Design and Implementation of Electromyography (EMG) based Real-Time Pattern Recognition model for Prosthetic hand Control

Pasan Yashoda Jayaweera

hkpasan@gmail.com

Liverpool John Moores University, UK

International College of Business and Technology, Sri Lanka

Table of Contents

List of Figures 5
List of Tables 7
Chapter 1 – Introduction 8
1.1 Background of the project: 8
1.2 Problem identification: 8
1.3 Aim and Objectives 9
1.4 Scope and Limitations: 9
Chapter 2 - Literature Review 10
2.1 Introduction 10
2.2 Basis for using EMG signals 10
2.3 EMG signal acquisition 10
2.4 Filtering the EMG Signal 11
2.5 Control configurations 11
- Direct control 11
- Pattern Recognition 12

2.6 Feature extraction 12
- Mean Absolute Value (MAV) 14
- Autoregressive Coefficients 14
- Integrated EMG (IEMG) 14
- Zero crossings (ZC) 14
- Slope Sign Changes (SSC) 15
- Skewness 15
- Myopulse Percentage Rate (MYOP) 15
- Log Detector 15
- Temporal Moment (TM) 16
- Log Difference Absolute Mean Value (LDAMV) 16
- Log Teager Kaiser Energy Operator (LTKEO) 16
- Maximum Fractal Length (MFL) 16
- Modified Mean Absolute value (MMAV) 17
- Mean Absolute Deviation (MAD) 17
- v-Order (VO) 17
- Willison Amplitude (WA) 18
- Average Amplitude Change (AAC) 18
- Hjorth Mobility and complexity 18

- Difference Mean Absolute Value (DMAV) 19
- Artificial Neural Networks (ANN) 19
- Linear Discriminant Analysis (LDA) 19
- Principle Component Analysis (PCA) 19

2.7 Classification 20
- K Nearest Neighbors (kNN) 20
- Support Vector Machine (SVM) 20
- Ensemble learning algorithms 21
- Artificial Neural Networks (ANN) 21

2.8 Postprocessing 23
- Majority Vote technique 23
- Bayesian Decision Theory 23

2.10 Design of the Prosthetic Controller 24
- Structural Materials 24
- Actuators 26
- Mechanisms of motion 26
- Sensors 27

Chapter 3 – Methodology 28

Conceptual design 1 28

Conceptual design 2 29

Conceptual design 3 30

Chapter 4 - Design and Implementation 32

4.1 Pattern recognition system design 32
- Implementation using entire signals as samples 33
- Implementation of windowed signal approach for real-time classification 33

4.2 Proposed Design of the Prosthetic Controller 35
- Actuators 36
- EMG Sensors 38
- Control system 39
- Power delivery 40
- Full circuit diagram of the proposed design 41

Chapter 5 - Results and Discussion 42

Implementation using entire signals as samples 42
- K-Nearest Neighbors Classifier (kNN) 42
- Support Vector Machine Classifier (SVM) 44

- Artificial Neural Networks (ANN) – feedforward neural network 46

Implementation of windowed signal approach for real-time classification 48

- Feature Extraction Delays 48
- K-Nearest Neighbors Classifier (kNN) 48
- Support Vector Machine Classifier (SVM) 49
- Ensemble Learning 50
- Artificial Neural Networks (ANN) – feedforward neural network 51

Postprocessing 56

- Majority Voting 56
- Bayesian Decision Theory 57

Chapter 6 - Conclusion and Further development 58

Conclusion 58

Further development 58

References 59

Appendices 62

List of Figures

Figure 1 Disjoint windowing (Spiewak, et al., 2018) 12
Figure 2 Overlapped windowing (Spiewak, et al., 2018) 13
Figure 3 ZC 14
Figure 4 SSC 15
Figure 5 LTKEO 16
Figure 6 SVM Algorithm (scikit-learn, 2020) 20
Figure 7 Structure of a neural network (IBM, 2020) 21
Figure 8 Bayes theorem 23
Figure 9 Human Hand Structure (Institute for Quality and Efficiency in Health Care, 2010) 24
Figure 10 Properties of PLA (Flynt, 2017) 25
Figure 11 Properties of ABS (Flynt, 2017) 25
Figure 12 conceptual design 1 block diagram 28
Figure 13 conceptual design 2 block diagram 29
Figure 14 conceptual design 3 block diagram 30
Figure 15 Frequency Response of the bandpass filter 32
Figure 16 Pattern recognition pipeline 34
Figure 17 InMoov hand and forearm Design (Langevin, 2014) 35
Figure 18 Fishing line 36
Figure 19 MG996R Servo Motor 36
Figure 20 Interfacing PCA9685 with servo motors 37
Figure 21 AD8232 Bio-potential Amplifier with Electrodes 38
Figure 22 Interfacing AD8232 with Microcontroller development board 38
Figure 23 Interfacing a Raspberry Pi to an Arduino 39
Figure 24 300W buck converter 40
Figure 25 7.4V Li-Po battery 40
Figure 26 Interfacing Battery with Buck converter 41
Figure 27 Completed circuit diagram 41
Figure 28 kNN accuracy chart 42
Figure 29 kNN implementation with PCA 43
Figure 30 kNN confusion matrix 43
Figure 31 SVM Implementation 44
Figure 32 SVM with PCA 45
Figure 33 SVM Confusion Matrix 45
Figure 34 kNN Accuracy vs Window Length Chart 49
Figure 35 SVM Accuracy vs Window Length Chart 50
Figure 36 Ensemble Learning Accuracy vs Window Length Chart 51
Figure 37 Feedforward-ANN Accuracy vs Window Length 52
Figure 38 All Algorithms Accuracy vs Window Length Chart 53
Figure 39 All Algorithms Size vs Window Length 54
Figure 40 All Algorithms Prediction delay vs Window Length 54
Figure 41 Bandpass filter implementation 62
Figure 42 Down sampling implementation 62
Figure 43 Feature Extraction Implementation Part 1 62
Figure 44 Feature Extraction Implementation Part 2 63
Figure 45 Feature Extraction Implementation Part 3 63
Figure 46 Feature Extraction Implementation Part 4 63

Figure 47 Classification implementation in MATLAB 63
Figure 48 Classification implementation in TensorFlow Part 1 63
Figure 49 Classification implementation in TensorFlow Part 2 63
Figure 50 Classification implementation in TensorFlow Part 3 63

List of Tables

Table 1 Parameters of tendons 36
Table 2 MG996R Servo Motor Specifications 36
Table 3 Components and their operating Voltages 40
Table 4 Feedforward ANN Hyperparameters 47
Table 5 Feature Extraction Delays 48
Table 6 kNN Optimized Hyperparameters with Window Lengths 48
Table 7 SVM Optimized Hyperparameters with Window Lengths 49
Table 8 Ensemble Learning Optimized Hyperparameters with Window Lengths 50
Table 9 Feedforward ANN Optimized Hyperparameters with Window Lengths 52
Table 10 Selection of suitable classifiers with Majority voting 56

Chapter 1 – Introduction

1.1 Background of the project:

The development and advancement of bionics play a crucial role in the augmentation or restoration of the physical function of a differently-abled person. Current development heavily focuses on the area of Rehabilitation; such as those with congenital defects or amputations of the upper limbs, however, there is a lack in the functionality and dexterity of a human hand/arm. The focus of this project is to design and implement a bionic arm using a much more robust, intuitive, or biomimetic approach with the usage of EMG signal as an effective command source for the control so that there will be an increased demand for the industry of prosthetics with an effective biomimetic background.

1.2 Problem identification:

Upper-limb amputation can cause a great deal of functional impairment for patients, particularly for those with amputation at or above the elbow. Although prostheses have been used for many centuries, they are still lacking in the functionality and dexterity of a human hand/arm, this results in amputee individuals abandoning these devices, due to their diminished functional outcome. This pressurizes on the fact that the intuitiveness with regards to the control interface between her/himself and the prosthesis including the satisfaction of the amputee is essential for the success of the device.

Therefore, we can provide a solution to this by implementing an EMG signal-based Bionic arm by incorporating machine learning for pattern recognition in the EMG signals. The myoelectric platform of this may allow to intuitively, robustly control multiple degrees of freedom simultaneously and obtain arm movements (flexion/extension and pronation/supination) based on the processing of EMG signal.

Hence, we hope that the capability of this interface to communicate intended movements to the prosthesis will set the upper limit for the performance of a bionic arm.

1.3 Aim and Objectives

Aim**:**

The main aim of this project is to design and implement a pattern recognition model for prosthetic hand control using a robust, intuitive, or biomimetic approach with the usage of EMG signal as an effective command source for the control. This is so that there will be an increased demand for the industry of prosthetics, human-like robots that can be controlled by humans intuitively.

Objectives**:**

- To successfully develop a pattern recognition model for prosthetic hand control with a good amount of robustness, intuitiveness, and dexterity as that of a human upper limb.
- Develop a robust algorithm to recognize and utilize EMG signals for the specified tasks.
- Energy-efficient design

1.4 Scope and Limitations:

Scope**:**

- Data Acquisition via surface electrodes from muscles
- Preprocessing the EMG data
- Developing an algorithm to translate preprocessed EMG data to mechanical movement
- Design a robust, biomimetic robot arm
- Create a modular design that can benefit various amputees

Limitations**:**

- Lack of controlled environment for data acquisition
- The number of subjects that can be used for data acquisition is limited and the number of samples are also limited
- Lack of access to proper signal acquisition hardware.
- Financial limitations when prototyping
- Time limitations

Chapter 2 - Literature Review

2.1 Introduction

The literature review highlights Electromyography signal acquisition techniques, filtering techniques, feature extraction, techniques of signal classification using machine learning, the synergy of neuroscience with robotics, and the mechanical design of the proposed bionic arm.

2.2 Basis for using EMG signals

Voluntary movements are planned, controlled, and executed by the motor cortex. The signals originated in the motor cortex travel through the spinal cord (optional) to muscles of the affected region by motor neurons. These signals cause the muscles to contract/relax. There is a generation of electrical activity due to muscle contraction. Electromyography is a technique utilized to measure these electrical currents produced during these neuromuscular activities. (Reaz, et al., 2006). Since muscle contraction/relaxation is directly associated with voluntary movements, EMG signals are inclusive of data that are associated with voluntary movements. Bionic arms are mostly used as prosthetic devices for amputees or people born without limbs (arm). However, electric impulses can still be passed on to the limb as EMG signals can be acquired from residual muscles which can be used to control the prosthesis.

2.3 EMG signal acquisition

Surface / Gel (Skin) electrodes can be used to acquire the signal from the muscles. Signals need to be amplified via a bio-signal amplifier before sampling using an analog to digital converter. Sparkfun AD8232 bio-signal amplifier is a great choice for amplification due to its compact size and low cost. EMG data can be collected via multiple channels. More channels in the region will increase the quality of data and classification accuracy up to a point but, having multiple channels can be troublesome due to the unnecessary complexity, more points of failure, and requirement for more computational power. While most of these complications don't affect the designing process greatly, they can be impractical for the end-user. So, obtaining EMG signals from two channels provides a good compromise between complexity and accuracy. A microcontroller/development board with an analog to digital converter (at least 10-bit) can be used to sample and process the EMG data. Some suggested development boards include Arduino boards, Espressif (ESP32), and STM32 development boards, but there is no limitation.

2.4 Filtering the EMG Signal

Sampled EMG signal needs to be filtered to remove unwanted frequencies. Typical EMG signal has frequencies between 0-500Hz (LAXMI SHAW, 2012) and frequency is known to decrease with the fatigue of the muscle. When filtering, other factors such as ECG (electrocardiography) signal, powe r line noise, movement artifacts need to be considered as well. ECG signal frequencies can be between 0 to 20Hz and power line noise is 50Hz or 60Hz depending on the region. So, a bandpass filter between 20Hz and 500Hz and a notch filter at 50Hz or 60Hz need to be implemented. These bandpass and notch filters seemed like a good compromise between reducing noise, artefacts and preserving information. Infinite impulse response Butterworth filters are used to filter the signal due to their maximally flat (sharp roll-off with minimum peaking) characteristics and faster computation time.

2.5 Control configurations

There are two main control configurations to control prosthetics using EMG signals.

- Direct control

Direct control uses the magnitude of the EMG signal to execute a certain movement in the prosthetic device. Due to the stochastic nature of EMG signals, it's impossible to use raw EMG signals for this purpose. One solution is to take the mean absolute (MAV) value of predefined sample size and if the value crosses a certain threshold, the prosthetic device can be activated. Using multiple channels with different muscles, multi-gesture control can be implemented. The speed of the movement can be determined according to the MAV (proportional control). The algorithmic simplicity of direct control makes it easier to implement. Direct control is the control configuration used by most commercial prosthetics.

Whilst simpler to implement, it's not without disadvantages. Direct control can be the simplest and most robust for implementing a single movement such as 'hand open' and 'hand close'. However, as more gestures are added, the necessity for more EMG channels increases. This increases the hardware complexity, introduces more points of failure, reduction of accuracy due to muscle crosstalk, and less robustness overall (Linda Resnik, 2018).

- Pattern Recognition

Pattern recognition techniques use handcrafted or computer-generated features of the given EMG to control the prosthetic device. Due to the time series and numerical nature of the EMG signal, it's impossible to recognize quantifiable patterns by the human eye. To mitigate this issue, various feature extraction techniques (both hands crafted and using artificial neural networks) can be implemented. Then, Machine learning techniques can be implemented to classify the signals for given classes (gestures). Pattern recognition techniques are algorithmically complex and require more computational power for implementation than direct control. But, when implemented correctly, pattern recognition techniques are observed to be more robust with fewer points of failure, more resistance to muscle crosstalk, and work with multiple gestures.

Due to these reasons, this project will move forward with pattern recognition techniques instead of direct control.

2.6 Feature extraction

Signals need to be prerecorded and labelled with performed gestures and fixed time duration. A single sign al vector contains data for the whole range of motion of the relevant gesture. To implement real-time pattern recognition, it's not viable to use an entire for feature extraction because it will cause a severe delay between prosthetic device motion and the actual movement. Windowing the signal with a predefined window, small window size (50ms,100ms,150ms), and analyzing each window separately can reduce the delay significantly. Either an overlapping windowing scheme or a disjoint windowing scheme can be used for this purpose. (Englehart & Hudgins, 2003).

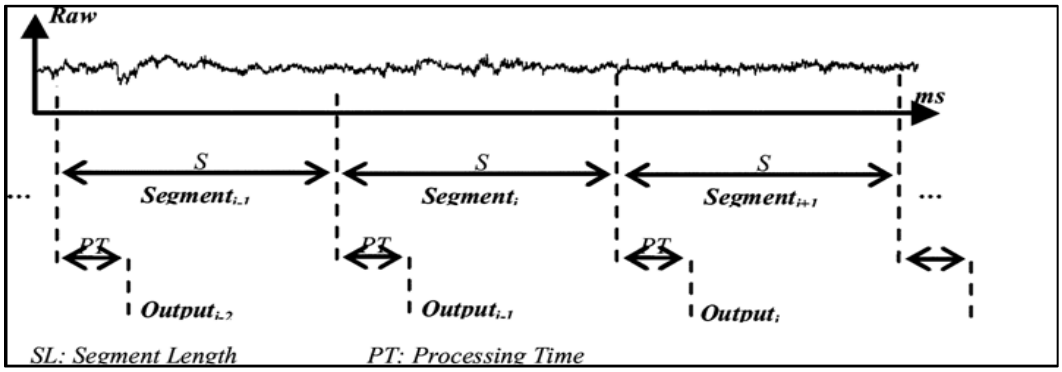

Figure 1 Disjoint windowing (Spiewak, et al., 2018)

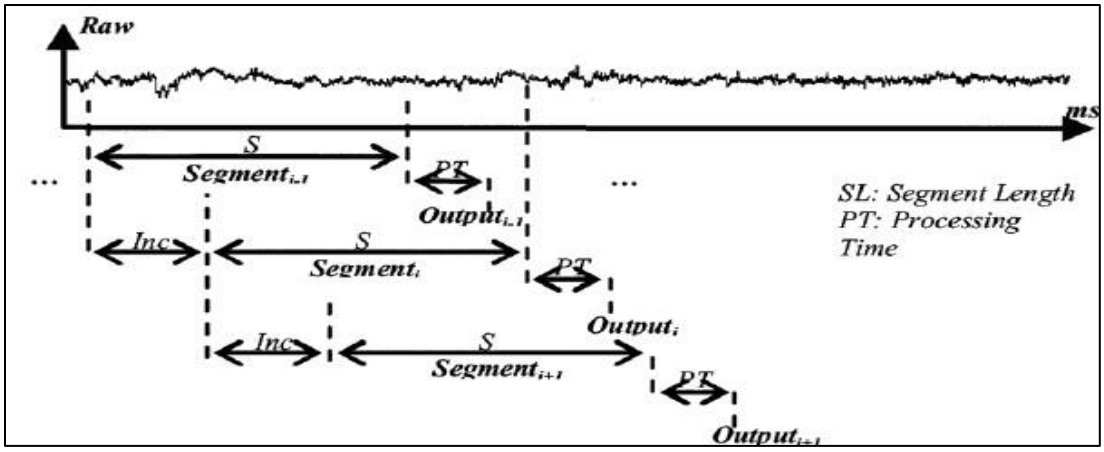

Figure 2 Overlapped windowing (Spiewak, et al., 2018)

It has been shown that overlapped windowing scheme can produce better classification accuracy than disjoint windowing, but it is also more computationally complex and can cause a delay in the process (Englehart & Hudgins, 2003). However, classification accuracy also depends on the window size, the number of training samples, and the classifier parameters itself (Rami N. Khushaba, 2012). Considering these factors, this project will focus on using a disjoint windowing scheme.

