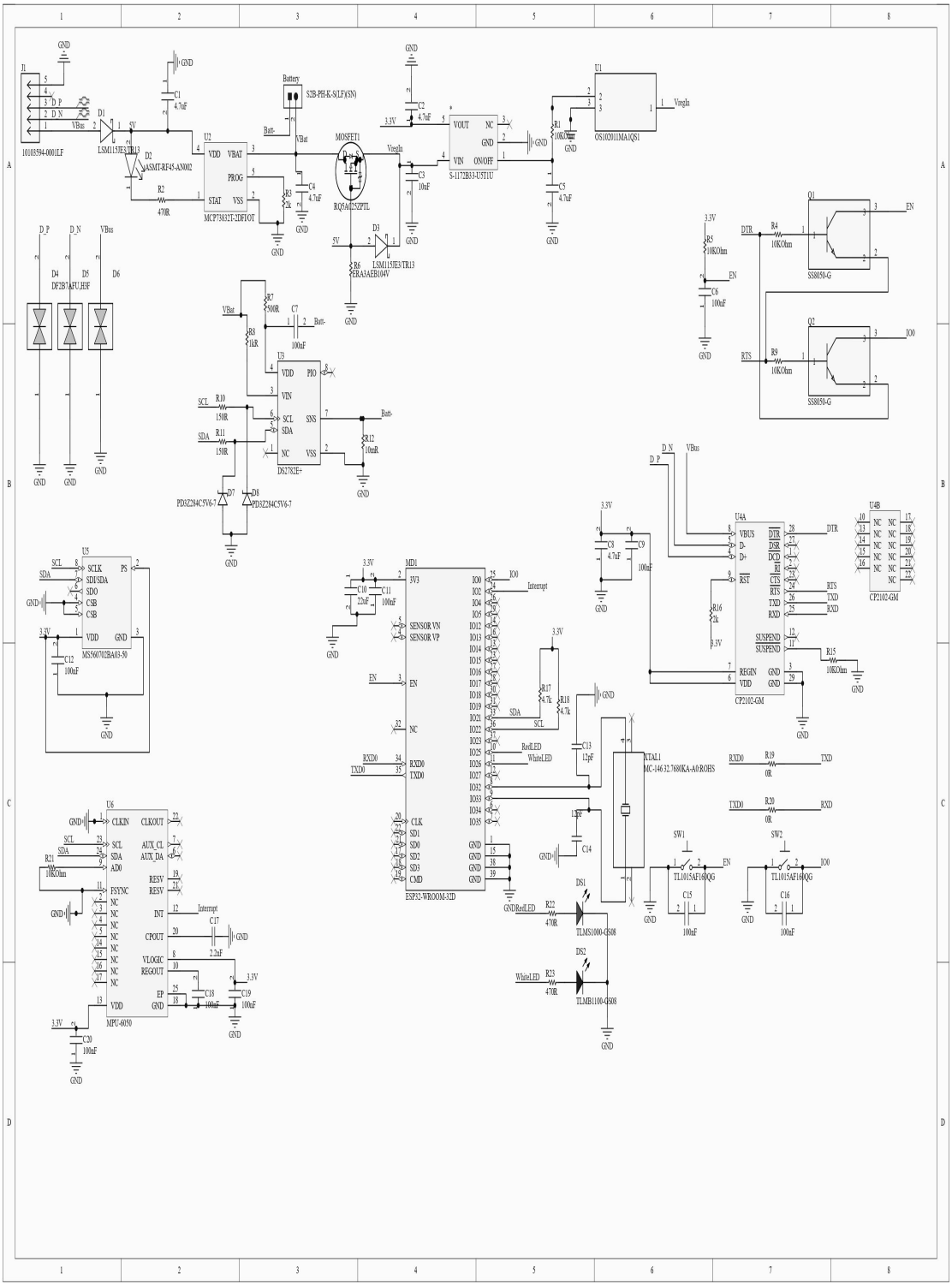
The fall detection device functions very well as is, however future efforts should be made to minimise the size of the design utilising a smaller battery and also a microcontroller with less power consumption. The casing for the fall detection device should also be made out of a more comfortable material to allow for better usability.

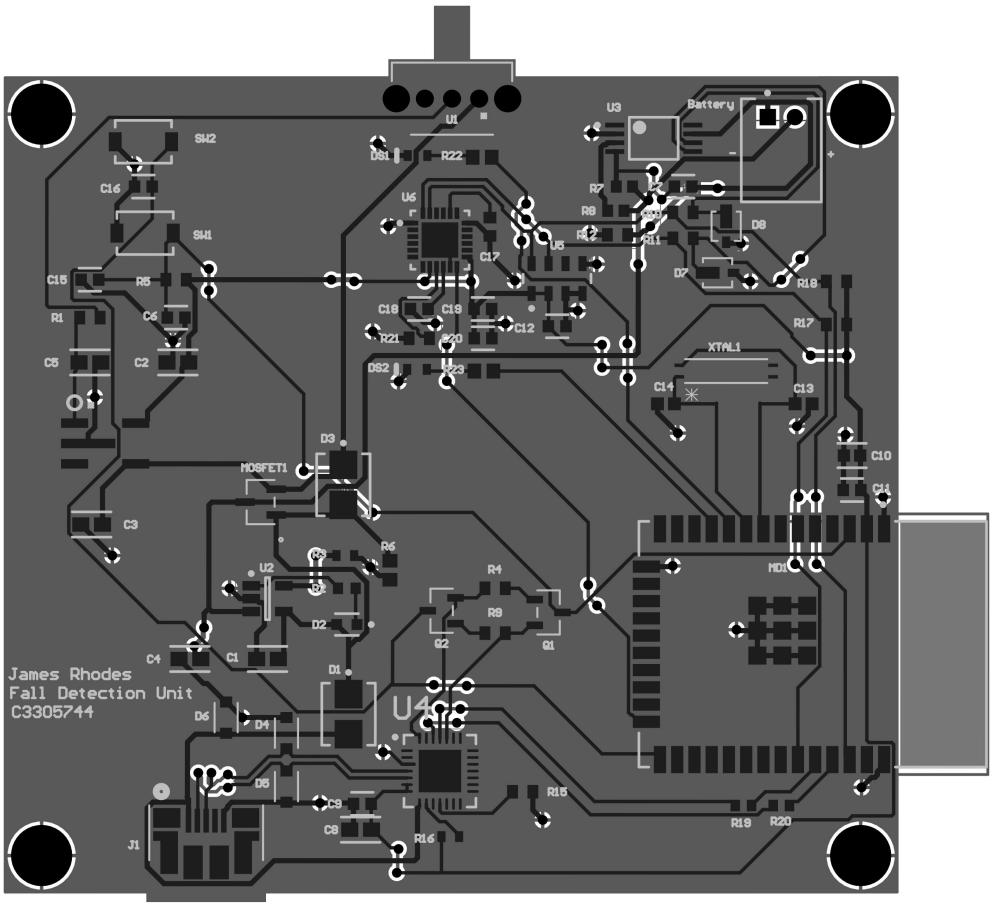
The data base that was created throughout this paper should be extended in the future to include not only a single persons measurements but multiple people from different backgrounds, ages, sexes and heights. This will allow for more robust methods of fall detection as the data set will be more representative of the entire population. Future efforts should also be made to increase the accuracy of the fall measurements (in a safe manor) such as performing the falls with a crash test dummy or otherwise. Another method that could be used to extend the data base is by creating an accurate simulation of a fall and activities of daily living to create thousands of different scenarios from differing height and body shapes.

