Imperial College London

Imperial College London

Department of Mechanical Engineering

MSc Thesis: Development of a motion and muscle sensing human-machine interface

Nawid Keshtmand

BSc, Chemistry, Imperial College London

A thesis submitted in partial fulfilment and requirements for the Master of Science Degree of Imperial College London and the Diploma of Membership of Imperial College

September 10 2018

Contents

List of Figures

List of Tables

Acknowledgements

Here, the author would like to sincerely express his gratitude to his supervisor, Dr. Ravi Vaidyanathan: Thank you for your support and guidance throughout the course of the project. Moreover, the author also wants to show his appreciation to Samuel Wilson for his continuous support and patience in teaching the author about the NU interface, C# code as well as general support and advice he has given the author on all the different parts of this project. In addition, the author would also like to show my gratefulness to the other members of Biomechatronics lab: Filip Paszkiewicz for his support in the implementation and optimisation of the machine learning and pre-processing of the data, Alexander Wolff for his suggestions for the segmentation of data and optimal features to use in classification, Ashwin Needham for supervising and testing the early development of the Serious game and Thomas Martineau for his guidance on the feature extraction approaches. Also, the author would like to thank Helena Santos Sousa for providing the results of the Support Vector Machine Classification. Furthermore, the author would like to show his gratefulness to Yuxuan Liu and Xinyang Yuan for their suggestions on approaches to take in the project and guidance on how to use the NU interface. Also, the author would like to express his gratitude to the test subjects. Finally, the author is grateful for the support from his family and friends, in particular thankful to his brothers, for their continuous support and guidance they provide him on a daily basis.

Abstract

Medical conditions such as Stroke can lead to severe disability in sufferers. Physical rehabilitation is generally in the the form of repetitive exercises which can be mundane and unmotivating. This work involves developing a Serious game for the purpose of motivating a sufferer of upper-limb control difficulty to perform hand gestures in an interactive and interesting context. Hand gestures are captured using a wearable sensor system consisting of Mechanomyography sensors and Inertial Measuring Units. The Linear Discriminant Analysis and Support Vector Machine classifiers were tested offline by 6 able-bodied individuals and was shown to have an average offline training accuracy of 91.7 % and 99.5 % respectively when using subject-specific training data. A potential issue in the supervised training of hand gesture classifiers, is that large subject-specific data sets can be difficult to obtain for an individual with upper-limb difficulties. Therefore to reduce the burden, the classification accuracy for the Linear Discrimiant Analysis and Support Vector Machine were examined when using a generic pooled dataset and a pooled dataset supplemented by the user's data. The pooled dataset provided an average accuracy of 32 % and 39 % for the Linear Discriminant Analysis and Support Vector Machine classifiers. Adding user-specific data increased the Linear Discriminant Analysis classification accuracy to 42 % and the Support Vector Machine classification accuracy to 55 - 60 %. The results show that there is potential in using user supplementary datasets in training MMG hand classifiers to minimise muscle burden on the user.

1 Introduction

1.1 Background

Human Machine interfaces (HMIs) are crucial in society as a means of enabling individuals to interact with cutting-edge technology. HMIs consist of 3 main parts:

- A stimuli made by the human user
- A sensor to measure the stimuli
- A response by a computer.

By improving the accessibility of cutting edge technology, individuals can gain a range of benefits such as a potential reduction in workplace injuries, as well as increasing the degree of independence for the elderly and the disabled [8]. Due to advances in modern day healthcare, there is an increase in the number of people living longer, with an increase in patient survival rates. Although this is often at the cost of a reduction in their functional capacity post operatively [9]. Therefore, HMIs which provide a means of replacing or bypassing an individuals motor impairment and thus increase their functional capacity are of particular importance and likely to become more significant in the future.

The NU interface is a motion and muscle sensing HMI built by Samuel Wilson in the Biomechatronics lab at Imperial College London. It has already proven to be applicable to control a range of different objects such as a phone, television, prosthetics, as well as a Baxter robot. Therefore, the likelihood that the NU interface could be used to provide assistance to the disabled or elderly is high. The NU interface consists of both hardware and a complementary Windows form application made in C#. The hardware is a wristband which consists of two parts: an Inertial Measuring Unit (IMU) which measures the motion (acceleration and rotation) of the arm and Mechanomyography (MMG) sensors to measure the muscle activity of the forearm muscle.

Figure 1.1: NU interface Windows form application

1.2 Motivation

Stroke is a major problem in the UK, with an estimated 111,000 first-time strokes occuring each year. Stroke can often cause severe physical disability such as attention deficiency, pain, weakness and paralysis, often on one side of the body. Such impairments can result in loss of ability to carry out typical day-to-day activities [10]. Post-stroke rehabilitation aims to help sufferers of stroke to restore movement in their limbs by carrying out exercises on a regular basis. However, patients have noted that performing these exercises can be laborious and mundane [11]. Therefore, activities which can encourage patients to perform these exercises can prove highly beneficial in a rehabilitation context. One potential method could be to develop a video game to encourage patients to perform these exercises.

Video games are a well-known means of providing excitement, motivation, challenge and enjoyment to those who play them. Depending on the design and content of the video game, they can be a unique medium in transferring important information. Video games which have been designed to be associated with education and the teaching of new skills and concepts are known as Serious games [12]. Many scientific studies have proven the positive effect of Serious games in a wide range of areas such as reaction time, cognitive ability, visual acuity, hand-eye coordination as well as the development of social skills [12]. Moreover, improvements in Virtual Reality technology enables different scenarios and environments to be constructed which may be impossible to reconstruct in real-life. This can improve the quality of learning from the game as this can allow users to experience simulations of past events as well as providing realism to the game which can encourage users to become fully immersed in the game. Due to the wide range of virtual environments that can be simulated, video games are a tool that can be orientated towards the treatment and education of different sectors of the population (including children and the disabled).

Serious games can also be used for several applications including:

- Training professionals
- Health prevention
- Upper or lower limb rehabilitation.

The design and difficulty of the Serious game will have to depend on the goal of the game. Games for limb rehabilitation would be designed to help during the rehabilitation process by making exercises more enjoyable and easier than traditional methods. However a Serious game for upper-limb rehabilitation would need to allow the user to go at their pace as the emphasis of the game should be for the user to perform the technical movements correctly rather than performing the movements quickly, especially as fast movements may be too difficult and it could cause the user to become frustrated. Due to these benefits of Serious games, the aims of this work were to:

- Develop a software algorithm to optimise the detection, processing and classification of hand gestures using the MMG data from the wearable sensor system
- Develop a Serious game in the Unity game engine which could be used for the purpose of upper-limb rehabilitation.
- Use the software algorithm in conjunction with the Serious game.

2 Literature Review

HMIs have expanded over the years and they have been instrumental in allowing humans to control complex machines such as robots and drones [13]. There are various different stimuli for HMIs such as light, sound, touch, motion and biological based stimuli [8]. Depending on the specific HMI, either one or several different stimuli can be detected.

In the past, gesture recognition was solely based on computer vision approaches where camera systems would record an individual performing actions. However, this approach was limited in range as it required the individual to be in the presence of a camera and additional equipment such as light reflective markers as seen in Fig. 2.1 [14].

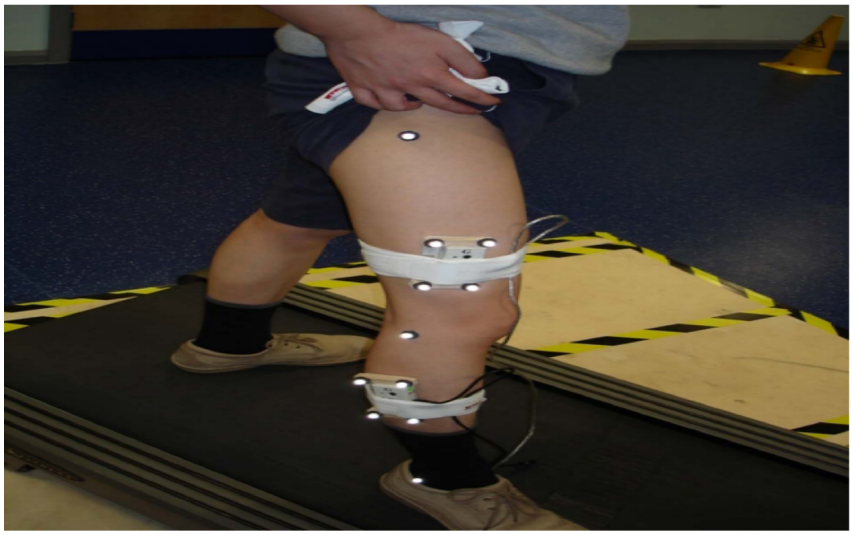


Figure 2.1: Light reflective markers on a test subject [1]

To overcome this limitation in range, biological based HMIs which enable an individual to control a machine through conscious control of their physiological signals can be used instead. This has been made possible due to the improvements in distinguishing and classifying different physiological signals into different classes by using pattern or non-pattern-based recognition approaches [15].

Research on Biological based HMIs have been primarily focused on muscle activity related to the extremities of an individual. This has been primarily due to the ease of controlling muscles on the hands or the feet [16]. However, other research has involved the measurement of brain signals [17] or tongue-movement ear pressure signals due to the movement of the tongue [6, 18], which can potentially be used for assisting physically impaired individuals who have limited body mobility. The most common method for the analysis of muscle signals is through the process of Electromyography (EMG), which measures the electrical stimulation resulting from the action potential produced by skeletal muscle contraction [19].

Non-pattern based methods of EMG signal recognition include 'on and off control' and 'proportional control' [20]. These methods are based on the amplitude of the EMG signal and the response is proportional to the amplitude of the voltage produced from the action potential. This generally only enables two different gestures to be identified such as an open clench and close clench gesture. It is possible for additional gestures to be identified using non-pattern based methods when EMG electrodes are used to identify individual finger movements. But, this is generally difficult to achieve without implantation of the EMG electrodes due to the crosstalk in EMG signals from the muscles relating to different fingers [20]. Pattern based recognition involves examining the amplitude as well as the shape of the signal, which is important as different gestures vary in both these ways as shown in Fig. 2.2.

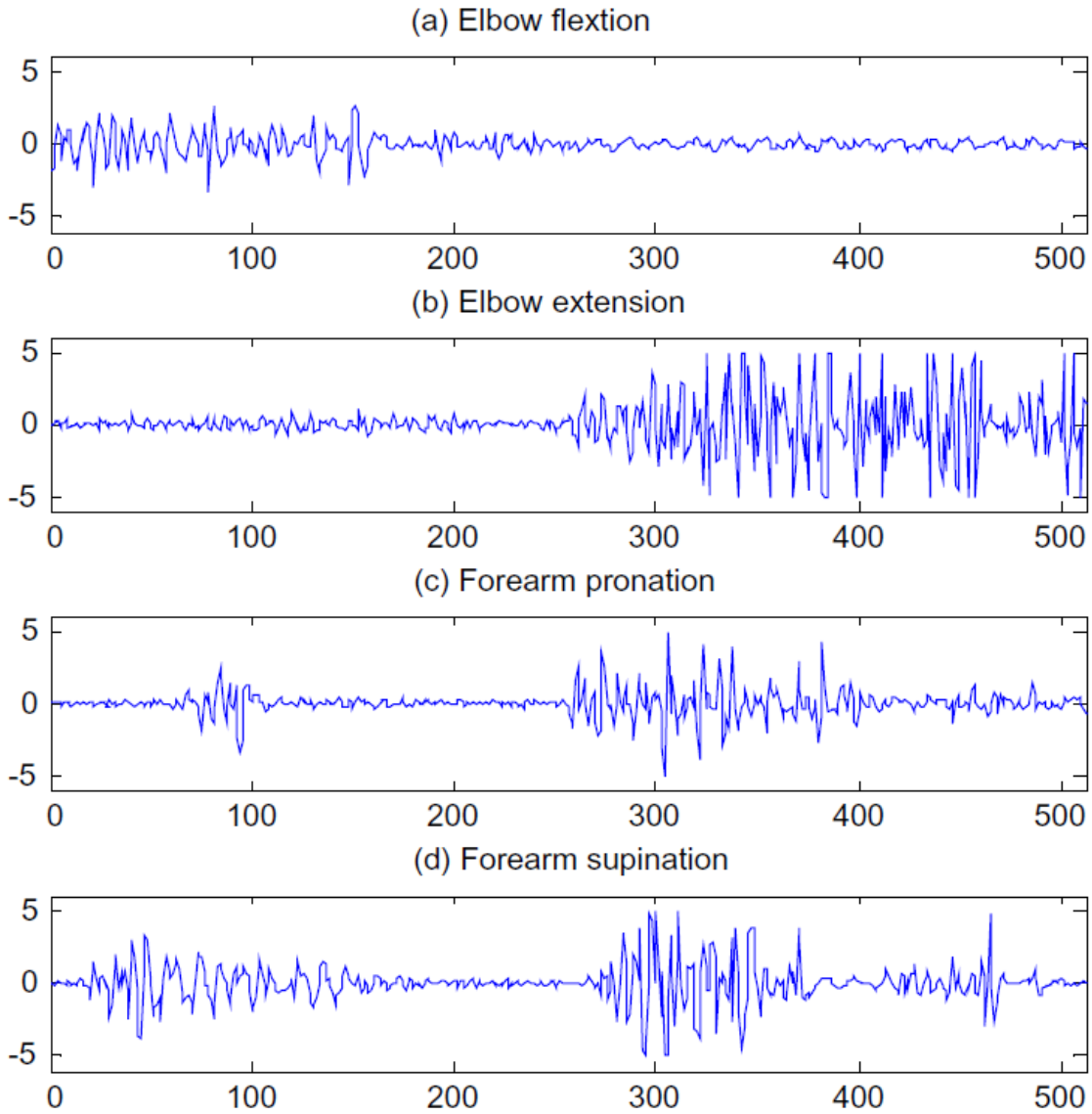


Figure 2.2: EMG data sample from 4 different arm movements [2]

Pattern recognition involves various steps such as [15]:

- Preprocessing of the signal such as filtering and smoothing by using a moving average
- Segmentation of the area of interest
- Feature extraction (most commonly on time domain features)
- Classification of the features.

Pattern recognition is generally superior than non-pattern based recognition as it is generally able to recognise a larger number of gestures with higher accuracy, however due to the high computational intensity, the effectiveness of these pattern recognition systems are limited in real-time [15, 16].

Pattern recognition of EMG signals have been studied in depth with previous works involving using surface EMG signals from the forearm muscle with different classifiers such as Multilayer Perceptrons [21], Hidden Markov models [22, 23] and other classifiers such as Linear Discriminant Analysis (LDA) [24].

Various studies have been completed on the classification of EMG signals and it can be seen that there is no minimum number of channels required to achieve high classification accuracy, as depending on the processing and classifier, high accuracies can be achieved. This can be proven as when 8 EMG channels were used, an average accuracy of 92% was achieved for classifying 7 gestures [24] whilst a classification accuracy of 90 % was seen when using 4 EMG channels to classify 6 different gestures [25]. However, there is a general trend that increasing the number of input channels increases the classification accuracy which is due to there being more information present from the different parts of the muscle being measured. The information obtained from EMG sensors is generally maximised by placing the sensors equidistant around the circumference of the muscle of interest as shown in Fig. 2.3.

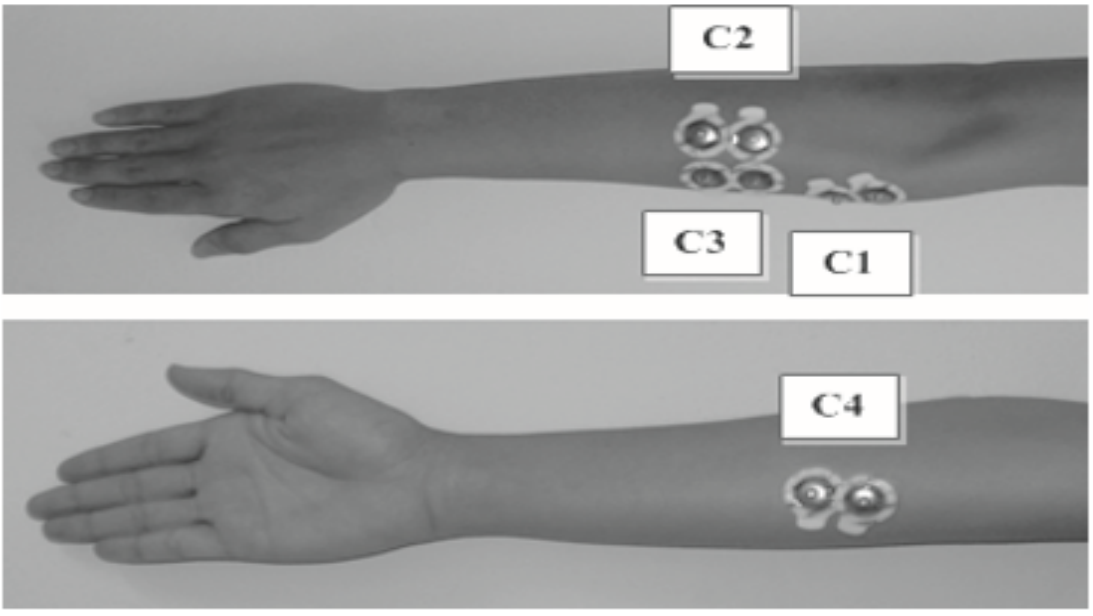


Figure 2.3: EMG sensors placed at different places on the forearm [3].

Furthermore, it has been seen that muscle movement of the biceps and triceps, which is unrelated to the forearm muscle, can also be classified effectively [2]. This shows that there is potential of using several EMG sensors on different parts of the arm which could allow a greater number of gestures to be recognised with a high accuracy.

Pattern recognition of EMG signals can also has been in conjunction with other sources of data to enhance classification accuracy of gesture recognition. This has been seen in the context of EMG and IMU signals being utilised in conjunction for the purpose of controlling prosthetics as well a range of different robots [14, 22].

One of the major issues affecting the use of EMG outside a clinical environment is that it is often subject to interference from physiological factors, such as skin impedance. Skin impedance can be affected by conditions such as the users level of scar tissue and sweat [26]. Sweat level is a particularly significant issue as sweat levels can change in time which can cause calibration errors in an EMG based control system, which therefore limits the long-term applicability of EMG [4]. An alternative approach to measuring muscle activity, which can overcome the issue of skin impedance, is Mechanomyography (MMG). MMG involves the measurement of low frequency vibrations between 2-200 Hz which are produced during the muscle contraction of skeletal muscles. Additional benefits of MMG compared to EMG include a higher signal-to-noise ratio and elimination of the need for shaving and conductive gel [27, 28].

Pattern recognition of MMG signals have been used in various contexts including gait analysis, control of prosthetics as well as the diagnosis of degenerative disorders where the signals from at least five different muscle actions have been identified as being distinct from one another [4, 29, 30].

Similarly to EMG pattern recognition, many studies have been conducted using MMG pattern recognition which have proven that MMG is a viable replacement for EMG in the controlling of multi-function devices. One study examines the pattern recognition of MMG signals for the purpose of controlling a commercial prosthetic hand (Bebionic Version 2) where 6 MMG channels are used in a template matching classification approach. It was seen that an offline classification accuracy of 83.5 % for 7 gestures was obtained when testing on 5 healthy individuals and 1 trans radial amputee [4]. This shows that MMG can be used as a successful alternative to EMG and is robust enough for use by both healthy individuals and amputees. Other studies show that when using more sophisticated classifiers such as LDA or a Convolutional Neural Network (CNN), accuracies of 93 % can be obtained for 8 different forearm motions and 94 % for 5 different hand gestures [13, 31].

Furthermore, previous research by Prociow et al [32] has studied the effects of using both EMG and MMG in conjunction for the classification of 7 gestures using a Learning Vector Quantisation Neural Network. This research showed that classification using both MMG and EMG had greater accuracy (93.83 %) than using solely EMG (91.36 %) or MMG (81.48 %). However, this improvement in classification accuracy when using both EMG and MMG may have been due to the fact that greater number of channels were used to collect more information from the forearm muscle rather than a synergistic effect of using both MMG and EMG channels [32] .

Despite the effectiveness and high classification accuracy of EMG and MMG for gestures classification purposes, there are several factors that prevent transition from the laboratory to clinical application. Generally, laboratory testing involves controlling several factors such as arm posture, fatigue and sensor position which can affect the data acquisition process from muscle contraction. However, in daily living, it is difficult to control these factors and this can lead to variation in the data collected and a decrease in gesture recognition accuracy [16]. However, there is potential for effective use of EMG and MMG outside a clinical environment as research has been conducted on using EMG and MMG over long periods of time in uncontrolled environments for the purpose of gait and lower-limb fatigue analysis [29, 33].

The three best known approaches to the pattern recognition are template matching, statistical classification and Neural Networks. Template matching in pattern recognition is used to determine the similarity between two entities (points or shapes). This involves a template being made for the different classes and then the entity to be recognised is matched against the different templates. A similarity measure is used to quantify the degree of similarity between an entity and the different templates, which then enables the entity to be classified into the group with the greatest similarity [34]. In a statistical classification approach, each pattern is represented in terms of D features (known as a feature set) and is viewed as a point in a D-dimensional space. The goal of statistical classification is to use the features of a pattern to identify which class it belongs to. The effectiveness of the feature set is determined by how well patterns from different classes can be separated from each other [5]. It is generally desired for the value of a feature to have low intra-class variation whilst having high inter-class variation. Given a set of training patterns from each class, the objective is to establish decision boundaries in the feature space which separate patterns belonging to different classes [34].

3 Hardware

In this project, the hardware used for the recording the MMG and inertial data was designed and fabricated by Samuel Wilson at the Biomechatronics lab of Imperial College London. To record the inertial and muscle activity, it was necessary to consider the following factors:

- The number of degrees of freedom (DOF)
- Range of the sensors
- Frequency at which the data is recorded
- Usability of the hardware

The inertial measuring unit (IMU) possesses 9 DOF as it can read 9 unique parameters to define itself. The IMU achieves this as it contains 3 accelerometers to define the linear acceleration along the 3 principal axis, 3 gyroscopes to define the rotational orientation and 3 magnetometers to measure the strength of the magnetic field.

The range of the accelerometers, gyroscopes and magnetometer was set to $\pm$2G, $\pm$500 degrees per second and $\pm$2 gauss respectively. These values correspond to the typical range of human movement and are effective in the calculation of the limbs orientation during manipulation tasks [4]. The sensor data (MMG and IMU data) was streamed via Bluetooth to the Windows form application where subsequent processing of the raw inertial and MMG data occurred.

The IMU was custom made with a maximum frequency of 1000 Hz to ensure that the sampling rate of the recording device was sufficiently high to prevent aliasing of the high frequency features, which could arise from the interaction of the different channels of the recorded MMG data [4].