Additionally, to implement real-time classification, a threshold function like direct control (by calculating mean absolute value in a predefined window) needs to be implemented to detect active segments of the continuously fed EMG signal. If the MAV exceeds the threshold, feature extraction and further steps need to be performed. If not, the segment can be ignored and moved on to the next segment.

Typical EMG signal features can be divided into the time domain, frequency domain, and time-frequency domain. To keep the computational complexity low, this project focuses more on time-domain features.

Various feature extraction techniques are selected based on available literature and they are stated below:

- Mean Absolute Value (MAV)

MAV is the moving average of the rectified signal calculated by summing all amplitudes in the segment and dividing by segment size.

$$MAV = \frac{1}{N}\sum_{i=1}^{N}|x_i|$$

N = sample size

x_i = i^{th} sample

- Autoregressive Coefficients

Autoregressive model describes each sample of the segment as a linear combination of previous samples and an error term. It is implemented as based on (Dart & Chan, 2004).

$$y(t) = \sum_{i=1}^{p} a(i) * y(t-i) + e(t)$$

Where $a(1), a(2), a(3)$.. are coefficients, and p is the order model. $e(t)$ is the error term.

- Integrated EMG (IEMG)

IEMG is the area under the curve of the rectified EMG segment. It can be expressed as the summation of all absolute values of amplitude (Spiewak, et al., 2018).

$$IEMG = \sum_{i=1}^{N}|x_i|$$

N = sample size

x_i = i^{th} sample

- Zero crossings (ZC)

The number of times the samples of the EMG segment change the sign (crosses zero levels) is calculated. A threshold value for the minimum amplitude is required to minimize the noise-induced errors. (Hudgins, et al., 1993)

$$\{x_k > 0 \text{ and } x_{k+1} < 0\} \text{ or } \{x_k < 0 \text{ and } x_{k+1} > 0\}) \text{ and } |x_k - x_{k+1}| \geq \varepsilon$$

Figure 3 ZC

- Slope Sign Changes (SSC)

The number of times the samples of EMG segment changes sign the slope (derivative) is calculated. A threshold value for a minimum amount of change is required to minimize noise-induced errors (Hudgins, et al., 1993).

$$\{x_k > x_{k-1} \text{ and } x_k > x_{k+1}\} \text{ or } \{x_k < x_{k-1} \text{ and } x_k < x_{k+1}\}$$
$$\text{and } |x_k - x_{k+1}| \geq \varepsilon \text{ or } |x_k - x_{k-1}| \geq \varepsilon$$

Figure 4 SSC

- Skewness

Skewness measures asymmetry/distortion of a distribution. When amplitudes of samples of EMG segment are represented as a distribution, skewness can be measured.

$$skewness = \frac{\sum_i^N (X_i - X')^3}{(N-1) * \sigma^3}$$

Where σ is the standard deviation and N is the number of variables, and X' is the mean of the distribution.

- Myopulse Percentage Rate (MYOP)

MYOP is defined as the mean absolute value of the EMG segment but with a threshold value for the amplitude of each sample (Phinyomark, et al., 2012).

$$MYOP = \frac{1}{N}\sum_{i=1}^{N} if\ (|x_i| > threshold)$$

N = sample size

x_i = i^{th} sample

- Log Detector

Provides an estimate of the exerted muscle force. This is a non-linear feature (Tkach & Huang, 2010).

$$logDetector = e^{\frac{1}{N}\sum_{i=1}^{N} \log(|x_i|)}$$

N = sample size

x_i = i^{th} sample

- Temporal Moment (TM)

The temporal moment is a statistical analysis technique that can be used as a feature. Since the first and second order of TM is MAV and Variance, 3rd order and above is taken (Phinyomark, et al., 2012).

$$TM = |\frac{1}{N}\sum_{i=1}^{N} x_i^{order}|$$

N = sample size

x_i = i^{th} sample

- Log Difference Absolute Mean Value (LDAMV)

According to (Phinyomark, et al., 2014),

$$\text{LDAMV} = \log\frac{\sum_{i=1}^{N-1}|x_{i+1} - x_i|}{N}$$

N = sample size

x_i = i^{th} sample

- Log Teager Kaiser Energy Operator (LTKEO)

Teager Kaiser Energy operator measures instantaneous changes in energy of the signals composed of a single time-varying frequency (Khushaba, et al., 2017). Low computational time and predictive power make it suitable for use in embedded applications.

$$f_7 = \log(\Psi) = \log\left(\sum_{j=0}^{N-2} x^2[j] - x[j-1]x[j+1]\right)$$

N = sample size

x_i = i^{th} sample

Figure 5 LTKEO

- Maximum Fractal Length (MFL)

MFL measures the strength of muscle contraction even though it is very similar to Wavelength. MFL includes a logarithmic scale, which makes it less vulnerable to noise (Arjunan & Kumar, 2010).

$$MFL = \log\sqrt{\sum_{i=1}^{N-1}(x_{i+1} - x_i)^2}$$

N = sample size

x_i = i^{th} sample

- Modified Mean Absolute value (MMAV)

MMAV is an expansion of MAV with a continuous weighted window function (w_i) (Phinyomark, et al., 2012)

$$MAV = \frac{1}{N}\sum_{i=1}^{N} w_i |x_i|$$

N = sample size

x_i = i^{th} sample

Where,

$$w_i \begin{cases} 1, if\ 0.25N \leq i \leq 0.75N \\ \frac{4i}{N}, elseif\ i < 0.25N \\ \frac{4(i-N)}{N}, otherwise \end{cases}$$

- Mean Absolute Deviation (MAD)

The mean absolute deviation measures the average distance between each sample of the EMG segment and the mean (Verma & Gupta , 2020).

$$MAD = \frac{1}{N}\sum_{i=1}^{N} |x_i - x'|$$

N = sample size

x_i = i^{th} sample

x' = mean

- v-Order (VO)

v-Order implicitly estimates the force of muscle contraction. It is a non-linear function (Phinyomark, et al., 2012).

$$VO = \left(\frac{1}{N}\sum_{i=1}^{N} x_i^{order}\right)^{\frac{1}{order}}$$

N = sample size

x_i = i^{th} sample

- Willison Amplitude (WA)

Willison amplitude measures the frequency information in the EMG segment. It calculates the number of times a difference between two adjacent amplitudes exceeds a certain threshold. It is found to be related to muscle contraction force and firing of motor unit action potentials (Phinyomark, et al., 2012).

$$WA = \sum_{i=1}^{N} [f(|x_i - x_{i+1}|)]$$

N = sample size

x_i = i^{th} sample

$$f(x) = \begin{cases} 1, if\ x \geq threshold \\ 0, otherwise \end{cases}$$

- Average Amplitude Change (AAC)

(Phinyomark, et al., 2012)

$$AAC = \frac{1}{N} \sum_{i=1}^{N} |x_{i+1} - x_i|$$

N = sample size

x_i = i^{th} sample

- Hjorth Mobility and complexity

Hjorth parameters are statistical indicators used in time domain signal properties, it was introduced by Bo Hjorth (1970).

Hjorth Mobility represents the mean frequency of the frequency spectrum of the EMG segment. It is defined as the square root of the variance of the first derivative divided by the variance of the signal.

$$Mobility = \sqrt{\frac{var\left(\frac{dy(t)}{dt}\right)}{var(y(t))}}$$

Hjorth complexity represents the change in frequency

$$Complexity = \frac{Mobility\left(\frac{dy(t)}{dt}\right)}{Mobility(y(t))}$$

- Difference Mean Absolute Value (DMAV)

(Too, et al., 2019)

$$DMAV = \frac{1}{N-1}\sum_{i=1}^{N} |x_{i+1} - x_i|$$

- Artificial Neural Networks (ANN)

Supervised artificial neural networks can be used for both feature extraction and classification. ANN contains input layers, hidden layers, and an output layer. Increasing the number of hidden layers can improve accuracy at the cost of computation (training and classification) time.

The main advantage of ANN is the auto-identification of relevant features. However, those features may not mean anything to humans. There are several disadvantages of using ANN for this project. These include heavily increased training (computation) time, inability to intuitively understand features, the requirement of a large amount of space to store lookup tables, which is an inherent limitation of microcontrollers, and increased processing time.

- Linear Discriminant Analysis (LDA)

Linear Discriminant analysis can be used as a dimensionality reduction technique before classification. LDA can help mitigate the 'curse of dimensionality'. LDA finds a linear combination of features that explain the data best.

- Principle Component Analysis (PCA)

Principle component analysis can also be used to reduce the dimensionality of the feature set while minimizing information loss by creating less correlated features that maximize variance. PCA is an unsupervised technique that doesn't consider class labels, while LDA makes assumptions about class distribution.

2.7 Classification

After feature extraction, the discovery of a suitable classifier to classify the signal to pre-defined classes is needed. Luckily, there are a plethora of supervised machine learning algorithms to choose from. Some of the proven algorithms for EMG signal classification are:

- K Nearest Neighbors (kNN)

kNN is a non-parametric supervised learning algorithm that uses a majority vote of k number of nearest neighbors of a data point for classification. kNN relies on various distance metrics to assign weights of the contribution of each neighbor. kNN algorithm tends to be rather fast while training and slow during classification because it does not generate a model of the data. Instead, it uses training data directly when classification is requested (Taunk, et al., 2019).

- Support Vector Machine (SVM)

SVM classification algorithms classify labelled data according to a hyperplane in an N-dimensional hyperplane while N being the number of features. SVM can be used with linear classification as well as non-linear classification by using appropriate kernel functions (scikit-learn, 2020).

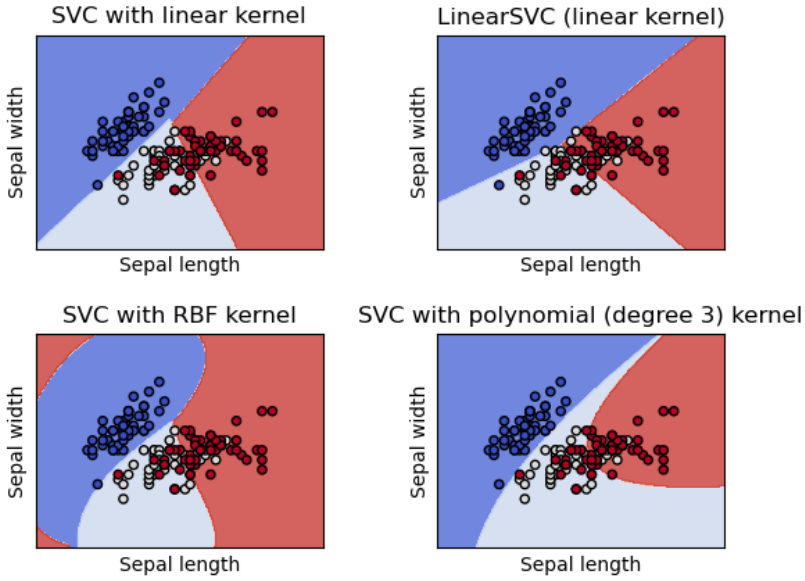

Figure 6 SVM Algorithm (scikit-learn, 2020)

- Ensemble learning algorithms

Ensemble learning methods combine multiple learning algorithms (learners) to achieve higher performance than using a single learning algorithm (Srivastava, 2015). Some common ensemble methods are,

i. Bagging – Using similar learners on a sample population and calculating the average of all predictions
ii. Boosting – Adjusting weights of observations according to the last classification.
iii. Stacking – Combining outputs from different learning models.

- Artificial Neural Networks (ANN)

Artificial neural networks try to mimic the signaling functions of biological neurons. ANNs are consist of an input layer, at least one hidden layer, and an output layer. Each layer has a predetermined set of nodes (neurons) that have associated weights and thresholds. Thresholds determine the activations of the relevant nodes. If activated, nodes send data to the next layer of the neural network (IBM, 2020). Like classical machine learning algorithms, ANNs also rely on training data to improve its' accuracy.

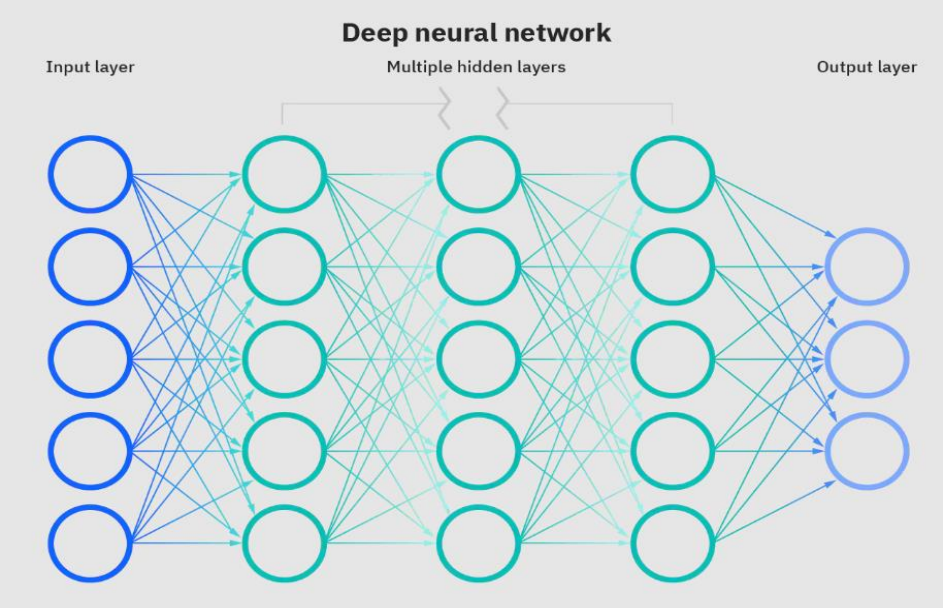

Figure 7 Structure of a neural network (IBM, 2020)

Feedforward neural networks are also called multi-layered perceptrons (MLPs). MLPs only transfers data in the forward direction. There are no loops, cycles, or feedback paths in the network. Instead, it relies on backpropagation with various optimization techniques to calculate the loss and train the algorithm. Feedforward neural networks were implemented on the dataset with various parameters and 5-fold cross-validation.

To find the most suitable algorithm and optimum algorithm parameters, repetitive training, and testing with different algorithms, testing different parameters is needed. Hyperparameter optimization techniques such as grid search or random search can be used to find the optimum parameters. One of the main problems that can arise while training is ‘overfitting’. Overfitting is when the classifier model fits exactly to the training data. This can happen when random fluctuations and noise is picked up by the algorithm. To reduce overfitting affecting results, validation techniques such as holdout validation and k-fold cross-validation can be implemented.

2.8 Postprocessing

Since the algorithm is going to be designed to take segments of EMG signal as input and continuous classification (if the MAV threshold value is reached), there is a tendency to overwhelm the prosthetic device with varying decisions and misclassifications. Therefore, postprocessing/smoothing techniques need to be applied to eliminate this issue. Techniques that are going to be experimented with are:

- Majority Vote technique

The majority simply means 'more than half. The Majority vote takes a predetermined amount of analysis windows and their decisions into consideration to give the smoothed decision which is the class with the highest number of occurrences. To make the Majority vote technique successful, at least 3 analysis window periods need to be considered. The analysis window is composed of data segment acquisition time, processing time, and the prosthetic device actuation delay. Processing time and actuation delay are yet to determined. One of the main disadvantages of the majority vote technique is treating every output decision in the classifier in a naïve manner. Increasing the number of analysis windows considered will result in better accuracy at the cost of increased delay (Rami N. Khushaba, 2012).

- Bayesian Decision Theory

Bayesian statistical methods are based on Bayes' theorem for calculating and updating probabilities after acquiring new data. Bayes' theorem explains the probability of an event based on prior knowledge of circumstances that could be related to the event.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Figure 8 Bayes theorem

Bayesian decision theory assumes statistical independence between each analyzing window in the ideal case. As the analyzing windows are disjoint, the assumption is not violated severely. With enough analysis windows and decisions, the resulting smoothed decision can be determined according to the previous decisions by the classifier. Additional weighting factors and normalization need to be included to favor the most recent decision output by the classifier.

2.10 Design of the Prosthetic Controller

The human hand is a highly complex system that contains over 30 intrinsic and extrinsic muscles with 27 bones which results in approximately 25 degrees of freedom. The human hand can perform complex and dexterous tasks with precision and force. (Institute for Quality and Efficiency in Health Care, 2010).

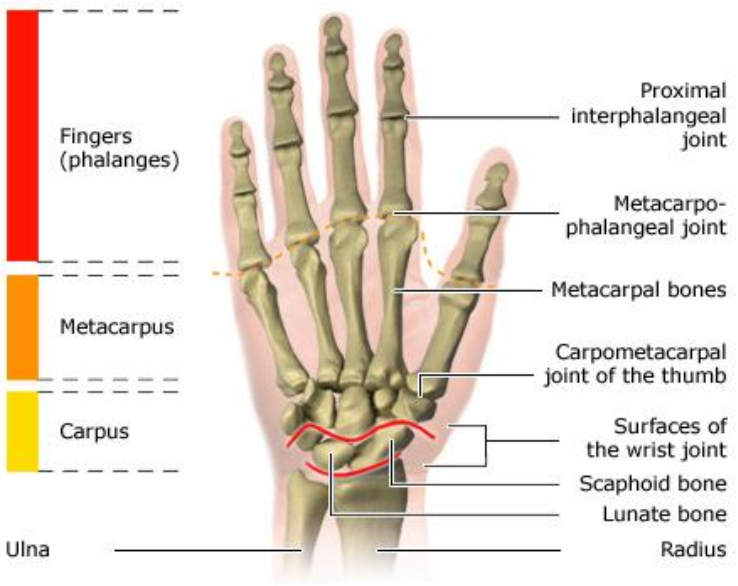

Figure 9 Human Hand Structure (Institute for Quality and Efficiency in Health Care, 2010)

Even though it is extremely challenging to incorporate most of the abilities of the human arm into an artificial bionic arm, it is not the limiting factor of functionality. The limiting factor lies in the data acquisition, classification, and overall algorithmic part of the process. Due to this reason, this project focuses on creating a structurally and mechanically robust bionic arm rather than achieving the highest degrees of freedom possible.