Other machine learning algorithms were also considered for this paper that could be explored in the future such as a convolutional neural network which can learn the relationships between the shapes of each of the data samples. A recurrent neural network could also be explored, a recurrent neural network is a neural network that has a memory and so in theory could potentially learn the relationship between the data samples leading up to a fall and the fall itself.

Appendices

Appendix A: PCB/Schematic




James Rhodes
Fall Detection Unit
C3305744
U1
U2
U3
U4
U5
U6
SW1
SW2
Battery
XTAL1
MOSFET1
MDI
J1
C1
C2
C3
C4
C5
C6
C8
C9
C10
C11
C12
C13
C14
C15
C16
C17
C18
C19
C20
R1
R4
R5
R6
R7
R8
R9
R15
R16
R17
R18
R19
R20
R21
R22
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2

Appendix B: Data Acquisition Unit Code

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2019-07-08 - Added Auto Calibration and offset generator
//       - and altered FIFO retrieval sequence to avoid using blocking code
//      2016-04-18 - Eliminated a potential infinite loop
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read process
//      2012-06-19 - completely rearranged DMP initialization code and simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
```

```
32 furnished to do so, subject to the following conditions:
33
34 The above copyright notice and this permission notice shall be
      included in
35 all copies or substantial portions of the Software.
36
37 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
      EXPRESS OR
38 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
      MERCHANTABILITY,
39 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
      SHALL THE
40 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
41 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
      ARISING FROM,
42 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
       IN
43 THE SOFTWARE.
44 ===============================================
45 */
46
47
48 //INCLUDES
49
50 #include "I2Cdev.h"
51 #include <BLEDevice.h>
52 #include <BLEUtils.h>
53 #include <BLEServer.h> //Library to use BLE as server
54 #include <BLE2902.h>
55 #include <Maxim_DS2782.h>
56 #include <MS5x.h>
57 #include <Wire.h>
58 #include "MPU6050_6Axis_MotionApps20.h"
59
60 //DEFINES
61 #define RED_LED 26
62 #define BLUE_LED 25
63 #define BLETIME 5000
64 #define BatteryService BLEUUID((uint16_t)0x180D)
65
66 #define SDA 22
67 #define SCL 21
68
69 #define BATTERY_SAMPLE_TIME 10000 //time between battery checks in ms
70
71 //Pressure Constants
72 #define SEA_LEVEL_PRESSURE 1013.25
73
74 //Scales for accleration and Gyro
75 const float ACC_SCALE = 16400;
76 const float GYRO_SCALE = 1450;
77
78 //FUNCTION DEFINITIONS
79 void InitBLE();
80
81
82 //GLOBAL VARS
```

```
83
84 //i2c var
85 TwoWire i2cLine = TwoWire(0);
86
87 //BLE VARS
88 bool _BLEClientConnected = false;
89 char *message;
90 bool initBLEtimer = 0;
91 unsigned long bleTime = 0;
92 BLECharacteristic BatteryLevelCharacteristic(BLEUUID((uint16_t)0x2A19)
   , BLECharacteristic::PROPERTY_READ | BLECharacteristic::
   PROPERTY_WRITE | BLECharacteristic::PROPERTY_NOTIFY);
93 BLECharacteristic HeartRateCharacteristic(BLEUUID((uint16_t)0x2A37),
   BLECharacteristic::PROPERTY_READ | BLECharacteristic::
   PROPERTY_WRITE);
94 BLEDescriptor BatteryLevelDescriptor(BLEUUID((uint16_t)0x2901));
95
96
97 // MPU CONTROL VARS
98 MPU6050 mpu;
99 bool dmpReady = false;  // set true if DMP init was successful
100 uint8_t devStatus;      // return status after each device operation
   (0 = success, !0 = error)
101 uint16_t packetSize;    // expected DMP packet size (default is 42
   bytes)
102 uint16_t fifoCount;     // count of all bytes currently in FIFO
103 uint8_t fifoBuffer[64]; // FIFO storage buffer
104
105 //BAROMETER VARS
106 MS5x barometer(&i2cLine);
107 double prevPressure=0; // The value of the pressure the last time the
   sensor was polled
108 double prevTemperature=0; // The value of the temperature the last
   time the sensor was polled
109 double seaLevelPressure = 0;
110 double initPressure;
111 double initTemp;
112
113
114 //BATTERY LEVEL VARS
115 Maxim_DS2782 fuel = Maxim_DS2782(&i2cLine,0x34,0.01);
116 int batterySamplePrev = 0;
117 int batterySample = 0;
118
119
120
121
122
123
124 // orientation/motion vars
125 //Quaternion q;           // [w, x, y, z]         quaternion container
126 //VectorInt16 aa;         // [x, y, z]            accel sensor
   measurements
127 //VectorInt16 gyro;
128 //VectorInt16 aaReal;     // [x, y, z]            gravity-free accel
   sensor measurements
129 //VectorInt16 aaWorld;    // [x, y, z]            world-frame accel
```

```
    sensor measurements
//VectorFloat gravity;    // [x, y, z]            gravity vector
//float euler[3];         // [psi, theta, phi]    Euler angle
    container
//float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll
    container and gravity vector