The size of IMU used in the project was 3.7cm x 2.5cm x 1.1cm and the weight was only 12g (including the battery and the shell box). Overall, the IMU was lightweight, small in size and easy to wear. Furthermore, Bluetooth was embedded in the hardware which enabled wireless use of the sensor system.

The MMGs were designed to ensure optimum data collection from the muscle vibrations. As described in the research by Samuel Wilson and Ravi Vaidyanathan, the MMGs consisted of a micro-elecro mechanical microphone placed in a printed case with the opening covered with a mylar membrane to optimum signal transmission from the skin to the microphone[4, 35]. 6 different MMGs were distributed evenly around the wristband used to collect data from different parts of the forearm muscles.

It has been seen that the amplitude of the MMG signal is maximised when the contact pressure between the sensor and the muscle is high and when the sensor is placed at the centre of the muscle [36]. Therefore, MMG signals are highly sensitive to contamination by motion artifact as moving limbs can affect the amplitude of the MMG signal as the movement can cause variations in the contact pressure as well as the relative position between the MMG sensor and the muscle. Furthermore, due to the need to move the IMU to provide directional commands, contamination of the signal by motion artefacts is likely to occur.

Therefore, to minimise the motion artifacts, a velcro strap was implemented during the design of the wearable sensor system which enabled the tightness to be optimised for each user to ensure a consistent contact pressure and minimise the relative movement of the MMG sensors on the muscle. The design and the different components of the wearable sensor system can be seen in Fig. 3.1. Overall, the sensor system used in this project satisfies all the basic requirements for the collection of data related to the muscle activity and limb orientation of a user.

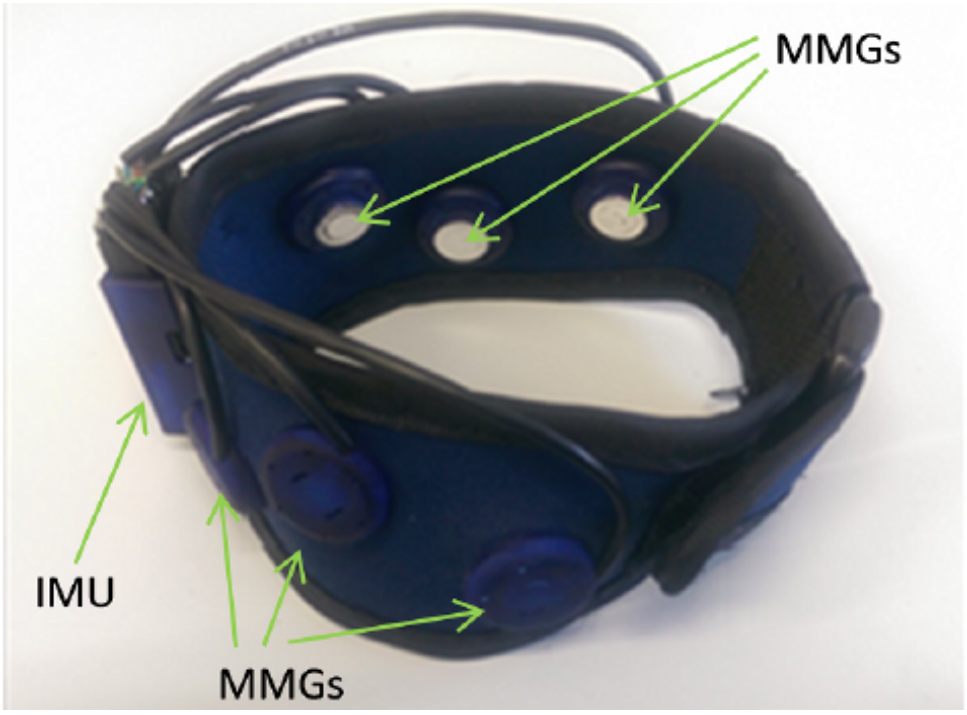
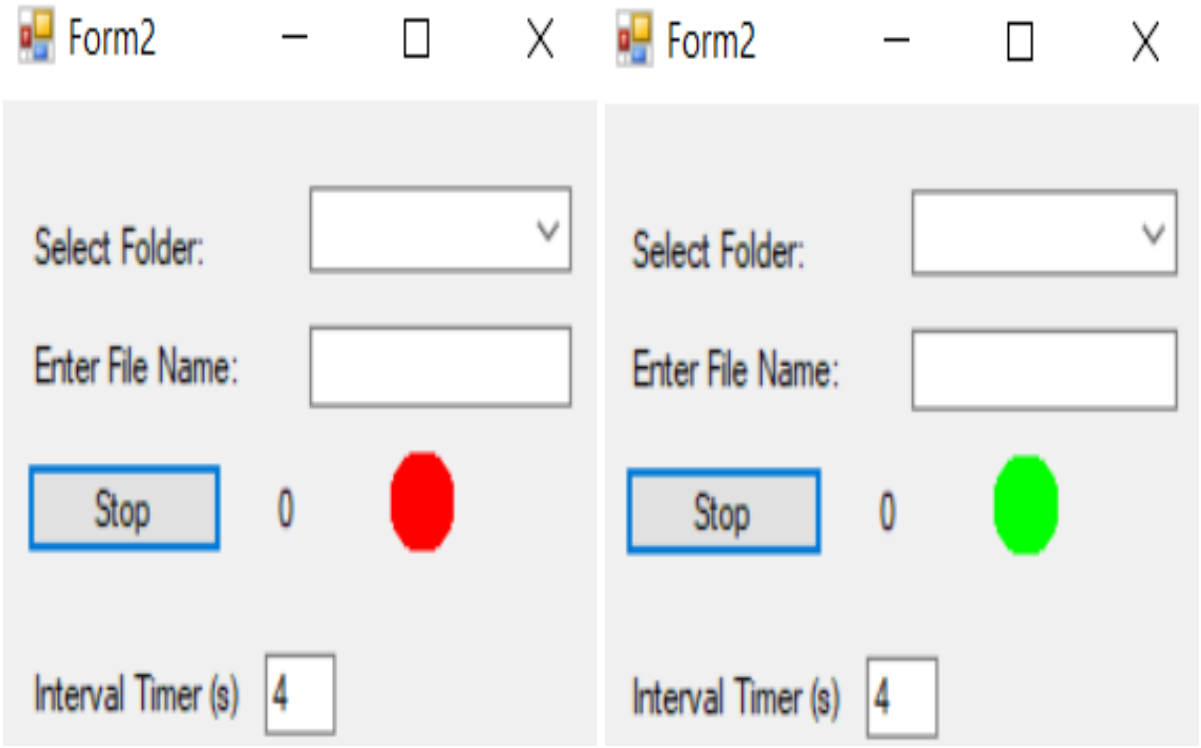


Figure 3.1: Armband showing sensor position [4]

4 Experimental protocol

Data for algorithm development and offline testing was collected by Samuel Wilson from 6 healthy subjects (3 male, 3 female). The data collection procedure occurred as described in the previous research performed by Samuel Wilson [4]: The data was measured using 1 IMU and 6 MMG sensors which were located on an armband which was placed on the subjects forearm, with the IMU uppermost on the arm when the subjects hand was held palm down. To minimise the motion artifacts from the movement of the limbs and to minimise the variance of the MMG signals due a variation of the position of the arm, each subject was seated at a desk, with their back straight and their elbow placed on the table. The forearm was held as vertical as possible, to ensure that the minimum level of muscle activity was necessary to maintain the hands position.

The data was recorded with the aid of a timer which would change in colour after a specific time interval to inform the user when to make a gesture. The length of time between the gestures were dependent on whether a gesture would require full or partial activation of the forearm muscles. Large gestures which require full activation of muscles, required a gap of 4 seconds, whereas small gestures would require a gap of 1-2 seconds between gestures. The different states of the timer can be seen in Fig. 4.1.



(a) 'Do not perform gesture' state of timer

(b) 'Perform gesture' state of timer

Figure 4.1: Different states of the timer

Immediately following the action of a gesture, the hand would relax to prevent a single gesture being recognised as 2 separate gesture events. Each subject performed 100 instances of each of the seven gestures shown in Fig. 4.2. The MMG and IMU data was recorded and sent to the NU interface where it would be used in real-time or saved as CSV files for offline analysis. The real-time data or the offline data would then be read in MATLAB R2017a for processing and classification.

Figure 4.2: Gestures used for Test 1 Top Row, Left to Right: Rest position, Open (1), Close (2), First Finger Pinch (3) Second Row, Left to Right: Middle Finger Pinch (4), Thumbs Up (5), Point (6), Finger Roll (7) [4]

5 Software algorithm

The wristband collects the MMG and the IMU data from the user and sends the data to the Windows form application to interpret the user's commands. The raw inertial data was processed using a gradient descent algorithm to calculate the world referenced orientation of the IMU [37], which enabled the calculation of the roll, pitch and yaw angles used for directional commands.

For the offline analysis, data was collected from each sensor and saved in CSV files where each CSV file contained 10 gesture instances each. This data was used to produce a matrix with 6 columns with each column containing the data of each MMG sensor varying in time. For the real-time analysis, the MMG data collected by the NU interface was transmitted to MATLAB R2017a by using a MATLAB application in the C# code. In MATLAB R2017a, the MMG data was processed and classified then the results of the classification was sent to Windows form application before being passed to Unity through an anonymous pipe. The gesture analysis consists of various stages which can be seen in Fig. 5.2.

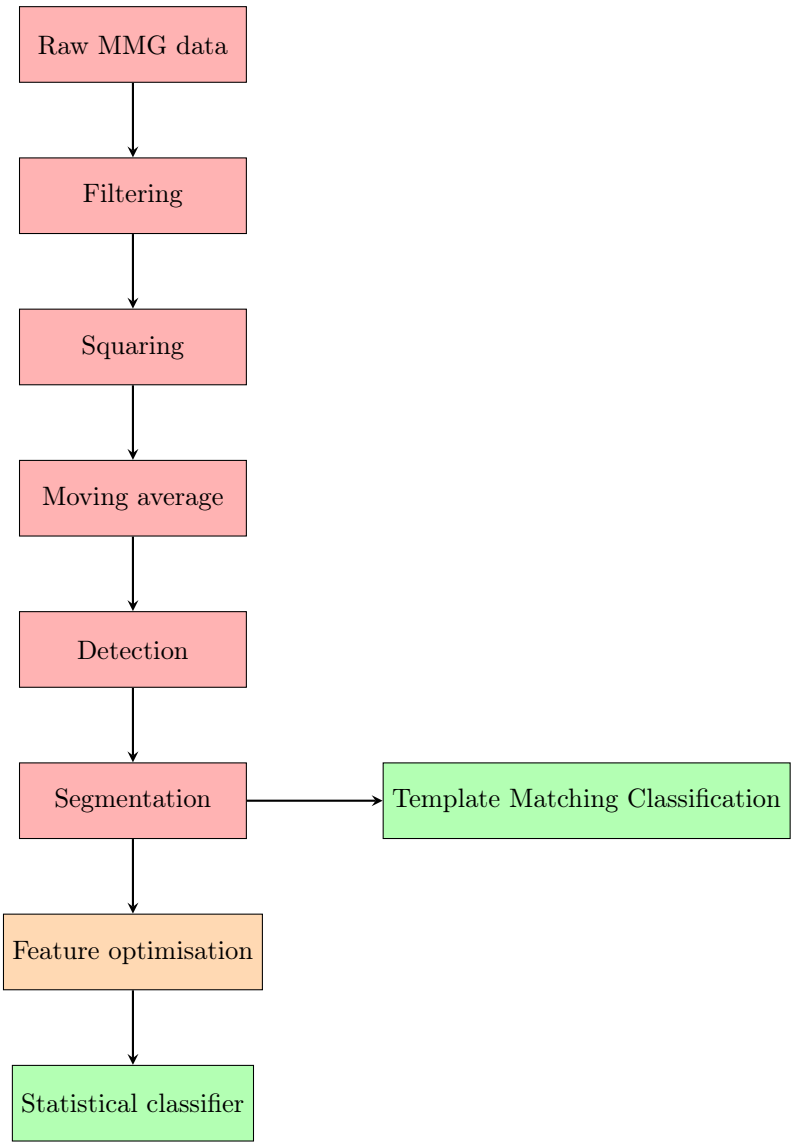


Figure 5.1: Flowchart showing the different steps in the gesture analysis

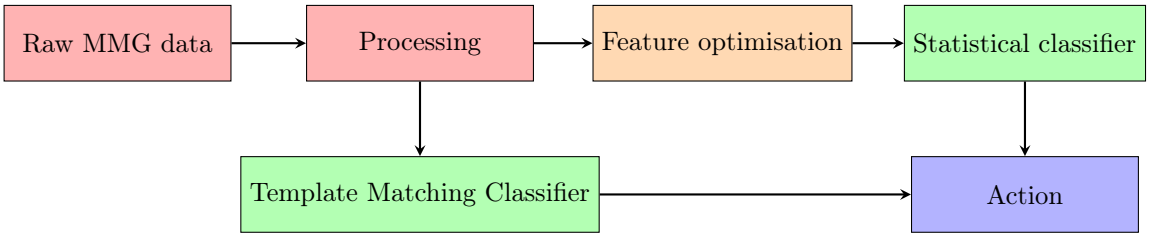


Figure 5.2: Flowchart showing the different steps in the gesture analysis

5.1 Processing data

The initial processing of the raw MMG data consisted of filtering, squaring and performing a moving average. After the initial processing, the amplitude of the processed data was examined to see if it went above a certain threshold which would signify a gesture instance event. Following this, the window of data corresponding to the gesture instance would be segmented. The extraction of the MMG data from the CSV files was achieved using the 'Raw MMG data extraction' function shown in Appendix. A. The filtering of the data was achieved in MATLAB by using the 'Filtering' function shown in Appendix. B and the detection of the gesture instance event was completed by the 'Segmenting' function showin in Appendix. C. The squaring, moving average and segmentation of the data was performed using the 'Processing data' function in Appendix. D.

5.1.1 Processing data: Initial processing and gesture detection

To minimise the motion artifacts present and remove other sources of noise in the MMG data, the matrix of raw data m was band-pass filtered between 1 – 100 Hz using a first order Butterworth filter to produce a matrix m*. The band-pass filtering was particularly effective as it was able to shift the baseline amplitude of the MMG signals to a value of 0 which made identifying the absolute amplitude of the signal more apparent. The effects of the filtering can be seen by looking at Fig 5.3 and Fig 5.4.

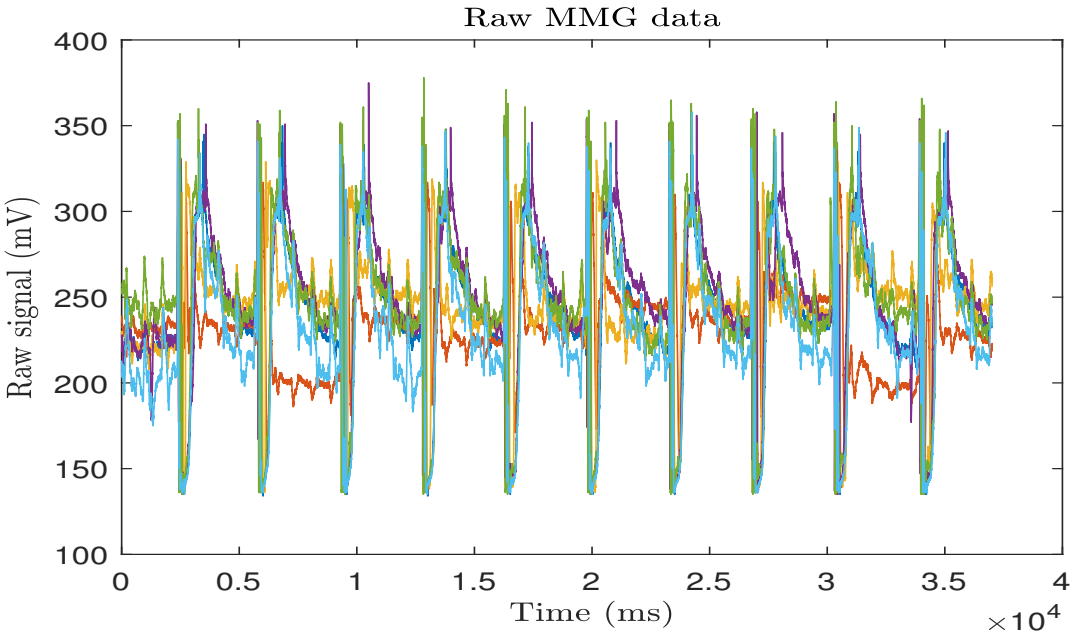


Figure 5.3: Raw MMG signal

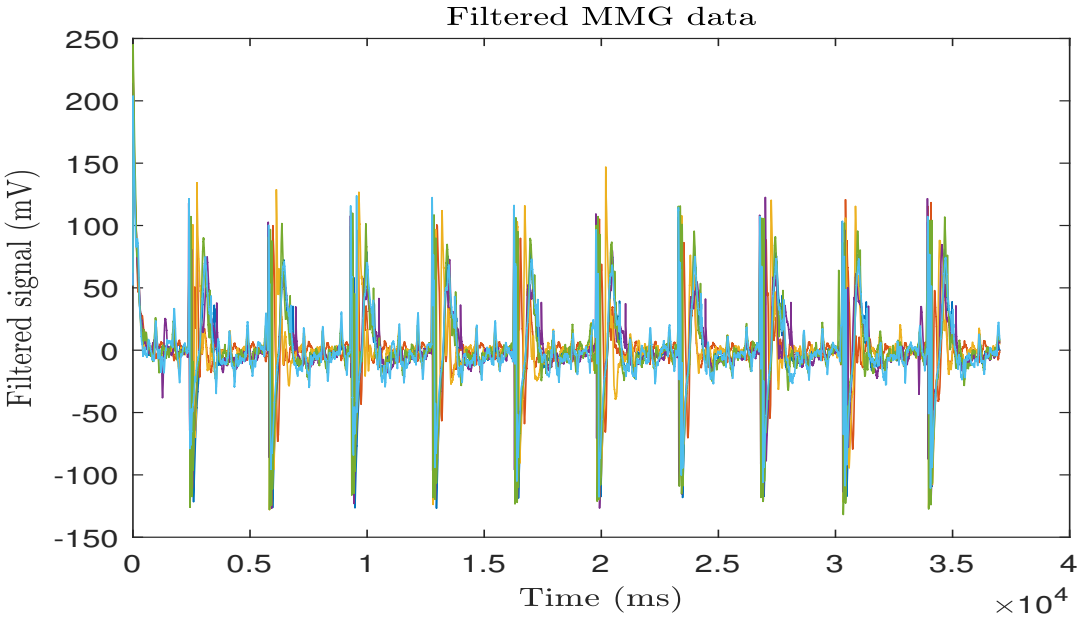


Figure 5.4: Band-pass filtered MMG signal

In order to detect whether a gesture had occurred, the power of the MMG signals were monitored to see if it exceeded a pre-defined threshold. The overall power of the gesture was defined as being the summation of the signals from all the MMG sensors. It was important to take into account all the different MMG channels of data as different gestures would lead to differing levels of activation from the different parts of the forearm muscle. As the quantities of interest in the signal was the total amplitude and the frequency of the MMG signal, the sign of the MMG signal made no effect on the total power of the signal, therefore, the elements in the matrix m* was squared to give the matrix $m^{squared}$. The effect of squaring the data can be seen in Fig. 5.5.

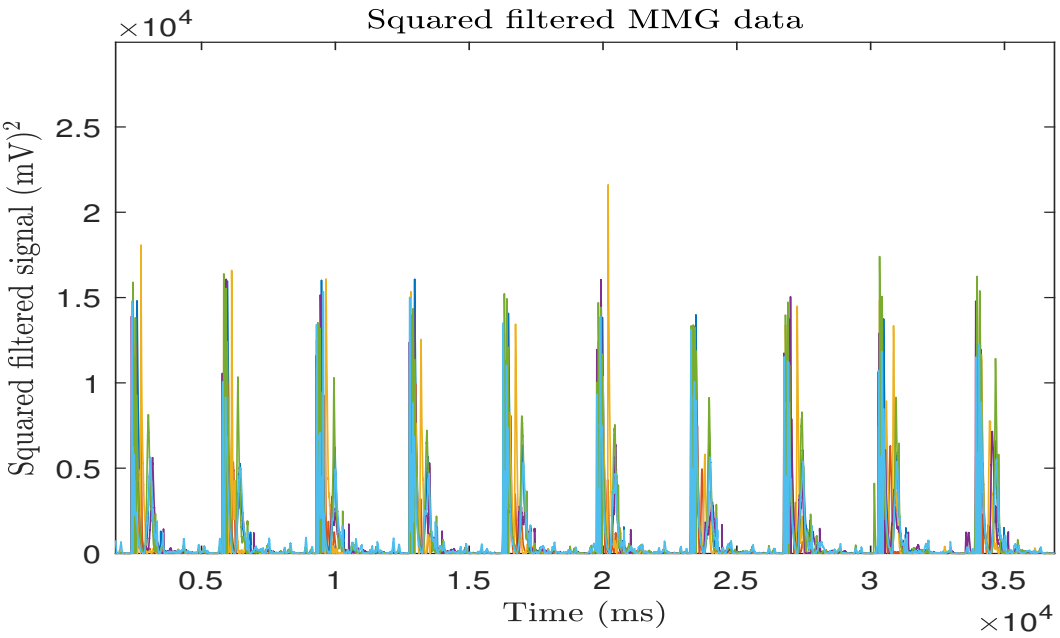


Figure 5.5: MMG data after filtering and subsequent squaring of the data

However, movements of the arm or incomplete gestures, could also lead to high amplitude MMG peaks which would lead to false positives. To mitigate against these type of false positives, a 150 point forward moving average was performed on matrix $m^{squared}$ to give matrix $m^{average}$. This lead to the amplitude of short lived peaks smoothening out, whereas long lived peaks from complete gesture were less affected and were able to reach the desired threshold [38]. 150 points were chosen for the moving average as it was generally seen that the peaks lasted approximately 150 ms, which can be seen in the second peak in Fig. 5.6. The smoothening effect of the moving average can be seen in Fig. 5.7.