- Structural Materials

  Mechanical design is can be modelled using Computer-Aided Design software and manufactured with a Fused deposition modeling (FDM) printer. Using 3D printing for prototyping and manufacturing would significantly decrease the costs. For early prototyping purposes, Polylactic acid (PLA) thermoplastic monomer derivatives can be considered, due to their versatility and ease of printing (Wikipedia, 2021). However, for the final prototype, Acrylonitrile butadiene styrene (ABS) thermoplastic derivatives are preferred.

The PLA has a lower glass transition temperature, lower strength and as a whole tends to be more fragile than ABS. Mostly it is the heat instability that stands as the main disadvantage for real-world applications. ABS derivatives are much stronger, more flexible, durable, and heat resistant compared to PLA plastics. However, it can be a challenge to print, as ABS materials are prone to warping, shrinking, and cracking in the cooling phase (Omnexus, 2021).

Property	Value
Full Name	Polylactic acid (PLA)
Melting Point	150 to 160 °C (302 to 320 °F)
Glass Transition	60-65 °C
Injection Mold Temperature	178 to 240 °C (353 to 464 °F)
Density	1.210–1.430 g·cm−3
Chemical Formula	(C3H4O2)n
Crystallinity	37%
Tensile Modulus	2.7–16 GPa
Solublility	Chlorinated solvents, hot benzene, tetrahydrofuran, and dioxane (not water soluable).

Figure 10 Properties of PLA (Flynt, 2017)

Property	Value
Full Name	Acrylonitrile Butadiene Styrene (ABS)
Melting Point	Amorphous (no true melting point). Commonly 230°C
Glass Transition	105 °C (221 °F)
Injection Mold Temperature	204 to 238 °C (400 to 460 °F)
Density	1.060–1.080 g·cm−3
Chemical Formula	(C8H8·C4H6·C3H3N)n
Tensile Modulus	310,000 PSI
Solublility	Esters, ketones, ethylene dichloride and acetone (not water soluable).

Figure 11 Properties of ABS (Flynt, 2017)

- Actuators

Electromechanical actuators of the Bionic arm serve a similar purpose to muscles in human arms. Muscles can perform both precise and forceful movements with ease. Although, attaining the same level of dexterity from electromechanical actuators is challenging; a satisfactory trade-off between speed and torque of the motions needs to be created. Some of the actuators are;

i. Servo Motors

Servo motors are very precise and efficient due to the in-built feedback mechanism which can output angular position, velocity, and acceleration. Servo motors tend to be bulkier than most other types of motors and require special instructions to work.

ii. Geared Motors

Geared motors are regular AC/DC motors with a gearbox that can generate a specific amount of torque required. Regular Geared motors do not have a feedback mechanism and implementing a direct feedback mechanism can be complicated. Geared motors are smaller than servo counterparts and have a straightforward working principle.

iii. Linear Actuators

Linear actuators convert rotation motion into linear push-pull motion and are available with or without feedback mechanisms. Linear actuators that can fit in a bionic arm are rare and tend to be expensive.

- Mechanisms of motion

i. Tendon Driven Mechanisms

Tendon-driven mechanisms are inspired by the bio-mechanics of the human hand. It uses threads to control multiple joints simultaneously by pulling the thread with an actuator. Servo motors and linear actuators are most suited for this process. Tendon-driven mechanisms allow control of multiple joints with a single actuator and allow an actuator to be placed remotely. This can be beneficial for finger movement due to the limited space inside the palm of the prosthetic hand. But, tendon-driven mechanisms tend to provide less power and are less robust than geared mechanisms. Excessive force on the threads can cause breakage, which leads to the malfunction of the device. If implemented properly, tendon drive mechanisms can be a great choice for the control of individual fingers.

ii. Geared Mechanisms

Geared mechanisms use actuators meshed/coupled with gears to obtain the required motion. Actuators need to be positioned at a specific joint, this requires more engineering and simulations than tendon-driven mechanisms for achieving perfection in the motion. Geared mechanisms are more robust and provide more torque than tendon-driven counterparts. Geared mechanisms are more suitable for wrist, elbow, and shoulder joints where high torque is required throughout.

- Sensors

Additionally, pressure, force sensors can be utilized to create a feedback mechanism that can be used for fine control of the movements and grasps.

Chapter 3 – Methodology

This section describes the stages and design decisions that can be taken during the design process of the bionic arm. The main stages of the control process are,

I. EMG signal acquisition
II. Signal processing
III. Mechanical output from the bionic arm

Additionally, labelled data acquisition, training the algorithm with labelled data needs to be done before the deployment of the algorithm.

Conceptual design 1

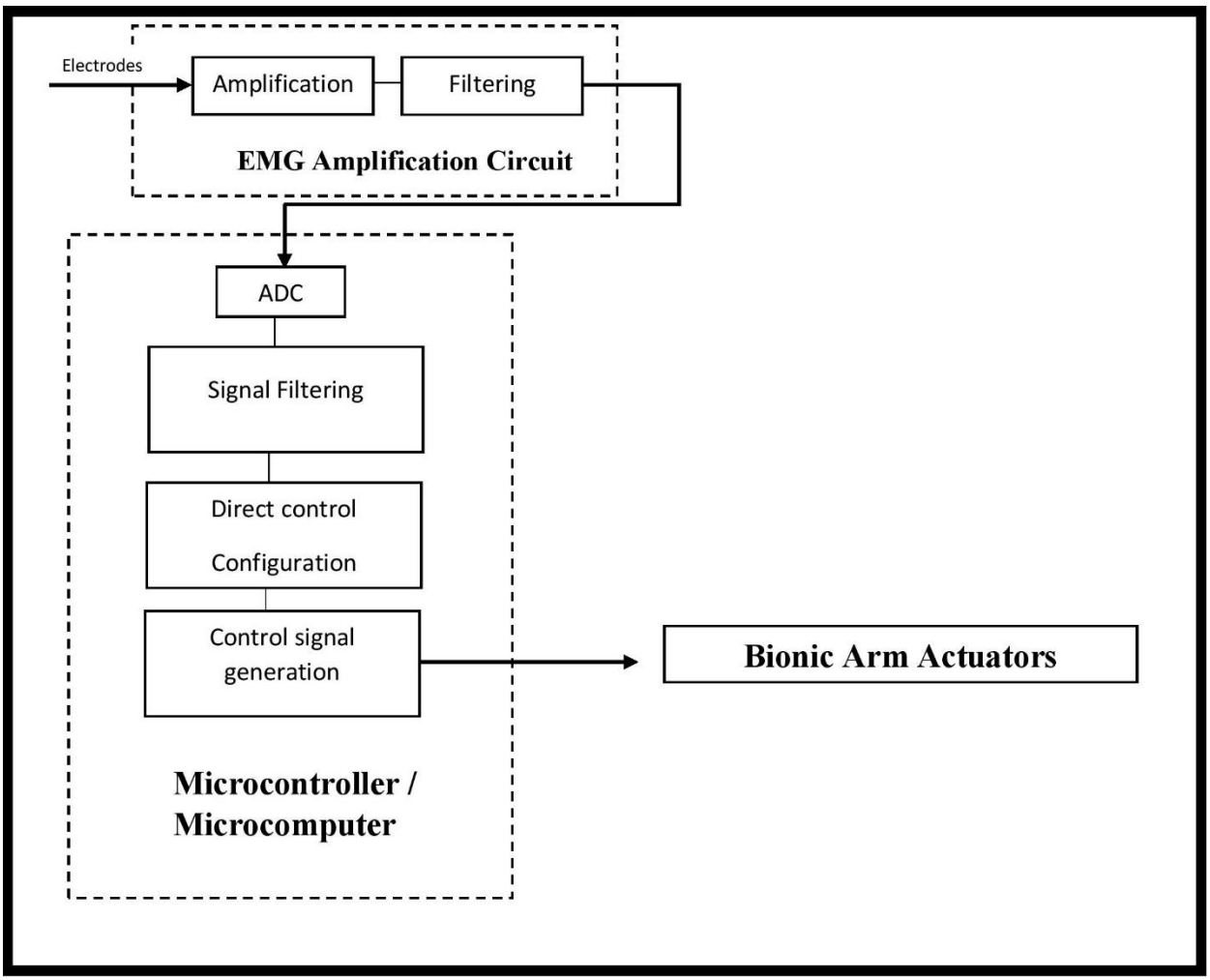

Figure 12 conceptual design 1 block diagram

The first conceptual design focuses on the utilization of direct control configuration to control the myoelectric prosthesis, where the magnitude of the EMG signal executes a certain movement in the prosthetic device. However, the stochastic nature of EMG signals limits the usage of raw EMG signals. Although there is a simplicity in terms of implementation, the disadvantage plays a dominant role such as the addition of more gestures, increase in requirement for more EMG channels. Thus, leading to a complexity in the hardware, decrease in accuracy and overall there is a hindrance in robustness.

Conceptual design 2

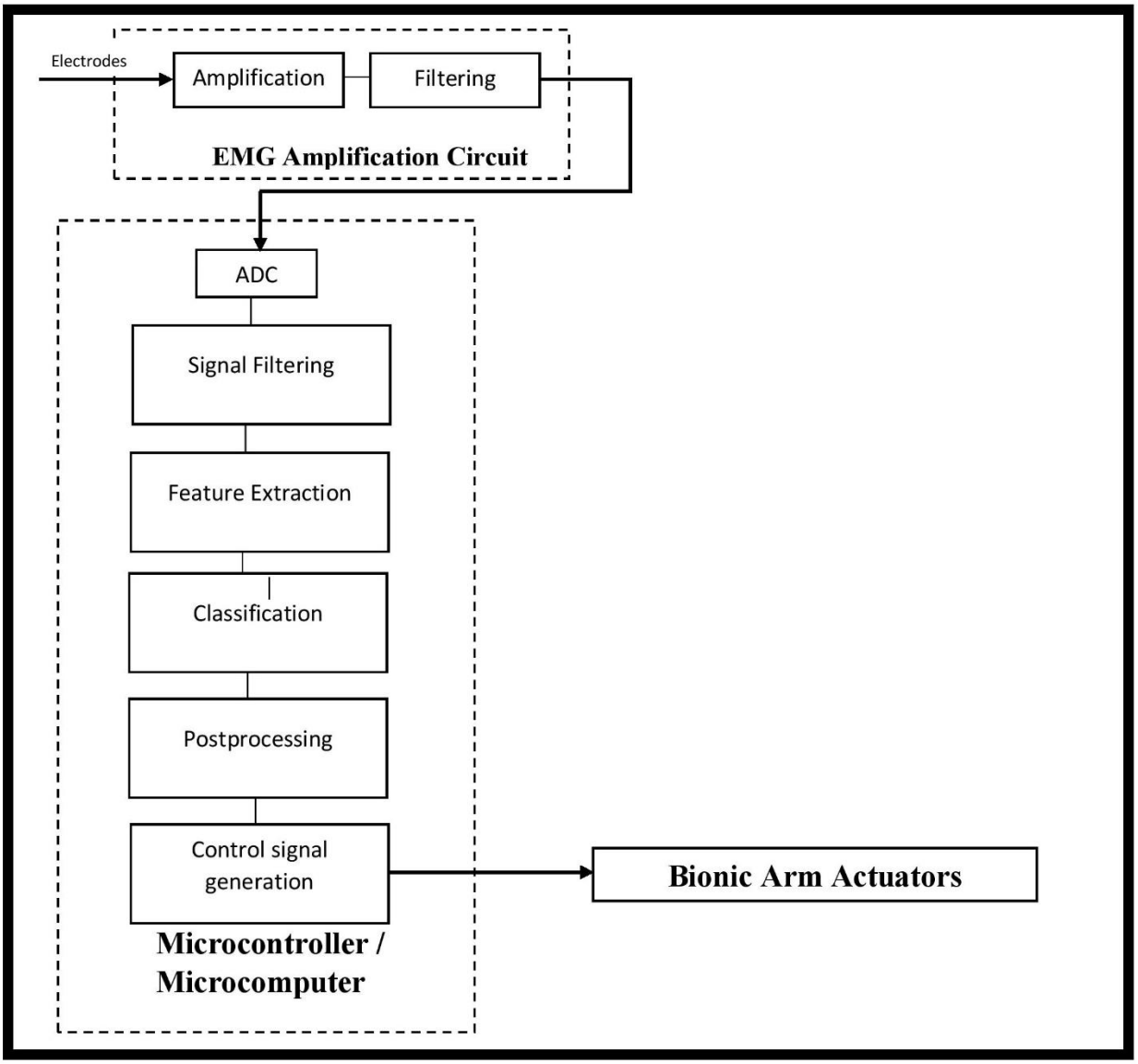

Figure 13 conceptual design 2 block diagram

The second conceptual design focuses on the utilization of Pattern recognition for controlling the myoelectric prosthesis; where computer-generated features of the given EMG take over the control of the prosthetic device. There is an algorithmic complexity in the Pattern recognition technique and computational power plays a key role in implementation when compared to direct control. However, accurate implementation of this technique, provides greater robustness, more accuracy due to the decrease in points of failure, greater resistance to muscle crosstalk, and functions with multiple gestures.

Conceptual design 3

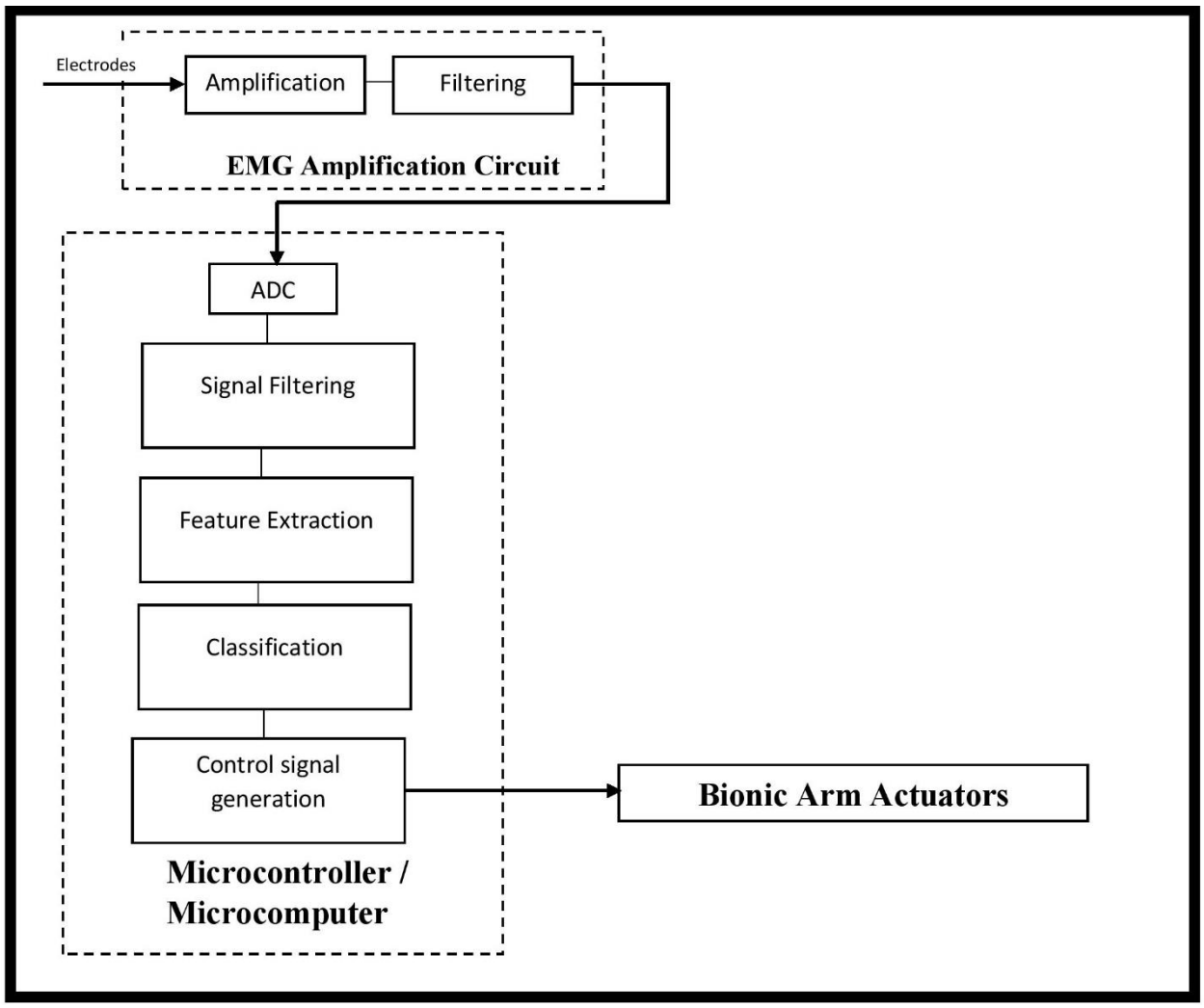

Figure 14 conceptual design 3 block diagram

The third conceptual design also focuses on the utilization of Pattern recognition for controlling the myoelectric prosthesis. However, postprocessing is not performed on the generated prediction. Since additional delays imposed by post-processing procedures are not available, this strategy allows for longer window lengths which can lead to increased accuracy per prediction. The downsides of this approach are increased chance of overwhelming the prosthetic controller with varying commands, susceptibility to more misclassifications, etc. Due to these concerns, Conceptual design 3 was not used in this project.