//Structures added for convenience in fall detection algorithms

typedef struct{
    VectorInt16 acc;
    VectorInt16 gyro;
    float accMagSq;
    double accMag;
    float gyroMagSq;
    double gyroMag;
    VectorInt16 accReal;
    Quaternion q;
    VectorFloat gravity;
    float ypr[3];
    double altitude=0;
    double pressure;
    double filteredPressure;
    double temp;
    double filteredTemp;
    float batteryCapacity = 0;
    unsigned long time;
}sensorData;

sensorData fallData;


void scaleAndCalcMags(sensorData *dat){

  //Calculate the magnitude of the gyroscope
  dat->gyroMagSq = (dat->gyro.x*dat->gyro.x) + (dat->gyro.y*dat->gyro.
    y) + (dat->gyro.z*dat->gyro.z);
  dat->gyroMag = sqrt(dat->gyroMagSq);
  //Calculate the magnitude of the accelerometer
  dat->accMagSq = (dat->acc.x*dat->acc.x) + (dat->acc.y*dat->acc.y) +
    (dat->acc.z*dat->acc.z);
  dat->accMag = sqrt(dat->accMagSq);
  }

void filterPressure(sensorData *dat){
  static double xk_1 = initPressure;
  static double vk_1 = 0;
  //Gains calculated in Python script -> Generalises to a steady state
    Kalman filter
  static const double a = 0.0829522865665463;
  static const double b= 0.0035910978084925382;


  dat->filteredPressure = alphaBetaFilter(dat->pressure, &xk_1,&vk_1,a
    ,b);

  }
```

```
180
181 void filterTemperature(sensorData *dat){
182   static double xk_1 = initTemp;
183   static double vk_1 = 0;
184
185
186   //Gains calculated in Python script -> Generalises to a steady state
        Kalman filter
187   static const double a = 0.019178547008903313;
188   static const double b= 0.00018569330684448104;
189
190   dat->filteredTemp = alphaBetaFilter(dat->temp, &xk_1,&vk_1,a,b);
191   }
192
193 double alphaBetaFilter(double filterItem,double *xk_1,double *vk_1,
       double a,double b){
194   double xk = *xk_1 + (*vk_1 * 0.01);
195   double vk = *vk_1;
196
197   double rk = filterItem - xk;
198
199   xk += a*rk;
200   vk += (b*rk)/0.01;
201
202   *xk_1 = xk;
203   *vk_1 = vk;
204
205   return xk;
206   }
207
208 double calcTempCompensatedAltitude(double pressure, double temp){
209   return ((pow((SEA_LEVEL_PRESSURE/pressure),1/5.257))-1)*(temp +
       273.15)/0.0065;
210   }
211
212
213
214
215 class MyServerCallbacks : public BLEServerCallbacks {
216     void onConnect(BLEServer* pServer) {
217       _BLEClientConnected = true;
218       Serial.println("Device Connected!!");
219       digitalWrite(RED_LED,LOW);
220       digitalWrite(BLUE_LED,HIGH);
221       initBLEtimer = 1;
222     };
223
224     void onDisconnect(BLEServer* pServer) {
225       _BLEClientConnected = false;
226       Serial.println("Device Disconnected!!");
227       digitalWrite(RED_LED,HIGH);
228       pServer->getAdvertising()->start();
229     }
230 };
231
232 void InitBLE() {
233   BLEDevice::init("Final Year Project");
```

```
  // Create the BLE Server
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  // Create the BLE Service
  BLEService *pBattery = pServer->createService(BatteryService);

  pBattery->addCharacteristic(&BatteryLevelCharacteristic);
  pBattery->addCharacteristic(&HeartRateCharacteristic);
  BatteryLevelDescriptor.setValue("Fall Detection Unit");
  BatteryLevelCharacteristic.addDescriptor(&BatteryLevelDescriptor);
  BatteryLevelCharacteristic.addDescriptor(new BLE2902());

 //BatteryLevelCharacteristic.setCallbacks(new MyCallbacks());

  pServer->getAdvertising()->addServiceUUID(BatteryService);

  pBattery->start();
  // Start advertising
  pServer->getAdvertising()->start();
}



// ===============================================================
// ===                      INITIAL SETUP                      ===
// ===============================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)


    pinMode(RED_LED,OUTPUT);
    pinMode(BLUE_LED,OUTPUT);

    digitalWrite(RED_LED,HIGH);

    Wire.begin(SCL,SDA,400000);

    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue
    immediately

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection
   successful") : F("MPU6050 connection failed"));

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

    // supply your own gyro offsets here, scaled for min sensitivity
```

```
289
290       mpu.setXAccelOffset(-2605);
291       mpu.setYAccelOffset(1358);
292       mpu.setZAccelOffset(1886);
293       mpu.setXGyroOffset(122);
294       mpu.setYGyroOffset(-25);
295       mpu.setZGyroOffset(25);
296
297     // make sure it worked (returns 0 if so)
298     if (devStatus == 0) {
299         // Calibration Time: generate offsets and calibrate our
    MPU6050
300 //        mpu.CalibrateAccel(10);
301 //        mpu.CalibrateGyro(10);
302         mpu.PrintActiveOffsets();
303         // turn on the DMP, now that it's ready
304         Serial.println(F("Enabling DMP..."));
305         mpu.setDMPEnabled(true);
306         dmpReady = true;
307
308         // get expected DMP packet size for later comparison
309         packetSize = mpu.dmpGetFIFOPacketSize();
310     } else {
311         // ERROR!
312         // 1 = initial memory load failed
313         // 2 = DMP configuration updates failed
314         // (if it's going to break, usually the code will be 1)
315         Serial.print(F("DMP Initialization failed (code "));
316         Serial.print(devStatus);
317         Serial.println(F(")"));
318     }
319     // configure LED for output
320     while(barometer.connect(MS5xxx_CMD_ADC_4096)>0) { // barometer.
    connect starts wire and attempts to connect to sensor
321       Serial.println(F("Error connecting..."));
322       delay(500);
323     }
324     Serial.println(F("Connected to Barometer"));
325     barometer.setPressMbar();
326
327
328     fallData.batteryCapacity = fuel.
    readRemainingStandbyAbsoluteCapacity();
329     InitBLE();
330
331
332 }
333
334
335 // ================================================================
336 // ===                      MAIN PROGRAM LOOP                     ===
337 // ================================================================
338
339 void loop() {
340
341   if(initBLEtimer){
342     bleTime = millis();
```