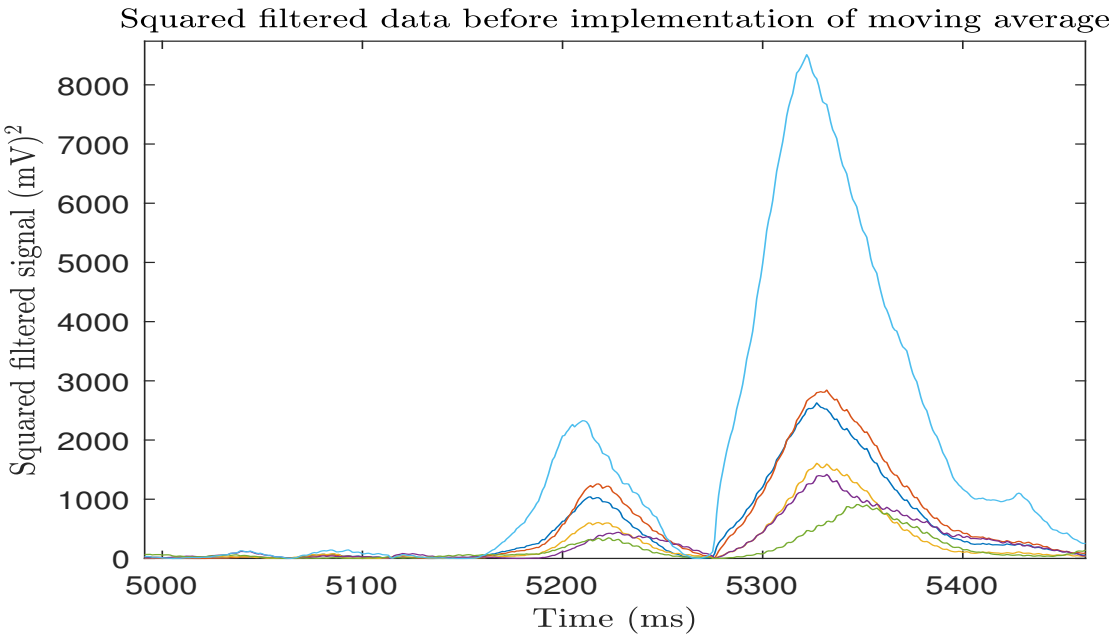


Figure 5.6: MMG signal peak before smoothing with a moving average

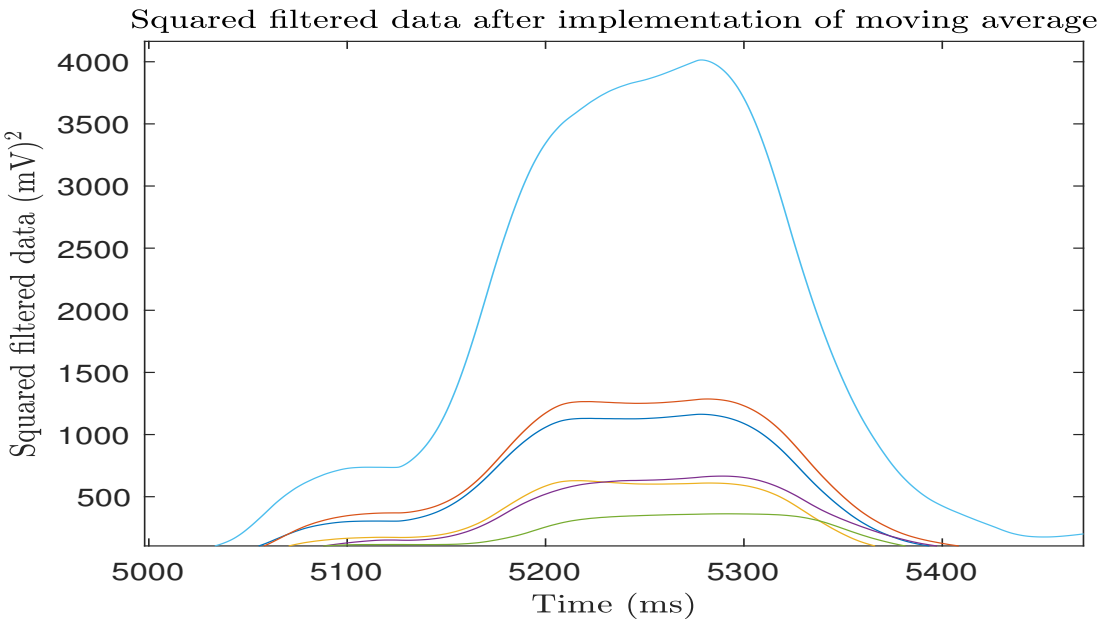


Figure 5.7: MMG signal peak after smoothing with a moving average

The energy of a gesture, e_{gest}, can be defined as:

$$e_{gest} = \sum_{j}^{N} m^{average} \tag{5.1}$$

Where N is the number of sensors that were used in the system. The offline data was analysed when deciding on a threshold to indicate whether a gesture had occurred. By looking at the MMG data from the different gestures, the goal was to find an energy threshold T_{MMG}, that marked the maximum number of gestures, whilst minimising the number of false positives across all the test subjects. This was difficult to achieve as the signal amplitude varied between different individuals depending on the position of the MMG sensor relative to the forearm, the skin thickness (fat content) as well muscularity of the individual. Furthermore, there was slight deviation in the amplitudes of signals from a single individual due to the level of force exerted by the individual and the level of fatigue that the individual would accumulate. However, a larger issue present was that different gestures performed by a single individual could vary quite significantly in terms of amplitude, therefore it was difficult to obtain an optimum threshold which could be used for all the different gestures. Small gestures and large gestures varied significantly in amplitude and an example of the difference in the MMG signal magnitude between a small gesture and a large gesture can be seen in Fig. 5.8 and Fig. 5.9

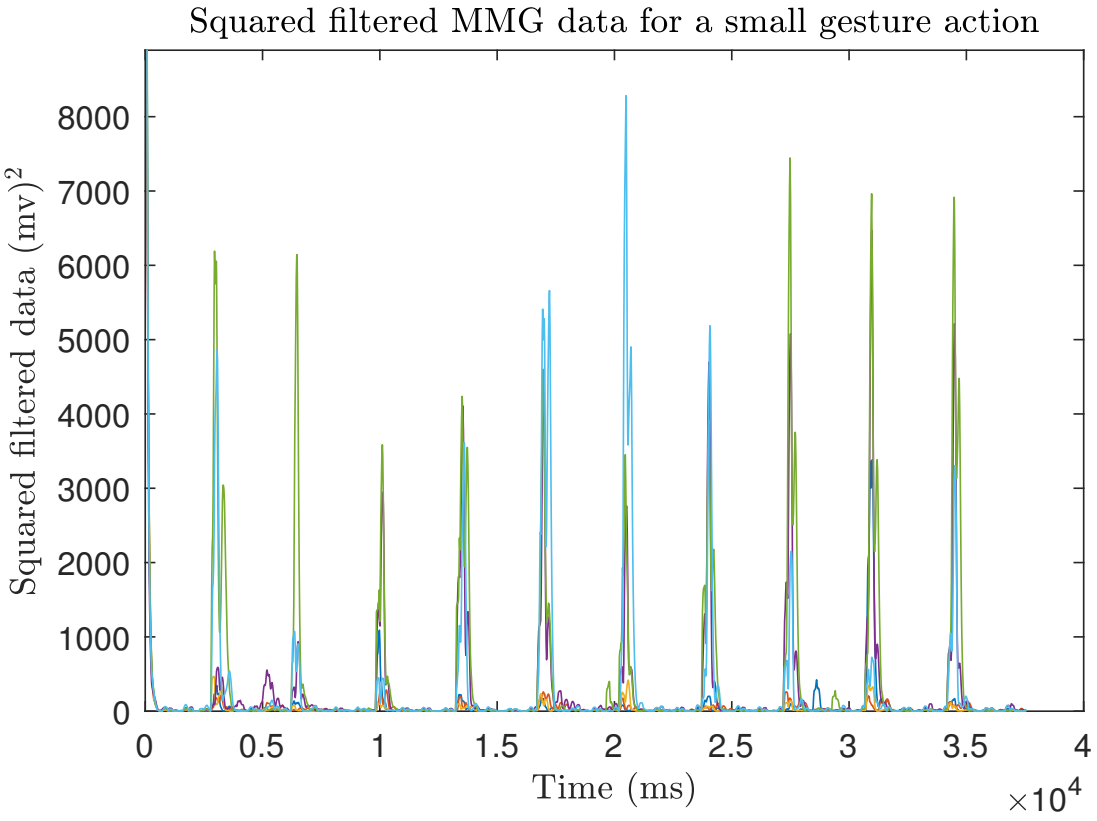


Figure 5.8: MMG signal magnitude for a small gesture

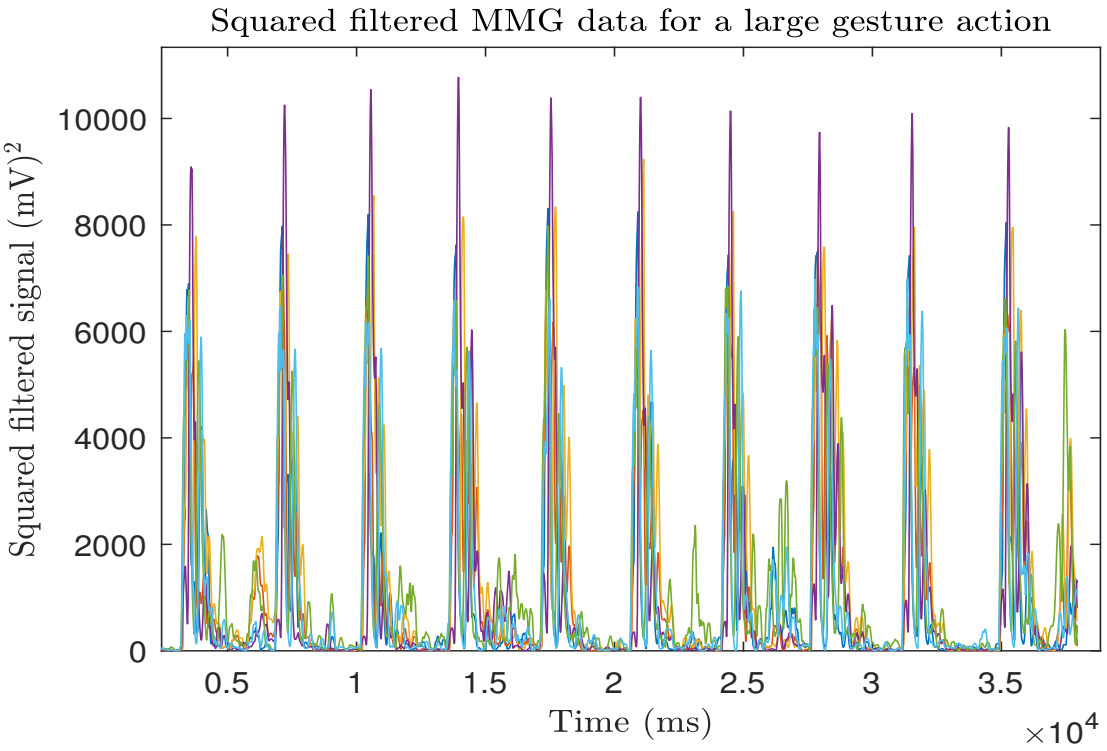


Figure 5.9: MMG signal magnitude for a large gesture

Therefore, having a low threshold which is suitable for small gesture activation could lead to false positives when a large gesture is performed as the smoothing effect of the moving average may not lower the amplitude sufficiently. However, having a high threshold suitable for the large gestures would lead to false negatives when a small gesture is performed as the threshold would not be reached or it would require the user to exert significantly which would not be sustainable due to fatigue and discomfort. Many previous works utilise calibration training step to find thresholds for a particular user, however this likely finds a single threshold which is a compromise between the optimum threshold for a large and a small gesture [39]. To overcome this issue in the offline analysis, the offline analysis algorithm used 2 different thresholds, one for the larger gestures and a second for the smaller gestures. This was done using the 'Segmenting' function shown in the Appendix. C. The function looks at different points of the matrix $\mathrm{m}^{average}$ to see if there are any points that exceeds the upper threshold $\mathrm{T}_{MMGUPPER}$ first. If the upper threshold was too high and there were no points in time where the summation of the MMG signals exceed the upper threshold, then $\mathrm{m}^{average}$ would be re-examined to see if there any points that exceed the lower threshold $\mathrm{T}_{MMGLOWER}$. Furthermore, to prevent false positives in the classification of gestures due to motion artifacts caused by the movement of limbs, Samuel Wilson implemented a threshold T_{gyro} into the C# code which was based on the gyroscopic energy (G_x,G_y,G_z). This gyroscopic threshold would ignore signals detected at the point when the rotational acceleration of the arm was above the threshold. A gesture was said to occur at time i if Occ = 1 where:

$$Occ = \begin{cases} 1, & when\ ||G|| < T_{gyro}\ \&\ e_{gest} > T_{MMGUPPER}\ or\ ||G|| < T_{gyro} \quad \& \quad e_{gest} > T_{MMGLOWER} \\ 0, & otherwise \end{cases} \tag{5.2}$$

5.1.2 Processing data: Gesture segmentation

The main goal of segmentation was to find the time instants in which abrupt change in the properties or dynamics of a signal occur. The 10 gesture instances within a CSV file were segmented using a change detection method where the data was processed sequentially and when a change was detected, the detector was restarted [40].

In this case, the matrix $m^{average}$ was processed sequentially until there was a point where Occ =1, then a segmentation point was marked to represent the occurrence of a gesture event. The detector was then restarted and skips the next 1500 points (1.5 seconds of data) after the previous segmentation point, in order to prevent the detector from misclassifying a single long lasting gesture event as 2 separate gestures events. This process of finding segmentation points was completed until 10 different segmentation points were found when using upper energy threshold, $T_{MMGUPPER}$. If 10 different segmentation points were not found using $T_{MMGUPPER}$, the process of finding segmentation points would be repeated from the start of the matrix $m^{average}$ whilst using the lower energy threshold $T_{MMGLOWER}$ instead. After the segmentation points were found, the gesture instances in the filtered data in matrix m* would be segmented into non-overlapping segments and collated into a matrix $m^{collated}$. The locations of the segmentation points in the $m^{average}$ and m* can be seen in the Fig. 5.10 and 5.11.

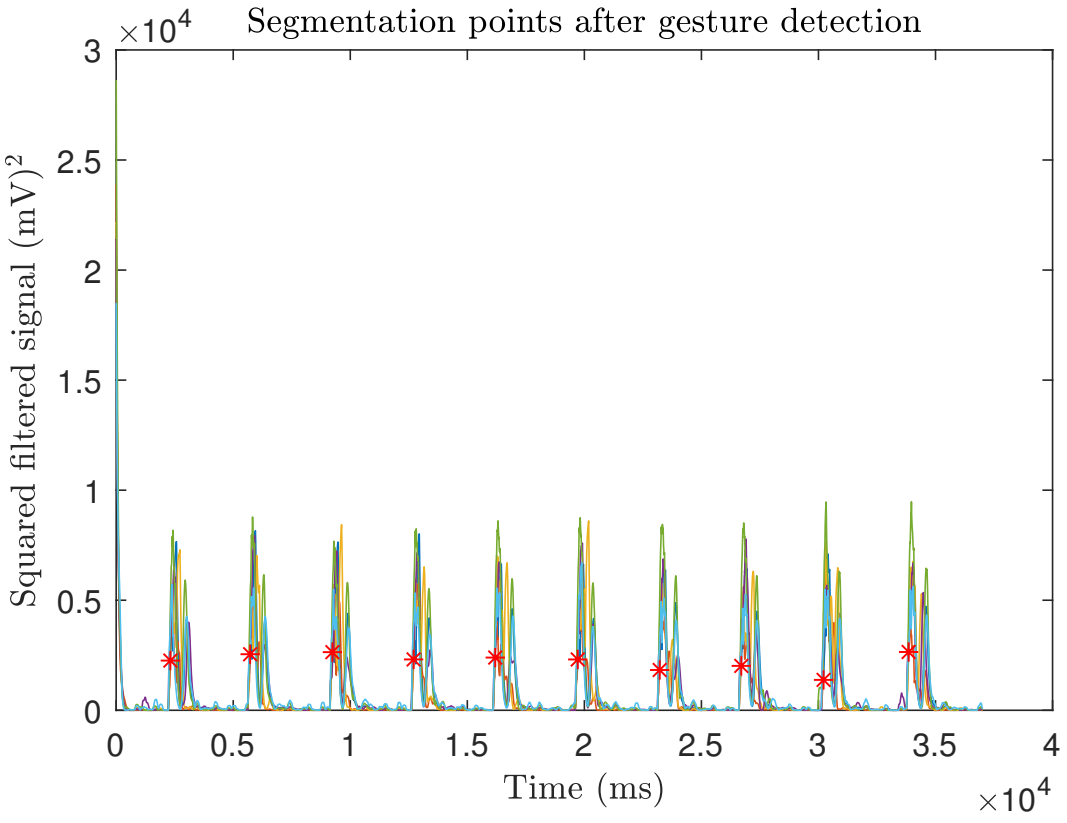


Figure 5.10: Marking of the segmentation points when MMG signal after the moving average reaches the threshold

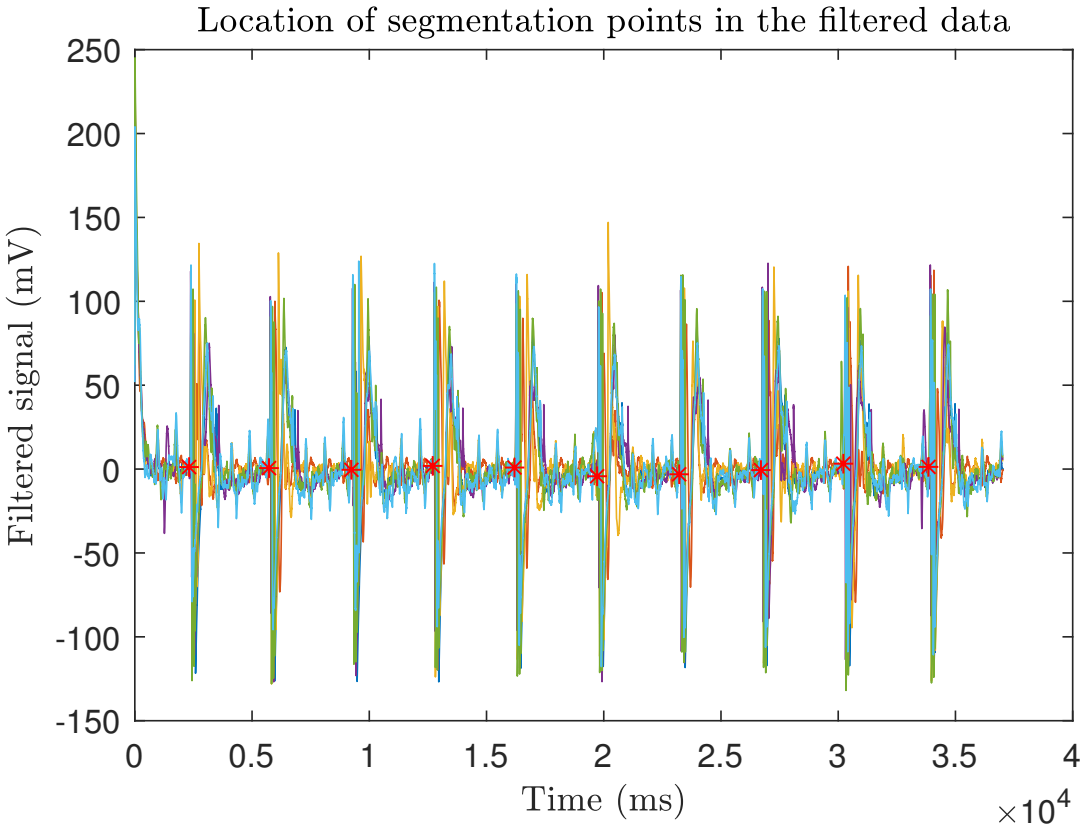


Figure 5.11: Segmentation points located of the filtered MMG data

Segmentation was important as it reduced the computational intensity of subsequent processing steps and improving the classification accuracy as it removed the unnecessary information such as the resting periods or additional motion which occurred outside the gesture period.

An MMG signal of a gesture event comprises of two states [38]:

- A transient state originating as a muscle goes from rest to a contraction
- A steady state originating during a constantly maintained contraction in a muscle.

Various methods have been used for segmentation such as finding the endpoint of an event and then segmenting backwards from the end point [18]. However, segmenting backwards from the endpoint of an MMG signal would capture mostly the steady state behaviour of the MMG signal. Therefore, in order to differentiate between the different gestures, it was necessary to segment the gesture forward in time from the initial point to capture the transient state of the MMG signal.

The period a of the transient state of the MMG signal from a gesture instance was found to be 600 ms, therefore a window length of 600 ms was used for the segmentation [31]. Additionally, it was found out by Samuel Wilson that the energy of the signal does not rise above the threshold until 0.05 seconds after the beginning of the gesture, defined as time b [4]. The signal that was recorded as the gesture for the N MMGs S_N was defined as:

$$S_N = [m^*_{N\ i-b}, ..., m^*_{N\ i+(a-b)}] \tag{5.3}$$

Therefore, each segment consisted of 50 points before the segmentation point followed by 550 points after the segmentation point. The collated segments can be seen in Fig. 5.12. To ensure that each gesture instance segment of the same gesture would be similar, it was important to ensure that the segmentation points appeared at a consistent location in the MMG signals of the different gesture instances. This was done by visually inspecting the location of the segmentation points as shown in Fig. 5.11.

