CYPRUS INTERNATIONAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

AI PROSTHETIC HAND CONTROL VIA THE PERIPHERAL NERVOUS SYSTEM

By

ABDELRAHMAN OMER

ABDELRAHMAN KHALAFALLA

MARIA RAJABALI

MELEK NUR KOÇ

HIBA BENKADDOUR

July, 2024

Nicosia, NORTH CYPRUS

CYPRUS INTERNATIONAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

AI PROSTHETIC HAND CONTROL VIA THE PERIPHERAL NERVOUS SYSTEM

By

ABDELRAHMAN OMER

ABDELRAHMAN KHALAFALLA

MARIA RAJABALI

MELEK NUR KOÇ

HIBA BENKADDOUR

July, 2024

Nicosia, NORTH CYPRUS

AI PROSTHETIC HAND CONTROL VIA THE PERIPHERAL NERVOUS SYSTEM

By

ABDELRAHMAN OMER

ABDELRAHMAN KHALAFALLA

MARIA RAJABALI

MELEK NUR KOÇ

HIBA BENKADDOUR

DATE OF APPROVAL: 24 July 2024

APPROVED BY:

ASST. PROF. DR. ZİYA DEREBOYLU

_____

ASSOC. PROF. DR.NAHIT RiZANER

_____

ACKNOWLEDGEMENTS

We extend our sincere appreciation to our Supervisor's **Asst. Prof. Dr. ZİYA DEREBOYLU** and **Assoc. Prof. Dr. NAHIT RIZANER** , for their continuous guidance, valuable insights, and unwavering support throughout this journey. Their expertise and constructive feedback have been instrumental in shaping the success of our project.

We are also grateful to our professors and mentors at **Cyprus International University** for equipping us with the knowledge and skills necessary to undertake this project. Their encouragement and dedication have greatly contributed to our learning experience.

A special thanks to our families and friends for their endless support, patience, and motivation. Their encouragement has been a source of strength and inspiration throughout this journey.

Our sincere thanks go to our entire team for their dedication and collaboration. Each member brought unique skills and perspectives, contributing to the successful integration of electronic and software systems. This project would not have been possible without their united commitment and hard work.

This project is a testament to the collective effort and support of everyone involved. Thank you all for being part of this journey.

ABSTRACT

Globally there are over 1 million limb amputations every year, one every 30 seconds. Having an access to prosthetic limbs shouldn't be based on income because this can happen to any of us or to our loved ones, in 30 seconds. This project aims to solve the non-functionality and high cost of prosthetic hands on the market. The prosthetic hand design is three-channeled (sEMG) and 3-D printed. The acquisition system of the dataset consists of Myoware 2.0, Arduino Uno microcontroller, and a Python program. Sensors are connected to 3 muscles (Brachioradialis, Flexor Carpi Ulnaris, and Flexor Carpi Radialis) and Arduino Uno, and this is connected to Raspberry Pi which is connected to the 3-D printed hand is able to make rest, fist, paper, OK gestures.

The 3-D printed hand design consists of 6 servo motors (one for thumb rotation) and a servo driver that controls the fingers. Whenever the patient does a gesture the fishing lines are pulled and pushed by these motors. This system is controlled by the Raspberry Pi and is powered by a LiPo battery. Its AI model includes ANN and KNN comparisons. In the end, ANN with a 300ms window was decided to be used due to its high accuracy and low prediction delay.

This project's results have been just the beginning of achieving the initial aim of the project. With more time and better compounds, it's promising. It would be customizable to each person's specific limb shape and the residual muscles available.