```
343     initBLEtimer = 0;
344   }
345   if((millis()-bleTime) > BLETIME){
346     digitalWrite(BLUE_LED,LOW);
347     }
348
349     // if programming failed, don't try to do anything
350     if (!dmpReady) return;
351     // read a packet from FIFO
352       if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest
      packet
353
354         mpu.dmpGetAccel(&fallData.acc, fifoBuffer);
355         mpu.dmpGetGyro(&fallData.gyro,fifoBuffer);
356         mpu.dmpGetQuaternion(&fallData.q, fifoBuffer);
357         mpu.dmpGetGravity(&fallData.gravity, &fallData.q);
358         mpu.dmpGetYawPitchRoll(fallData.ypr, &fallData.q, &fallData.
    gravity);
359
360       barometer.checkUpdates();
361       if (barometer.isReady()) {
362
363       fallData.temp = barometer.GetTemp(); // Returns temperature in C
364       fallData.pressure = barometer.GetPres(); // Returns pressure in
    mBar
365
366       }
367       if(prevPressure == 0){
368         prevPressure = fallData.pressure;
369         initPressure = prevPressure;
370
371         initTemp = fallData.temp;
372       }
373       else{
374         filterTemperature(&fallData);
375
376       if(abs(fallData.pressure - prevPressure) > 0.2){
377         fallData.pressure = prevPressure;
378         }
379       filterPressure(&fallData);
380       fallData.altitude = calcTempCompensatedAltitude(fallData.
    filteredPressure,fallData.filteredTemp);
381       prevPressure = fallData.pressure;
382       }
383
384
385
386
387
388
389         scaleAndCalcMags(&fallData);
390         fallData.time = millis();
391
392         asprintf(&message, "{\"yaw\":\"%f\",\"pitch\": \"%f\",\"roll
    \":\"%f\",\
393         \"acc_x\":\"%d\",\"acc_y\": \"%d\",\"acc_z\":\"%d\",\
394         \"gyro_x\":\"%d\",\"gyro_y\": \"%d\",\"gyro_z\":\"%d\",\"
```

```
    pressure\":\"%f\",\
395         \"temp\":\"%f\",\"altitude\":\"%f\",\"gravity_x\":\"%f\",\"
    gravity_y\":\"%f\",\"gravity_z\":\"%f\",\"batteryCapacity\":\"%f
    \",\"accMag\":\"%f\",\"gyroMag\":\"%f\",\"time\":\"%d\"}",fallData
    .ypr[0]*180/M_PI, fallData.ypr[1]*180/M_PI,fallData.ypr[2]*180/
    M_PI
396         ,fallData.acc.x, fallData.acc.y,fallData.acc.z,
397         fallData.gyro.x, fallData.gyro.y,fallData.gyro.z,fallData.
    pressure,fallData.temp,fallData.altitude,fallData.gravity.x,
    fallData.gravity.y,fallData.gravity.z,fallData.batteryCapacity,
    fallData.accMag,fallData.gyroMag,fallData.time);
398         BatteryLevelCharacteristic.setValue(message);
399         BatteryLevelCharacteristic.notify();
400         free(message);
401
402
403     }
404
405     batterySample = millis();
406     if(batterySample-batterySamplePrev>BATTERY_SAMPLE_TIME){
407       fallData.batteryCapacity = fuel.
    readRemainingStandbyAbsoluteCapacity();
408       batterySamplePrev = batterySample;
409     }
410 }
```

Appendix C: Time Series Class Code

```
import pickle
import time
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from collections import deque
from sklearn.utils import shuffle
from glob import iglob
import os
import sys
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler


class Time_Series_Model:

    def __init__(self,itemsToOffset=None,modelFileName=None,seqLength=None,allData=None,model=None,dataPath=None,columnsOfInterest=None,itemsToScale=None):
        self.model = model
        self.modelFileName = modelFileName
        self.dataPath = dataPath if dataPath else r"C:\Users\jrcen\Desktop\Uni Shit\Final Year Project\NEW Data Acquisition\NEW data files\All Data\*.csv"
        self.allDataDf = allData
        self.columnsOfInterest = columnsOfInterest
        self.seqLength = seqLength
        self.itemsToOffset = itemsToOffset
        self.itemsToScale = itemsToScale

        self.xData = None
        self.xTrain = None
        self.xTest = None

        self.yData = None
        self.yTrain = None
        self.yTest = None

        self.scaledItems = {}

        self.testMetrics = {"tp":0,"tn":0,"fp":0,"fn":0,"percentCorrect":None,"name":"Test","numFalls":0,"sensitivity":0,"specificity":0}
        self.trainMetrics = {"tp":0,"tn":0,"fp":0,"fn":0,"percentCorrect":None,"name":"Train","numFalls":0,"sensitivity":0,"specificity":0}

    def printMetrics(self,metric,file=sys.stdout):
        print(f"Percent Correct in {metric['name']} Set = {metric['percentCorrect']}%",file=file)
```

```
42          print(f"The {metric['name']} set had tp: {metric['tp']}, tn: {
    metric['tn']}, fp: {metric['fp']} and fn: {metric['fn']} with a
    total number of falls of {metric['numFalls']}",file=file)
43          print(f"{metric['name']} Sensitivity: {metric['sensitivity']}
    and Specificity: {metric['specificity']}",file=file)
44
45      def calculateMetrics(self, metric,x,y):
46          correct = 0
47          total = len(x)
48
49          for i in range(total):
50              predict = self.model.predict([x[i]])
51              if(y[i]):
52                  metric['numFalls'] += 1
53              if(predict and y[i]):
54                  metric['tp']+=1
55                  correct+=1
56              elif(predict and not y[i]):
57                  metric['fp']+=1
58              elif(not predict and y[i]):
59                  metric['fn']+=1
60              elif(not predict and not y[i]):
61                  metric['tn']+=1
62                  correct+=1
63          if((metric['tp'] + metric['fn']) != 0):
64              metric['sensitivity'] = metric['tp']/(metric['tp'] +
    metric['fn'])
65          else:
66              metric['sensitivity'] = 0
67
68          if((metric['tn'] + metric['fp']) != 0):
69              metric['specificity'] = metric['tn']/(metric['tn'] +
    metric['fp'])
70          else:
71              metric['specificity'] = 0
72
73          metric['percentCorrect'] = (correct/total)*100
74
75
76      def test(self,equalize=None):
77          #Initialise model if not already initialized
78          if(self.model is None):
79              self.model = pickle.load(open(self.modelFileName,'rb'))
80          if((self.xData is None) and (self.allDataDf is None)):
81              self.formAllData(equalize=equalize)
82          elif(self.allDataDf is not None and self.xData is None):
83              self.generateDataFromDf()
84          elif(self.allDataDf is not None or self.xData is not None):
85              pass
86          else:
87              raise Exception("No Data available")
88
89          print("Beginning Testing")
90
91          self.calculateMetrics(self.trainMetrics,self.xTrain,self.
    yTrain)
92          self.printMetrics(self.trainMetrics)
```