- **Verdict**

Due to its higher effectiveness/accuracy and fewer downsides, conceptual design 2 was chosen to be implemented in the project after reviewing all three conceptual designs.

Chapter 4 - Design and Implementation

4.1 Pattern recognition system design

A publicly available and gesture labelled electromyography dataset (Rami N. Khushaba, 2012) was used to design the system. The dataset contained 600 signals with two channels which are classified into ten classes that correspond to an individual or combined finger movements. Classes are,

i. Hand close
ii. Thumb finger flexion
iii. Middle finger flexion
iv. Index finger flexion
v. Little finger flexion
vi. Ring finger flexion
vii. Thumb and index finger flexion
viii. Thumb and middle finger flexion
ix. Thumb and ring finger flexion
x. Thumb and little finger flexion

Then, EMG signals were filtered between 20Hz and 500Hz using a 4th order Butterworth bandpass filter. 60Hz notch filter was implemented to remove the power line interference.

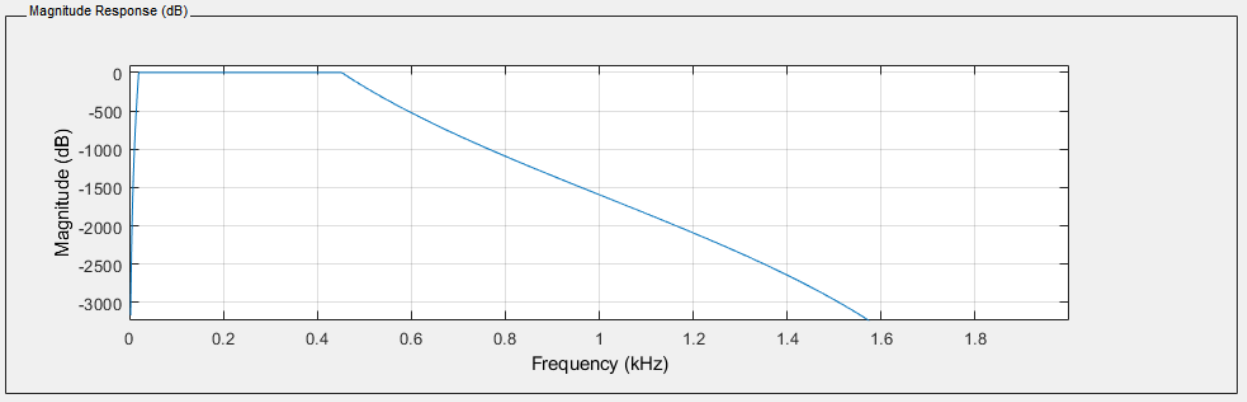

Figure 15 Frequency Response of the bandpass filter

To minimize computational costs, signals are down-sampled from 4000Hz to 1000Hz. The down-sampling process is done after filtering to avoid aliasing artefacts.

- Implementation using entire signals as samples

Each signal is with a duration of 5 seconds with a sampling rate of 1000Hz which corresponds to 5000 data points per signal. These signals were used as samples for feature extraction and classification to acquire a baseline accuracy level.

Then, 58 features are calculated using previously mentioned (Chapter 3) feature extraction techniques. These features are normalized and used with the following classification techniques,

i. k-Nearest-Neighbor (kNN)
ii. Support Vector Machines (SVM)
iii. Ensemble Learning methods
iv. Feedforward Artificial Neural Networks

Then, the effectiveness and accuracy level of each classifier was recorded for further analysis.

- Implementation of windowed signal approach for real-time classification

Each 5-second signal was windowed with a pre-defined set of window sizes using the disjoint windowing scheme. Then, training samples were labeled appropriately to generate the new dataset. The number of training samples can be calculated by the following equation,

$$No.of\ training\ samples = \frac{signal\ length - (signal\ length\ \%\ window\ length)}{window\ length} * 600$$

Chosen window lengths were 50, 100, 150, 200, 250, and 300 milliseconds. Separate datasets were created for each window length.

Then, 58 features are calculated using previously mentioned (Chapter 3) feature extraction techniques, and feature extraction delays including filtering and resampling delays were measured for each window size using a computer with an Intel Core i5 8265u 1.6GHz processor and 8GB random access memory. Prediction delays were measured for 100 separate samples, each measured 5 times, and the mean value was taken to minimize random errors. Because of the change in computation platform and interference caused by other background processes, these measurements may not represent real-time feature extraction delays when implemented in a standalone prosthetic device, but they can provide a relative understanding of how feature extraction delays can change with sample window size.

Then, these features are normalized and used with several classification techniques to discover the effectiveness and accuracy level of each classifier with the corresponding window size. Accuracy levels are measured with 5-fold cross-validation. The following block diagram indicates the pipeline of this method.

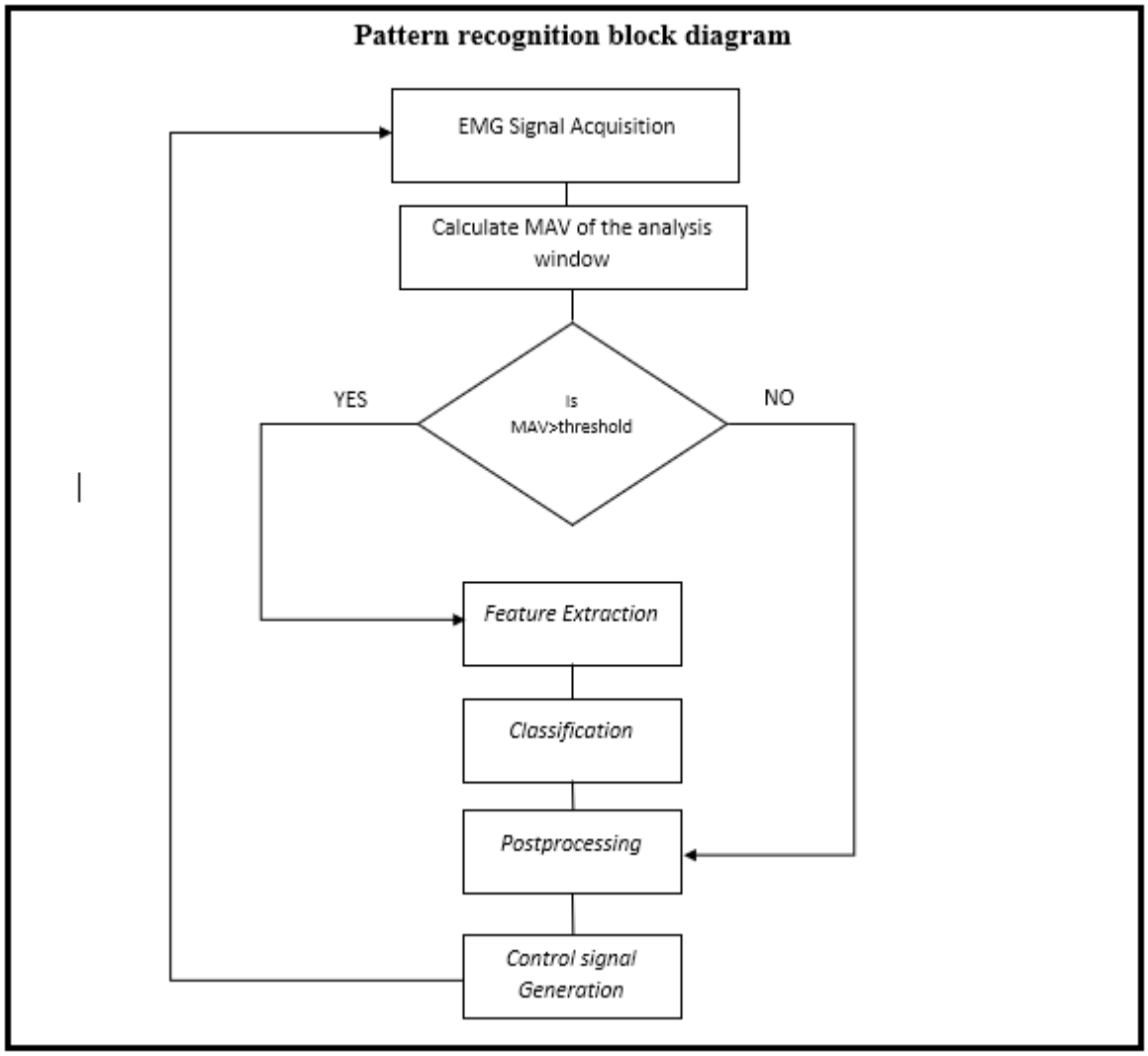

Figure 16 Pattern recognition pipeline

4.2 Proposed Design of the Prosthetic Controller

The prosthetic controller design was based on InMoov open-source 3D printed robot's hand and forearm by Gael Langevin (2014). The design uses a tendon-driven mechanism to control finger movements while using a geared mechanism to control wrist rotation. InMoov forearm allows sufficient degrees of freedom for the preset gestures and intuitive mechanical design.

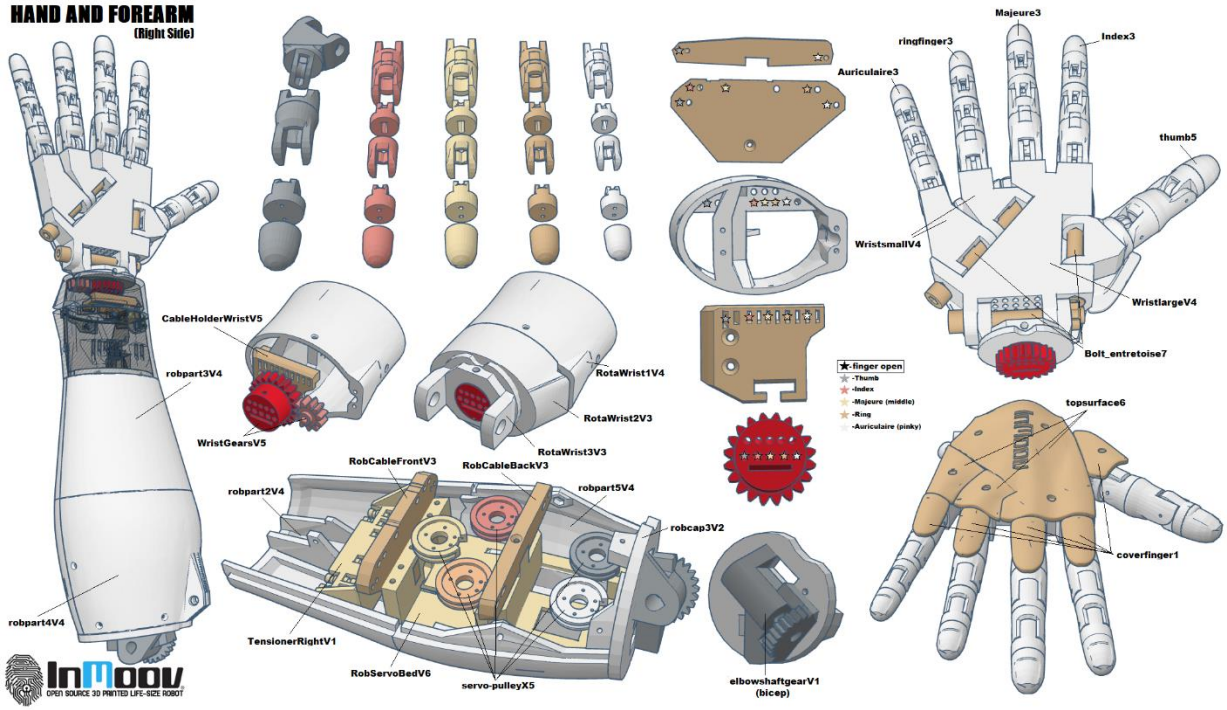

Figure 17 InMoov hand and forearm Design (Langevin, 2014)

The design allows 6 degrees of freedom including independent finger movement and wrist rotation. Mechanical and structural components were manufactured using an FDM 3D printer from a combination of ABS and PLA plastics. Because of its high strength, thermal stability, and toughness, ABS plastic was used to print structural components. However, due to the risk of warping during printing, ABS was not used to print mechanical components such as gears. PLA was used to print mechanical components.

Tendon-driven mechanisms use threads (wires) as tendons to control joints with actuators, and those tendons must be able to sustain high tension and resist deformation when force is applied. As a result, fishing lines were chosen as tendons. Characteristics of the chosen fishing line are as follows,

Table 1 Parameters of tendons

Material	High-Density Polyethylene
Line type	Braided (4 Strands)
Thickness	0.5mm
Test	80lb

Figure 18 Fishing line

- Actuators

To control the tendon-driven mechanism 5 digital servo motors were used. Another servo motor was used to control the wrist rotation. Chosen model for all servo motors is the Tower Pro MG996R Metal Gear servo with the following specifications.

Table 2 MG996R Servo Motor Specifications

Operation voltage	4.8V – 6V
Weight	56 grams
Speed	0.13 seconds / 60 degrees (6V)
Stall torque	11kg/cm (6V)
Gear types	Metal
Rotation	360º
Stall current	2.5A(6V)

Figure 19 MG996R Servo Motor

The MG996R can deliver adequate torque while remaining light and compact enough to fit inside the bionic arm. Additionally, the use of metal gears in MG996R motors allows increased resistance to wear and tear ensuring prolonged lifespan. All servos were controlled using a 12-bit 16 channel

PWM / Servo Driver PCA9685 (adafruit, 2015) which connects to the microcontroller development board via an I2C (Inter-Integrated Circuit) interface. Even though the direct connection of all 6 motors to the development board is possible, connecting via a servo driver increases upgradability (addition of more actuators) and streamlines the power delivery system for each motor. The following circuit diagram shows the implementation of the PCA9685 servo driver

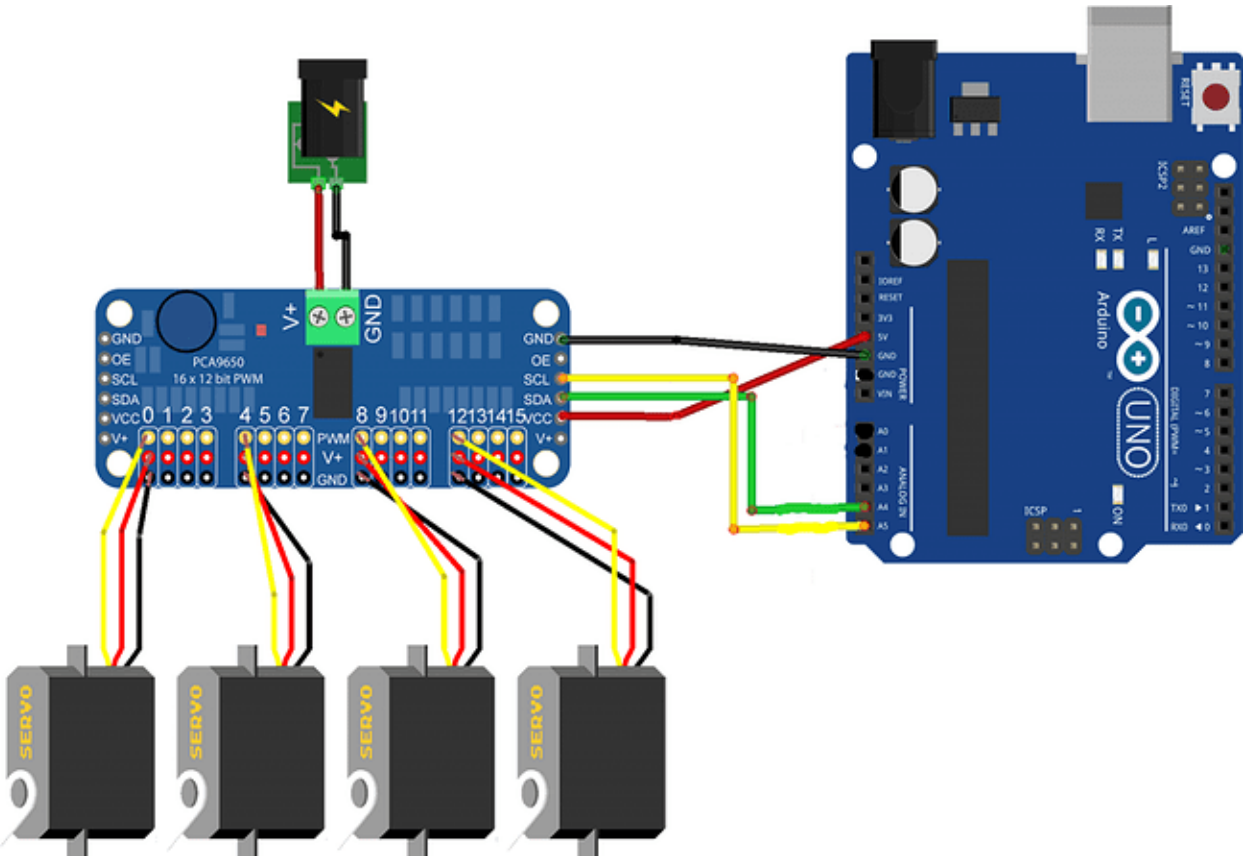

Figure 20 Interfacing PCA9685 with servo motors

and servo motors with a microcontroller development board (Arduino Uno).