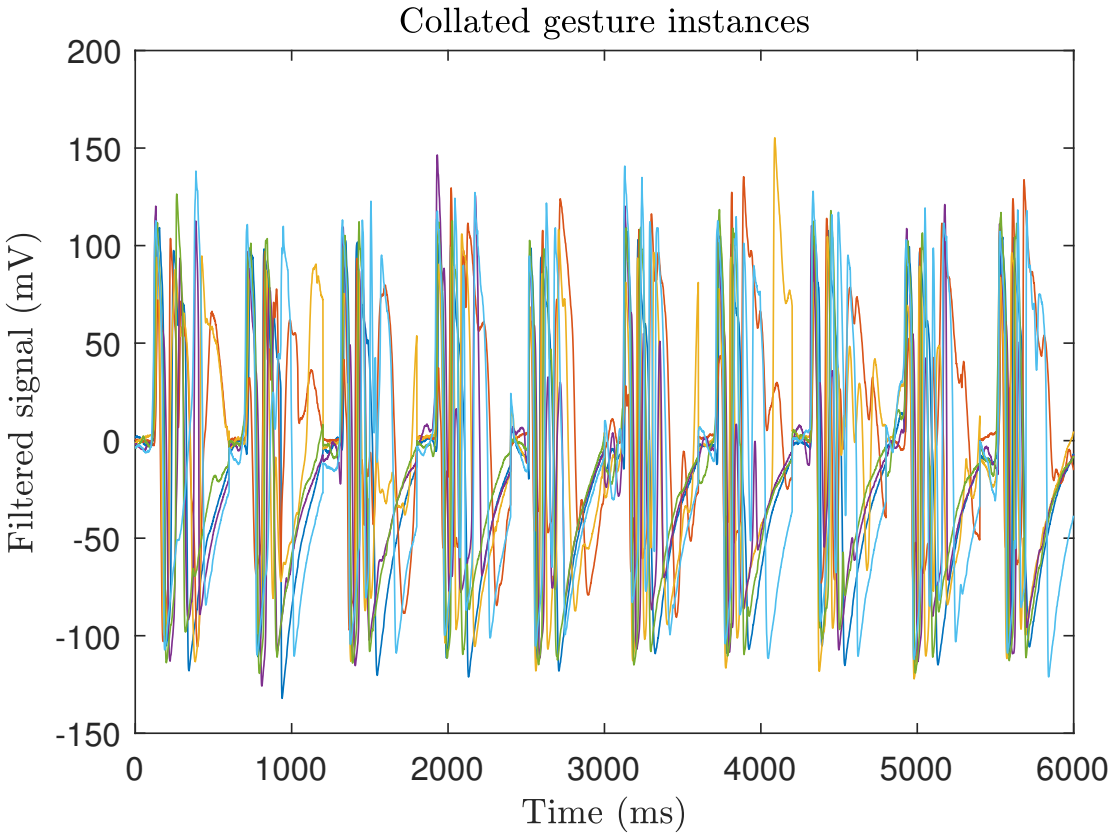


Figure 5.12: Segmented gesture instances which are collated into one matrix $\mathrm{m}^{collated}$

5.2 Feature optimisation

Obtaining an optimised set of features can be divided into two tasks:

- Feature selection
- Feature extraction

5.2.1 Feature optimisation: Feature selection

Transforming the input MMG signal into a representative feature set extracts the useful information that is present in the MMG signal and removes the unwanted aspects and interference [41]. Some features are robust across different kinds of noises and an appropriate feature set must be chosen to maximise class separability. This is important for obtaining effective discrimination between different gestures classes and for high classification accuracy [42]. There are 2 major ways to increase the information derived from MMG recognition systems [3]:

- Obtaining information from a greater number of different muscle positions
- Optimising the feature set such as by increasing the number of features or selecting more effective features for the data

Features are obtained from each gesture instance segment to create the total feature set used to represent the MMG data set. Therefore, the total number of features was determined by the number of features obtained from a gesture instance segment as well as the total number of segments present. Due to the non-stationary behaviour of the MMG signal, it was extremely difficult to extract a single feature which reflects the unique behaviour of an MMG signal perfectly, therefore it was desirable to use multiple features for the MMG pattern classification. However, the inclusion of features which do not contribute significantly to class separability may lead to a decrease in the classification performance as the higher dimensionality of the feature set may decrease the generalisability of the classifier [3]. Several factors were used to determine the optimum feature set chosen but the most important factors were those of computational complexity and class discrimination [21]. The acceptable computational complexity was limited by the response time of the system, which was especially important for control in real-time. Features in the analysis of MMG signals can be divided into 3 main groups:

- Time domain features
- Frequency domain features
- Time-scale domain features

A disadvantage of time domain features is that the input data is assumed to be a stationary signal, but MMG signals are non-stationary as their statistical properties change with time in the transient portion of the signal. Regardless of that issue, time domain features have been used widely in engineering research due to the features in the time domain being quick and easy to obtain as the data does not need to transform beforehand to obtain the features [43]. Therefore, to minimise the computational lag associated with obtaining the features, time domain features were used in this project. 10 time domain features constituting 13 values were obtained from each MMG channel (as four different autoregressive coefficients were obtained) in the MMG gesture instance segment. Therefore, each gesture instance segment would contain 78 (13 x 6) values to represent the features obtained. The features used to represent the MMG signals can be seen in Table. 5.1:

Table 5.1: Mathematical definitions of features selected for the MMG pattern recognition. Let x_n represent the MMG data in a segment. a_i is the auto-regressive coefficient. w_n is the white noise error. N denotes the length of the signal [3]

Feature	Definition
Waveform Length (WL)	$\mathrm{WL} = \sum_{n=1}^{N-1} \lvert x_{n+1} - x_n \rvert$
Root Mean Squared (RMS)	$\mathrm{RMS} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x_n^2}$
Integrated Absolute Value (IAV)	$\mathrm{IAV} = \frac{1}{N}\sum_{n=1}^{N} \lvert x_n \rvert$
Mean Absolute Value (MAV)	$\mathrm{MAV} = \frac{1}{N}\sum_{n=1}^{N} \lvert x_n \rvert$
Variance (VAR)	$\mathrm{VAR} = \frac{1}{N-1}\sum_{n=1}^{N} x_n^2$
Interquartile Range (IQR)	$\mathrm{IQR} = Q_3 - Q_1$
Skewness (SKW)	$\mathrm{SKW} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \overline{x})^3 / \left(\frac{1}{N}\sum_{n=1}^{N}(x_n - \overline{x})^2\right)^{\frac{3}{2}}$
Wilison Amplitude (WAMP)	$\mathrm{WAMP} = \sum_{n=1}^{N-1} f(\lvert x_n - x_{n+1} \rvert) \begin{cases} 1, & if\ x \geq threshold \\ 0, & otherwise \end{cases}$
Slope Sign Change (SSC)	$\mathrm{SSC} = \sum_{n=2}^{N-1} [f[(x_n - x_{n-1}) \times (x_n - x_{n+1})]] \begin{cases} 1, & if\ x \geq threshold \\ 0, & otherwise \end{cases}$
Auto Regressive Coefficients (a_i)	$\hat{x}_n = -\sum_{i=1}^{p} a_i x_{n-i} + w_n$

5.2.2 Feature optimisation: Feature extraction

Following the initial selection of the MMG feature set, it was then necessary to employ a feature extraction (also known as feature reduction) technique to reduce the dimensionality of the feature set obtained per gesture instance. Ideally, feature extraction proceeds in a manner that reduces the redundancies in the feature set. In addition to improving the signal and reducing the noise, feature extraction can lead to greater generalisation in the classification and hence a higher classification accuracy [3].

The feature extraction methods that have been implemented in this study were Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). PCA is an unsupervised learning feature extraction method since it only involves a set of features $X = [X_1, X_2, ..., X_p]$ and no associated class label [5]. Unsupervised learning approaches are used to find patterns or intrinsic structure within a dataset. PCA refers to the process by which a low dimensional feature space, $Y = [Y_1, Y_2, ..., Y_p]$, is produced from a linear combination of the p - features of the original feature set. The linear combination of features for the first principle component and the p^{th} principal component can be seen in Eqn. 5.4 and Eqn. 5.5 respectively [44]:

$$Y_1 = a_{11}X_1 + a_{21}X_2 + ... + a_{n1}X_n = a_1^T X \tag{5.4}$$

$$Y_m = a_{1m}X_1 + a_{2m}X_2 + ... + a_{nm}X_n = a_m^T X \tag{5.5}$$

The purpose of the low dimensional feature space in PCA is to extract the variation in the original feature set. The first principle component (Y_1) extracts the maximum variance from the original set of features and the loading vector (a_1) defines a direction in the feature space along which the data varies the most. The second principal component (Y_2) is the linear combination of X_1,...,X_p that has the maximum variance out of all linear combinations that are uncorrelated with Y_1. This constraint leads to the different principal components being orthogonal to one another. An example of the principal components of a data set can be seen in Fig. 5.13.

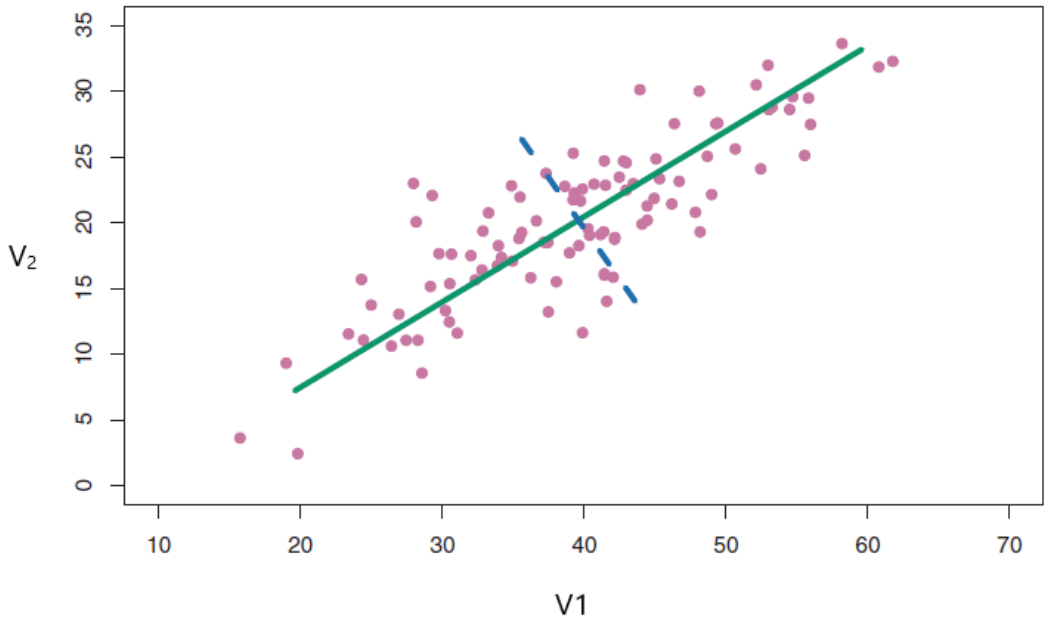


Figure 5.13: Graph showing the relationship between two variables, V_1 and V_2. The green solid line indicates the first principal component and the blue dashed line indicates the second principal component [5].

In this research, the number of principal components chosen in the dimensionality reduction was 20 features, as using 20 features was shown to capture 90% of the total variance in the data. This reduced the dimensionality of the feature set from 78 values to 20 values. An alternative feature extraction method used was LDA. LDA is a well known technique used in pattern recognition which can be used for both feature extraction as well as classification. It has been widely used in many applications such as facial recognition [45] as well as gesture classification of EMG signals [25]. LDA performs feature extraction by taking a high dimensional feature set from k different classes of data and then finds the optimum linear combination of features that maps the raw features into a lower dimensional space of k-1 features [5]. LDA differs from PCA as LDA is a supervised method for feature extraction. Supervised learning is a branch of machine learning, where the desired outputs are fed with the inputs of the machine learning algorithms, so that the underlying function that maps the inputs to the outputs can be approximated. Whereas PCA extracts the feature space which maximise the variance of the original set of features, LDA extracts a feature space which maximises the discrimination between the different classes. This is achieved as LDA chooses the linear combination of features which minimise the intra-class variation whilst simultaneously maximising the inter-class variation [3].

5.3 Classification methods

Three different classifiers were tested during this work:

- Template matching
- Linear Discriminant Analysis (LDA)
- Support Vector Machine (SVM)

The template matching classifier was used after the data had been segmented whilst the LDA and SVM classifiers were used after the feature extraction step as shown in Fig. 5.2.

5.3.1 Template matching

One initial approach taken to the classification of the data was using a template matching classification approach where templates were constructed during the training stage. Template matching techniques measure the similarity between a template and a sample [46]. The class of the template which had the greatest similarity or lowest difference was the class in which the sample was classified into [5, 46].

A careful examination of the collated segmented signals in Fig. 5.12 shows that instances of the same gesture exhibit variation from trial to trial. These variations include differences in the amplitudes and in the waveform due to inconsistencies in the intensity and speed of the hand gesture. Due to these variations in the MMG data, an averaging approach was used in the creation of the gesture templates. Averaging is useful as it can reduce the effects of random noise, which leads to an improvement in the signal-to-noise ratio of the template [47]. However, signals should be aligned in time, otherwise the averaging process can lead to a blurring or loss of peaks and valleys in the signal [47]. Therefore, MMG data from gesture instances was initially aligned such that latency discrepancies between the two MMG signals were minimised before averaging signals during template generation. This was achieved using pairwise cross-correlation averaging. As described in research conducted by Ravi Vaidyanathan et al [6], pairs of signals were successively averaged after finding the maximum cross-correlation of the pair via dot product of the signals. This process was then repeated with the mean of the pair of signals to form the template, $\overline{h}_L(k)$, as show in Fig. 5.14. In Fig. 5.14, $\mathrm{h}_{m,i}(\mathrm{k})$ represented a signal; k =1,2, ...,N was the different time instants of a signal; i = 1,2,..., L, was the i^{th} gesture instance of class m, m = 1,2,....M , where M was the number of gesture classes. L was the number of signals used to build the template when using pairwise cross-correlation averaging and $\overline{h}_{m\rightarrow n}(k)$ was the average of signals n to m after the cross-correlation.

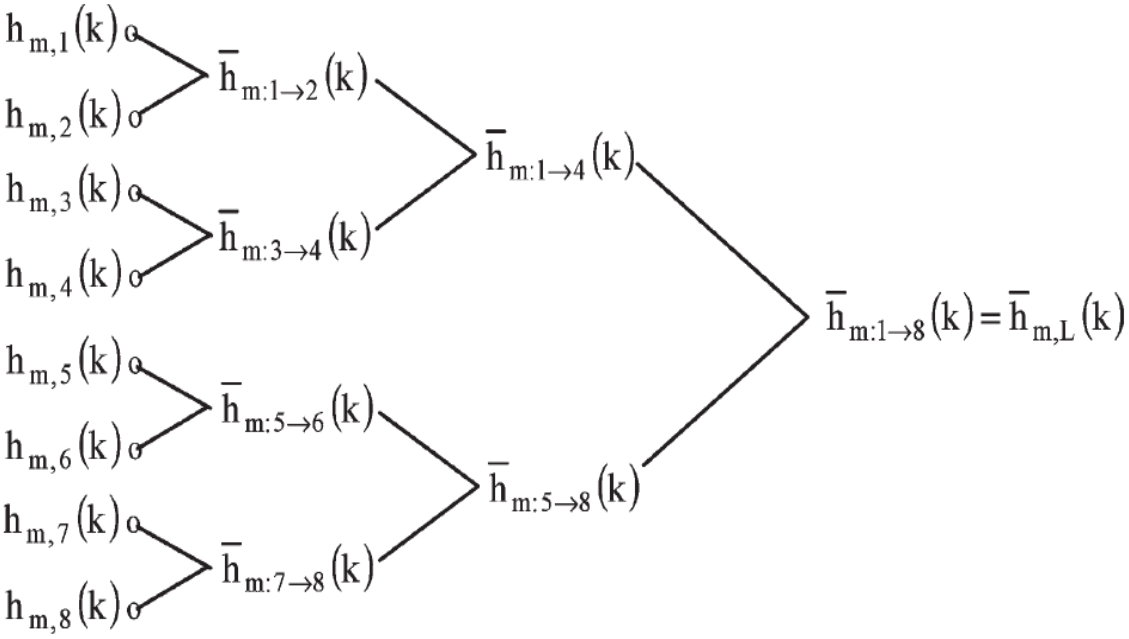


Figure 5.14: Pairwise cross-correlation averaging for L = 8 [6]

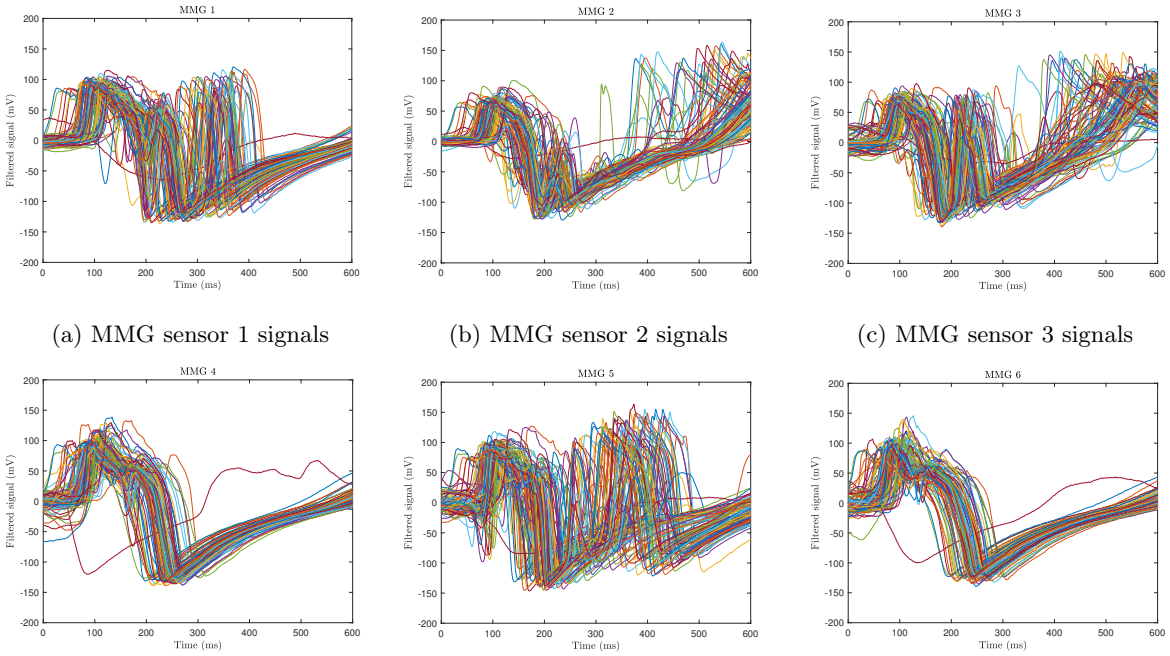
However, finding the optimum alignment via this approach was computationally intensive and would not be feasible for the real-time classification purposes. Also, alignment of signals was shown not to be a significant issue after a visual inspection of the segmentation points was done to ensure that the segmentation point was located at the same point within the different gesture instances. Therefore, the template generation method was changed from a pairwise cross-correlation averaging to a simple direct average of all the gesture instances, similar to work done previously by Samuel Wilson [4].

The overall template, $\overline{S}_{overall}$, was made from templates from each MMG channel. For g gesture instances, the template for the $N_t h$ MMG channel, $\overline{S}_N$, can be seen in Eqn. 5.6 [4]:

S

$$\overline{S}_N = \frac{1}{g}\sum_{j=1}^{g} s_{Nj} \tag{5.6}$$

Where s_{Nj} was the j^{th} signal used in template production from the N^{th} MMG. The averaging was optimised by visually analyzing the degree of overlap of the overlapping gesture instances which would form the template, as shown in Fig. 5.15. From Fig. 5.15, the significant overlap between the different instances of the same gesture also reaffirms that the segmentation was successful and no additional alignment approach was required.



(a) MMG sensor 1 signals (b) MMG sensor 2 signals (c) MMG sensor 3 signals

(d) MMG sensor 4 signals (e) MMG sensor 5 signals (f) MMG sensor 6 signals

Figure 5.15: 20 instances of the same gesture with the separate MMG signals overlapping

Following the creation of the template, gesture recognition was performed using an absolute difference method to measure the difference between a template and a sample to be classified. Absolute difference measures the discrepancy between the data points of a test gesture instance and all the gesture templates. For the test gesture signal, s(k), and the ith template $\overline{S}_{overall,i}(k)$, the absolute difference D_i over K points was defined as:

$$D_i = \sum_{k=1}^{K} \left| s(k) - \overline{S}_{overall,i}(k) \right| \tag{5.7}$$

The gesture would then be classified into the class where the difference, D_i, was at a minimum.

5.3.2 Linear Discriminant Analysis classification

After the dimensionality reduction of the original feature set, LDA itself can be used as a classifier. LDA uses the lower dimensional feature space as a training set to estimate the parameters of the discriminant function, such as the mean and the scatter of the features in the feature space. The discriminant functions are then used to determine the decision boundary between the various classes [5].

The classification of an observation into one of k classes was achieved by approximating the terms present in Bayes rule which is shown in Eqn. 5.8 [2]:

$$\hat{P}(Y = k|X = x) = \frac{\hat{P}(X = x|Y = k)\hat{P}(Y = k)}{\hat{P}(X = x)} == \frac{\hat{P}(X = x|Y = k)\hat{P}(Y = k)}{\sum_j \hat{P}(X = x|Y = j)\hat{P}(Y = j)} \tag{5.8}$$

$\hat{P}(X = x|Y = k)$ denotes the probability distribution function that the feature X = x came from the k^{th} class, $\hat{P}(Y = k)$ was the overall probability that a randomly chosen observation came from the k^{th} class and $\hat{P}(X = x)$ was the overall probability that an observation X = x. $\hat{P}(Y = k|X = x)$ was the posterior probability that an observation X = x belonged to the k^{th} class. $\hat{P}(Y = k|X = x)$ was obtained by estimating both $\hat{P}(X|Y)$ as well as the probability of each class, $\hat{P}(Y)$. In LDA, it was assumed that $\hat{P}(X = x|Y = k) = \hat{f}_k(x)$, where $\hat{f}_k(x)$ was a multivariate normal distribution with a general form shown by Eqn. 5.9 [5]:

$$f_k(x) = \frac{1}{(2\pi)^{\frac{p}{2}} |\sum|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)^T \sum^{-1}(x-\mu_k)} \tag{5.9}$$

For the case of LDA, the feature vectors of different classes could have different mean values from one another, however the covariance matrix of the different classes were identical and the covariance matrix was calculated using a pooled estimate [2]. Furthermore $\hat{P}(Y = k) = \hat{\pi}_k$ and this was estimated by the fraction of training samples that come from the k^{th} class. By Bayes rule, the probability of category k, given the input feature x was:

$$P(Y = k|X = x) = \frac{f_k(x))\pi_k}{P(X = x)} \tag{5.10}$$

The denominator term does not depend on the response k, so it was be considered as a constant. By inputting Eqn. 5.9 and taking the logarithm of both sides, the discriminant function $\delta(x)$ was:

$$\delta_k(x) = log\pi_k - \frac{1}{2}\mu_k^T \sum^{-1} \mu_k + x^T \sum^{-1} \mu_k \tag{5.11}$$

The input feature, x, was classified as being in the class which would give the highest value for $\delta_k(x)$ [25]. The different classes of data were separated by decision boundaries based on the discriminant functions as shown in Fig. 5.11.