```
93
94          self.calculateMetrics(self.testMetrics,self.xTest,self.yTest)
95          self.printMetrics(self.testMetrics)
96
97          startTime = time.time()
98          self.model.predict([self.xTrain[0]])
99          print(f"The time to execute one prediction is {time.time() -
    startTime}")
100
101     def train(self,equalize=None):
102         if((self.allDataDf is None) and (self.xData is None)):
103             self.formAllData(equalize=equalize)
104         elif(self.allDataDf is not None and self.xData is None):
105             self.generateDataFromDf()
106         elif(self.allDataDf is not None or self.xData is not None):
107             pass
108         else:
109             raise Exception("No Data available")
110         if(self.model is None):
111             raise Exception("No Model Initialized!!!")
112         print("Beginning Training")
113         self.model.fit(self.xTrain,self.yTrain)
114         print("\nFinished Training")
115         print(f"Saving Model to {self.modelFileName}")
116         pickle.dump(self.model,open(self.modelFileName,'wb'))
117         print("Finished Saving Model")
118
119     def formAllData(self,equalize=None,scaleType="MinMax"):
120         xData = []
121         yData = []
122         folderAmount = self.calc_folder_amount(self.dataPath.replace("
    \*.csv",""))
123         folderCount = 0
124         print("Beginning Data Cleaning and Preperation")
125         for path in iglob(self.dataPath):
126             print(f'{(folderCount/folderAmount)*100:2.2f}% Through
    Files',end='\r')
127             x,y = self.formXandY(path,self.columnsOfInterest)
128             xData += x
129             yData += y
130             folderCount+=1
131         print("\n")
132         xData,yData = shuffle(xData,yData)
133
134         self.xData,self.yData = shuffle(xData,yData)
135
136         if(self.itemsToOffset is not None):
137             print("Removing Offsets...")
138             for feature in self.itemsToOffset:
139                 self.removeOffset(feature)
140             print("Finished Removing Offsets")
141         if(self.itemsToScale is not None and scaleType=="MinMax"):
142             print("Scaling via Min Max...")
143             for feature in self.itemsToScale:
144                 self.minMaxScaling(feature)
145             print("Finished Scaling")
146
```

```
147         self.xTrain,self.xTest,self.yTrain,self.yTest =
    train_test_split(self.xData,self.yData,test_size=0.4)
148         if(equalize == 'Oversample'):
149             ros = RandomOverSampler()
150             self.xTrain,self.yTrain = ros.fit_resample(self.xTrain,
    self.yTrain)
151             self.xTest,self.yTest = ros.fit_resample(self.xTest,self.
    yTest)
152         elif(equalize == 'Undersample'):
153             ros = RandomUnderSampler()
154             self.xTrain,self.yTrain = ros.fit_resample(self.xTrain,
    self.yTrain)
155             self.xTest,self.yTest = ros.fit_resample(self.xTest,self.
    yTest)
156 
157         return self.xTrain,self.xTest,self.yTrain,self.yTest
158 
159     def removeOffset(self,feature):
160         ind = self.columnsOfInterest.index(feature)
161         lengthOfInterest = len(self.columnsOfInterest)-1
162         for i in range(len(self.xData)):
163             offset = self.xData[i][ind]
164             for j in range(len(self.xData[i])):
165                 if (j%lengthOfInterest) == ind:
166                     self.xData[i][j] -= offset
167     def generateDataFromDf(self):
168         print("Converting DF to X Data and Y Data")
169         self.xData = self.allDataDf.drop(columns=['fall']).to_numpy()
170         self.yData = self.allDataDf['fall'].to_numpy()
171         print("Shuffling Data")
172         self.xData,self.yData = shuffle(self.xData,self.yData)
173         self.xTrain,self.xTest,self.yTrain,self.yTest =
    train_test_split(self.xData,self.yData,test_size=0.4)
174         print("Data Has Been Initialised")
175 
176     def logData(self,testName):
177         if(self.testMetrics['percentCorrect'] is None):
178             raise Exception("No Data to Log!")
179         else:
180             with open(fr"{os.path.dirname(__file__)}\Results.txt",'a+'
    ) as f:
181                 print("*"*10,file=f)
182                 print(f"{testName}\n\n",file=f)
183                 self.printMetrics(self.testMetrics,file=f)
184                 if(self.scaledItems):
185                     print("Scaled Items:\n",file=f)
186                     for key in self.scaledItems:
187                         print(f"{key} -> {self.scaledItems[key]}",file
    =f)
188                 if(self.modelFileName is not None):
189                     print(f"Directory -> {self.modelFileName}",file=f)
190                 print("*"*10,file=f)
191 
192     def getFeatureofSequence(self,feature):
193         ind = self.columnsOfInterest.index(feature)
194         lengthOfInterest = len(self.columnsOfInterest)-1
195         data = np.zeros((len(self.xData),self.seqLength))
```