- EMG Sensors

For EMG signal acquisition, 2 AD8232 single lead bio-signal monitors with gel electrodes were used. AD8232 bio-signal amplifier is used to collect, amplify and filter bio-signals. Compact design and the ability to be powered from the microcontroller development board itself makes it the ideal choice. To acquire EMG signals from two channels, two amplifiers are necessary.

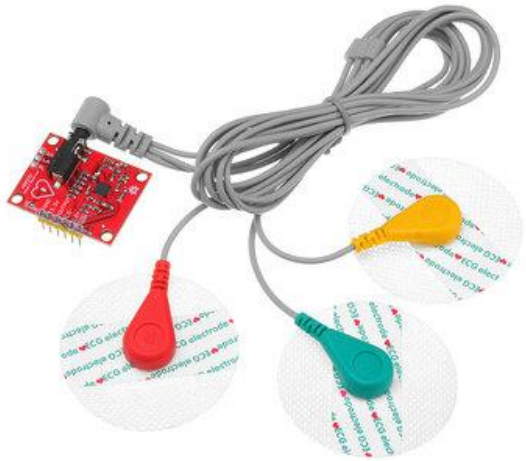

Figure 21 AD8232 Bio-potential Amplifier with Electrodes

The outputs of the amplifiers are interfaced with the microcontroller development board as below,

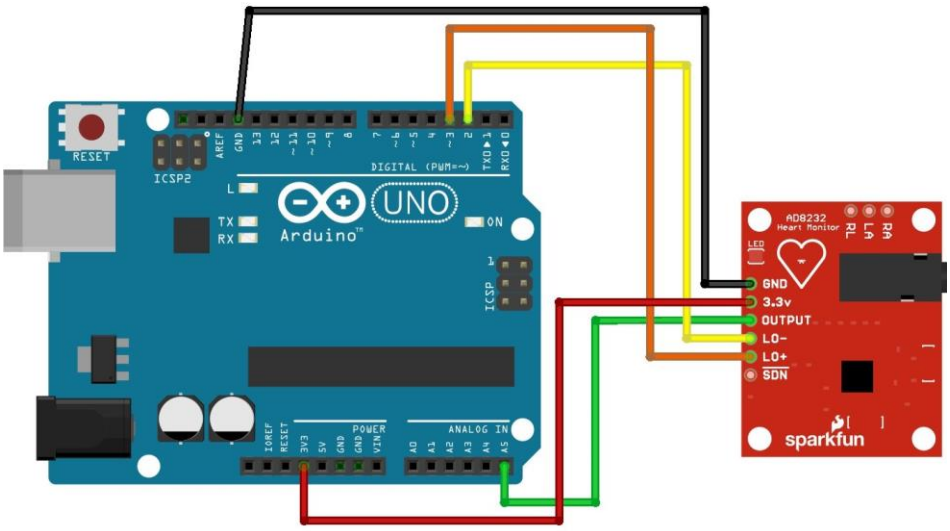

Figure 22 Interfacing AD8232 with Microcontroller development board

- Control system

A combination of microcontroller-microcomputer (Raspberry Pi – Arduino) platforms was used to implement the control system. This method allows the combining of computing power / wireless capabilities of Raspberry Pi with more versatile input/output capabilities of a dedicated microcontroller / Arduino. Raspberry Pi will host the intelligence component (preprocessing, classification) while Arduino will control the inputs (signal acquisition) and outputs (motor control).

Using the USART serial communication protocol, Arduino and Raspberry Pi will communicate with each other to exchange information (EMG signals, control commands) via USB ports on the Arduino and Raspberry Pi. The circuit diagram for interfacing a Raspberry Pi to an Arduino is shown below.

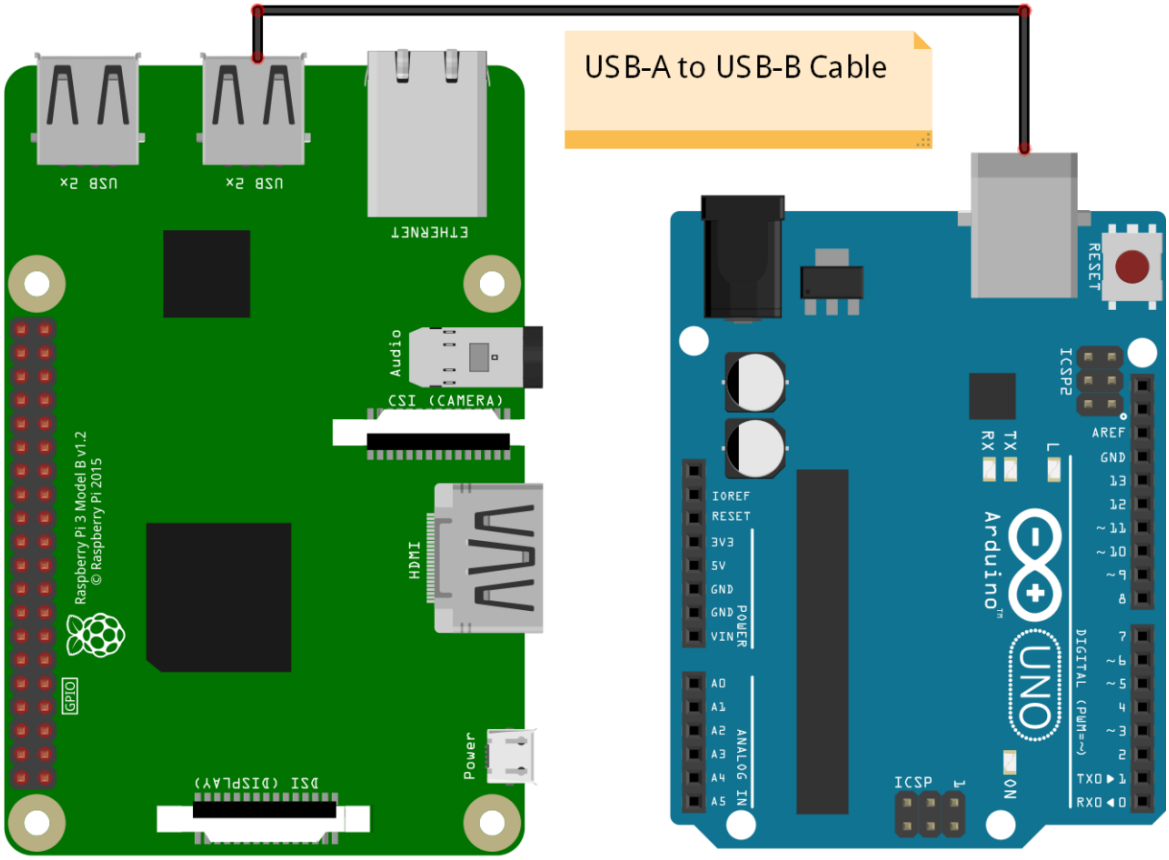

Figure 23 Interfacing a Raspberry Pi to an Arduino

- Power delivery

The operating voltage ranges of the components are displayed in the table below.

Table 3 Components and their operating Voltages

Component	Voltage
Arduino Uno	5V (6V – 20V via barrel jack)
Raspberry Pi	5V
MG996R Servo Motor	4.8V – 6V
AD8232 Bio-potential amplifier	3.3V (via Arduino)

Here, Arduino Uno is powered through the USB cable connected to the Raspberry Pi. EMG amplifiers are powered via the 3.3V output of the Arduino. Servo motors and Raspberry Pi require power delivery directly from a power source. Even though Servo motors can handle voltages up to 6V, Raspberry Pi's maximum input voltage is 5V (4.75-5.25). So, it was decided to operate servo motors at 5V to reduce unnecessary complexity from additional power converters. For the prosthetic arm to be portable, the whole system was powered via a 7.4V 1500mAh rechargeable lithium-polymer battery. Battery voltage was controlled to 5V using a high-power (300W) DC-DC adjustable buck converter. Buck converter enables the precise control of voltage which is required by sensitive electronic components in the system.

Figure 24 300W buck converter

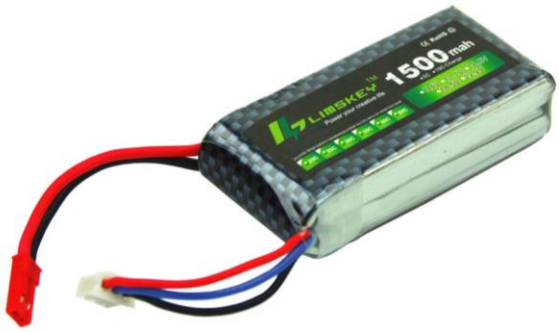

Figure 25 7.4V Li-Po battery

Following diagram shows the connection of battery with the step – down (buck) converter.

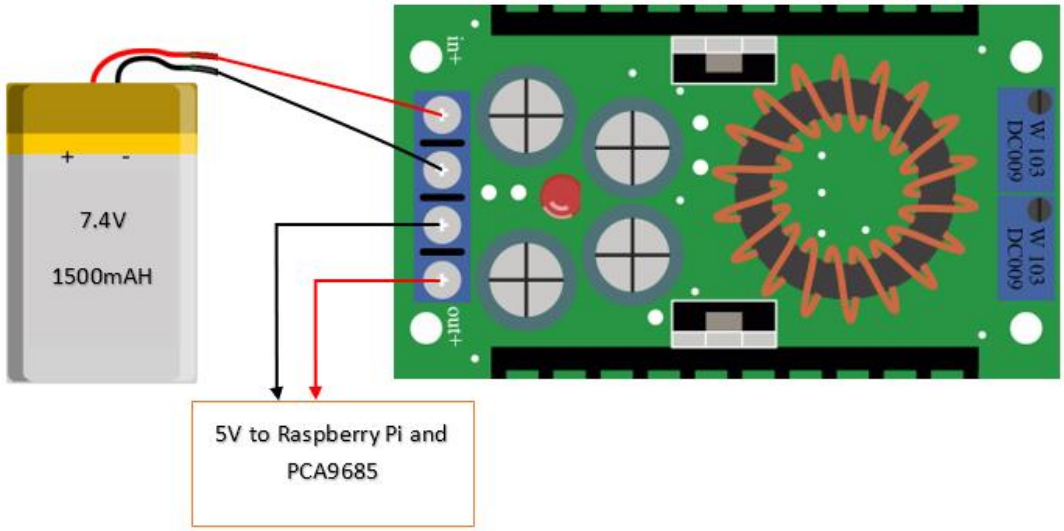

Figure 26 Interfacing Battery with Buck converter

- Full circuit diagram of the proposed design

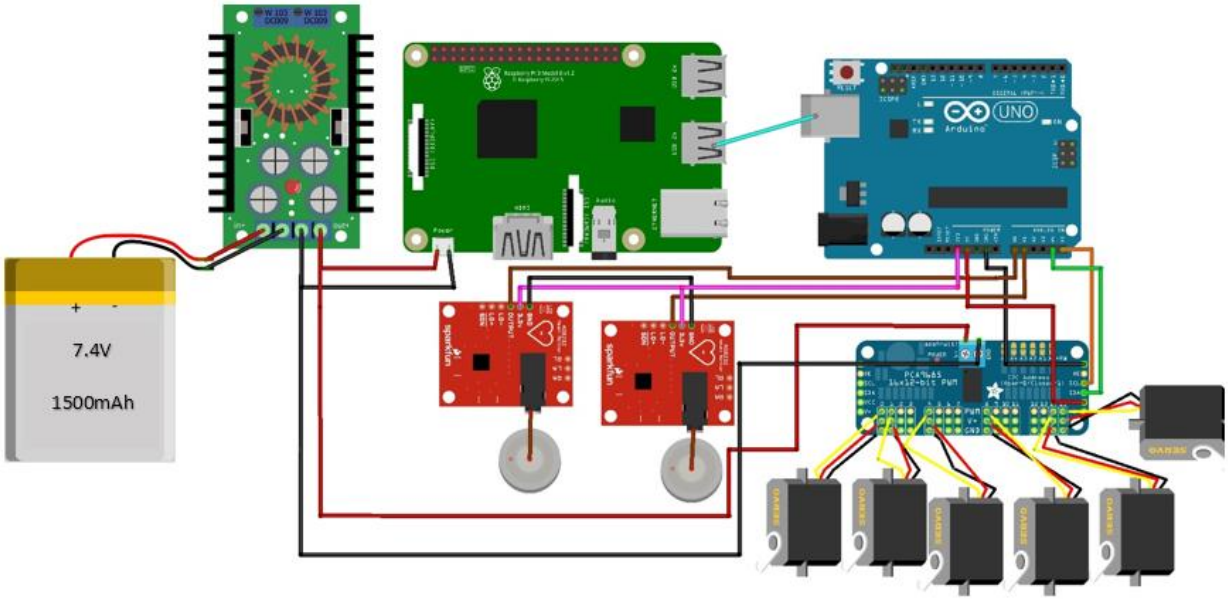

Figure 27 Completed circuit diagram

Chapter 5 - Results and Discussion

Implementation using entire signals as samples

- K-Nearest Neighbors Classifier (kNN)

kNN algorithm was implemented with various parameters and trained with 5-fold cross-validation to obtain the accuracy values as below.

1 ☆ KNN Last change: 'Distance weight' = 'Inverse'	Accuracy: 82.3% 58/58 features
2 ☆ KNN Last change: Medium KNN	Accuracy: 51.2% 58/58 features
3 ☆ KNN Last change: Coarse KNN	Accuracy: 33.0% 58/58 features
4.1 ☆ KNN Last change: Fine KNN	Accuracy: **83.0%** 58/58 features
4.2 ☆ KNN Last change: Medium KNN	Accuracy: 51.2% 58/58 features
4.3 ☆ KNN Last change: Coarse KNN	Accuracy: 33.0% 58/58 features
4.4 ☆ KNN Last change: Cosine KNN	Accuracy: 51.2% 58/58 features
4.5 ☆ KNN Last change: Cubic KNN	Accuracy: 52.3% 58/58 features
4.6 ☆ KNN Last change: Weighted KNN	Accuracy: 77.8% 58/58 features
5 ☆ KNN Last change: Optimizable KNN	Accuracy: 81.0% 58/58 features

Figure 28 kNN accuracy chart

The highest accuracy can be observed when classifier parameters are set to,

i. Number of neighbor/s – 1
ii. Distance metric – Euclidean
iii. Distance weight – Equal

After selecting the optimal parameters, the classifier was trained again with extracted features from principal component analysis. PCA was implemented several times to keep the various number of numeric features at several classification sessions.

9 ☆ KNN Last change: PCA keeping 12 numeric components	Accuracy: 81.8% 12/58 features (PCA on)
10 ☆ KNN Last change: PCA keeping 15 numeric components	Accuracy: 84.5% 15/58 features (PCA on)
11 ☆ KNN Last change: PCA keeping 20 numeric components	Accuracy: 83.3% 20/58 features (PCA on)
12 ☆ KNN Last change: PCA keeping 30 numeric components	Accuracy: 84.8% 30/58 features (PCA on)
13 ☆ KNN Last change: PCA keeping 57 numeric components	Accuracy: 84.7% 57/58 features (PCA on)
14 ☆ KNN Last change: PCA keeping 5 numeric components	Accuracy: 75.8% 5/58 features (PCA on)
15 ☆ KNN Last change: PCA keeping 10 numeric components	Accuracy: 82.2% 10/58 features (PCA on)
16 ☆ KNN Last change: PCA keeping 9 numeric components	Accuracy: **85.2%** 9/58 features (PCA on)
17 ☆ KNN Last change: PCA keeping 8 numeric components	Accuracy: **85.2%** 8/58 features (PCA on)
18 ☆ KNN Last change: PCA keeping 9 numeric components	Accuracy: **85.2%** 9/58 features (PCA on)

Figure 29 kNN implementation with PCA

The highest classification accuracy was obtained when PCA was used to extract 9 numeric features. A confusion matrix was obtained for the training session with the highest accuracy.

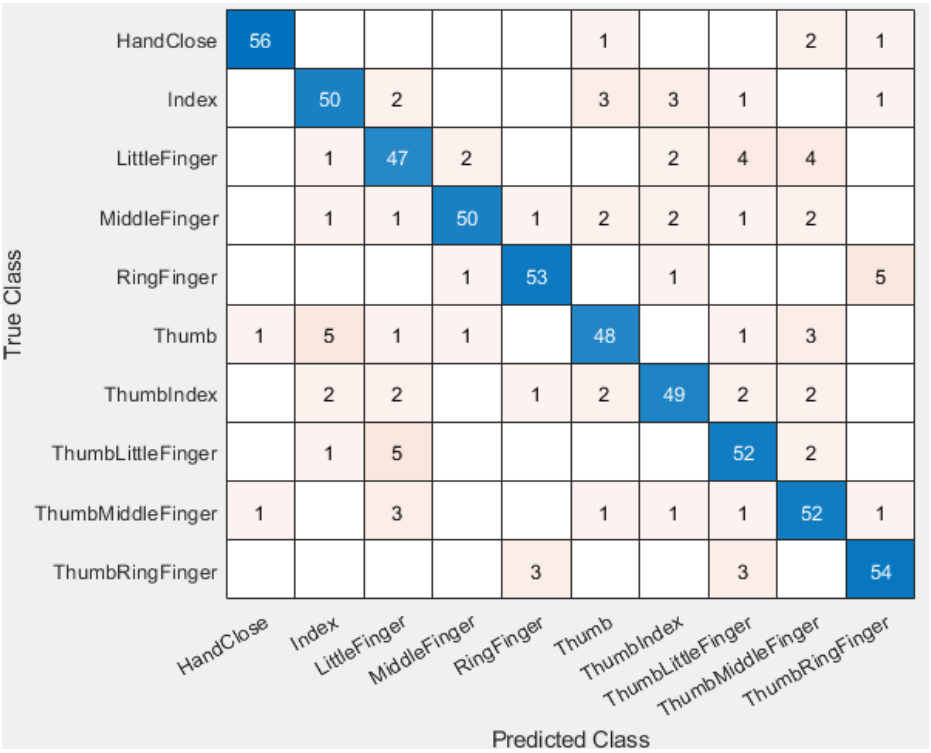

Figure 30 kNN confusion matrix

- Support Vector Machine Classifier (SVM)

SVM algorithm was implemented on the dataset with various parameters and 5-fold cross-validation.