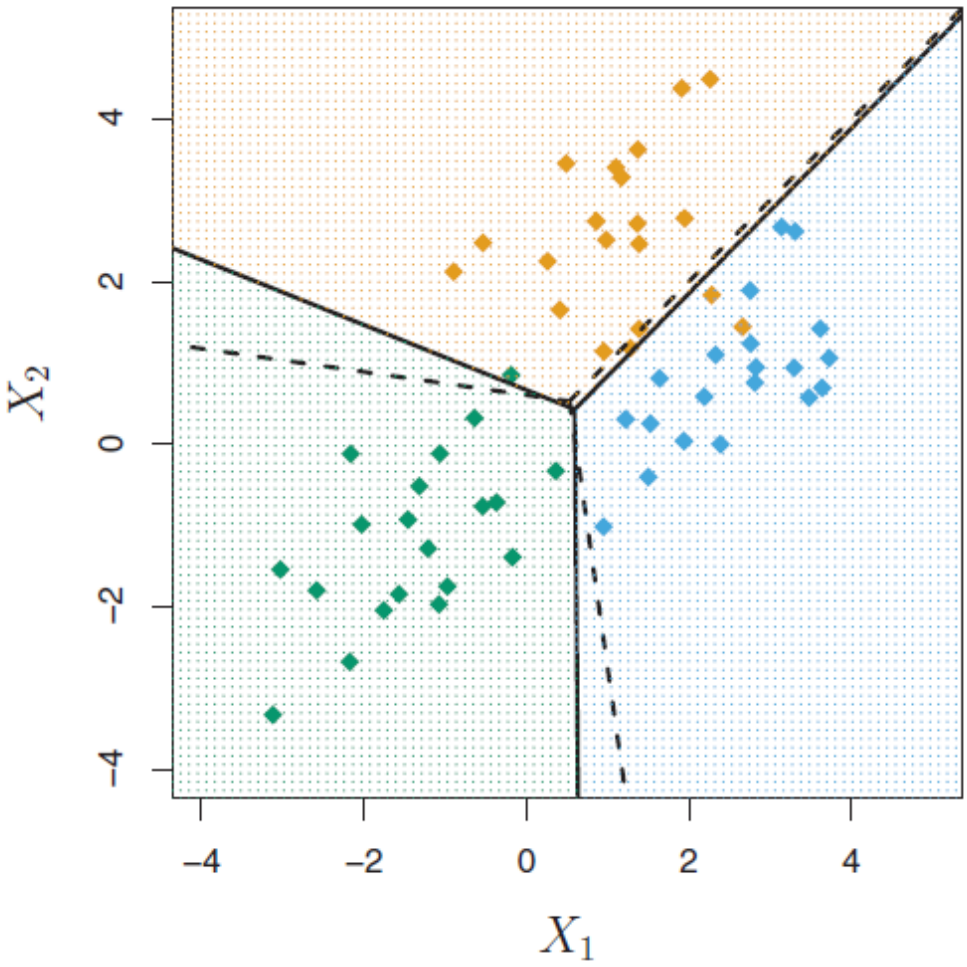


Figure 5.16: LDA classification of three classes. The observations from each class have two different features. The dashed lines are the Bayes decision boundaries and the LDA decision boundaries are indicated using the solid black lines [5].

5.3.3 Support Vector Machine Classifier

Support Vector Machines (SVM) are traditional two-class classifiers made by Vapnik based on the ideas of maximising margins and mapping data into a higher dimensional subspace. Classification occurs by taking an n-dimensional feature space as an input and then making a n-1 dimensional hyperplane to separate the different classes [2]. The data points of the different classes which are nearest to the hyperplane are known as Support Vectors. There are two main types of classifiers in SVM:

- Maximal margin
- Soft margin

The aim of maximal margin classifier is to orient the hyperplane in such a way that the margin between the hyperplane and the Support Vectors is maximised whilst ensuring that there are only observations from a single class on either side of the hyperplane. The general form of a linear hyperplane can be defined by Eqn. 5.12 [48]:

$$f(x) = \omega.x + b \tag{5.12}$$

Where $f(x)$ is the decision function used to represent the hyperplane, x is an input of dimension n, ω is a n-dimensional vector which defines the hyperplane and b is a bias term which determines the position of the hyperplane. The optimal hyperplane can be obtained as a solution to the optimisation problem of minimising $\frac{1}{2}(\omega.\omega)$ subject to the constraint [48]:

$$y_i((\omega.x_i + b)) - 1 \geq 0,\ i = 1, ..., l \tag{5.13}$$

Where l is the number of training examples and x_i are the Support Vectors obtained from training. The solution of the constrained quadratic programming optimisation problem can be obtained as [7]:

$$\omega = \sum \nu_i x_i \tag{5.14}$$

Which leads to the decision function being equal to [7]:

$$f(x) = \sum_{i=1}^{l} (x.x_i + b) \tag{5.15}$$

An example of a maximal margin hyperplane can be seen in Fig. 5.17.

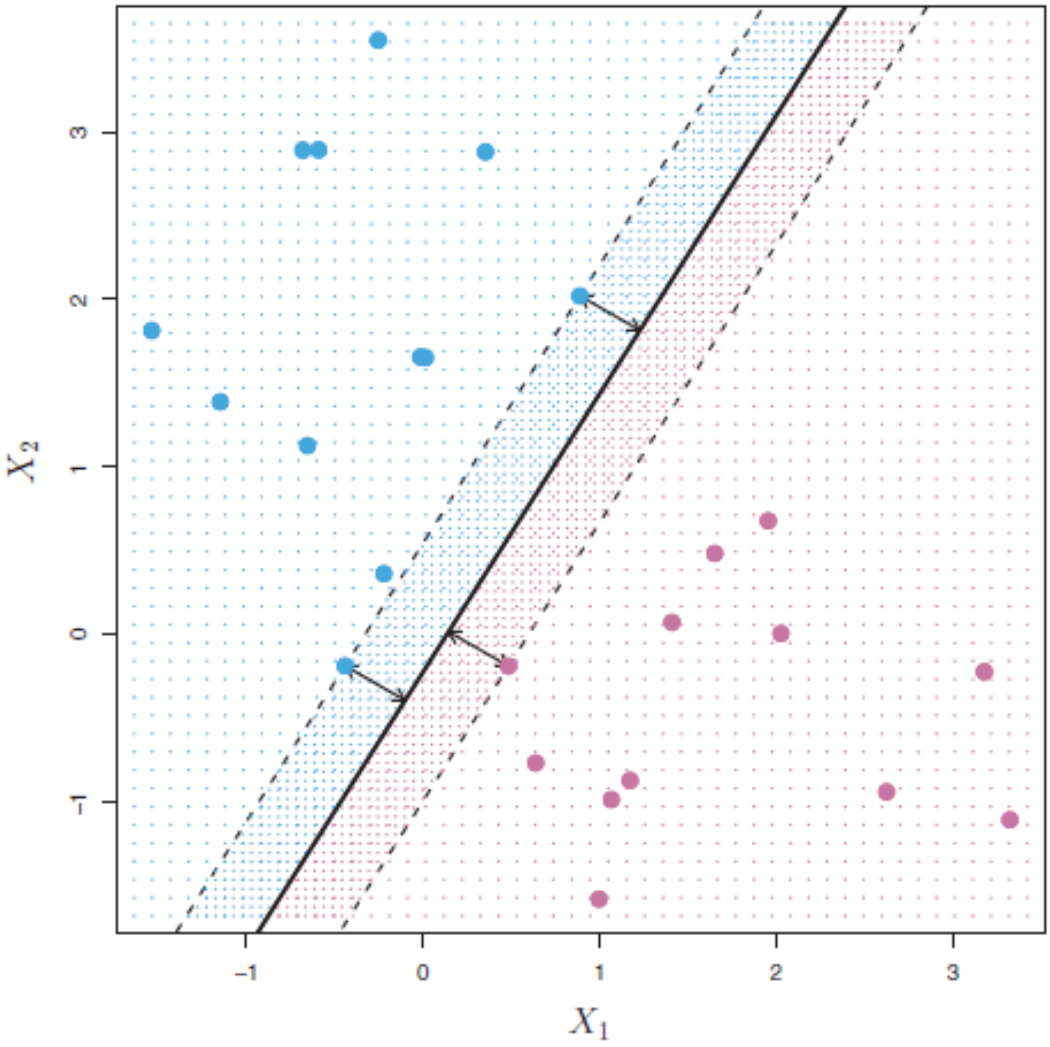


Figure 5.17: There are two classes of observations, shown in blue and in purple. The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the Support Vectors, and the distance from those points to the hyperplane is indicated by arrows [5].

An issue with the position of a hyperplane of maximal margin classifiers is that it can be highly susceptible to outliers, therefore a soft margin classifier be used instead. Soft margin classifiers also aims to maximise the margins between the nearest training examples of the different classes, however, it can also allow observations from different classes to be on the same side of the hyperplane. This is taken into account by using slack variables, ϵ_i, and an error penalty, C. A soft margin classifier is generally more effective than a maximal margin classifier as it has greater robustness to individual observations and better classifcation of most of the training observations. The soft margin hyperplane derived will differ based on the magnitude of ϵ_i and C, as shown by Fig. 5.18

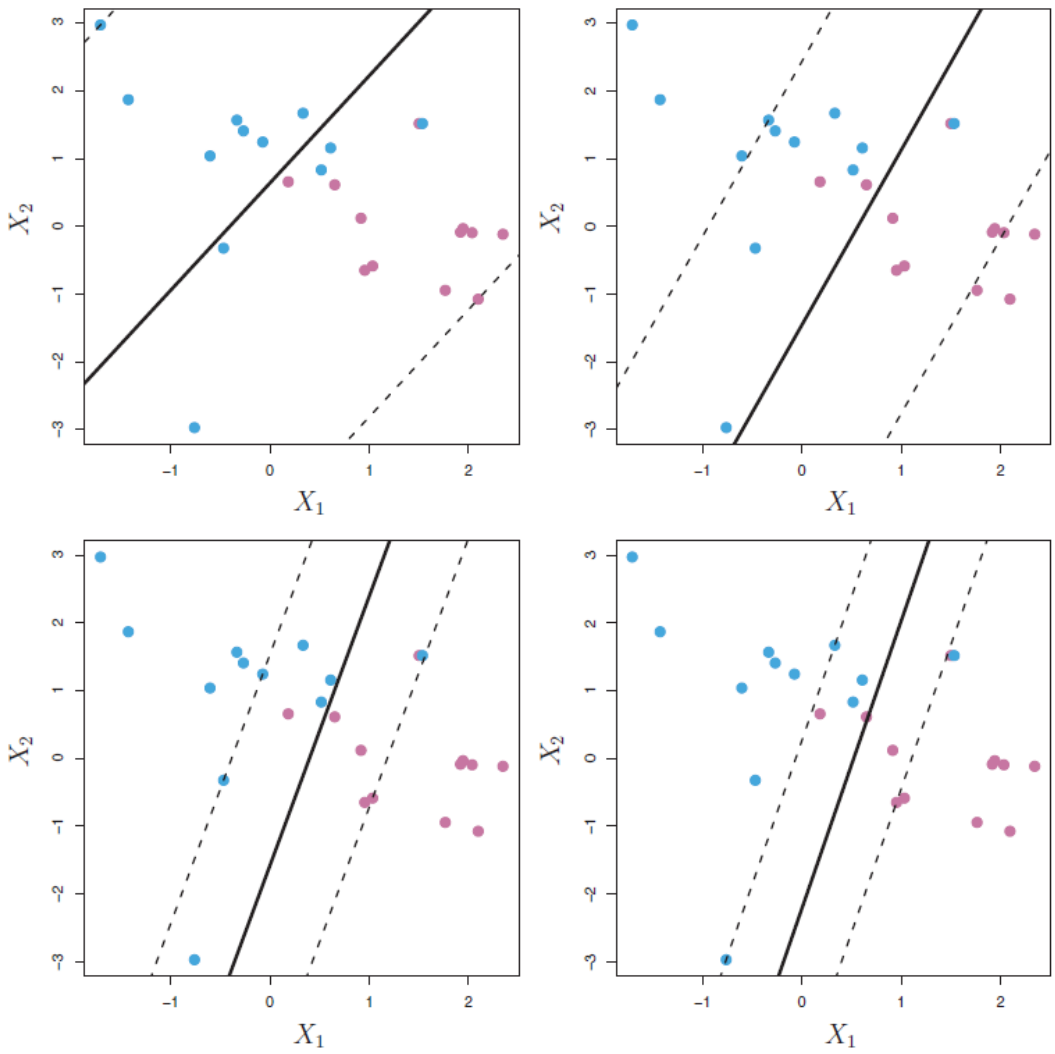


Figure 5.18: A soft margin classifier was fit using 4 different values of the tuning parameter, C. The largest value of C was used in the top left panel, and smaller values were used in the top right, bottom left, and bottom right panels. When C is large, there is a high tolerance for observations being on the wrong side of the margin, and so the margin will be large. As C decreases, the tolerance for observations being on the wrong side of the margin decreases, and the margin narrows [5].

The optimal soft margin classifier can be obtained as a solution to the optimisation problem, where $\phi(w) = \frac{1}{2}(\omega.\omega) + C\sum_{i=1}^{l}\xi_i$ is minimised subject to [2]:

$$y_i(\omega.x_i + b) - 1 + \xi_i \geq 0 \tag{5.16}$$

For cases where even a soft margin linear boundary in the input space is insufficient to separate the classes effectively, due to a non-linear distribution of data, it is possible to create a linear hyperplane that allows for separation in a higher dimensional space. This is achieved through the use of a transformation $\phi(x)$ that converts the data from a N-dimensional feature space to a higher dimension, Q-dimensional feature space [5, 25]. This causes a non-linear boundary in the N-dimensional input space to transform into a linear boundary in the Q-dimensional feature space as shown by Fig. 5.19.

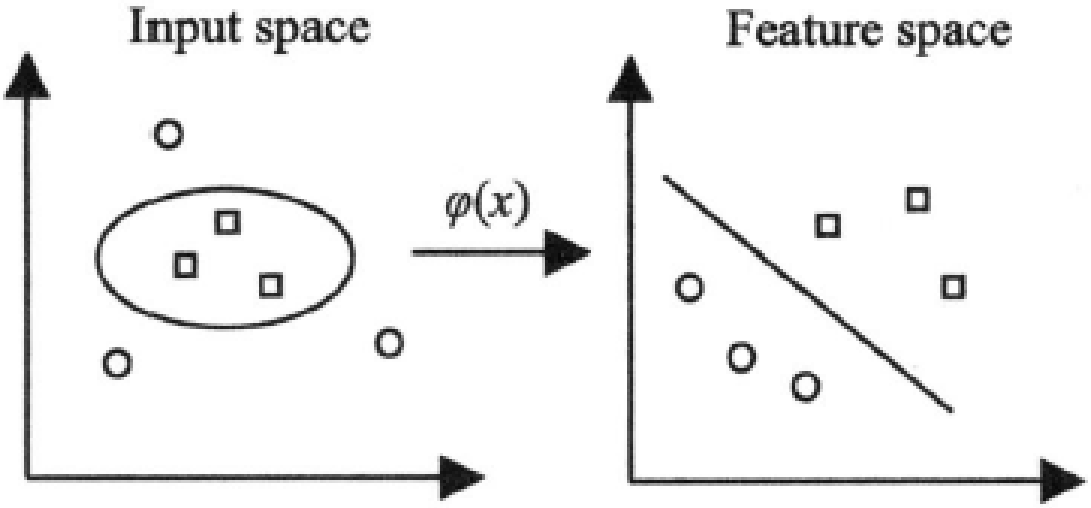


Figure 5.19: Conversion of non-linear low dimensional feature space to a linear high dimensional feature space [7]

Substituting the transformation $\phi(x)$ into Eqn. 5.12 provides a new decision function which can be used to represent a non-linear decision boundary [7]. The new decision function can be seen in Eqn. 5.17.

$$f(x) = \sum_{i=1}^{l} \nu_i(\phi(x).\phi(x_i)) + b) \tag{5.17}$$

This transformation to a higher dimensional space can be provided by a kernel function. The choice of kernel function is significant as the correct choice of kernel function can improve the classification accuracy of the SVM. Kernel functions are similarity functions that quantify the similarity between a test and training observation by calculating the inner product between the vectors after transforming them to a higher transformational space [5, 7]. The kernel $K(x_i.x_j)$ is defined by Eqn.5.18 [49]:

$$K(x_i.x_j) = \phi(x_i).\phi(x_j) \tag{5.18}$$

In this project, 2 main types of kernels were used by my colleague, Helena Santos Sousa, during her testing using a SVM classifier. The kernels used were [49]:

- The polynomial kernel function: $K(x.x_i) = [(x.x_i) + 1]^d$
- The radial basis kernel function: $K(x.x_i) = exp\left(\frac{-|x-y|^2}{2\sigma^2}\right)$

6 Classification tests

For all the classification tests conducted offline, $T_{MMGUPPER}$ was chosen as 20,000 $(mV)^2$ and $T_{MMGLOWER}$ was chosen as 10,000 $(mV)^2$. For the real-time classification, a single threshold of 15,000 $(mV)^2$ was chosen. Furthermore the segmentation window length was 600 points (600 ms of data) for both the offline and real-time classification. Both feature extraction and classification was completed in MATLAB R2017a for both the offline and real-time classification. The only significant difference in the real-time analysis was that a rolling buffer was used to capture the data in real-time. The template matching classification was achieved by using the 'Offline template generation', 'Absolute difference measure' and the 'Offline template classification' MATLAB scripts seen in Appendix .E - G. The offline LDA classification was achieved using the 'Offline LDA classification' MATLAB Script seen in Appendix. H. The real-time LDA classification was achieved using the 'Real-time extracting training features' and 'Real-time LDA classification' MATLAB functions shown in Appendix. I and J. The LDA classification algorithms were based on Professor Adrian Chan's pattern recognition library available at http://www.sce.carleton.ca/faculty/chan [24].

6.1 Template matching and statistical classification comparison

For the comparison between the template matching and the statistical classification, the author performed 50 instances of the gestures shown in Fig. 4.2. The segmented gestures instances were split into a training set consisting of 40 % of the gestures and a test set consisting of the remaining 60%. The template matching classification approach involved using the absolute difference between a sample and a template to perform a classification. As the accuracy of the template matching was not above 50 % when more than 4 gestures were used, only 4 gestures were used in the offline analysis when using the gesture classification. These gestures were:

1. Open
2. Close
3. Thumbs up
4. Point

These gestures were chosen as they provide a mixture of large and small gestures which would increase the likelihood of obtaining distinct templates which would lead to high classification accuracy. Table. 6.1 summarises the results of the offline classification.

Table 6.1: Results of the offline classification of the four gestures using a template matching approach

		Gesture recognised			
		Open	Close	Thumbs Up	Point
Actual gesture	Open	80.0 %	0.0 %	10.0 %	10.0 %
	Close	3.3 %	86.7 %	3.3 %	6.7 %
	Thumbs Up	10.0 %	0.0 %	83.3 %	6.6 %
	Point	0.0 %	0.0 %	6.7 %	93.3 %
			Total accuracy		85.8 %

For the real-time classification, as the accuracy of the template matching was not above 50 % for more than three gestures, only three gestures were used in the real-time classification. The 3 gestures used in the real-time classification were:

1. Open
2. Close
3. Point

Table. 6.2 summarises the results of the real-time classification using the template matching approach.

Table 6.2: Results of the real-time classification of the three gestures using a template matching approach

		Gesture recognised		
		Open	Close	Point
Actual gesture	Open	66.7 %	0.0 %	33.3 %
	Close	0.0 %	86.7%	13.3 %
	Point	0.0 %	0.0 %	100.0 %
		Total accuracy		84.5 %

Similarly, results were also obtained for the offline and real-time classification of the seven gestures shown in Fig. 4.2. The offline and real-time classification results can be seen in Table. 6.3 and 6.4.

Table 6.3: Results of the offline classification of the seven gestures using LDA classifications

		Gesture recognised						
		Open	Close	First finger pinch	Second finger pinch	Thumbs up	Point	Finger roll
Actual gesture	Open	96.3 %	1.2 %	0.0 %	2.5 %	0.0 %	0.0 %	0.0 %
	Close	0.0 %	98.8 %	0.0 %	0.0 %	0.0	0.0 %	1.2 %
	First finger pinch	0.0 %	0.0 %	98.8 %	0.0 %	0.0 %	1.2 %	0.0 %
	Second finger pinch	0.0 %	0.0 %	18.8 %	71.2 %	10.0 %	0.0 %	0.0 %
	Thumbs Up	0.0 %	0.0 %	0.0 %	2.5 %	90.0 %	7.5 %	0.0 %
	Point	1.3 %	0.0 %	6.3 %	0.0 %	11.2 %	81.2 %	0.0 %
	Finger roll	2.5 %	0.0 %	6.6 %	0.0 %	0.0 %	0.0 %	97.5 %
						Total accuracy		90.5 %

From the results seen in Table. 6.1 - 6.4, it can be seen that real-time classification accuracy was generally slightly lower than offline classification accuracy. This was due to the offline classification using gestures at different levels of fatigue for training, whilst real-time classification would generally have training data from gestures at low level of fatigue whilst testing data would be obtained later in time where the level of fatigue would be higher. Furthermore LDA had a much higher classification accuracy in both the offline and real-time classification than the templalte matching classification. As a result, the following tests focused on using statistical classifiers such as LDA and SVM rather than a template matching classifier. To assess the suitability of the LDA and SVM classifiers for MMG hand gesture analysis, 3 different investigation were conducted offline. During these investigations, the results of the SVM classification was performed by the author's colleague, Helena Santos Sousa. Several different kernels were tested and it was seen that the linear and quadratic kernels yielded the results of highest accuracy.

Table 6.4: Results of the real-time classification of the seven gestures using LDA classifications

		Gesture recognised						
		Open	Close	First finger pinch	Second finger pinch	Thumbs up	Point	Finger roll
Actual gesture	Open	100.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %
	Close	0.0 %	93.3 %	0.0 %	6.7 %	0.0 %	0.0 %	0.0 %
	First finger pinch	0.0 %	6.6 %	66.7 %	26.7 %	0.0 %	0.0 %	0.0 %
	Second finger pinch	0.0 %	0.0 %	6.7 %	93.3 %	0.0 %	0.0 %	0.0 %
	Thumbs Up	10.0 %	0.0 %	0.0 %	0.0 %	86.7 %	3.3 %	0 %
	Point	0.0 %	0.0 %	23.3 %	16.7 %	0 %	60.0 %	0 %
	Finger roll	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	100.0 %
						Total accuracy		85.7 %

6.2 Preliminary testing

Before starting the 3 different investigations with the different classifiers, it was necessary to:

- Compare the effectiveness of the different feature extraction methods
- Find the optimum number of samples for offline training.

These two goals were achieved by comparing the classification accuracy as a function of the number of the training examples when using a LDA classifier with 3 different feature extractions. The different feature extractions were:

- LDA reduction to a 6 dimensional space
- PCA reduction to 6 dimensional space
- PCA reduction to 20 dimensional space which corresponds to 90% of the explained variance.

The classification results of Subject 1 when using the different feature extractions can be seen in Fig. 6.1 whilst the results for the other test subjects can be seen in Appendix K.