```
196         for i in range(len(self.xData)):
197             pos=0
198             for j in range(len(self.xData[i])):
199                 if (j%lengthOfInterest) == ind:
200                     data[i][pos] = self.xData[i][j]
201                     pos+=1
202         return data
203 
204     def calcFeatureInsights(self,featureName,arr):
205         self.scaledItems[featureName] = {"min":arr.min(),"max":arr.max
    (),"std":arr.std(),"mean":arr.mean(),"variance":arr.var()}
206         return self.scaledItems
207 
208     def minMaxScaling(self,feature):
209         arr = self.getFeatureofSequence(feature)
210         self.calcFeatureInsights(feature,arr)
211         ind = self.columnsOfInterest.index(feature)
212         lengthOfInterest = len(self.columnsOfInterest)-1
213         for i in range(len(self.xData)):
214             for j in range(len(self.xData[i])):
215                 if (j%lengthOfInterest) == ind:
216                     self.xData[i][j] = self.calcMinMax(self.xData[i][j
    ],self.scaledItems[feature])
217 
218     @staticmethod
219     def calcMinMax(val,scaleParams):
220         return (val - scaleParams['min'])/(scaleParams['max']-
    scaleParams['min'])
221 
222     @staticmethod
223     def getSequential(df,seqLength):
224         #ensure format of df is such that the y column is the last one
    !
225         fall = df['fall'].values
226         df.drop(columns=['fall'],inplace=True)
227         yData = np.zeros(len(df.index) - seqLength + 1)
228         xData = np.zeros((len(df.index) - seqLength + 1,(len(df.
    columns)*seqLength)))
229 
230         timeSequence = deque(maxlen=seqLength)
231         counter = 0
232         for index,row in enumerate(df.values):
233             timeSequence.append(row)
234             if(len(timeSequence) == seqLength):
235                 xData[counter] = np.array(timeSequence).flatten()
236                 yData[counter] = fall[index]
237                 counter+=1
238 
239         return list(xData),list(yData)
240 
241     def formXandY(self,path,COLUMNS_OF_INTEREST):
242         datadf = pd.read_csv(path,usecols=COLUMNS_OF_INTEREST)
243         datadf = datadf[COLUMNS_OF_INTEREST]
244 
245         for column in datadf.columns:
246             if(column in ['acc_x','acc_y','acc_z']):
247                 #Below scales between -1 and 1
```

```
248                 #datadf[column] = datadf[column]/(2*8200)
249
250                 # Below Scales between 0 and 1
251                 datadf[column] = 0.5*((datadf[column]/(2*8200))+1)
252
253             elif(column in ['gyro_x','gyro_y','gyro_z']):
254                 #Below scales between -1 and 1
255                 # datadf[column] = datadf[column]/1450
256
257                 # Below Scales between 0 and 1
258                 datadf[column] = 0.5*((datadf[column]/1450)+1)
259             else:
260                 datadf[column] = datadf[column]
261
262         xData,yData = self.getSequential(datadf,self.seqLength)
263
264         return shuffle(xData,yData)
265
266     @staticmethod
267     def calc_folder_amount(path):
268         return len(os.listdir(path))
```

Appendix D: Server Code

```
"""Classes for different fall detection techniques"""
from abc import ABC,abstractmethod
import pandas as pd
import numpy as np
import pickle
from collections import deque
from sklearn.decomposition import PCA
import json
import os
import sys
import tensorflow as tf
from tensorflow import keras

class Fall_Methods(ABC):


    def __init__(self):
        self.queue = None
        self.scaleParams = None
        self.seqLen = None
        self.colsOfInterest = None
        self.xData = None

    def prepData(self,data,itemToScale):
        """Will prepare the data for prediction, input data must be a dataframe"""
        ACC_TO_SCALE = ['acc_x','acc_y','acc_z']
        GYRO_TO_SCALE = ['gyro_x','gyro_y','gyro_z']
        index = self.colsOfInterest.index(itemToScale)
        data.drop(columns=[column for column in data.columns if column not in self.colsOfInterest],inplace=True)
        data = data[self.colsOfInterest].astype(float)

        for column in data.columns:
            if((column in ACC_TO_SCALE) or (column == 'accMag')):
                data[column] = 0.5*((data[column]/(2*8200))+1)
            elif((column in GYRO_TO_SCALE) or (column == 'gyroMag')):
                data[column] = 0.5*((data[column]/1450)+1)

        self.queue.extend(data.to_numpy().flatten())
        # xData = list(queue)
        lengthOfInterest  = len(self.colsOfInterest)
        if(len(self.queue) == lengthOfInterest*self.seqLen):
            xData = np.array(self.queue)
            xData = xData.reshape(self.seqLen,lengthOfInterest)
            xData[:,index] = xData[:,index] - xData[0,index]
            xData[:,index] = self.calcMinMax(xData[:,index],self.scaleParams)
            return xData.flatten()
        return []
```

```
48
49      @staticmethod
50      def generateData(request):
51          return pd.DataFrame(json.loads(request.data.decode()))
52
53      @staticmethod
54      def calcMinMax(val,scaleParams):
55          """Calculates the min max scaling of the input data"""
56          return (val - scaleParams['min'])/(scaleParams['max']-
    scaleParams['min'])
57
58      @abstractmethod
59      def init(self):
60          """This method will initialise the current model of choice and
     the pca"""
61          pass
62
63      @abstractmethod
64      def predict(self,request):
65          """This method will perform the prediction on the current data
    """
66          pass
67
68
69
70  class SVM(Fall_Methods):
71
72      def __init__(self):
73          self.seqLen = 225
74          self.itemToScale = 'pressure'
75          self.colsOfInterest = ['acc_x','acc_y','acc_z','gyro_x','
    gyro_y','gyro_z','pressure','accMag','gyroMag']
76          self.queue = deque(maxlen=self.seqLen*len(self.colsOfInterest)
    )
77          self.model = None
78          self.pca = None
79          self.scaleParams = {'min':-0.6900020000001632,'max'
    :0.8299560000000383}
80
81          self.MODEL_PATH = r".\SVM_PCA_Pressure_No_Oversampling_2250ms.
    sav"
82          self.PCA_PATH = r".\PCA_Pressure_No_Oversampling_2250ms.sav"
83
84      def init(self):
85          """Initialises the SVM model and pca"""
86          self.model = pickle.load(open(self.MODEL_PATH,'rb'))
87          self.pca = pickle.load(open(self.PCA_PATH,'rb'))
88
89      def predict(self,request):
90          """Performs the prediction for SVM"""
91          self.xData = self.prepData(self.generateData(request),
    itemToScale=self.itemToScale)
92
93          if(len(self.xData) == self.seqLen*len(self.colsOfInterest)):
94              try:
95                  xData = self.pca.transform([self.xData])
96                  return self.model.predict(xData)[0]
```