ÖZET

Küresel olarak her yıl, her otuz saniyede bir, 1 milyon kişi ampütasyon oluyor. Prostetik uzuvlara erişim maddi gelire bağlı olmamalı çünkü bu, bize veya sevdiğimiz birinin başına gelebilir, 30 saniye. Bu proje marketteki fonksiyonellikten uzak fakat yüksek fiyatlı prostetik elleri çözümlemeyi amaçlıyor. Projemiz üç kanallı (yüzey elektromiyografi sensörlü) ve 3-D baskılı prostetik el tasasımına sahip. Veri toplama sistemimiz Myoware 2.0, Arduino Uno mikrodenetleyici ve bir Python programından oluşmaktadır. Sensörler; 3 tane kasa (Brachioradialis, Flexor Carpi Ulnaris, and Flexor Carpi Radialis) ve Arduno Uno'ya bağlı ve bu ikisi de Raspberry Pi'ya bağlı, bu da 3-D baskılı ele bağlanarak 'rest', 'fist', 'paper', 'OK' hareketlerini gerçekleştirebiliyor. 3-D baskılı el tasarımı 6 servo motor (biri baş parmak rotasyonu için) ve de parmakları kontrol eden servo sürücüsü içermekte. Hasta bir el işareti yapınca olta ipleri bu motorlar tarafından çekilip itiyor. Bu system LiPo bataryasından güç alarak Raspberry Pi tarafından kontrol ediliyor. Projemizin AI modeli ANN ve KNN'i karşılaştırarak, nihayetinde yüksek doğruluk ve düşük tahmin süresi nedenleriyle 300ms pencereleme tekniği kullanılmaya karar verildi. Bu projenin neticeleri, projemizin ilk amacının sadece bir başlangıcı oldu. Daha fazla zaman ve daha iyi parçalarla umut verici. Eğer daha fazla zaman ve kaynaklarla devam edilirse, kişinin geriye kalan uzuvunun şekline ve kaslarına göre özelleştirilebilir

Contents

List of Tables

List of Figures

List of Code Blocks

CHAPTER ONE
INTRODUCTION

Our goal for this project was to build a prosthetic hand controlled by the peripheral nervous system by using an AI model to predict specific hand gestures. The human hand is responsible for a wide range of movements that are required for the daily life of a human being. The loss of the human hand can highly impact one's degree of autonomy and capacity to perform daily tasks. Amputation due to traumatic injuries is considered as one of the main causes of upper limb amputation .

The first step of this project was to conduct researches and construct a literature review. The research conducted helped us understand and gather the information related to the goal of this project as well as highlight all the different steps required to complete the project, these include; Materials, Methodology, Deliverables and even funds.

In this report the Realistic constraints that will be discussed are; Design constraints, Engineering Standards and lifelong learning, Economical Analysis, Sustainability, Ethical Issues, Social and Political issues, Manufacturability and Risk management and change management. These are discussed in detail below

The methodology of this project is divided into four parts, which include the design and implementation of the acquisition system which was used to acquire data sets, followed by the actual acquisition of EMG (electromyography) signals from different people and different gestures (data acquisition), the design and functionality of the prosthetic hand and the AI model which will be the interface between the signals from the EMG sensors and the output gestures.

CHAPTER TWO
LITERATURE REVIEW

The human hand is capable of a wide range of intricate movements that let us engage with our surroundings and speak with one another. The opposable thumb, a rarity in nature, has helped us achieve high levels of dexterity allowing our evolution to proceed rapidly over other creatures. We must synthesize a vast amount of somaesthetic information about our surroundings, such as temperature, proprioception, vibration, pain, and fine touch, in order to perform complex hand movements. The loss of upper limbs may have a major impact on one's degree of autonomy and capacity to perform daily tasks like working and volunteering. Prosthetic devices only partially address the issues that restrict hand grip, such as force or tactile feedback deficiencies and limitations in the interfaces used to control the prosthesis. There are three main types of prosthetic arms

- Cosmetic Prosthetic arms
- Active Prosthetic arms (Cables)
- Myoelectric Prosthetic arms

Despite significant advancements in innovation over the past half-century, upper appendage prostheses of today are still subject to relevant confinements. Implanting actuators, sensors, and electronic components into a prosthesis that is the same size and weight as the replaced hand or appendage is one of the largest engineering challenges in the development of prosthetic devices. Another is advancing prosthesis control, which has a significant impact on the restoration of

someone's daily abilities (Lee et al., 2022) To create intuitive control, the user's intention can be extracted from signals recorded through methods such as surface EMG electrodes, ultrasound imaging, or other methods such as implantable EMG sensors or neural interfaces from the peripheral or central nervous system. We need a well-developed AI model in order to improve the prosthetic hand's behaviour and abilities. . (Reaz et al., 2006) There are many obstacles and problems in the way of developing and using functional prosthetic hands. The following are a few concerns regarding prosthetic hands: First: People who could benefit from Advanced AI prosthetic hands that can recognize different gestures and individual finger control, may not be able to access them due to the high costs associated with the increase of EMG sensors required to provide this high functionality. Second: The workings of the prosthetic hand using conventional and theoretical methods may not suit all patients which indicates that configuration of prosthetic hand is required for each patient