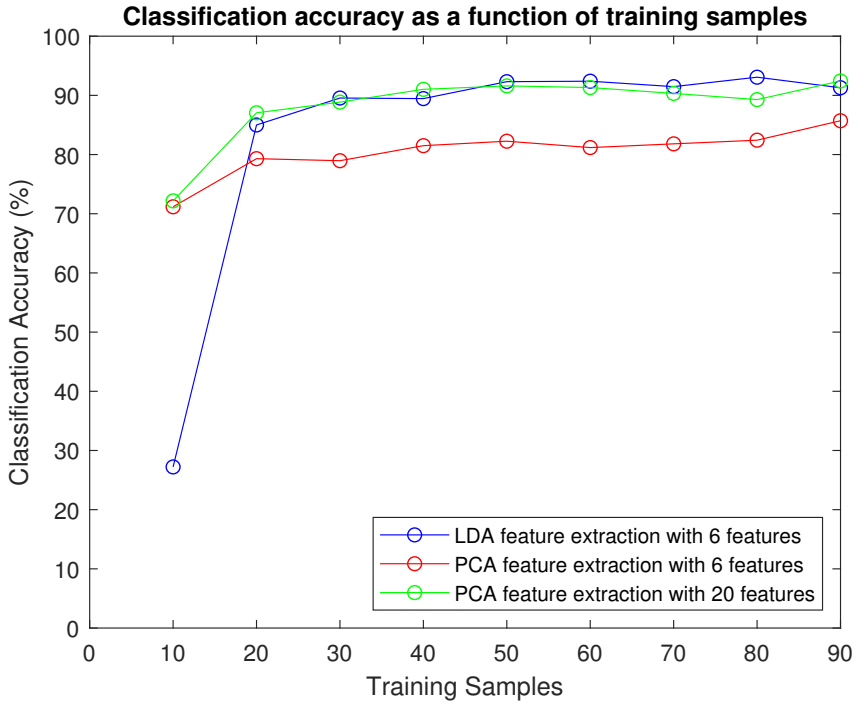


Figure 6.1: Classification accuracy as a function of training examples for 3 different feature extractions

From Fig. 6.1, the classification accuracy when using the PCA reduction was generally better than LDA reduction at lower training samples as even when the PCA and LDA both use a 6-dimensional subspace, the PCA has a much higher classification accuracy. However, LDA quickly improved and obtained a greater accuracy than even the PCA reduction to a 20-dimensional space. LDA achieving a higher classification accuracy was expected as LDA is a supervised dimensionality reduction whereas PCA is unsupervised. Furthermore, it can be seen that classification initially rapidly improved at a lower number of training samples and then generally plateaued at higher values such as 70 to 80 training samples. Therefore, 80 training samples was used for the first investigation and LDA dimensionality reduction was used for all the different tests.

6.3 User specific classifiers

The first investigation involved user specific classifiers being trained for each test subject. The dataset of each subject consisted of 100 samples for each gesture and it was divided into 80 % training and 20 % test sets as a training set of 80 samples was shown to be effective during preliminary testing. Due to the limited amount of data present from each individual, the user specific classifiers were tested using k-fold cross validation. K-fold cross validation enables all the data to be used for the training of the classifier as well as in testing, and it is a widely used technique to estimate the generalisation error of a classifier. K-fold cross validation consists of [5, 50]:

- Dividing the sample data into k equal sized subsets. Each subset is referred to as a fold. Let the folds be named as f_1,f_2,. . . ,f_k.
- For i = 1:k, perform the classification using the fold f_i as a test set and keep the remaining k-1 folds as a training set.

The overall classification accuracy was calculated by the average of the k predicted accuracies. The results of the user-specific classifiers for the different subjects using five-fold cross validation can be seen in Table. 6.5.

Table 6.5: User specific classification error for the different classifiers

Subject	Accuracy for LDA (%)	Accuracy for Linear SVM (%)	Accuracy for Quadratic SVM (%)
1	91.8	98.6	99.3
2	84.5	99.3	99.3
3	93.7	100.0	100.0
4	89.9	99.3	97.9
5	96.9	100.0	100.0
6	93.1	100.0	100.0
Average	91.7	99.5	99.4

From the results of the first test, it can be seen that the LDA, linear SVM and quadratic SVM classifier were able to achieve high accuracies of 84.5 to 96.9 % , 98.6 to 100.0 % and 97.9 to 100.0 % respectively. This suggests that both the LDA and SVM classifiers were easily able to find distinct patterns in the data when a single test subject provides 80 instances of data for each gesture as the training set.

6.4 Pooled dataset

The second investigation aimed to assess the applicability of using a pooled dataset for testing hand gesture classification on new users. This involved collating data from 5 of the test subjects as a training data set and then using the data from the sixth subject as a test set. This process was repeated until data from each subject was used as a testing set against classifiers which had not been trained on the testing subject's data. The results of the classification using the pooled dataset for training of the classifiers and testing on unseen subjects can be seen in Table. 6.6

Table 6.6: Classification error when training the different classifiers using a pooled dataset and testing on unseen subjects

Subject	Accuracy for LDA (%)	Accuracy for Linear SVM (%)	Accuracy for Quadratic SVM (%)
1	40.6	35.4	33.3
2	33.1	65.7	70.3
3	18.0	26.0	14.1
4	37.7	31.4	30.3
5	31.6	65.9	70.2
6	31.6	12.0	14.0
Average	32.1	39.4	38.7

From Table. 6.6, it can be seen that the accuracy of the classifiers were significantly less and the range was significantly higher than the user-specific classifiers as there was at least a 50% decrease in the average classification accuracy. This behavior differs from that seen in Needham et al where the accuracy of the user-specific testing was low whilst the pooled dataset gave high accuracies [50]. This difference could be due to the gait of different individuals being similar to one another whilst there being more variation in performing hand gestures. Another possible reason for the discrepancy in the findings could be that in the work produced by Ashwin et al, there was less user-specific data as there fewer instances per gait per individual meaning the user-specific classifiers were less effective. However, the number of test subjects was twice the number used in this work, which could mean that the SVM algorithm was able to get a better generalisation of the behaviour of the different muscle movements whilst in this case the data was more specialised for an individual.

6.5 Pooled dataset supplemented by user data

The third investigation aimed to quantify the benefit of introducing a small quantity of subject specific data to the pooled data set. This was done, by removing 20 % of the previously unseen participant's data from the test set and adding it to the pooled data set. The classifiers were then tested on the remaining data in the test set. The effect of this can be seen by comparing the classification accuracy of the pooled data set before and after the addition of the user-specific data. This can be seen for the different classifiers in Tables. 6.7 - 6.9.

Table 6.7: Classification error for the LDA classifier when training using a pooled dataset before and after being supplemented by user-specific data

Subject	Accuracy when trained on generic pooled data (%)	Accuracy when training on generic pooled data supplemented by user data (%)
1	40.6	51.1
2	33.1	38.4
3	18.0	33.6
4	37.7	43.7
5	31.6	44.7
6	31.6	42.3
Average	32.1	42.3

Table 6.8: Classification error for the linear SVM classifier when training using a pooled dataset before and after being supplemented by user-specific data

Subject	Accuracy when trained on generic pooled data (%)	Accuracy when training on generic pooled data supplemented by user data (%)
1	35.4	61.6
2	65.7	62.3
3	26.0	60.7
4	31.4	60.7
5	65.9	62.3
6	12.0	26.3
Average	39.4	55.7

Table 6.9: Classification error for the Quadratic SVM classifier when training using a pooled dataset before and after being supplemented by user-specific data

Subject	Accuracy when trained on generic pooled data (%)	Accuracy when training on generic pooled data supplemented by user data (%)
1	33.3	67.0
2	70.3	68.6
3	14.1	64.6
4	30.3	62.7
5	70.2	68.8
6	14.0	28.2
Average	38.7	60.0

From Table. 6.7 - 6.9, it can be seen that adding in user-specific data, the average accuracy increased from 32.1 to 43.5 %, 39.4 to 55.7 % and 38.7 to 60.0 % for the LDA, linear SVM and Quadratic SVM respectively. This shows that adding supplementary user-specific data to the training set leads to an increase in the average accuracy for all the classifiers. This increase in accuracy was likely due to the increase in the generalisation of the classifier from the increased number of different individuals used in the training set but it also could be due to the different instances of a user being similar to one another and so test instances were more likely to be recognised if the user data was present in the training set.

7 Serious game

7.1 Game design

The Serious game was designed to aid the rehabilitation of individuals with difficulty in the control of their upper-limbs. The Serious game involved an individual controlling the movement of virtual Kuka robot arm via the position of 2 IMUs which would control different joints of the robot. The Kuka robot can be seen in Fig. 7.1

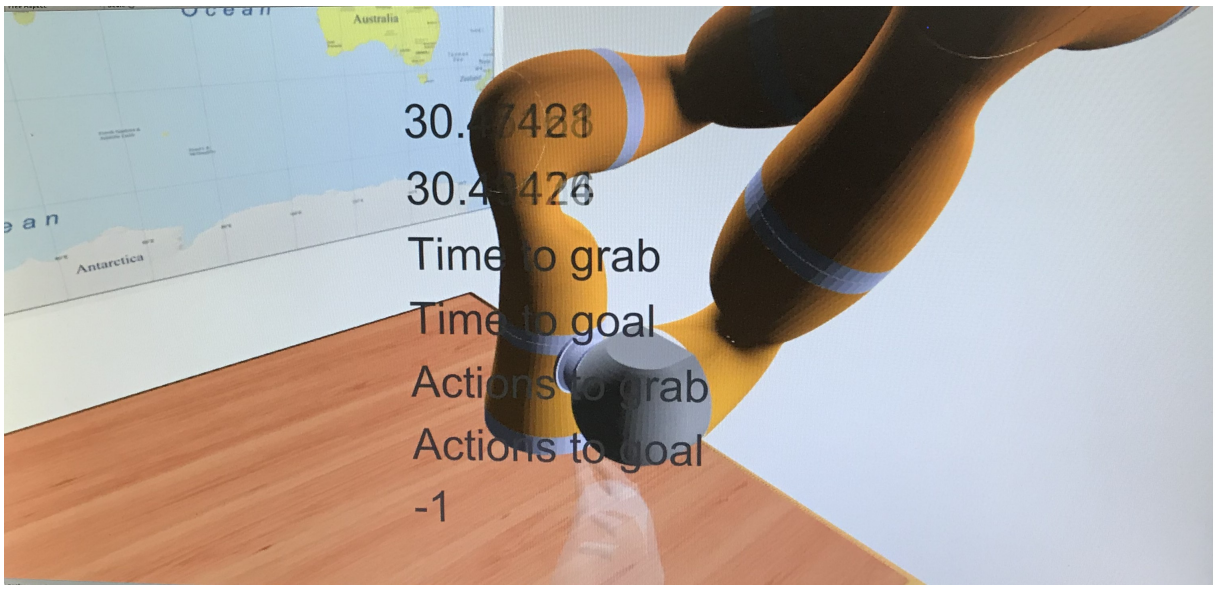
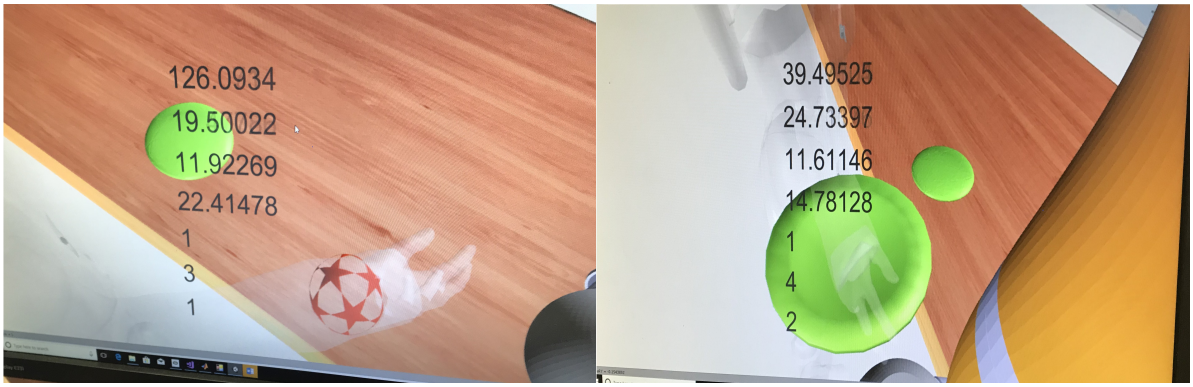


Figure 7.1: Kuka robot arm alongside transparent avatar used to aid the user in controlling the movement of the Kuka robot

The kinematics of the Kuka robot was coded previously by Samuel Wilson. When the Kuka robot would become sufficiently close to either of the 2 objects, a plate or a ball, the object would glow red and be picked up once the NU interface recognised the required gesture performed by the user. This can be seen in Fig. 7.2a. Following the object being picked up by the Kuka robot, once the picked up object was sufficiently close enough to the goal which was present in the middle of the table, the object would glow green and it could be placed in the goal after the recognition of an 'Open' gesture by the user. This can be seen in Fig. 7.2b. This process of picking up an object and placing it in the goal would be repeated for ten objects before the game would be completed. The NU interface was trained using 30 gesture instances for each of the 3 gestures and the data was classified using the LDA classifier whilst using a LDA feature extraction. The hardware required for this game consisted of two IMUs, 6 MMG sensors, an Oculus rift headset and 2 rift controllers. The hardware can be seen in Fig. 7.3



(a) Ball close to Kuka arm

(b) Plate close to goal

Figure 7.2: Change in state of the objects to represent proximity to object and goal

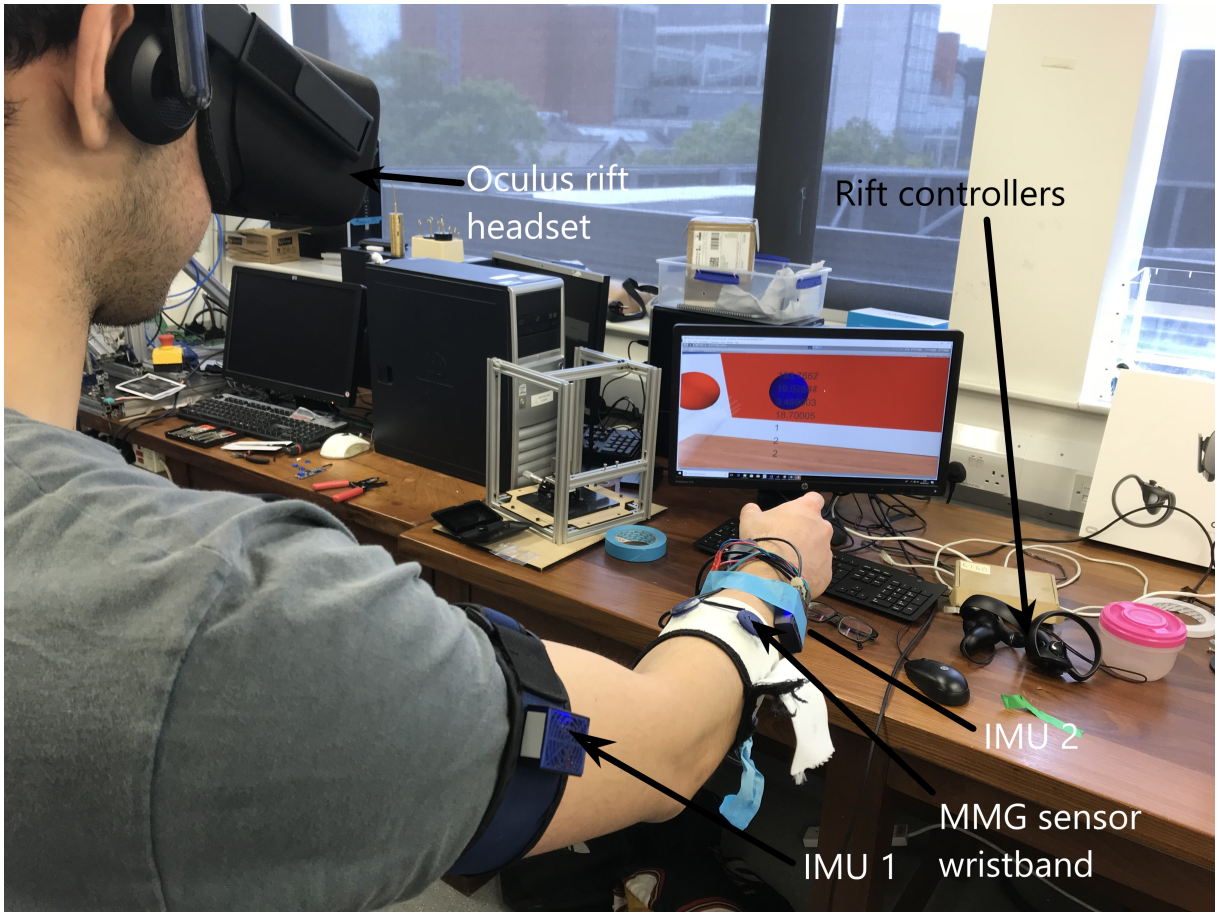


Figure 7.3: Author using the NU interface along with the Oculus hardware

7.2 Results of the Serious game

The performance of the player of the Serious game was analysed by looking at four criteria:

- Time required to grab an object
- Time required to place in goal (includes grabbing object and placing in goal)
- Number of actions used to grab an object
- Number of actions used to put object in goal (includes actions used to grab object and place in goal)

The Serious game was completed using the motion and muscle sensing NU interface as well as the Oculus Rift Controllers. The game was completed 5 times when using the NU interface and the Rift controllers by the Author.

The results of the first trial of the Serious game using the Rift Controllers can be seen in Table. 7.2 and the results for the subsequent trials can be seen in Appendix L. The results of the Serious game when using the Rift Controllers show that the game was relatively simple for an able-bodied individual as the time take to place object in the goal was generally less than 10 seconds and the number of actions used to put object in goal was generally 2 whilst some mistakes can occur in the button pressed making this value increase to 3 or 4. The results of the first trial of the game using the NU interface can be seen in Table. 7.3, whilst the other results can be seen in Appendix M. The IMU motion control enabled the Kuka robot to reach the objects relatively quickly, therefore the time taken to grab an object or place an object in the goal mostly reflects the time taken for the NU interface to recognize the correct gesture required to perform the desired action.

The time taken to pick up an object varied from 1.17 to 79.4 seconds whilst the time to place an object in the goal varied from 3.40 to 82.69 seconds. This large variation in the results was most likely due to the inconsistency of the gestures instances during the training phase. This could be suggested as the results from the NU interface improved during the later trials when the author became more familiar with performing the gestures in a consistent manner. The general trend in the Table. 7.3 and Appendix M showed that the time taken and the number of actions performed generally increased for the objects later on in the trial. This could be due to the effect of fatigue lowering the gesture classification. Furthermore, the movement of the user's arm to different positions was also likely to affect the gesture recognition accuracy which could be the reason for poor gesture classification for objects early on in a trial.

Table 7.1: Comparison between the results of the game when using the NU interface and Oculus rift controllers

Trial	Average time to grab an object (s)	Average time to place object in goal (s)	Average number of actions to grab object	Average number of actions used to put object in goal
NU interface 1	25.15	30.74	5.4	6.7
NU interface 2	16.64	18.00	2.2	3.5
NU interface 3	12.64	24.21	1.5	3.1
NU interface 4	6.020	17.47	1.2	2.3
NU interface 5	4.29	7.78	1.4	2.2
Rift controllers 1	3.09	4.52	1.2	2.2
Rift controllers 2	2.75	3.64	1.2	2.2
Rift controllers 3	5.08	8.22	1.7	2.7
Rift controllers 4	3.19	4.78	1.2	2.3
Rift controllers 5	3.42	4.39	1.4	2.4
NU interface Average	12.94	19.64	2.3	3.6
Rift controllers Average	3.5	5.1	1.3	2.4

Table 7.2: Measurements from game when using the Rift controllers for the first trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	6.42	7.84	1	2
2	2.30	3.61	1	2
3	3.78	5.15	1	2
4	2.54	4.29	1	2
5	3.81	5.43	2	3
6	3.59	5.09	1	2
7	2.32	3.32	2	3
8	3.65	5.70	1	2
9	1.01	2.02	1	2
10	1.49	2.72	1	2

Table 7.3: Measurements from game when using the NU interface for the first trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	17.75	20.72	3	4
2	8.52	10.83	3	4
3	11.70	37.51	2	5
4	15.86	17.93	5	6
5	27.89	30.85	1	2
6	39.37	44.85	6	7
7	17.38	21.76	3	4
8	1.87	5.13	1	3
9	31.7	35.1	13	14
10	79.4	82.69	17	18

8 Conclusion and future work

Serious games have been shown to be important in the rehabilitation of certain specific individuals, such as those stroke victims and those with difficulty controlling their upper limbs, through an engaging and practical application. A Serious game has been designed for individuals such as stroke victims with difficulties controlling their upper-limbs. Completing the game with the NU interface was tested 5 times by the author. It was shown that initially the game was difficult to complete due to a low classification accuracy due to performing gestures in an inconsistent manner and the variation of the MMG signals because of the variation of the user's arm. After becoming more familiar with the Serious game and performing gestures in a consistent manner, the time required complete the task had decreased. Due to the variety of different classifiers available to train the game, it was first necessary to compare the effectiveness of template based and statistical classification to see which one achieves higher classification accuracy. The statistical classifier was more effective as it was able to achieve an average accuracy of 90.5 % and 85.7 % for the offline and real-time classification of seven gestures whilst template matching only achieved 85.8 % and 84.5 % for 4 and 3 gestures. After deciding on using a statistical classifier for the game, three investigations were conducted to see various ways to train the statistical classifier for the game. The first study examined using 80 instances of user specific data for the training of each gesture and this resulted in average accuracies above 90 % for the LDA, linear SVM and quadratic SVM classifiers. However, training with a large number of gesture instances can be often be time consuming, fatiguing and mundane, especially to someone with upper-limb difficulties. A potential solution for the need for a large sets of subject-specific training data was to use a generic pooled data set of 5 different training subjects for the testing of an unseen user. But this lead to an average classification accuracy below 50 % for the 7 gestures which shows that each individuals MMG signals are distinct from one another due to factors such as skin thickness, muscularity and way of performing the gesture and it is difficult to interpret a pattern in the data when using gesture instances from only five different individuals in the training set. The third investigation examined the effect of supplemented the generic pooled dataset with a small set of subject-specific data which increased the classification accuracy of the pooled dataset by at least 10 %, although the accuracy was still not able to reach that of the subject specific dataset. Throughout the 3 investigations it can be seen that the SVM classifiers attained higher classification accuracy than the LDA classifier which agrees with the literature and suggests MMG data is not normally distributed [50].

Following this study, a more comprehensive dataset should be gathered which contains data from larger number of subjects to see if this enables LDA and SVM classifiers to find a clear pattern in the different gestures which goes beyond a specific individual. Additionally, EMG and MMG could perhaps be used in conjunction with the same total number of sensors to see if there is a synergistic effect. Furthermore, additional classifiers such as Neural Networks could be tested to see if they are better able to find patterns in the pooled dataset which can enable higher classification accuracy from the pooled dataset. Finally, to test the robustness of the classifiers, testing should also be done on transradial amputees and individuals with difficulty with upper-limb control. The game should be further optimised to perform a larger number of gestures to allow an individual to enjoy more variety and a greater degree of challenge.