```
            except:
                print("Caught error")
                return 0
        else:
            return 0

class NN(Fall_Methods):

    def __init__(self):
        self.seqLen = 225
        self.itemToScale = 'pressure'
        self.colsOfInterest = ['acc_x','acc_y','acc_z','gyro_x','gyro_y','gyro_z','pressure','accMag','gyroMag']
        self.queue = deque(maxlen=self.seqLen*len(self.colsOfInterest))
        self.model = None
        self.pca = None
        self.scaleParams = {'min':-0.6900020000001632,'max':0.8299560000000383}

        self.MODEL_PATH = r"C:\Users\jrcen\Desktop\Uni Shit\Final Year Project\Website Files\ML_API\NN_Models\Final_Results_Optuna_2"
        self.PCA_PATH = r"C:\Users\jrcen\Desktop\Uni Shit\Final Year Project\Website Files\ML_API\NN_PCA_2250ms.sav"

    def init(self):
        """Initialises the NN Model"""
        self.model = tf.keras.models.load_model(self.MODEL_PATH)
        self.pca = pickle.load(open(self.PCA_PATH,'rb'))
    def predict(self,request):
        """This will produce the NN prediction"""
        self.xData = self.prepData(self.generateData(request), itemToScale=self.itemToScale)

        if(len(self.xData) == self.seqLen*len(self.colsOfInterest)):
            try:
                xData = self.pca.transform([self.xData])
                return self.model.predict(xData)[0][0]
            except:
                print("Caught error")
                return 0
        else:
            return 0
```

Listing D.1: ML Classes

```
from flask import Flask, request,jsonify
from flask_cors import CORS
from Fall_Detection_Classes import SVM,NN

app = Flask(__name__)
CORS(app)
model = None


@app.route("/")
def index():
    return "<h1>This is the ML API for an FYP</h1>"
```