Third: Although progress has been made, it is still difficult to replicate natural sensory feedback in prosthetic hands. Research is still being done to achieve a realistic sense of touch, temperature, and proprioception.

Fourth: Careful engineering and design are needed for the seamless integration of AI technologies with prosthetic devices. It is a constant challenge to make sure the AI system enhances the prosthetic hand's performance in real time Lastly, prosthetic hands must be able to tolerate normal wear and tear. It is essential to guarantee the robustness of the embedded AI technology as well as the physical components. Furthermore, for long-term usability, accessible maintenance and repair services are necessary. To guarantee safety, effectiveness, and moral application, regulatory standards must be met during the development and implementation of AI prosthetic hands. Complying with intricate regulatory frameworks can be difficult for manufacturers and researchers. (Parajuli et al., 2019) In this project we're going to record signals relating different gestures and individual finger control using an EMG machine, these signal data

sets are acquired by the Microcomputer (Raspberry Pi) by utilizing an ADC. The AI model is then trained by using these signal data sets so the model will distinguish the different gestures and individual finger control. After training the AI model using the EMG Machine, we will need to also train it for our EMG sensors on different datasets so that we can relate it to the data from the EMG machine. Upon the completion of the AI model training, We are able to test our device. With the inputs from the EMG sensors, we will be able to control our servo motors for fine control of the prosthetic arm by the patient's command. Utilizing an I2C Communication board the communication between the servo and the Microcomputer will allow us to produce fine control for the Prosthetic arm.

2.1 EMG SIGNALS

When muscles contract and relax, they generate electrical signals due to the neuromuscular activities involved. These electrical signals are called electromyographic (EMG) signals. Electromyography is the technique used to measure and record these electrical activities in muscles. According to Reaz et al. (2006), the generation of EMG signals is closely tied to voluntary muscle movements. Bionic arms, which serve as prosthetic devices for individuals who have lost limbs or were born without them, utilize EMG signals for control. Even in the absence of a complete limb, residual muscles in the remaining part of the limb can produce EMG signals when they contract. These signals can be detected and interpreted to control the movements of a bionic arm. By harnessing the EMG signals from these residual muscles, the bionic arm can be maneuvered to perform various tasks, providing users with functional capabilities similar to those of a natural arm. Muscle Contraction: Residual muscles in the limb contract and produce EMG signals. EMG Signal Detection: Sensors detect the EMG signals generated by the residual muscles. Signal Processing: The detected EMG signals are processed and analysed by the prosthetic device's

control system. Prosthesis Movement: The processed signals are used to control the movements of the bionic arm, enabling the user to perform desired actions. EMG signals play a crucial role in enabling the control of bionic arms, thereby significantly improving the quality of life for users by providing them with the ability to perform everyday tasks with their prosthetic devices.

2.2 EMG SIGNAL ACQUISITION:

To acquire EMG signals from muscles, electrodes are placed on the skin overlying the muscles. These envelope signals must be amplified before they can be sampled using an built in analog-to-digital converter (ADC) on the Arduino. The MyoWare 2.0 sensor is a particularly effective tool for this purpose due to its user-friendly design, compact size, and integrated functionalities. This sensor is tailored for measuring muscle activity and is ideal for wearable applications. The MyoWare 2.0 sensor simplifies the process by incorporating the necessary amplification circuitry directly within the sensor unit. This integration ensures that the EMG signals are amplified and outputted as clean, analog signals that can be easily read by a microcontroller or development board. EMG data collection can be done using multiple channels, but too many channels can complicate the system and introduce additional points of failure. For practical applications, acquiring EMG signals from two channels strikes a good balance between data quality and system simplicity. This approach improves classification accuracy while minimizing complexity, making the system more user-friendly. Having the three channels complicates the system but adds more information and differentiation that is needed for the AI model. Due to solving the shipment issues of the reference cables, three channels were decided to be used like in the original plan. To sample and process the EMG data, a microcontroller or development board equipped with an ADC (Various options are available, such as Arduino board and ESP32.) is required. EMG Signal Generation: Muscles produce electrical signals during contraction and relaxation. Electrode Use: Electrodes