References

[1] G. Cooper, I. Sheret, L. McMillian, K. Siliverdis, N. Sha, D. Hodgins, L. Kenney, and D. Howard, "Inertial sensor-based knee flexion/extension angle estimation," *Journal of biomechanics*, vol. 42, no. 16, pp. 2678–2685, 2009.

[2] A. Alkan and M. Günay, "Identification of emg signals using discriminant analysis and svm classifier," *Expert Systems with Applications*, vol. 39, no. 1, pp. 44–47, 2012.

[3] A. Phinyomark, H. Hu, P. Phukpattaranont, and C. Limsakul, "Application of linear discriminant analysis in dimensionality reduction for hand motion classification," *Measurement Science Review*, vol. 12, no. 3, pp. 82–89, 2012.

[4] S. Wilson and R. Vaidyanathan, "Upper-limb prosthetic control using wearable multichannel mechanomyography," in *Rehabilitation Robotics (ICORR), 2017 International Conference on*. IEEE, 2017, pp. 1293–1298.

[5] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[6] R. Vaidyanathan, B. Chung, L. Gupta, H. Kook, S. Kota, and J. D. West, "Tongue-movement communication and control concept for hands-free human–machine interfaces," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 37, no. 4, pp. 533–546, 2007.

[7] Z. Wu, X. Wang, Z. Gao, and G. Ren, "Automatic digital modulation recognition based on support vector machines," in *Neural Networks and Brain, 2005. ICNN&B'05. International Conference on*, vol. 2. IEEE, 2005, pp. 1025–1028.

[8] J. Cannan and H. Hu, "Human-machine interaction (hmi): A survey," *University of Essex*, 2011.

[9] K. Tai, S. Blain, and T. Chau, "A review of emerging access technologies for individuals with severe motor impairments," *Assistive Technology*, vol. 20, no. 4, pp. 204–221, 2008.

[10] R. Anderson, *The aftermath of stroke: the experience of patients and their families*. Cambridge University Press, 2006.

[11] J. W. Burke, M. McNeill, D. Charles, P. J. Morrow, J. Crosbie, and S. McDonough, "Augmented reality games for upper-limb stroke rehabilitation," in *2010 Second International Conference on Games and Virtual Worlds for Serious Applications*. IEEE, 2010, pp. 75–78.

[12] N. A. Bartolome, A. M. Zorrilla, and B. G. Zapirain, "Can game-based therapies be trusted? is game-based education effective? a systematic review of the serious games for health and education," in *IEEE Intelligent Systems*. IEEE, 2011, pp. 275–282.

[13] Y. Ma, Y. Liu, R. Jin, X. Yuan, R. Sekha, S. Wilson, and R. Vaidyanathan, "Hand gesture recognition with convolutional neural networks for the multimodal uav control," in *Research, Education and Development of Unmanned Aerial Systems (RED-UAS), 2017 Workshop on*. IEEE, 2017, pp. 198–203.

[14] M. T. Wolf, C. Assad, M. T. Vernacchia, J. Fromm, and H. L. Jethani, "Gesture-based robot control with variable autonomy from the jpl biosleeve," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE, 2013, pp. 1160–1165.

[15] S. Theodoridis, K. Koutroumbas *et al.*, "Pattern recognition," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, p. 376, 2008.

[16] A. D. Roche, H. Rehbaum, D. Farina, and O. C. Aszmann, "Prosthetic myoelectric control strategies: a clinical perspective," *Current Surgery Reports*, vol. 2, no. 3, p. 44, 2014.

[17] R. A. Ramadan and A. V. Vasilakos, "Brain computer interface: control signals review," *Neurocomputing*, vol. 223, pp. 26–44, 2017.

[18] M. Mace, K. Abdullah-al Mamun, A. A. Naeem, L. Gupta, S. Wang, and R. Vaidyanathan, "A heterogeneous framework for real-time decoding of bioacoustic signals: Applications to assistive interfaces and prosthesis control," *Expert Systems with Applications*, vol. 40, no. 13, pp. 5049–5060, 2013.

[19] R. H. Chowdhury, M. B. Reaz, M. A. B. M. Ali, A. A. Bakar, K. Chellappan, and T. G. Chang, "Surface electromyography signal processing and classification techniques," *Sensors*, vol. 13, no. 9, pp. 12 431–12 466, 2013.

[20] P. Geethanjali, "Myoelectric control of prosthetic hands: state-of-the-art review," *Medical Devices (Auckland, NZ)*, vol. 9, p. 247, 2016.

[21] B. Hudgins, P. Parker, and R. N. Scott, "A new strategy for multifunction myoelectric control," *IEEE Transactions on Biomedical Engineering*, vol. 40, no. 1, pp. 82–94, 1993.

[22] S.-O. Shin, D. Kim, and Y.-H. Seo, "Controlling mobile robot using imu and emg sensor-based gesture recognition," in *Broadband and Wireless Computing, Communication and Applications (BWCCA), 2014 Ninth International Conference on.* IEEE, 2014, pp. 554–557.

[23] A. D. Chan and K. B. Englehart, "Continuous myoelectric control for powered prostheses using hidden markov models," *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 1, pp. 121–124, 2005.

[24] A. D. Chan and G. C. Green, "Myoelectric control development toolbox," *CMBES Proceedings*, vol. 30, no. 1, 2017.

[25] O. De Silva, G. Mann, R. Gosine *et al.*, "Pairwise linear discriminant analysis of electromyographic signals," 2011.

[26] C. J. De Luca, "The use of surface electromyography in biomechanics," *Journal of applied biomechanics*, vol. 13, no. 2, pp. 135–163, 1997.

[27] T. W. Beck, "Technical aspects of surface mechanomyography," *Applications of Mechanomyography for examining muscle function. Kerala: Transworld Research Network*, pp. 95–107, 2010.

[28] M. Stokes, "Acoustic myography: applications and considerations in measuring muscle performance," *Isokinetics and Exercise Science*, vol. 3, no. 1, pp. 4–15, 1993.

[29] R. B. Woodward, S. J. Shefelbine, and R. Vaidyanathan, "Pervasive monitoring of motion and muscle activation: Inertial and mechanomyography fusion," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 5, pp. 2022–2033, 2017.

[30] P. Angeles, Y. Tai, N. Pavese, S. Wilson, and R. Vaidyanathan, "Automated assessment of symptom severity changes during deep brain stimulation (dbs) therapy for parkinson's disease," in *Rehabilitation Robotics (ICORR), 2017 International Conference on.* IEEE, 2017, pp. 1512–1517.

[31] N. Alves and T. Chau, "Classification of the mechanomyogram: its potential as a multifunction access pathway," in *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE.* IEEE, 2009, pp. 2951–2954.

[32] P. Prociow, A. Wolczowski, T. G. Amaral, O. P. Dias, and J. Filipe, "Identification of hand movements based on mmg and emg signals." in *BIOSIGNALS (2)*, 2008, pp. 534–539.

[33] R. R. Manero, A. Shafti, B. Michael, J. Grewal, J. L. R. Fernández, K. Althoefer, and M. J. Howard, "Wearable embroidered muscle activity sensing device for the human upper leg," in *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the.* IEEE, 2016, pp. 6062–6065.

[34] A. K. Jain, R. P. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[35] A. Posatskiy and T. Chau, "Design and evaluation of a novel microphone-based mechanomyography sensor with cylindrical and conical acoustic chambers," *Medical engineering & physics*, vol. 34, no. 8, pp. 1184–1190, 2012.

[36] ——, "The effects of motion artifact on mechanomyography: A comparative study of microphones and accelerometers," *Journal of Electromyography and Kinesiology*, vol. 22, no. 2, pp. 320–324, 2012.

[37] S. O. Madgwick, A. J. Harrison, and R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm," in *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on.* IEEE, 2011, pp. 1–7.

[38] H.-B. Xie, Y.-P. Zheng, and J.-Y. Guo, "Classification of the mechanomyogram signal using a wavelet packet transform and singular value decomposition for multifunction prosthesis control," *Physiological measurement*, vol. 30, no. 5, p. 441, 2009.

[39] J. Silva, W. Heim, and T. Chau, "Mmg-based classification of muscle activity for prosthesis control," in *Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE*, vol. 1. IEEE, 2004, pp. 968–971.

[40] F. Gustafsson, *Segmentation of signals using piecewise constant linear regression models.* Linköping University, 1994.

[41] R. Boostani and M. H. Moradi, "Evaluation of the forearm emg signal features for the control of a prosthetic hand," *Physiological measurement*, vol. 24, no. 2, p. 309, 2003.

[42] M. A. Oskoei, H. Hu *et al.*, "Support vector machine-based classification scheme for myoelectric control applied to upper limb." *IEEE Trans. Biomed. Engineering*, vol. 55, no. 8, pp. 1956–1965, 2008.

[43] S. Negi, Y. Kumar, and V. Mishra, "Feature extraction and classification for emg signals using linear discriminant analysis," in *Advances in Computing, Communication, & Automation (ICACCA)(Fall), International Conference on.* IEEE, 2016, pp. 1–6.

[44] W. Caesarendra, S. U. Lekson, K. A. Mustaqim, A. R. Winoto, and A. Widyotriatmo, "A classification method of hand emg signals based on principal component analysis and artificial neural network," in *Instrumentation, Control and Automation (ICA), 2016 International Conference on.* IEEE, 2016, pp. 22–27.

[45] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," Yale University New Haven United States, Tech. Rep., 1997.

[46] S. Nikan and M. Ahmadi, "Partial face recognition based on template matching," in *Signal-Image Technology & Internet-Based Systems (SITIS), 2015 11th International Conference on.* IEEE, 2015, pp. 160–163.

[47] L. Gupta, D. L. Molfese, R. Tammana, and P. G. Simos, "Nonlinear alignment and averaging for estimating the evoked potential," *IEEE Transactions on Biomedical Engineering*, vol. 43, no. 4, pp. 348–356, 1996.

[48] A. Wang, W. Yuan, J. Liu, Z. Yu, and H. Li, "A novel pattern recognition algorithm: Combining art network with svm to reconstruct a multi-class classifier," *Computers & Mathematics with Applications*, vol. 57, no. 11-12, pp. 1908–1914, 2009.

[49] A. Shigeo, "Support vector machines for pattern classification," *Advances in Pattern Recognition, Springer, Heidelberg*, 2005.

[50] A. H. Needham, F. P. Paszkiewicz, M. F. M. Alias, S. Wilson, A. A. Dehghani-Sanij, B. C. Khoo, and R. Vaidynanathan, "Subject-independent data pooling in classification of gait intent using mechanomyography on a transtibial amputee," 2018.

Appendices

Appendix A Raw MMG data extraction - MATLAB Script

```
function A = data(filename) % Function which extracts MMG data from the raw data
raw = csvread(filename); % Reads the raw data saved the MMG and IMU
data = zeros(size(raw,1),8); % Creates an array of zeros

for n = 1:size(raw,1)
for m = 10:17
data(n,m-9) = raw(n,(m*2))*256+raw(n,(m*2)+1); % Extracts the MMG data
if(data(n,m-9)>(2^15)-1);
data(n,m-9) = data(n,m-9 ) - 2^16;
end
end % This produces the input data. Need to now put the input data into a butterworth bandpass filter.
end % The input data for each MMG is in its own particular column.
relevant_data = zeros(size(raw,1),6);
relevant_data(:,1) = data(:,1); % MMG 1
relevant_data(:,2) = data(:,2); % MMG 2 data
relevant_data(:,3) = data(:,3); % MMG 3 data put into the array.
relevant_data(:,4) = data(:,4); % MMG 4
relevant_data(:,5) = data(:,5); % MMG 5 data
relevant_data(:,6) = data(:,6); % MMG 6 data
lin = linspace(1,size(raw,1),size(raw,1)); % Made a vector which has a range of size(raw,1)
A = relevant_data;
```

Appendix B Filtering - MATLAB Script

```
function C = Filtering_only_of_original_data(M)
[b,a] = butter(1,[1 100]/(1000/2),'bandpass'); % This is a 2nd order bandpass filter( as n = 1 ,
%The lower cutoff frequency is 1 hz and a higher cutoff frequency is 100 hz.
%Sampling frequency is 1000 hz
filtered_data = filter(b,a,M(:,:));
C = filtered_data;
```

Appendix C Segmenting - MATLAB Script

```
function B = Segmentation_moving_data_600_10_instances(M)
 % This function finds points which corresponds to gesture instances.
 % The filtered data is then segmented at these segmentation points.
array = zeros(1,10);% Made an array of zero as this will
index = 1; % Starting index
windowlength = 600;  % Length of the segmentation window

i = 1000; % Iteration starts at point 1000. The first 1000 ms is not used
    .

while i <= size(M,1)- windowlength -51 && index < 11;
    % iterates till the end of the data in the file or until 10
        segmentation points are found
     if M(i,1)+ M(i,2)+M(i,3)+ M(i,4)+ M(i,5)+M(i,6) > 20000
%Looks if the sum of the channels at a point in time exceeds the
    threshold
         array(1,index) = i  ; % Marks the segmenation point
         index = index + 1;
         i =  i + 1549; % Skips 1549 ms of data after a segmentation point
     else
         i = i + 1;
     end
end
if array(1,10) == 0 % This checks if there are 10 segmentation points
    % , If there are not 10 segmentaiton points, the iteration occurs
        from
    % the start whilst using the lower threshold
     array = zeros(1,10);
     index = 1;
     i = 1000;

else
     index = 11;
end
while  i <= size(M,1)-windowlength -51 && index < 11% Iterates through
    the data
     if M(i,1) + M(i,2) + M(i,3)+ M(i,4)+ M(i,5) + M(i,6) > 10000;
% Checks if the sum of the channels exceed the lower threshold of 10,000
         array(1,index) = i  ;
         index = index + 1;
         i =  i + 1549; % Skips 1549 ms of data after the segmentation
```

```
            point
    else
        i = i +1;
    end
end
% This part of the code checks if there is 10 segmentation points
% If there is not a tenth segmentation point, then the 9th segementation
% point is used twice. This is repeated for the other indexes also.
if array(1,2) == 0;
    array(1,2) = array(1,1);
end
if array(1,3) == 0;
    array(1,3) = array(1,2);
end
if array(1,4) == 0;
    array(1,4) = array(1,3);
end
if array(1,5) == 0;
    array(1,5) = array(1,4);
end
if array(1,6) == 0;
    array(1,6) = array(1,5);
end
if array(1,7) == 0;
    array(1,7) = array(1,6);
end
if array(1,8) == 0;
    array(1,8) = array(1,7);
end
if array(1,9) == 0;
    array(1,9) = array(1,8);
end
if array(1,10) == 0;
    array(1,10) = array(1,9);
end

B = array; % This is the output array
end
```

Appendix D Processing data - MATLAB Script

```
function A = processing_data_600_10_instances_LDA(filename); % Filename is the input
% Function used to filter,square, segment and perform a moving average on
% the data.
windowlength = 600; % Size of the segmentation window
original_data = data(filename); % Extracts the raw data from the input file
only_filtered_data = Filtering_only_of_original_data(original_data); % Filters the raw data
filtered_squared_data = only_filtered_data.^2; % Squares the filtered data
moving_average_original_data = movmean(filtered_squared_data,[0 150],1); %
% This calculates a 150 point forward moving average on the squared filtered data.
segmentation_points = Segmentation_moving_data_600_10_instances(moving_average_original_data);
% This calculates the segmentation points based on the points in which
% moving average data exceeds the threshold

data_array = zeros(windowlength*10,6);
% This collates the 10 gesture instances in a file and collates it

for i = 0:9

    data_array(1+(windowlength*i):windowlength+(windowlength*i),1:6) = filtered_squared_data(segmentation_points(1,i+1)-50:segmentation_points(1,i+1)+windowlength -51,1:6);
% This uses the segmentation points to collate the 10 gesture instances.
end
A = data_array;
end
```

Appendix E Offline template generation- MATLAB Script

```
function A =
    Overlapping_instances_filtered_average_600_10_instances_confuse(
    samples)% This function generates the sample names in a random order
    which can then be used for the test data and input data
instances = 30;

full_data_array = zeros(600,6*instances)
% Looks at the different files from the input 'samples'
filename1 = samples(1).name
filename2 = samples(2).name
filename3 = samples(3).name

% Collates the gestures instances in the different files
data_array_1 = processing_data_600_10_instances_LDA(filename1)
data_array_2 = processing_data_600_10_instances_LDA(filename2)
data_array_3 = processing_data_600_10_instances_LDA(filename3)

% Adds the collated gesture instances into one large array of data
full_data_array(1:600,1:60) =data_array_1(1:600,1:60)
full_data_array(1:600,61:120) =data_array_2(1:600,1:60)
full_data_array(1:600,121:180) =data_array_3(1:600,1:60)

%Saves the value from MMG1 and makes an average of these values
MMG1 = zeros(600,instances)
MMG1 = full_data_array(1:600,1:6:instances*6)
average_MMG1 = mean(MMG1,2) % Mean(MMG1,2) makes the mean of the data
    along the rows making a one new column vector
plot(MMG1)

%Saves the value from MMG2 and makes an average of these values
MMG2 = zeros(600,instances)
MMG2 = full_data_array(1:600,2:6:instances*6)
average_MMG2 = mean(MMG2,2)

%Saves the value from MMG3 and makes an average of these values
MMG3 = zeros(600,instances)
MMG3 = full_data_array(1:600,3:6:instances*6)
average_MMG3 = mean(MMG3,2)

%Saves the value from MMG4 and makes an average of these values
MMG4 = zeros(600,instances)
```

```
MMG4 = full_data_array(1:600,4:6:instances*6)
average_MMG4 = mean(MMG4,2)

%Saves the value from MMG5 and makes an average of these values
MMG5 = zeros(600,instances)
MMG5 = full_data_array(1:600,5:6:instances*6)
average_MMG5 = mean(MMG5,2)
figure; plot(MMG5)

%Saves the value from MMG6 and makes an average of these values
MMG6 = zeros(600,instances)
MMG6 = full_data_array(1:600,6:6:instances*6)
average_MMG6 = mean(MMG3,2)
figure; plot(MMG6)

% Creation of template from the mean of each channel
Template = zeros(600,instances)
Template(:,1) = average_MMG1(:,1)
Template(:,2) = average_MMG2(:,1);
Template(:,3) = average_MMG3(:,1)
Template(:,4) = average_MMG4(:,1)
Template(:,5) = average_MMG5(:,1);
Template(:,6) = average_MMG6(:,1)
figure; plot(Template)
A = Template
```

Appendix F Absolute difference measure- MATLAB Script

```
function B = Euclidean_norm_initial_point_no_dot_product_6_gestures(instance1 ,M1,M2,M3,M4,M5,M6) % The inputs will be the particular instance and the 5 template matrices
% This looks at the absolute difference between a instance and the different templates
% Finds the sum of the absolute difference between instance and template 1
absolute_difference1_M1 = abs(difference1_M1)
summation1_M1 = sum(absolute_difference1_M1)
euclidean_norm1_M1 = sum(summation1_M1)

% Finds the sum of the absolute difference between instance and template 2
difference1_M2 = instance1 - M2
absolute_difference1_M2 = abs(difference1_M2)
summation1_M2 = sum(absolute_difference1_M2)
euclidean_norm1_M2 = sum(summation1_M2)

% Finds the sum of the absolute difference between instance and template 3
difference1_M3 = instance1 - M3
absolute_difference1_M3 = abs(difference1_M3)
summation1_M3 = sum(absolute_difference1_M3)
euclidean_norm1_M3 = sum(summation1_M3)

% Finds the sum of the absolute difference between instance and template 4
difference1_M4 = instance1 - M4
absolute_difference1_M4 = abs(difference1_M4)
summation1_M4 = sum(absolute_difference1_M4)
euclidean_norm1_M4 = sum(summation1_M4)

% Finds the sum of the absolute difference between instance and template 5
difference1_M5 = instance1- M5
absolute_difference1_M5 = abs(difference1_M5)
summation1_M5 = sum(absolute_difference1_M5)
euclidean_norm1_M5 = sum(summation1_M5)

% Finds the sum of the absolute difference between instance and template 6
```

```
difference1_M6 = instance1 - M6
absolute_difference1_M6 = abs(difference1_M6)
summation1_M6 = sum(absolute_difference1_M6)
euclidean_norm1_M6 = sum(summation1_M6)

%Makes an array of all the different sum of differences
Values = [euclidean_norm1_M1; euclidean_norm1_M2; euclidean_norm1_M3;
    euclidean_norm1_M4; euclidean_norm1_M5; euclidean_norm1_M6]
% Finds the minimum value and which template has the minimum value
[M, I] = min(Values)
B = I
```

Appendix G Offline template classification- MATLAB Script

```
% Number of instances
gestures = 6;

% Dataset for gesture 1
rand_data1 = training_test_dataset('G1'); % Generation of the samples in
    a random order
training_samples = 3; % 30 instances used as 3 files used

training_data1 = rand_data1(1:training_samples);
% Test data
test_data1 = rand_data1(training_samples+1:10);
% Creates a template for Gesture 1 from the training data
t1 = Overlapping_instances_filtered_average_600_10_instances_confuse(
    training_data1); % Generation of the template from the first 8 random
     samples

%Dataset for gesture 2
rand_data2 = training_test_dataset('G2');
% Gesture 2 training data
training_data2 = rand_data2(1:training_samples);
% Gesture 2 test data
test_data2= rand_data2(training_samples+1:10);
% Creates a template for Gesture 1 from the training data
t2 = Overlapping_instances_filtered_average_600_10_instances_confuse(
    training_data2);


% Dataset for gesture 3
rand_data3 = training_test_dataset('G3');
```

```
training_data3 = rand_data3(1:training_samples);
test_data3= rand_data3(training_samples+1:10);
% Generation of gesture 3 template
t3 = Overlapping_instances_filtered_average_600_10_instances_confuse(
    training_data3);

% Dataset for gesture 4
rand_data4 = training_test_dataset('G4');
training_data4 = rand_data4(1:training_samples);
test_data4= rand_data4(training_samples+1:10);
% Generation of gesture 4 template
t4 = Overlapping_instances_filtered_average_600_10_instances_confuse(
    training_data4);

% Dataset for gesture 5
rand_data5 = training_test_dataset('G5');
training_data5 = rand_data5(1:training_samples);
test_data5= rand_data5(training_samples+1:10);
% Generation of gesture 5 template
t5 = Overlapping_instances_filtered_average_600_10_instances_confuse(
    training_data5);

% Dataset for gesture 6
rand_data6 = training_test_dataset('G6');
training_data6 = rand_data6(1:training_samples);
test_data6= rand_data6(training_samples+1:10);
% Generation of gesture 6 template
t6 = Overlapping_instances_filtered_average_600_10_instances_confuse(
    training_data6);



% Sets the original value for the counter.
% This tells us where the test gestures get classified into
i = 0;
j = 0;
k = 0;
l = 0;
m = 0;
n = 0;
% Calculates which template has the minimum difference with the test
    gesture
```

```
for a = 1:7 % This looks at the 7 data files present in the testdata
    data_array = processing_data_600_10_instances_LDA(test_data6(a).name)
        ;
    for b = 1:10 % This looks at four instances in sample file
        if Euclidean_norm_initial_point_no_dot_product_6_gestures(
            data_array(1:600,6*b-5:6*b),t1,t2,t3,t4,t5,t6) == 1;
            i = i + 1;
        elseif Euclidean_norm_initial_point_no_dot_product_6_gestures(
            data_array(1:600,6*b-5:6*b),t1,t2,t3,t4,t5,t6) ==  2;
            j = j + 1;
        elseif Euclidean_norm_initial_point_no_dot_product_6_gestures(
            data_array(1:600,6*b-5:6*b),t1,t2,t3,t4,t5,t6) == 3;
            k = k + 1;
        elseif Euclidean_norm_initial_point_no_dot_product_6_gestures(
            data_array(1:600,6*b-5:6*b),t1,t2,t3,t4,t5,t6) == 4;
            l = l + 1;
        elseif Euclidean_norm_initial_point_no_dot_product_6_gestures(
            data_array(1:600,6*b-5:6*b),t1,t2,t3,t4,t5,t6) == 5;
            m = m + 1;
        elseif Euclidean_norm_initial_point_no_dot_product_6_gestures(
            data_array(1:600,6*b-5:6*b),t1,t2,t3,t4,t5,t6) == 6;
            n = n + 1;
        end
    end
end

confusion = [i j k l m n] % Confusion matrix
B =  confusion
```