Figure 31 SVM Implementation

The highest accuracy can be observed when classifier parameters are set to,

i. Kernel function - cubic
ii. Kernel scale mode - auto
iii. Box constraint level / Highest penalty – 1
iv. Multi-class method – one vs one

After selecting the optimal parameters, the classifier was trained again with extracted features from principal component analysis. PCA was implemented several times to keep various numbers of numeric features at several classification sessions.

7 ☆ SVM Last change: PCA keeping 9 numeric components		Accuracy: 79.2% 9/58 features (PCA on)
8 ☆ SVM Last change: PCA keeping 10 numeric components		Accuracy: 78.8% 10/58 features (PCA on)
9 ☆ SVM Last change: PCA keeping 12 numeric components		Accuracy: 74.2% 12/58 features (PCA on)
10 ☆ SVM Last change: PCA keeping 8 numeric components		Accuracy: **79.7%** 8/58 features (PCA on)
11 ☆ SVM Last change: PCA keeping 7 numeric components		Accuracy: 76.8% 7/58 features (PCA on)
12 ☆ SVM Last change: PCA keeping 5 numeric components		Accuracy: 68.3% 5/58 features (PCA on)
13 ☆ SVM Last change: PCA keeping 15 numeric components		Accuracy: 76.2% 15/58 features (PCA on)
14 ☆ SVM Last change: PCA keeping 20 numeric components		Accuracy: 77.3% 20/58 features (PCA on)
15 ☆ SVM Last change: PCA keeping 25 numeric components		Accuracy: 77.5% 25/58 features (PCA on)
16 ☆ SVM Last change: PCA keeping 11 numeric components		Accuracy: 76.0% 11/58 features (PCA on)

Figure 32 SVM with PCA

Marginal classification accuracy improvement can be observed when the classifier is trained with 8 PCA-generated numeric features.

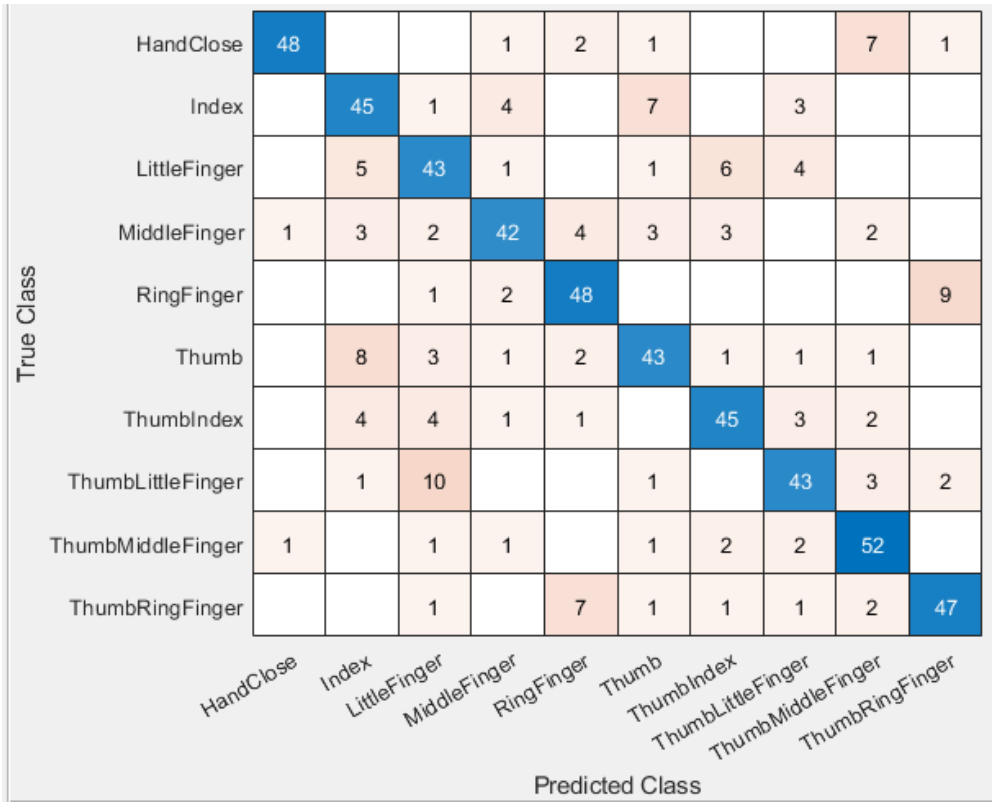

Figure 33 SVM Confusion Matrix

Other classical classification algorithms such as Decision tree algorithms, Naïve Bayes, and ensemble learning were also implemented on the dataset. Almost all of them failed to generate satisfactory accuracy levels with the exception being ensemble learning algorithm with an accuracy of 76 % with the following parameters,

i. Ensemble method – bagged trees
ii. Learner type - decision tree
iii. Maximum number of splits – 599
iv. Number of learners – 30

- Artificial Neural Networks (ANN) – feedforward neural network

ANNs were implemented on the TensorFlow machine learning platform with Keras deep-learning library. The network consisted of an input layer, output layer, and 5 hidden layers. Dropout with a 0.1 probability of output retention was implemented on first 4 layers as a regularization parameter to reduce overfitting. Any value greater or smaller than 0.1 was shown to reduce the accuracy levels of the algorithm. Input layer and hidden layers consisted of 58 neurons each. Determining the number of neurons for each layer was done via trial and error. Other optimized parameters include,

i. Activation function – Rectified linear unit (ReLU)
ii. Activation function (output layer) – SoftMax (normalized exponential function)
iii. Optimizer – Adam (stochastic gradient decent)
iv. Loss function – Categorical cross entropy
v. Epochs (cycles through dataset) – 1000

The compiled algorithm was trained various batch sizes and threshold values to get the following results

Table 4 Feedforward ANN Hyperparameters

Batch size	**Threshold**	**Accuracy**
1	0.2	86%
10	0.15	86%
100	0.1	87%
200	0.05	88.6%
200	0.2	86%

Application of PCA greatly reduced classifier performance hence not considered useful. With these results, it may be concluded that the optimized feedforward ANN classifier achieves the highest accuracy level of 88.6% while the optimized kNN classifier is at second place with 85.7% accuracy while optimized SVM achieved 79.7 % accuracy. But these results are not useful for real-time classification because the delay period between muscle contraction and classification is unacceptable.

Implementation of windowed signal approach for real-time classification

- Feature Extraction Delays

Table 5 Feature Extraction Delays

Window Length	**50ms**	**100ms**	**150ms**	**200ms**	**250ms**	**300ms**	**Full signal**
Delay	261ms	234ms	236ms	240ms	250ms	242ms	232ms

According to these measurements, feature extraction delays have not changed by a significant amount with window length. Even though the feature extraction delay was unchanged by window lengths in this situation, no conclusions can be taken because numerous uncontrolled variables can alter measured data.

- K-Nearest Neighbors Classifier (kNN)

kNN classifier was implemented with various parameters on each dataset. Optimized parameters are displayed below.

Table 6 kNN Optimized Hyperparameters with Window Lengths

Window Length	**Number of Neighbors**	**Distance Metric**	**Distance weight**
50ms	100 (Coarse)	Euclidean	Equal
100ms	10 (Weighted)	Euclidean	Squared Inverse
150ms	10 (Weighted)	Euclidean	Squared Inverse
200ms	10 (Weighted)	Euclidean	Squared Inverse
250ms	10 (Weighted)	Euclidean	Squared Inverse
300ms	10 (Weighted)	Euclidean	Squared Inverse

Classifier accuracies for each window length were calculated and plotted below.

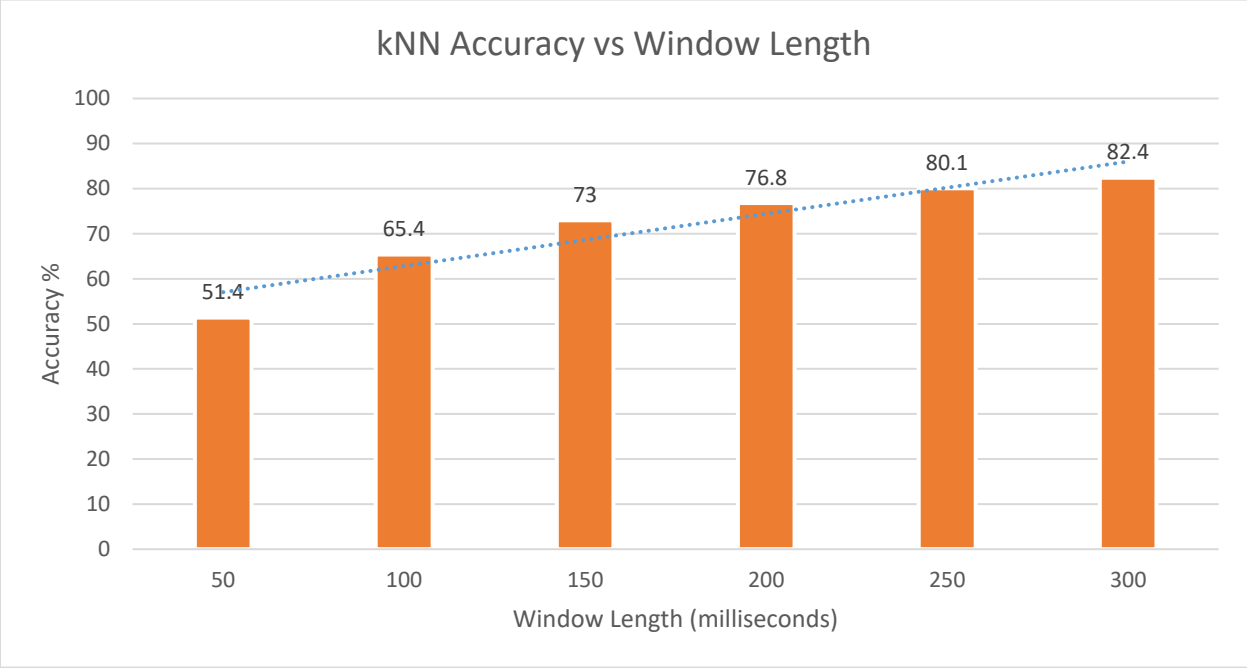

Figure 34 kNN Accuracy vs Window Length Chart

- Support Vector Machine Classifier (SVM)

The SVM classifier was implemented with various parameters on each dataset. Optimized parameters are displayed below. Kernel scale mode was set to auto.

Table 7 SVM Optimized Hyperparameters with Window Lengths

Window Length	**Kernel method**	**Box constraint level**	**Multi-class method**
50ms	Cubic	1	one vs one
100ms	Cubic	1	one vs one
150ms	Cubic	1	one vs one
200ms	Cubic	1	one vs one
250ms	Cubic	1	one vs one
300ms	Cubic	1	one vs one

Classifier accuracies for each window length were calculated and plotted below.

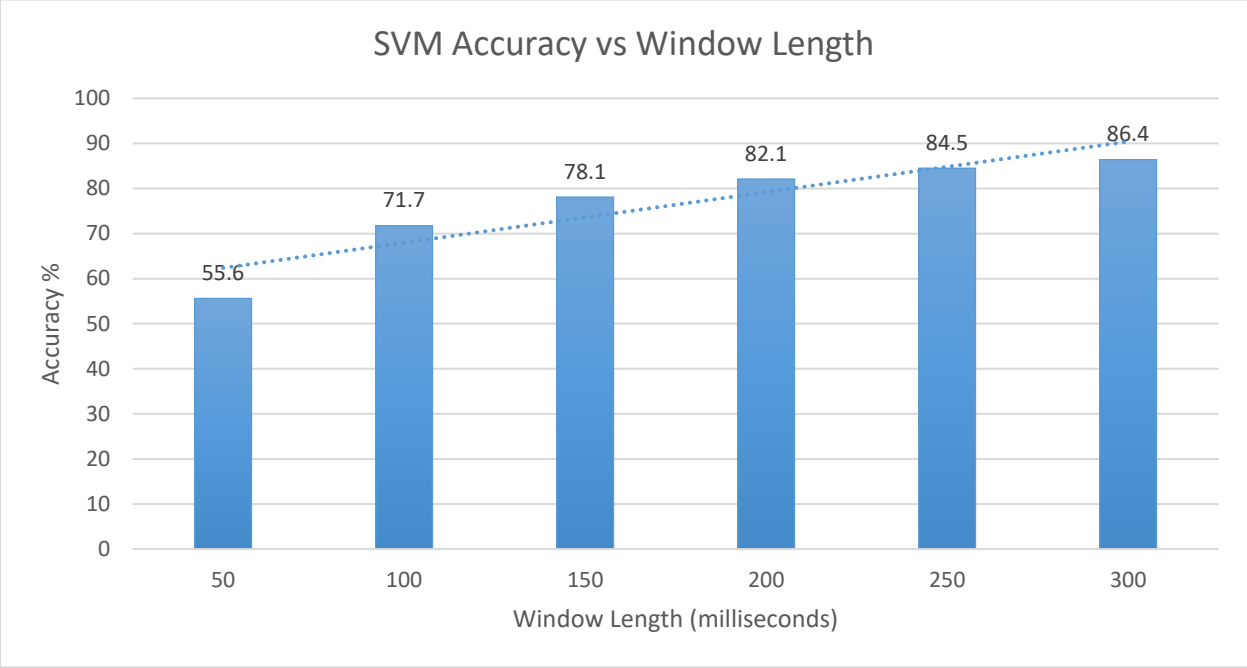

Figure 35 SVM Accuracy vs Window Length Chart

- Ensemble Learning

An ensemble learning algorithm was implemented with various parameters on each dataset. Optimized parameters are displayed below. Kernel scale mode was set to auto.

Table 8 Ensemble Learning Optimized Hyperparameters with Window Lengths

Window Length	**Ensemble method**	**Learner type**	**Max number of splits**	**Number of Learners**
50ms	Bagged trees	Decision tree	59,999	30
100ms	Bagged trees	Decision tree	29,999	30
150ms	Bagged trees	Decision tree	19,799	30
200ms	Bagged trees	Decision tree	14,999	30
250ms	Bagged trees	Decision tree	11,999	30
300ms	Bagged trees	Decision tree	9599	30

Classifier accuracies for each window length were calculated and plotted below.

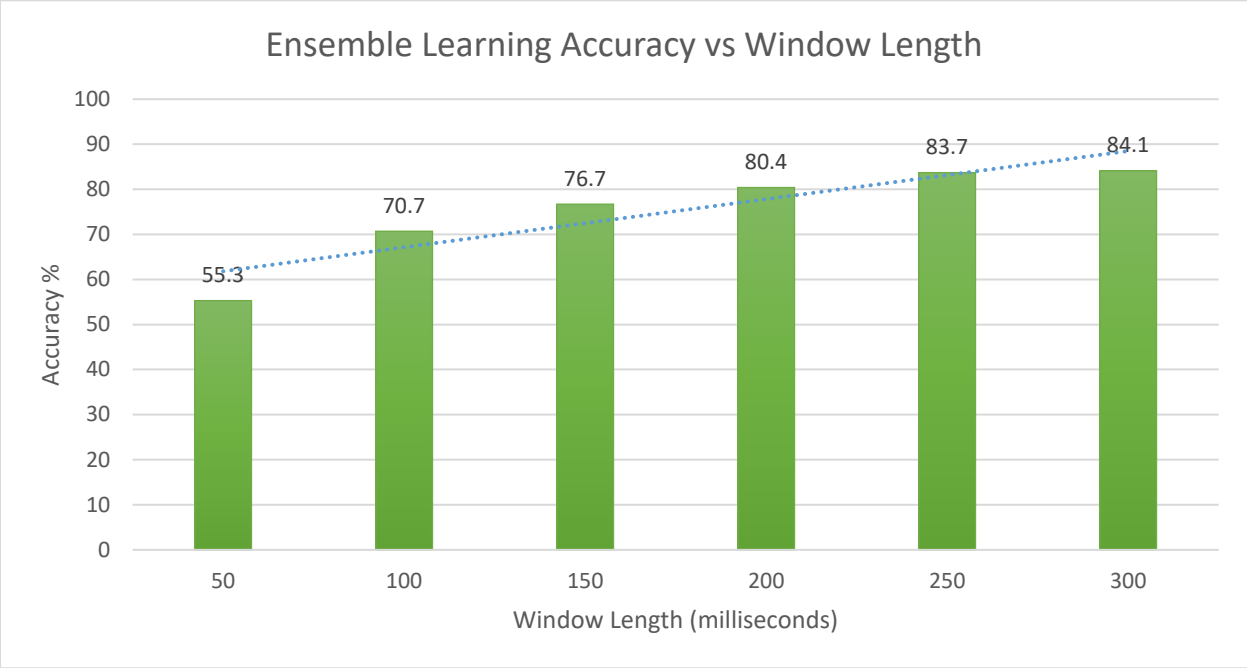

Figure 36 Ensemble Learning Accuracy vs Window Length Chart

- Artificial Neural Networks (ANN) – feedforward neural network

Feedforward ANN was implemented with similar architecture to the previous implementation with the following optimized parameters,

i. Activation function – Rectified linear unit (ReLU)
ii. Activation function (output layer) – SoftMax (normalized exponential function)
iii. Optimizer – Adam (stochastic gradient descent)
iv. Loss function – Categorical cross-entropy

Other parameters are optimized for each window length.