detect these signals, which require amplification. Amplification and Processing: The MyoWare 2.0 sensor amplifies EMG signals and provides a clean output for microcontrollers. Prosthetic Control: Processed EMG signals enable the control of bionic arms, restoring functionality.

CHAPTER THREE
REALISTIC CONSTRAINTS

During the undertaking of this project we were faced with several limitations. These limitations can be described as the realistic constraints that we faced. They were crucial and had to be effectively managed in order to ensure the completion of our project.

3.1 DESIGN CONSTRAINTS

During the undertaking of this project several limitations may be faced. These will be crucial and will have to be effectively managed in order to ensure the completion of our project. One of the first limitations we can be faced with is the unavailability of materials within Northern Cyprus. Due to this factor we had to order our EMG sensors from Germany with the risk of long delivery time and not receiving the complete order. In order to avoid further associated risks mentioned above we made sure to place our order during the semester break so that we could get the sensors right at the beginning of the semester which we did. However, the reference cables that were supposed to come with the sensors were missing, hence we had to order reference cables separately from England and get them delivered to us, for efficiency we used DHL to get the cables to Northern Cyprus. This last bit had a great impact on the lifecycle of our project as it delayed our data acquisition process.

3.2 ENGINEERING STANDARDS AND LIFELONG LEARNING

Despite the challenges 5 of us managed to overcome and organize the task list with simultaneous work, support and communication. Throughout the runtime of this project, we learnt how to effectively communicate with each other, allowing us to learn concepts beyond our individual field of study from each other.

3.3 ECONOMICAL ANALYSIS

Problem: prosthetic hands are expensive and inaccessible to many Objectives: -Selecting a design for 3-D printing of the hand and putting it together -Selecting the components that will be used for the hand -Research and acquiring model for the datasets from the patients -Creating an AI model to predict the gesture from the sensors and gives a good accuracy while doing it. Cost and benefits: A functional prosthetic hand could cost from $20,000 to $100,000. This hand design costs less than $2,000. By scanning the residual limb the 3-D print of the design and the socket can be customised for each patient. The residual limb can be even 3-D scanned and sent to the station to be printed and shipped, this way the access to remote areas and transportation cost and time can be reduced by both parties.

3.4 SUSTAINABILITY

The sustainability of this project relies on a thorough analysis of social, economic, legal, cultural, educational, and political factors. Socially, the project aims to enhance users' quality of life by providing affordable and accessible prosthetic options that improve mobility and independence. Community engagement through awareness programs fosters acceptance and support, while ongoing feedback helps refine the product. It is evident that the structure of the

project is still modular through design, which means that to increase the sustainability of manufacturing this project improvements should be done. These include but are not limited to; using better sensors such as intramuscular sensors instead of surface EMG sensors, further training the AI with more data sets for better predictions subsequently followed by overall software updates, usage of more durable and sustainable materials for the prosthetic hand itself. Economically, the focus is on affordability, with cost-effective production and distribution to ensure access for low-income populations. Collaborations with healthcare providers and insurance companies will help with financial support, making the prosthetics more accessible. Exploring various funding sources, such as grants and partnerships, further ensures financial sustainability. Legally, the project will comply with medical device regulations to ensure safety, Strong data privacy measures will protect user information, building trust with users and stakeholders. By addressing these sustainability aspects, the AI prosthetic hand project aims to make a lasting positive impact on users and communities, fulfilling its mission to empower individuals with innovative, AI-driven solutions.