```
13
14 @app.route("/initModel", methods = ["POST"])
15 def initModel():
16     global model
17     data = request.data.decode('utf-8').replace('"','')
18     modelType = data
19     if(modelType == "SVM"):
20         model = SVM()
21         model.init()
22     elif(modelType == "NN"):
23         model = NN()
24         model.init()
25     return jsonify(isError= False,
26                    message= "Success",
27                    statusCode= 200,
28                    data= data), 200
29
30 @app.route("/Predict",methods=["POST"])
31 def makePredict():
32     global model
33     fall = model.predict(request)
34     print(fall)
35     return jsonify(isError= False,
36                 message= f"{fall:.5f}",
37                 statusCode= 200), 200
38
39
40 if __name__ == "__main__":
41     app.run(host="0.0.0.0", debug=False)
```

Listing D.2: Server Code

References

[1] L. M. S. d. Nascimento, L. V. Bonfati, M. L. B. Freitas, J. J. A. Mendes Junior, H. V. Siqueira and S. L. Stevan, 'Sensors and Systems for Physical Rehabilitation and Health Monitoring—A Review', *Sensors*, vol. 20, no. 15, 2020, ISSN: 1424-8220. DOI: 10.3390/s20154063. [Online]. Available: https://www.mdpi.com/1424-8220/20/15/4063.

[2] A. I. of Health and Welfare. (2018). 'Older Australia at a glance', [Online]. Available: https://www.aihw.gov.au/reports/older-people/older-australia-at-a-glance/contents/service-use/health-care-hospitals (visited on 21/08/2021).

[3] A. B. of Statistics. (2016). 'Disability, Ageing and Carers, Australia: Summary of Findings, 2015', [Online]. Available: https://www.abs.gov.au/ausstats/abs@.nsf/Lookup/4430.0main+features302015#:~:text=Most%5C%20older%5C%20Australians%5C%20(94.8%5C%25),all%5C%20older%5C%20people%5C%20lived%5C%20alone.. (visited on 21/08/2021).

[4] A. I. of Health and Welfare. (2018). 'Australia's health 2018', [Online]. Available: https://www.aihw.gov.au/reports/australias-health/australias-health-2018/contents/table-of-contents (visited on 20/08/2021).

[5] ——, (2019). 'Trends in hospitalised injury due to falls in older people 2007–08 to 2016–17', [Online]. Available: https://www.aihw.gov.au/reports/injury/trends-in-hospitalised-injury-due-to-falls/summary (visited on 21/08/2021).

[6] J. Burton. (2019). 'Healthcare Innovation: The Use of Fall Detectors to Detect Unwitnessed Falls and Reduce a Long Lie', [Online]. Available: http://drjennaburton.com/healthcare-innovation-the-use-of-fall-detectors-to-detect-unwitnessed-falls-and-reduce-a-long-lie/ (visited on 21/08/2021).

[7] N. Kolban, *ESP32_BLE_Arduino*, https://github.com/nkolban/ESP32_BLE_Arduino, 2017.

[8] F. Beaufort. (2015). 'Communicating with Bluetooth devices over JavaScript', [Online]. Available: https://web.dev/bluetooth/ (visited on 17/09/2021).

[9] Q. T. Huynh, U. D. Nguyen, L. B. Irazabal, N. Ghassemian and B. Q. Tran, 'Optimization of an accelerometer and gyroscope-based fall detection algorithm', *Journal of Sensors*, vol. 2015, 2015.

[10] D. Shiffman. (2012). 'The Nature of Code', [Online]. Available: https://natureofcode.com/book/ (visited on 20/08/2021).

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] Mart'ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[13] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, 'Optuna: A next-generation hyperparameter optimization framework', in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[14] V. Jakkula, 'Tutorial on support vector machine (svm)', *School of EECS, Washington State University*, vol. 37, 2006.

[15] MathWorks. (). 'Support Vector Machine (SVM)', [Online]. Available: `https://au.mathworks.com/discovery/support-vector-machine.html` (visited on 27/08/2021).

[16] A. Kumar, L. Bi, J. Kim and D. D. Feng, 'Chapter Five - Machine learning in medical imaging', in *Biomedical Information Technology (Second Edition)*, ser. Biomedical Engineering, D. D. Feng, Ed., Second Edition, Academic Press, 2020, pp. 167–196, ISBN: 978-0-12-816034-3. DOI: `https://doi.org/10.1016/B978-0-12-816034-3.00005-5`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B9780128160343000055`.

[17] G. Zhang. (2018). 'What is the Kernel Trick? Why is it Important?', [Online]. Available: `https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d` (visited on 27/08/2021).

[18] J. Han, M. Kamber and J. Pei, '9 - Classification: Advanced Methods', in *Data Mining (Third Edition)*, ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber and J. Pei, Eds., Third Edition, Boston: Morgan Kaufmann, 2012, pp. 393–442, ISBN: 978-0-12-381479-1. DOI: `https://doi.org/10.1016/B978-0-12-381479-1.00009-5`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B9780123814791000095`.

[19] S. Sreenivasa. (2020). 'Radial Basis Function (RBF) Kernel: The Go-To Kernel', [Online]. Available: `https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a` (visited on 07/09/2021).

[20] S. Patel. (2017). 'Chapter 2 : SVM (Support Vector Machine) — Theory', [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72 (visited on 07/09/2021).

[21] dishaa.agarwal. (2021). 'Introduction to SVM(Support Vector Machine) Along with Python Code', [Online]. Available: https://www.analyticsvidhya.com/blog/2021/04/insight-into-svm-support-vector-machine-along-with-code/ (visited on 07/09/2021).

[22] V. Jain. (2019). 'Everything you need to know about "Activation Functions" in Deep learning models', [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253 (visited on 10/09/2021).

[23] S. Ganesh. (2020). 'What's The Role Of Weights And Bias In a Neural Network?', [Online]. Available: https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f (visited on 10/09/2021).

[24] Hyunjulie. (2018). 'Activation Functions: A Short Summary', [Online]. Available: https://medium.com/hyunjulie/activation-functions-a-short-summary-8450c1b1d426 (visited on 10/09/2021).

[25] ——, (2020). 'Neural Networks', [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks (visited on 10/09/2021).

[26] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[27] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG].

[28] N. Qian, 'On the momentum term in gradient descent learning algorithms', *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[29] G. Wang, Z. Liu and Q. Li, 'Fall Detection with Neural Networks', in *2019 IEEE International Flexible Electronics Technology Conference (IFETC)*, 2019, pp. 1–7. DOI: 10.1109/IFETC46817.2019.9073718.

[30] V. Mallawaarachchi. (2017). 'Introduction to Genetic Algorithms — Including Example Code', [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (visited on 07/09/2021).

[31] R. Penoyer, 'The Alpha-Beta Filter', *C Users J.*, vol. 11, no. 7, pp. 73–86, Jul. 1993, ISSN: 0898-9788.

[32] J. Painter, D. Kerstetter and S. Jowers, 'Reconciling steady-state Kalman and alpha-beta filter design', *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 6, pp. 986–991, 1990. DOI: 10.1109/7.62250.

[33] I. T. Jolliffe and J. Cadima, 'Principal component analysis: A review and recent developments', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20 150 202, 2016.

[34] R. Vashisht. (2020). 'What is data leakage? And how to mitigate it?', [Online]. Available: https://www.atoti.io/what-is-data-leakage-and-how-to-mitigate-it/ (visited on 12/09/2021).

[35] Espressif, *Arduino-esp32*, https://github.com/espressif/arduino-esp32/blob/master/libraries/Wire/src/Wire.h, 2006.

[36] *ESP32_DevKitc_V4*, ESP32_DevKitc_V4, Espressif, 2017. [Online]. Available: https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf (visited on 14/09/2021).

[37] *MPU-6000 and MPU-6050 product Specification Revision 3.4*, MPU6050, InvenSense, 2013. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf (visited on 14/09/2021).

[38] *MS5607-02BA03 Barometric Pressure Sensor, with stainless steel cap*, MS5607-02BA03, TE CONNECTIVITY SENSORS, 2020. [Online]. Available: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=MS5607-02BA03&DocType=Data+Sheet&DocLang=English (visited on 15/09/2021).

[39] *S-1172 Series HIGH RIPPLE-REJECTION LOW DROPOUT HIGH OUTPUT CURRENT CMOS VOLTAGE REGULATOR*, S-1172 Series, ABLIC, 2015. [Online]. Available: https://www.ablic.com/en/doc/datasheet/voltage_regulator/S1172_E.pdf (visited on 15/09/2021).

[40] *Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers*, MCP73832, Microchip, 2005. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP73831-Family-Data-Sheet-DS20001984H.pdf (visited on 15/09/2021).

[41] *Stand-Alone Fuel Gauge IC*, DS2782, Maxim, 2008. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/DS2782.pdf (visited on 15/09/2021).

[42] *SINGLE-CHIP USB-TO-UART BRIDGE*, CP2102/9, Silicon Labs, 2017. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf (visited on 15/09/2021).

[43] J. Rowberg, *MPU6050_6Axis_MotionApps20*, https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU6050/MPU6050_6Axis_MotionApps20.h, 2019.

[44] Abishur, *Ms5x*, https://github.com/abishur/ms5x, 2021.

[45] A. Quick, 'Derivation relating altitude to air pressure', *Portland State, Aerospace Society*, 2004.

[46] M. Laboratories. (). 'Engine Speed Monitoring: The Alpha-Beta Filter', [Online]. Available: https://www.mstarlabs.com/control/engspeed.html (visited on 18/09/2021).

[47] G. Welch, G. Bishop *et al.*, 'An introduction to the Kalman filter', 1995.

[48] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, 'Array programming with NumPy', *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[49] ChartJs and contributers, *Chart.js*, https://github.com/chartjs/Chart.js, 2021.

[50] A. Ronacher. (2021). 'Flask Web Development', [Online]. Available: https://flask.palletsprojects.com/en/2.0.x/ (visited on 17/09/2021).

[51] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[52] J. Lima, N. Gracias, H. Pereira and A. Rosa, 'Fitness Function Design for Genetic Algorithms in Cost Evaluation Based Problems.', Jan. 1996, pp. 207–212. DOI: 10.1109/ICEC.1996.542362.

[53] S. Sharma, A. Gosain and S. Jain, 'A Review of the Oversampling Techniques in Class Imbalance Problem', in. Jan. 2022, pp. 459–472. DOI: 10.1007/978-981-16-2594-7_38.

[54] G. Lemaître, F. Nogueira and C. K. Aridas, 'Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning', *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html.

[55] Angular, *Angular: The Modern Web-Developer's Platform*, https://github.com/angular/angular, 2016.