Table 9 Feedforward ANN Optimized Hyperparameters with Window Lengths

Window Length	Epochs	Batch size	Threshold value
50ms	300	1000	0.3
100ms	300	1000	0.35
150ms	300	1000	0.35
200ms	1000	1000	0.35
250ms	300	200	0.4
300ms	1000	1000	0.35

Classifier accuracies for each window length were calculated and plotted below.

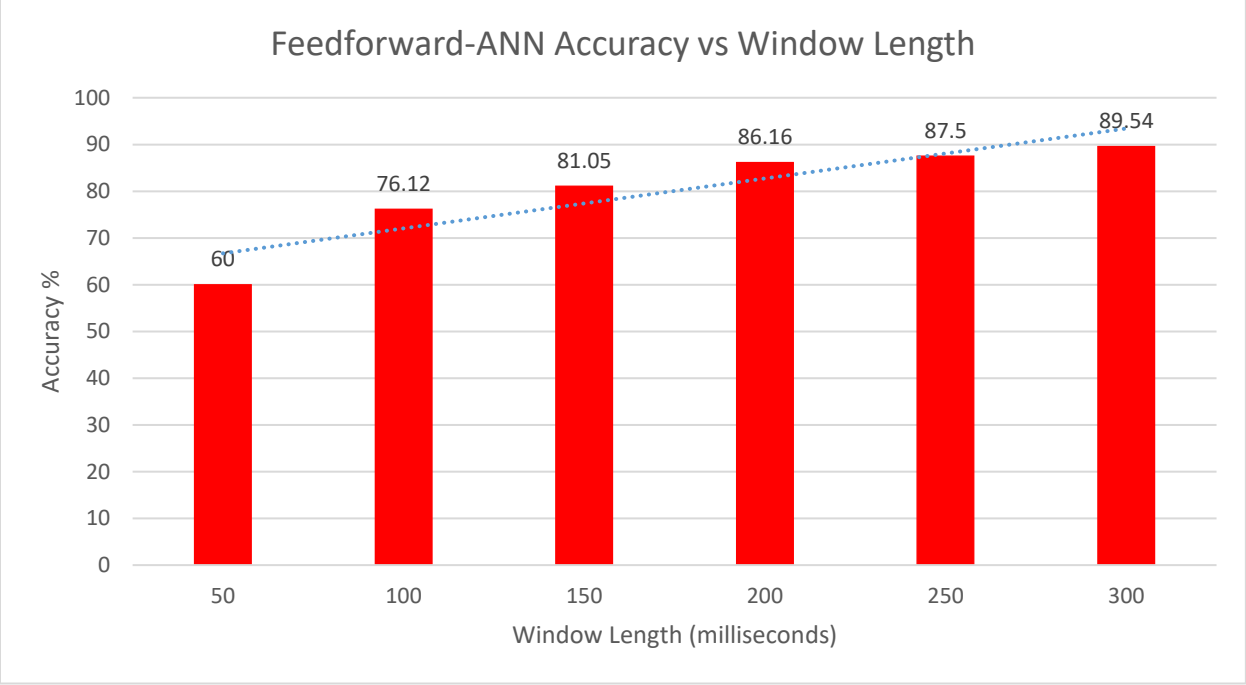

Figure 37 Feedforward-ANN Accuracy vs Window Length

Other classification algorithms such as decision tree, naïve Bayes, and discriminant analysis were also implemented on the dataset, but observed accuracy levels were significantly lower. Accuracy levels with PCA were significantly lower for all the implemented algorithms and computational cost is increased during predictions. Therefore, it may conclude that PCA may not be suitable for the real-time classification of signals

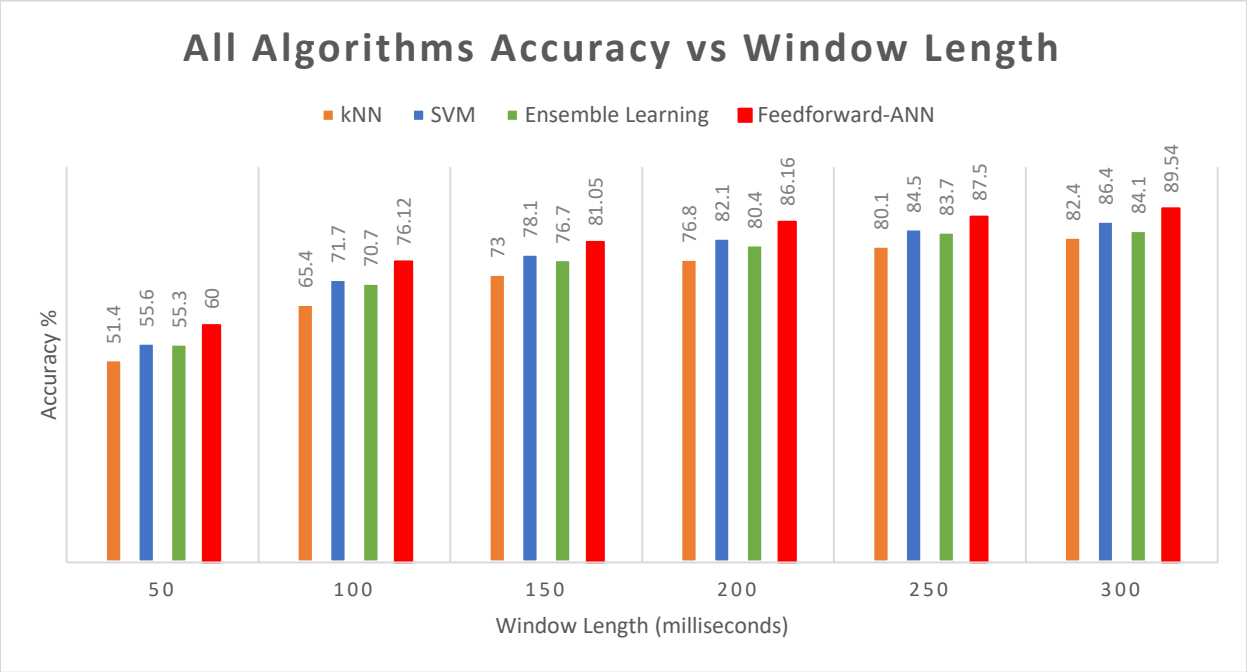

Figure 38 All Algorithms Accuracy vs Window Length Chart

Choosing the window length for real-time predictions should be based on several factors such as accuracy level, post-processing technique and, processing delay. But, the poor classification accuracy of 50ms window length causes it to be not acceptable for real-time predictions. kNN algorithms' inherent nature causes it to scan the complete training set for each prediction. This might cause lead to increased delays and memory limitations during real-time predictions. These reasons combined with low classification accuracy cause it to be not suitable for real-time predictions.

Window lengths higher than 200ms are more suitable for application without implementing post-processing techniques because they are more robust to varying decisions in a short period. Besides accuracy, the most important parameters for selecting the best model are prediction delay and model size. Those parameters were measured using a workstation Intel core i5 8265u 1.6GHz processor and 8GB random access memory. Measuring delay can be affected by various other processes running during the prediction. To minimize that error, the mean value of 100 prediction delays is taken. Because the processing platform will change, these values do not show the prediction delays when these algorithms are applied for real-time classification in a prosthetic device. Nonetheless, these can be used to identify the fastest algorithm.

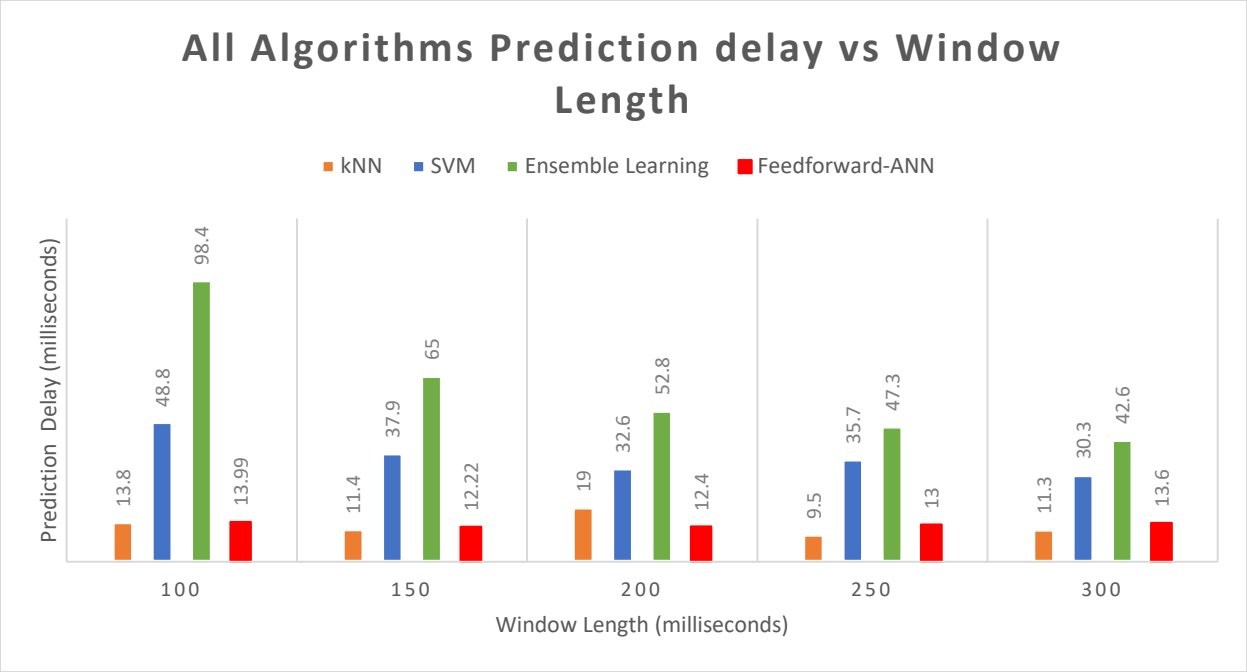

Figure 40 All Algorithms Prediction delay vs Window Length

Surprisingly, the kNN classifier had the smallest prediction delay. But the insufficiency of accuracy makes it less suitable for predictions. Despite having sufficient accuracies, SVM and ensemble learning classifiers have the longest prediction delays. In terms of accuracy and prediction delay, the feed-forward ANN classifier shows promising results. Window sizes used to train each classifier don't seem to have a significant impact on prediction delay. Sizes of the exported classifiers were also recorded as below.

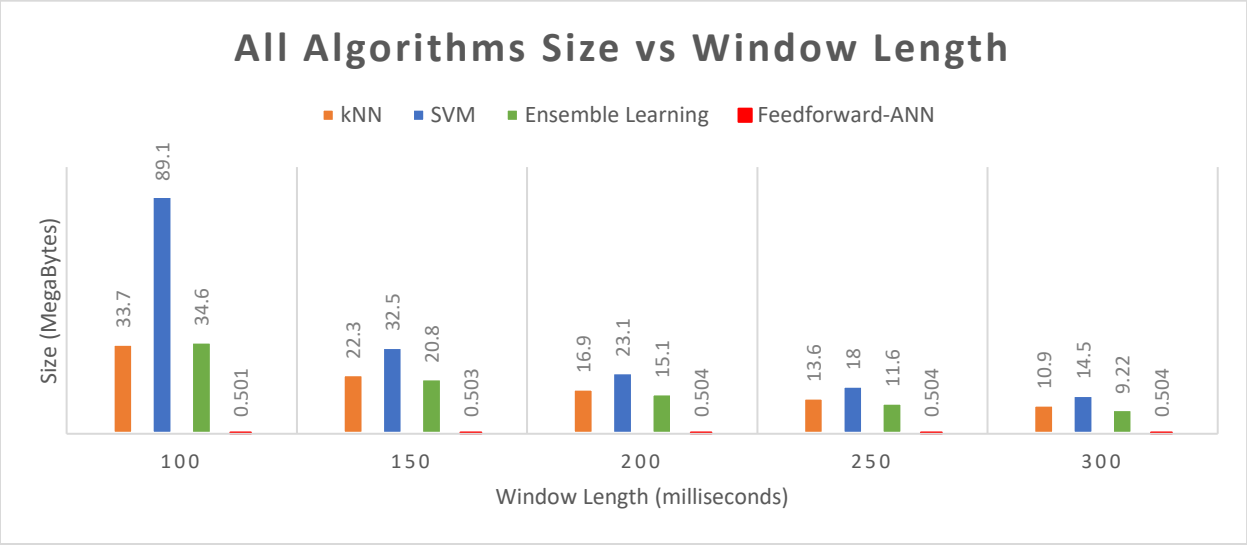

Figure 39 All Algorithms Size vs Window Length

The size of exported classifiers has a significant impact on their ability to be used on standalone prosthetic devices. None of the classifiers are acceptable for usage with any 8-bit microcontrollers or most 32-bit controllers due to their sizes. The maximum flash memory on even the most capable microcontroller development boards, such as the ESP32 Series and Arduino Nano RP2040 Connect, is 16 Megabytes. Even though the Feedforward ANN classifier file is almost 500 kilobytes in size, it requires the TensorFlow lite libraries to function. However, TensorFlow lite can only be implemented on certain development boards such as Arduino Nano BLE Sense, STM32F746 Discovery, etc. (TensorFlow, 2021).

One solution would be to use a single board computer such as Raspberry Pi Zero for computation. But, raspberry Pis' limited capabilities for direct hardware interfacing such as,

i. Real-time constraints are caused by the operating system's resource management.
ii. Lack of analog input (ADC) for signal acquisition (Robotics Back-End, 2021).

Because of these constraints, the Raspberry Pi cannot be used as the entire control system. However, a Raspberry Pi linked to a suitable microcontroller platform will be able to accomplish the task.

Postprocessing

Postprocessing techniques were used to smooth class decisions in 100ms, 150ms, 200ms window lengths. Larger window lengths were less prone to overwhelm actuators due to increased prediction delays.

- Majority Voting

Since feature extraction delays and prediction delays did not show significant variance within window sizes, classifier accuracies associated with window sizes were used to determine the number of decisions to be used in the majority voting technique for each selected classification algorithm. The nature of majority voting requires at least 3 votes to function. Thus, the minimum number of votes was set to three. To retain real-time prediction capability, the maximum allowable delay between the start of signal acquisition and prediction was set to 1500ms.

$$Total\ delay = (Feature\ extraction\ delay + Prediction\ delay + Window\ length) * number\ of\ votes$$

Feedforward ANN and SVM classifiers were selected due to their high accuracy levels. Even between these two, Feedforward ANN is preferred since it achieves the best levels of accuracy with the shortest prediction delay. The following table demonstrates some suitable classifiers, window size, accuracy per vote, and approximated total delay combinations.

Table 10 Selection of suitable classifiers with Majority voting

Classifier	Window length	Number of votes	Accuracy	Total delay
SVM	100ms	3	71.7 %	1149ms
SVM	100ms	4	71.7 %	1531ms
SVM	150ms	3	78.7%	1272ms
SVM	200ms	3	82.1%	1418ms
ANN	100ms	3	76.12%	1045ms
ANN	100ms	4	76.12%	1393ms
ANN	150ms	3	81.05%	1195ms
ANN	200ms	3	86.16%	1357ms
ANN	250ms	3	87.15%	1539ms

Odd numbers of votes are naturally preferred over even numbers of votes in majority voting because two classes with an identical number of votes may result in a tie, leaving the program indecisive. If all the votes have different classes, it can also result in an indecisive state. To address these concerns, the predicted class of the final vote was chosen as the decision in the situation of indecisiveness. Based on these factors, the following combination was chosen to be used with the majority voting technique.

i. Classifier – Feedforward ANN
ii. Window Size – 150ms
iii. Number of votes – 3
iv. Approximated total delay – 1195ms

Even though this combination doesn't offer the highest accuracy or lowest offers a satisfactory compromise between those.

- Bayesian Decision Theory

Even though Bayesian decision theory can offer a more sophisticated approach for postprocessing, it requires a higher number of votes to function properly. Increasing the number of votes increases the total delay linearly which hinders real-time prediction capabilities. So, Bayesian decision theory was not implemented as a post-processing technique.

Chapter 6 - Conclusion and Further development

Conclusion

This project focused on the development of an EMG-based real-time pattern recognition model for prosthetic hand control. To achieve real-time recognition, EMG data was split according to specified window lengths. Then features were extracted according to the discussed feature extraction techniques. Then various accuracy levels of different optimized classifiers were compared to identify the optimum classifier. Then majority voting technique was implemented to identify the best classifier-window length combination according to total delay and accuracy. The proposed model (ANN, 150ms window length, 3 votes) was able to achieve 81.05 % accuracy per classification with a feasible amount of delay. The designed model can be implemented on the proposed prosthetic controller or any other suitable prosthetic controller for real-time demonstration. With current results, it can be concluded that the designed model is a success.