Appendix H Offline LDA classification - MATLAB Script

```
win_size = 600; % Size of the window where features are extracted from
win_inc = 600; % Increment in which data is moved to another window
gesture_no = 7; % This is the number of gestures that are being recorded
 l = 7 % Number of files used for training

% Generation of the files within a folder in a random order
rand_data1 = training_test_dataset('G1');
rand_data2 = training_test_dataset('G2');
rand_data3 = training_test_dataset('G3');
rand_data4 = training_test_dataset('G4');
rand_data5 = training_test_dataset('G5');
rand_data6 = training_test_dataset('G6');
rand_data7 = training_test_dataset('G7');

% Taking 7 files for each gesture as training
training_data1 = rand_data1(1:l);
training_data2 = rand_data2(1:l);
training_data3 = rand_data3(1:l);
training_data4 = rand_data4(1:l);
training_data5 = rand_data5(1:l);
training_data6 = rand_data6(1:l);
training_data7 = rand_data7(1:l);

% Collating all the data from the training set into one large array
Training_data = Collated_instances_gestures_training_data_260718_LDA(
    training_data1,training_data2, training_data3, training_data4,
    training_data5, training_data6, training_data7);
training_file_no = size(training_data1,1);% Number of files in the
    training set
Training_motion = zeros(training_file_no*gesture_no*10,1);
% Training_motion tells us which gesture is being performed in the
    training data
for i = 1:training_file_no*10
    Training_motion(i,1) = 1;
    Training_motion(i+(training_file_no*10),1) = 2;
    Training_motion(i+(2*training_file_no*10),1) = 3;
    Training_motion(i+(3*training_file_no*10),1) = 4;
    Training_motion(i+(4*training_file_no*10),1) = 5;
    Training_motion(i+(5*training_file_no*10),1) = 6;
    Training_motion(i+(6*training_file_no*10),1) = 7;
end
```

```
% Training index tells of the start of a gesture
Training_index = zeros(training_file_no*10*gesture_no,1);

for i = 0:(training_file_no*10*gesture_no)-1 % Iterating to add all the
    points when the gesture begins
    Training_index(i+1,1) = 1 + (600*i);
end
% Extracts the features of the training data with a window size of 600
    points and
% increment of 600 points
feature_training = extract_feature(Training_data,win_size,win_inc); %
    This is used to extract the RMS values and the autoregressive
    coefficients. The order of the autoregressive coefficient is 4 and
    the RMS is one value so there is a total of 5 features per channel so
     there is 30 columns of feature data. The number of rows comes from
    the number of times the window goes against the data.
% Gets the class labels of training sets
class_training = getclass(Training_data,Training_motion,Training_index,
    win_size,win_inc); % I am not sure what class_training



%Generating test data
testing_data1 = rand_data1(l+1:10);
testing_data2 = rand_data2(l+1:10);
testing_data3 = rand_data3(l+1:10);
testing_data4 = rand_data4(l+1:10);
testing_data5 = rand_data5(l+1:10);
testing_data6 = rand_data6(l+1:10);
testing_data7 = rand_data7(l+1:10);

% Collates the data in a large test file
testing_data = Collated_instances_gestures_training_data_260718_LDA(
    testing_data1,testing_data2, testing_data3, testing_data4,
    testing_data5, testing_data6, testing_data7); % This is all the test
    data added together
testing_file_no = size(testing_data1,1);
% Obtains the label for the specific test gestures
testing_motion = zeros(testing_file_no*10*gesture_no,1); % This shows the
     gestures that is occuring
% Labels the index of the test instances
for i = 1:testing_file_no * 10
```

```
    testing_motion(i,1) = 1;
    testing_motion(i+(testing_file_no * 10),1) = 2;
    testing_motion(i+(testing_file_no * 10*2),1) = 3;
    testing_motion(i+(testing_file_no * 10*3),1) = 4;
    testing_motion(i+(testing_file_no * 10*4),1) = 5;
    testing_motion(i+(testing_file_no * 10*5),1) = 6;
    testing_motion(i+(testing_file_no * 10*6),1) = 7;
end

testing_index = zeros(testing_file_no*10*gesture_no,1);
for i = 0:(testing_file_no*10*gesture_no)-1
    testing_index(i+1,1) = 1 + (win_size*i);
end
% Extracts the features from the test set
feature_testing = extract_feature(testing_data,win_size,win_inc); % This
    gets all the gestures
class_testing = getclass(testing_data,testing_motion,testing_index,
    win_size,win_inc); %

Nfeat = 6; % number of features to reduce to
%[feature_training,feature_testing] = pca_feature_reduction(
    feature_training,Nfeat,feature_testing);
[feature_training,feature_testing] = ulda_feature_reduction(
    feature_training,Nfeat,class_training,feature_testing);
% LDA feature reduction



% LDA classsification which outputs the training error and testing error
[error_training,error_testing,classification_training,
    classification_testing]...
    = ldaclassify(feature_training,feature_testing,class_training,
        class_testing,'linear');

% Produces confusion matrix of the data.
collated_confusion_matrix = confmat(class_testing,classification_testing)
    ;
```

Appendix I Real-time extracting training features - MATLAB Script

```
function [feature_training,class_training] =
    extract_training_features_v3_010918(Training_data,gesture_no,
    gesture_instances)
% Inputs are the training data, number of gestures and the number of
    instances per gesture
win_size = 600;% Size of the window where features are extracted from
win_inc = 600;  % Increment in which data is moved to another window


Training_motion = zeros(gesture_instances*(gesture_no),1);
% Training_motion tells us which gesture is being performed in the
    training data
for i = 1:gesture_instances
    Training_motion(i,1) = 0;
    Training_motion(i+(gesture_instances),1) = 1;
    Training_motion(i+(2*gesture_instances),1) = 2;
    Training_motion(i+(3*gesture_instances),1) = 3;
    Training_motion(i+(4*gesture_instances),1) = 4;
    Training_motion(i+(5*gesture_instances),1) = 5;
    Training_motion(i+(6*gesture_instances),1) = 6;

end

Training_index = zeros(gesture_instances*gesture_no,1); % Training_index
    tells us in the training data when a specific gesture begins
for i = 0:(gesture_instances*(gesture_no))-1 % Iterating to add all the
    points when the gesture begins
    Training_index(i+1,1) = 1 + (600*i);
end
% Obtains the features of the training data and the class labels
feature_training = extract_feature(Training_data,win_size,win_inc);
class_training = getclass(Training_data,Training_motion,Training_index,
    win_size,win_inc); % I am not sure what class_training

end
```

Appendix J Real-time LDA classification- MATLAB Script

```
function classification_testing  = LDA_real_time_classification(
    testing_data,feature_training,class_training,testing_motion)
% Inputs are the testing data, the features of the training data and the
% label of the test data
win_size = 600;%% Size of the window where features are extracted from
win_inc = 600;  % Increment in which data is moved to another window
testing_index = 1; % This is the start of the gesture
% Extracts the features of the test data and the labels
feature_testing = extract_feature(testing_data,win_size,win_inc); % This
    gets all the gestures
class_testing = getclass(testing_data,testing_motion,testing_index,
    win_size,win_inc); %

% Dimensionality reduction using LDA
Nfeat = 6; % number of features to reduce to
%[feature_training,feature_testing] = pca_feature_reduction(
    feature_training,Nfeat,feature_testing);
[feature_training,feature_testing] = ulda_feature_reduction(
    feature_training,Nfeat,class_training,feature_testing);

% Classification using LDA
[error_training,error_testing,classification_training,
    classification_testing]...
    = ldaclassify(feature_training,feature_testing,class_training,
        class_testing,'linear');

end
```

Appendix K Feature extraction comparison

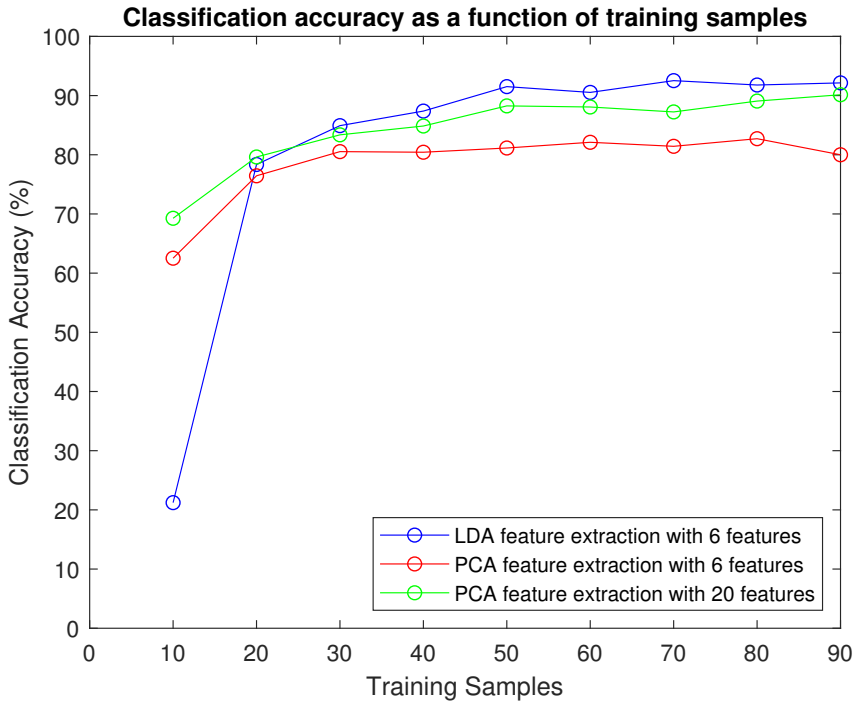


Figure K.1: Results of Subject 2

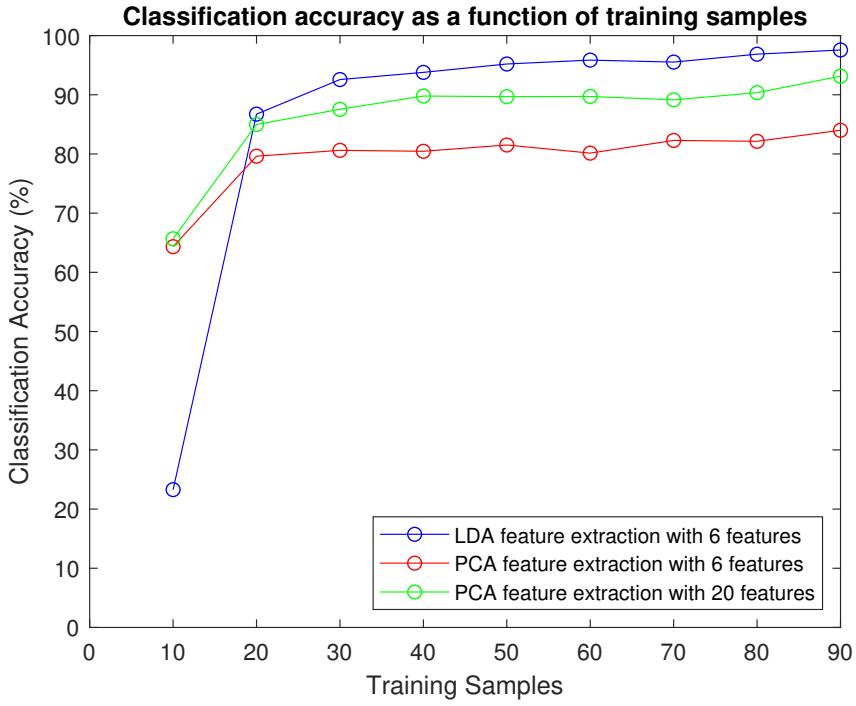


Figure K.2: Results of Subject 3

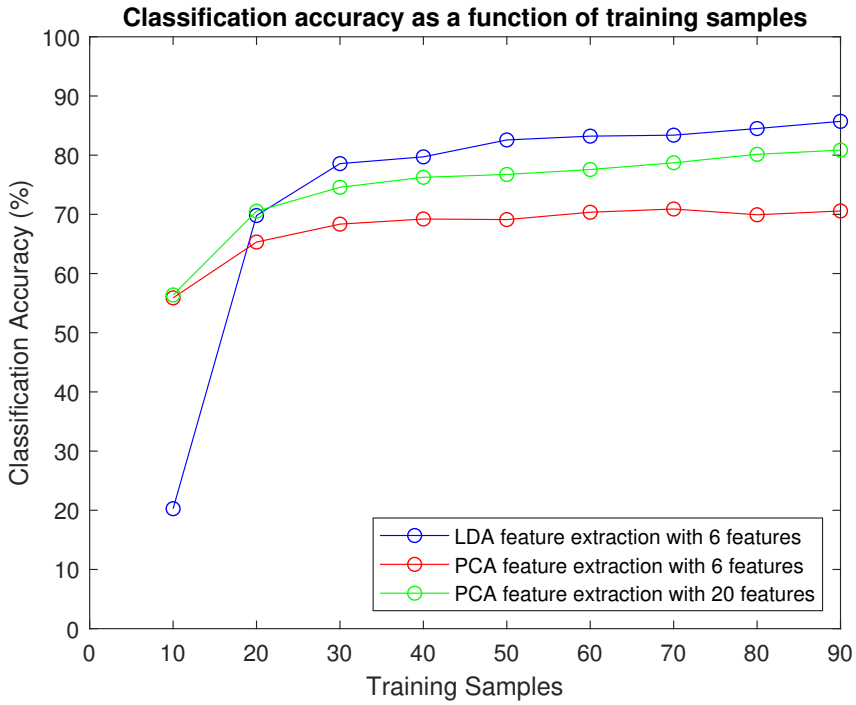


Figure K.3: Results of Subject 4

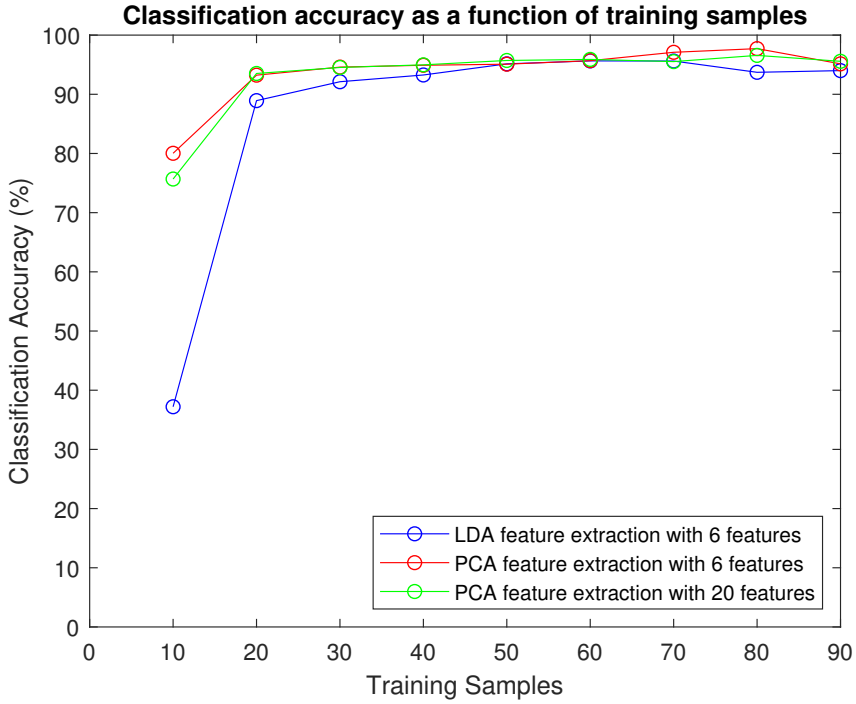


Figure K.4: Results of Subject 5

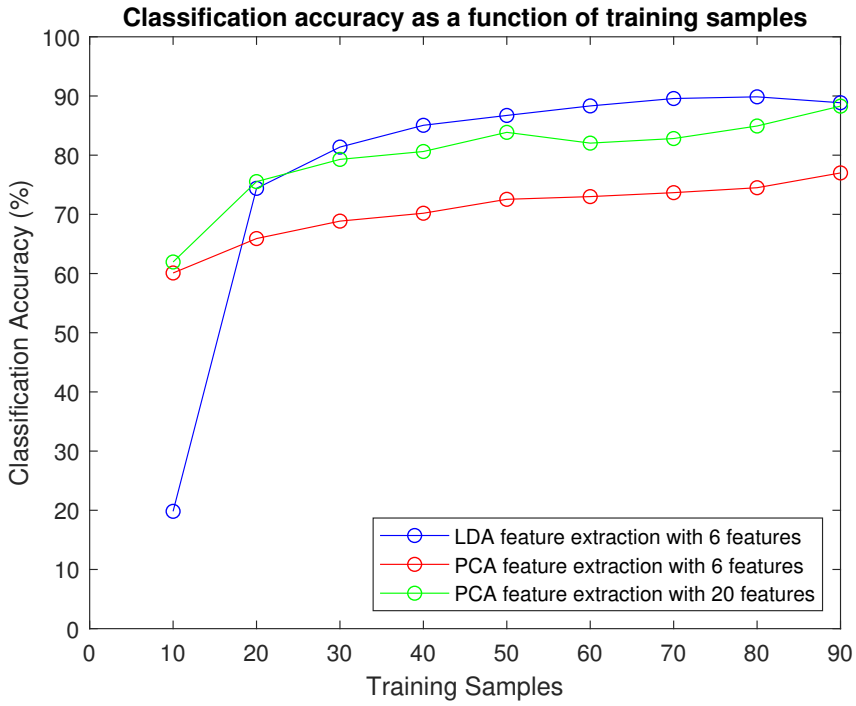


Figure K.5: Results of Subject 6

Appendix L Rift controller game results

Table L.1: Measurements from game when using the rift controllers for the second trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	5.93	6.61	2	3
2	0.86	1.87	1	2
3	5.27	6.86	2	3
4	2.41	3.90	2	3
5	4.28	5.68	1	2
6	3.15	5.15	2	3
7	1.80	3.37	1	2
8	1.08	2.49	1	2
9	2.38	4.89	1	2
10	1.78	3.14	1	2

Table L.2: Measurements from game when using the rift controllers for the third trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	8.09	10.18	1	2
2	1.52	2.84	1	2
3	1.23	2.49	1	2
4	4.90	5.77	1	2
5	1.47	2.47	1	2
6	1.60	2.40	2	3
7	2.61	3.46	1	2
8	1.42	2.13	1	2
9	1.42	3.31	2	3
10	0.64	1.38	1	2

Table L.3: Measurements from game when using the rift controllers for the fourth trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	3.18	4.17	1	2
2	1.18	2.21	1	2
3	1.57	2.54	1	2
4	1.97	3.34	1	2
5	2.26	3.44	2	3
6	4.38	6.09	5.13	8.26
7	5.13	8.26	2	4
8	1.07	2.67	1	2
9	6.41	7.54	1	2
10	4.72	7.52	1	2

Table L.4: Measurements from game when using the rift controllers for the fifth trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	2.35	3.87	1	2
2	1.17	2.81	1	2
3	10.20	12.34	2	3
4	16.61	17.33	2	3
5	1.32	2.18	1	2
6	3.82	4.88	4	5
7	4.08	6.68	1	2
8	5.99	7.41	2	3
9	1.91	13.68	2	3
10	3.31	11.08	1	2

Appendix M NU interface game results

Table M.1: Measurements from game when using the NU interface for the second trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	6.43	11.58	1	3
2	3.82	6.56	1	2
3	5.03	11.06	2	3
4	8.20	13.01	1	3
5	56.55	58.51	3	4
6	22.53	26.24	5	7
7	10.83	12.58	1	2
8	20.57	21.66	5	6
9	2.86	4.49	1	2
10	3.61	14.32	2	3

Table M.2: Measurements from game when using the NU interface for the third trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	6.25	14.78	2	4
2	11.61	41.12	1	4
3	8.76	14.56	1	3
4	20.89	55.01	1	2
5	8.74	17.18	1	2
6	3.28	4.80	1	2
7	18.38	20.13	1	2
8	8.42	25.59	4	5
9	38.29	42.77	5	6
10	1.74	6.23	1	2

Table M.3: Measurements from game when using the NU interface for the fourth trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	16.97	18.82	2	3
2	1.76	3.40	1	2
3	1.74	3.50	1	2
4	6.12	17.39	2	4
5	2.41	10.61	1	2
6	2.08	4.16	2	1
7	2.84	4.70	2	3
8	5.36	7.54	3	4
9	1.86	3.93	1	2
10	1.76	3.60	1	2

Table M.4: Measurements from game when using the NU interface for the fifth trial

Object	Time taken to grab an object (s)	Time taken to place object in goal (s)	Number of actions to grab object	Number of actions used to put object in goal
1	5.65	7.41	1	2
2	1.96	6.89	1	2
3	5.58	44.30	2	3
4	3.61	9.51	1	2
5	2.84	5.03	1	2
6	7.33	11.05	2	3
7	1.85	22.41	1	3
8	11.92	26.90	1	2
9	15.96	22.54	1	2
10	3.49	18.70	1	2