3.5 ETHICAL ISSUES

In overseeing the development of an AI-driven prosthetic hand, ethical considerations take precedence to guarantee user safety, privacy protection, and inclusivity. It's crucial to implement stringent safety measures to thoroughly test the prosthetic. Privacy concerns necessitate secure management of sensitive user data, accompanied by transparent consent processes. Achieving equity involves ensuring that the technology is accessible and affordable for everyone, irrespective of their background.

3.6 SOCIAL AND POLITICAL ISSUES

In the realm of AI prosthetic hand projects, navigating social and political issues is crucial for success. Project managers must address ethical concerns such as user consent and privacy while ensuring compliance with regulatory requirements. They can effectively use positive political tactics by building alliances, communicating benefits clearly, and aligning project goals with organizational objectives. This approach ensures smoother implementation, fosters innovation in healthcare technology, and maintains stakeholder support and regulatory adherence throughout the project lifecycle.

3.7 MANUFACTURABILITY

The constraints related to manufacturability that we faced was that the market does not provide a lot of options when it comes to different portable EMG sensors as we have struggled to find a suitable one for our project. Furthermore, we didn't have an appropriate acquisition system that worked with our sensors available in the CIU labs. This also caused immense delays in our data acquisition process, as we had to design and implement our own acquisition system to our methodology. Due to this we had some irregularities within our data sets, such as absurdly high values and inconsistent sampling frequencies. Further details regarding the specific challenges encountered within the acquisition system are elaborated in the methodology part. Some constraints regarding the design and 3D printing of the hand were also present. We planned to print the hand using the 3D printer of the electronics lab, but the printer was broken so we had to request access and use the one on the mechanical lab and due to time constraints as well as availability of the 3D printer in the mechanical lab, it was apparent that a complex 3D design was not feasible for us to print and implement into the project

3.8 RISK MANAGEMENT AND CHANGE MANAGEMENT

During the course of this project we were faced with a lot of changes. Starting from the issue that only two reference cables got delivered so we had to take the initial datasets with only 2 channels and not 3 as planned. However after a while we got the 3rd cable and started taking data sets with 3 channels but overall the process of taking data sets was delayed. Furthermore, we planned to use the Biopac to acquire data sets but after testing we saw that it was impossible to use data acquired with Biopac to train the AI model, therefore, we had to design our own acquisition system to acquire datasets. We also planned on having at least 9 gestures but after repetitive data acquisitions and testing with the data only 3 gestures where recognizable and the AI model was trained using 3 non rest gestures and 1 rest gesture.

3.9 USAGE CONSTRAINTS

The Myoware 2.0 sensors we used to acquire the signals from are surface EMG sensors and the sensors are quite susceptible to noise which does not make them reliable for 24 hours use. The electrodes which are used are also surface electrodes, which are not reusable and are easily susceptible to sweat, tear, hence their adhesion to the skin are easily compromised throughout daily usage. The complexity of the electronics and how it is not fully integrated into the hand itself or into one system is also a main factor for the usage constraint of the prosthetic hand. Furthermore, the material used for the actual Prosthetic hand such as the ABS was used for 3D printing, the paracord and the fishing line and their durability are also a huge factor, these materials can easily be worn and torn, which makes it hard to sustain long periods of usage. Furthermore, the ABS used to print the hand is slippery which reduces the functionality of the hand while grabbing objects.

3.10 COST ANALYSIS

Component	Quantity	Turkish Lira	Euro
Arduino	1	662.11TL	18.53 EUR
Raspberry Pi	1	3114.15TL	88.03 EUR
Myoware EMG Sensor	3	4344.94TL	120 EUR
Myoware Reference Electrodes	1	10	3 EUR
USB Isolator	1	1151.11TL	32.22 EUR
MG995 Servo Motor	5	48TL	20.66 EUR
SG90 Servo Motor	1	220 TL	Data 45
Fishing line	5 meters	400 TL	11 EUR
DC-DC Step Down Converter	1	555.79TL	15.35 EUR
12V Power Adapter	1	750TL	20.73 EUR
16-Channel 12-bit PWM/Servo Driver	1	397.28 TL	11.12 EUR
Combined Shipping Fees	-	6517.06 TL	180 EUR
Total	Data 79	Data 80	510 EUR

Table 3.1: Cost Analysis Table

CHAPTER FOUR
METHODOLOGY

4.1 ACQUISITION SYSTEM

4.1.1 Design

The Biopac acquisition system was initially going to be used to acquire data sets for training the AI system, however, due to the difference between the sensing capabilities of the sensors used, a system was designed and implemented to acquire physiological muscle signals using the Myoware muscle sensors and Arduino microcontroller to acquire data sets.