Further development

- The classification accuracies can be improved if the model is implemented with more EMG channels because more channels can provide more information.
- The algorithm can be further optimized to improve accuracy and decrease prediction delays by removing less meaningful features, the addition of more features, decreasing redundant calculations, etc.
- The model was implemented and tested on a desktop workstation. To successfully use this model in a prosthetic controller, it needs to be optimized for the relevant control platform.

References

adafruit, 2015. *Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685.* [Online]
Available at: https://www.adafruit.com/product/815
[Accessed 1 09 2021].

Al-Timemy, A. H., Khushaba, R. . N. & Escudero, J., 2016. *A comparison of post-processing techniques on the performance of EMG based pattern recognition system for the transradial amputees.* Beirut, Middle East Conference on Biomedical Engineering (MECBME).

Arjunan, S. P. & Kumar, D. K., 2010. Decoding subtle forearm flexions using fractal features of surface electromyogram from single and multiple sensors. *JOURNAL OF NEUROENGINEERING AND REHABILITATION,* 7(1), p. Article 53.

Brownlee, J., 2019. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks.* [Online]
Available at: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
[Accessed 23 08 2021].

Dart, A. & Chan, C., 2004. *Investigating classification parameters for continuous myoelectrically controlled prostheses.* Quebec City, Canadian Medical and Biological Engineering Conference.

Englehart, K. & Hudgins, B., 2003. A Robust, Real-Time Control Scheme for Multifunction Myoelectric Control. *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING,* 50(7), pp. 848-854.

Flynt, J., 2017. *Acrylonitrile Butadiene Styrene (ABS): A Tough and Diverse Plastic.* [Online]
Available at: https://3dinsider.com/what-is-abs/
[Accessed 14 09 2021].

Flynt, J., 2017. *Polylactic Acid (PLA): The Environment-friendly Plastic.* [Online]
Available at: https://3dinsider.com/what-is-pla/
[Accessed 14 09 2021].

Goel, A., 2018. *Servo Motor : types and working principle explained.*. [Online]
Available at: https://engineering.eckovation.com/servo-motor-types-working-principle-explained/
[Accessed 09 08 2021].

Hudgins, B., Parker, P. & Scott, R. N., 1993. A New Strategy for Multifunction Myoelectric Control. *IEEE Transactions on Biomedical Engineering,* 40(1), pp. 82-94.

IBM, 2020. *Neural Networks.* [Online]
Available at: https://www.ibm.com/cloud/learn/neural-networks
[Accessed 23 08 2021].

Institute for Quality and Efficiency in Health Care, 2010. *How do hands work?.* [Online]
Available at: https://www.ncbi.nlm.nih.gov/books/NBK279362/
[Accessed 07 08 2021].

Khushaba, R. N., Al-Timemy, A. H., Al-Ani, A. & Al-Jumaily, A., 2017. A Framework of Temporal-Spatial Descriptors-Based Feature Extraction for Improved Myoelectric Pattern Recognition. *IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING,* 25(10), pp. 1821-1831.

Langevin, G., 2014. *Open Source 3D printed life-size robot.* [Online]
Available at: http://inmoov.fr/hand-and-forarm/
[Accessed 04 07 2021].

LAXMI SHAW, S. B., 2012. ONLINE EMG SIGNAL ANALYSIS FOR DIAGNOSIS OF NEUROMUSCULAR DISEASES BY USING PCA AND PNN. *International Journal of Engineering Science and Technology*, 04(10), pp. 4453-4459.

Linda Resnik, H. (. H. A. W. D. L. C. F. Z. a. N. W., 2018. Evaluation of EMG pattern recognition for upper limb prosthesis control: a case study in comparison with direct myoelectric control. *Journal of NeuroEngineering and Rehabilitation*, 15(23), pp. 1-13.

Omnexus, 2021. *A Detailed Guide on Acrylonitrile Butadiene Styrene.* [Online]
Available at: https://omnexus.specialchem.com/selection-guide/acrylonitrile-butadiene-styrene-abs-plastic
[Accessed 07 08 2021].

Phinyomark, A., Phukpattaranont, P. & Limsakul, C., 2012. Feature reduction and selection for EMG signal classification. *Expert Systems with Applications*, 39(8), pp. 7420-7431.

Phinyomark, A. et al., 2014. Feature extraction of the first difference of EMG time series for EMG pattern recognition. *Computer Methods and Programs in Biomedicine*, 117(2), pp. 247-256.

Rami N. Khushaba, S. K. M. T. G. D., 2012. Toward improved control of prosthetic fingers using surface electromyogram (EMG) signals. *Expert Systems with Applications*, 39(12), pp. 10731-10738.

Reaz, M. B. I., Hussain, M. S. & Mohd-Yasin, F., 2006. Techniques of EMG signal analysis: detection, processing, classification and applications. *Biological Procedures Online*, 8(1), pp. 11-35.

Robotics Back-End, 2021. *When to Use Arduino vs Raspberry Pi.* [Online]
Available at: https://roboticsbackend.com/when-to-use-arduino-vs-raspberry-pi/
[Accessed 27 08 2021].

Rodriguez, J. A. T. et al., 2019. Mechanical Properties of 3D-Printing Polylactic Acid Parts subjected to Bending Stress and Fatigue Testing. *Materials (Basel)*, 12(23), p. 3859.

scikit-learn, 2020. *1.4. Support Vector Machines.* [Online]
Available at: https://scikit-learn.org/stable/modules/svm.html
[Accessed 24 08 2021].

Spiewak, C., Islam, M. & Assad-Uz , M., 2018. A Comprehensive Study on EMG Feature Extraction and Classifiers. *Open Access Journal of Biomedical Engineering and Biosciences*, 1(1), pp. 17-26.

Srivastava, T., 2015. *Basics of Ensemble Learning Explained in Simple English.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/
[Accessed 24 08 2021].

Taunk, K., De, S., Verma, S. & Swetapadma, A., 2019. *A Brief Review of Nearest Neighbor Algorithm for Learning and Classification.* Madurai, International Conference on Intelligent Computing and Control Systems (ICCS).

TensorFlow, 2021. *TensorFlow Lite for Microcontrollers.* [Online]
Available at: https://www.tensorflow.org/lite/microcontrollers
[Accessed 27 08 2021].

Tkach, D. & Huang, H., 2010. Study of stability of time-domain features for electromyographic pattern recognition. *Journal of NeuroEngineering and Rehabilitation,* 21 (7).

Too, J., Abdullah , A. R. & Saad, N. M., 2019. Classification of Hand Movements based on Discrete Wavelet Transform and Enhanced Feature Extraction. *International Journal of Advanced Computer Science and Applications(IJACSA),* 10(6).

Wikipedia, 2021. *Polylactic acid.* [Online]
Available at: https://en.wikipedia.org/wiki/Polylactic_acid
[Accessed 07 08 2021].

Appendices

1. Digital signal filter implementation in MATLAB

```
fs = 4000; %sampling frequency
L = 20;%lower cut-off frequency
H = 500;%upper cut-off frequency
filtered_normal_ch1 = filterEMG(Subject_ch1,fs,L,H);
filtered_normal_ch2 = filterEMG(Subject_ch2,fs,L,H);
```

Figure 41 Bandpass filter implementation

2. Downsampling signals implementation in MATLAB

```
%Downsampling the signal
fs = 4000; %old sampling rate
fs_new = 1000; %new sampling rate
sample_rate = fs/fs_new;%downsampling factor.
signals_downsampled_ch1_normal = filtered_normal_ch1(1:sample_rate:end);
signals_downsampled_ch2_normal = filtered_normal_ch2(1:sample_rate:end);
```

Figure 42 Down sampling implementation

3. Feature extraction implementation in MATLAB

```
%%%%FEATURE EXTRACTION%%%%
%Mean Absolute Value Sum
MAV_ch1= jMeanAbsoluteValue(filtered_normal_ch1);
MAV_ch2= jMeanAbsoluteValue(filtered_normal_ch2);
%autoregressive model - parameters, order1
opts.order = 1;
AR_ch1 = jAutoRegressiveModel(filtered_normal_ch1,opts);
AR_ch2 = jAutoRegressiveModel(filtered_normal_ch2,opts);

 %Sum IEMG
IEMG_ch1 = jIntegratedEMG(filtered_normal_ch1);
IEMG_ch2 = jIntegratedEMG(filtered_normal_ch2);

%zero Crossing
opts.threshold = 0.01;
ZC_ch1 = jZeroCrossing(filtered_normal_ch1,opts);
ZC_ch2 = jZeroCrossing(filtered_normal_ch2,opts);

%SSC
 opts.threshold = 0.01;
SSC_ch1 = jSlopeSignChange(filtered_normal_ch1,opts);
SSC_ch2 = jSlopeSignChange(filtered_normal_ch2,opts);

%Skewness of the signal
skewness_ch1 = skewness(filtered_normal_ch1);
skewness_ch2 = skewness(filtered_normal_ch2);
```

Figure 43 Feature Extraction Implementation Part 1

```
%MyoPulsePercentageRate
opts.threshold = 0.016;
MyoPR_ch1 = jMyopulsePercentageRate(filtered_normal_ch1,opts);
MyoPR_ch2 = jMyopulsePercentageRate(filtered_normal_ch2,opts);

%LogDetector
log_ch1 = jLogDetector(filtered_normal_ch1);
log_ch2 = jLogDetector(filtered_normal_ch2);

%TemporalMoment
opts.order = 3;
TM_ch1 = jTemporalMoment(filtered_normal_ch1,opts);
TM_ch2 = jTemporalMoment(filtered_normal_ch2,opts);

%Log Differance Absolute Mean Value
LDMAV_ch1 = jLogDifferenceAbsoluteMeanValue(filtered_normal_ch1);
LDMAV_ch2 = jLogDifferenceAbsoluteMeanValue(filtered_normal_ch2);

%LogTeagerKaiserEnergyOperator
LTKEOP_ch1 = jLogTeagerKaiserEnergyOperator(filtered_normal_ch1);
LTKEOP_ch2 = jLogTeagerKaiserEnergyOperator(filtered_normal_ch2);

%Maximum Fractal Length
MFL_ch1 = jMaximumFractalLength(filtered_normal_ch1);
MFL_ch2 = jMaximumFractalLength(filtered_normal_ch2);
```

Figure 44 Feature Extraction Implementation Part 2

```
%Modified MAV2
MMAV2_ch1 = jModifiedMeanAbsoluteValue2(filtered_normal_ch1);
MMAV2_ch2= jModifiedMeanAbsoluteValue2(filtered_normal_ch2);

%Mean Absolute Deviation
MAD_ch1 = jMeanAbsoluteDeviation(filtered_normal_ch1);
MAD_ch2 = jMeanAbsoluteDeviation(filtered_normal_ch2);

%V order
opts.order = 2;
VOrder_ch1 = jVOrder(filtered_normal_ch1,opts);
VOrder_ch2 = jVOrder(filtered_normal_ch2,opts);

%Willison Amplitude
opts.threshold = 0.01;
WillisonAmplitude_ch1= jWillisonAmplitude(filtered_normal_ch1,opts);
WillisonAmplitude_ch2 = jWillisonAmplitude(filtered_normal_ch2,opts);

%Average Amplitude Change
Avg_Amplitude_change_ch1 = jAverageAmplitudeChange(filtered_normal_ch1);
Avg_Amplitude_change_ch2 = jAverageAmplitudeChange(filtered_normal_ch2);

%Hjorth Parameters - refer to the function
[Hjorth_Mobility_ch1,Hjorth_Complexity_ch1] = HjorthParameters(filtered_normal_ch1);
[Hjorth_Mobility_ch2,Hjorth_Complexity_ch2] = HjorthParameters(filtered_normal_ch2);
```

Figure 45 Feature Extraction Implementation Part 3

```
%Difference Absolute Mean Value
DMAV_ch1 = jDifferenceAbsoluteMeanValue(filtered_normal_ch1);
DMAV_ch2 = jDifferenceAbsoluteMeanValue(filtered_normal_ch2);

%Adding all the features together
total_features = vertcat(AR_ch1,AR_ch2,Avg_Amplitude_change_ch1,Avg_Amplitude_change_ch2,...
   DMAV_ch1,DMAV_ch2,log_ch1,log_ch2,...
    IEMG_ch1,IEMG_ch2,LDMAV_ch1,LDMAV_ch2,...
    LTKEOP_ch1,LTKEOP_ch2,MAD_ch1,MAD_ch2,MAV_ch1,MAV_ch2,...
    MFL_ch1,MFL_ch2,MMAV2_ch1,MMAV2_ch2,...
    MyoPR_ch1,MyoPR_ch2,skewness_ch1,skewness_ch2,SSC_ch1,SSC_ch2,...
    TM_ch1,TM_ch2,...
    VOrder_ch1,VOrder_ch2,WillisonAmplitude_ch1,WillisonAmplitude_ch2,...
    ZC_ch1,ZC_ch2,Hjorth_Complexity_ch1,Hjorth_Complexity_ch2,...
    Hjorth_Mobility_ch1,Hjorth_Mobility_ch2 );
```

Figure 46 Feature Extraction Implementation Part 4

4. Classification implementation in MATLAB

```
%Predicting
y_pred = classifier.predictFcn(total_features)
%Real_value
y_real = label_names_new(i);
y_pred = y_pred{1};
```

Figure 47 Classification implementation in MATLAB

5. Classification implementation in Python using TensorFlow (ANN)

```
#Multiclass classification
import tensorflow.keras
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
import numpy as np
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
import pickle
# Importing the dataset
dataset_X = pd.read_csv('total_features_smoll_windowed_300.csv',header = None)
X = dataset_X.iloc[:,:].values
dataset_Y = pd.read_csv('Category_names_600_windowed_300.csv',header = None)
y = dataset_Y.iloc[:, :].values
y = y.T
#X = X.T
# Encoding categorical data (output class)
# encode class values as integers
encoder = LabelEncoder()
y = encoder.fit_transform(y)
y = np_utils.to_categorical(y)
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X = sc.transform(X)
```

Figure 48 Classification implementation in TensorFlow Part 1

```
#Applying PCA
# from sklearn.decomposition import PCA
# num_components = 9
# pca = PCA(n_components=num_components)
# X_train = pca.fit_transform(X_train)
# X_test = pca.transform(X_test)
# explained_variance = pca.explained_variance_ratio_
# print(sum(explained_variance))

model = Sequential()
model.add(Dense(units = 58,activation = 'relu',input_dim = 58))
model.add(Dropout(0.1))
model.add(Dense(units = 58,activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(units = 58,activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(units = 58,activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(units = 58,activation = 'relu'))
#model.add(Dropout(0.1))
model.add(Dense(units = 58,activation = 'relu'))
# model.add(Dropout(0.1))

model.add(Dense(units = 10,activation = 'softmax'))#softmax for classification
sgd = SGD(learning_rate = 0.01,decay  =1e-6,momentum  = 0.9,nesterov  = True) #optimizer,lr = learning rate

model.compile(loss = 'categorical_crossentropy',optimizer = 'adam', metrics = ['accuracy'])


model.fit(X_train,y_train,epochs = 1000 , batch_size = 1000)
i = 0
while i <1:
    y_pred = model.predict(X_test)
    y_pred_thres = (y_pred>i) #threshold is 0.5
    y_pred_thres = y_pred_thres*1
    y_pred2 =y_pred_thres.argmax(axis=1) #converting from encoded mode to normal mode
    y_test2 = y_test.argmax(axis=1)#converting from encoded mode to normal mode
    from sklearn.metrics import confusion_matrix #this is a function not a class
    cm = confusion_matrix(y_test2,y_pred2)
```

Figure 49 Classification implementation in TensorFlow Part 2

```
    from sklearn.metrics import accuracy_score
    accuracy  = accuracy_score(y_test2, y_pred2)
    print(f'accuracy is: {accuracy} when threshold is: {i}')
    i=i+0.05
y_pred = model.predict(X_test)
y_pred_thres = (y_pred>0.15) #threshold is 0.5
y_pred_thres = y_pred_thres*1
y_pred2 =y_pred_thres.argmax(axis=1) #converting from encoded mode to normal mode
y_test2 = y_test.argmax(axis=1)#converting from encoded mode to normal mode
from sklearn.metrics import confusion_matrix #this is a function not a class
cm = confusion_matrix(y_test2,y_pred2)

from sklearn.metrics import accuracy_score
accuracy  = accuracy_score(y_test2, y_pred2)
print(f'accuracy is: {accuracy}')

#K-Fold Cross Validation
def baseline_model():
    model = Sequential()
    model.add(Dense(units = 58,activation = 'relu',input_dim = 58))
    model.add(Dropout(0.1))
    model.add(Dense(units = 58,activation = 'relu'))
    model.add(Dropout(0.1))
    model.add(Dense(units = 58,activation = 'relu'))
    model.add(Dropout(0.1))
    model.add(Dense(units = 58,activation = 'relu'))
    model.add(Dropout(0.1))
    model.add(Dense(units = 58,activation = 'relu'))
    #model.add(Dropout(0.1))
    model.add(Dense(units = 58,activation = 'relu'))
    #model.add(Dropout(0.1))
    model.add(Dense(units = 10,activation = 'softmax'))#softmax for classification
    sgd = SGD(learning_rate = 0.01,decay  =1e-6,momentum  = 0.9,nesterov  = True) #optimizer,lr = learning rate

    model.compile(loss = 'categorical_crossentropy',optimizer = sgd, metrics = ['accuracy'])
    return model
estimator = KerasClassifier(build_fn=baseline_model, epochs=1000, batch_size=100)

kfold = KFold(n_splits=5, shuffle=True)
results = cross_val_score(estimator, X, y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Figure 50 Classification implementation in TensorFlow Part 3

END OF THE PROJECT