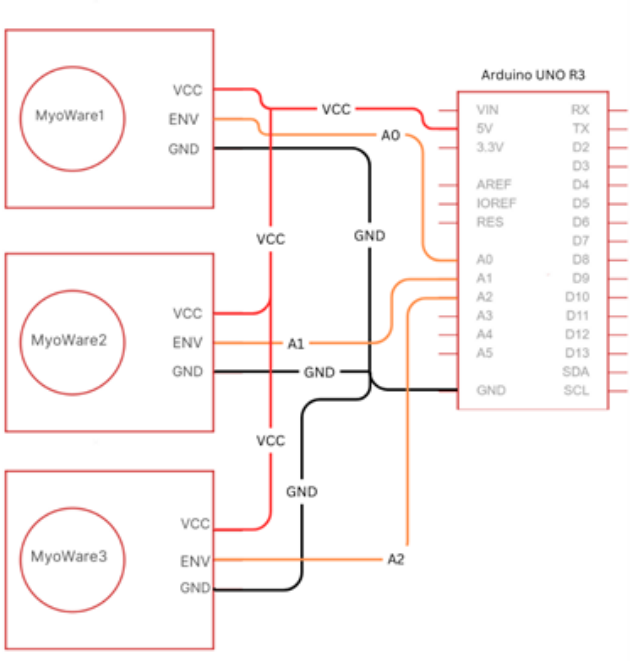

Figure 4.1: Acquisition System Design

The Arduino is configured as an ADC that is used to get the analog values from the Myoware sensor readings and send the data via the serial port that's connected to the computer. In the computer a code was developed using Python which used the PySerial library to capture the values from the Arduino Serial Port. A special way to print the values serially was needed since printing the values one by one in a comma separated manner would affect the loop time drastically and therefore the sampling rate therefore sprintf function was used to efficiently send the values through Serial. Ample time was also given so the Arduino ADC could settle to produce accurate readings.

4.1.2 Output validation

EMG Signals have a frequency range of 20-500Hz and according to the Nyquist Theorem for the minimum sampling rate required to recreate the signal is 2*B where B is the bandwidth of the signal. This results in a sampling rate of 1000Hz. The way to control the sampling rate in the Arduino is by adjusting the loop runtime so in order to control the sampling rate the delay function was used to control the loop runtime. In this system the Myoware library gracefully provides a way to test the sampling rate of your system using the code In Appendix 1. The sampling rate of the system is 1100+/-100Hz this was validated by checking the output which was the csv file. The csv file generation is illustrated in the flow chart below. The Python code can be found in Appendix 2

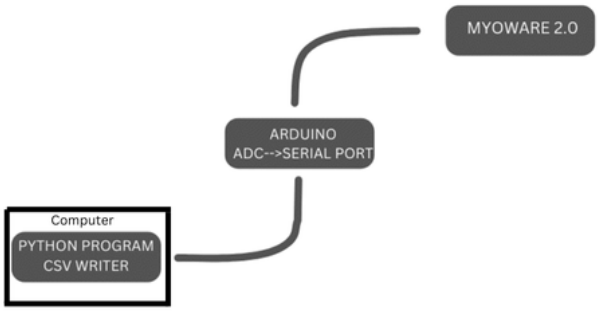

Figure 4.2: CSV file generation flowchart

4.2 DATA ACQUISITION

4.2.1 Anatomical background

In order to detect electromyography signals EMG sensors would have to be placed on the forearm. These locations were carefully selected based on the muscles of the forearm and the hand. For this project the Brachioradialis, Flexor carpi radialis and Flexor carpi ulnaris muscles were used to acquire signals.

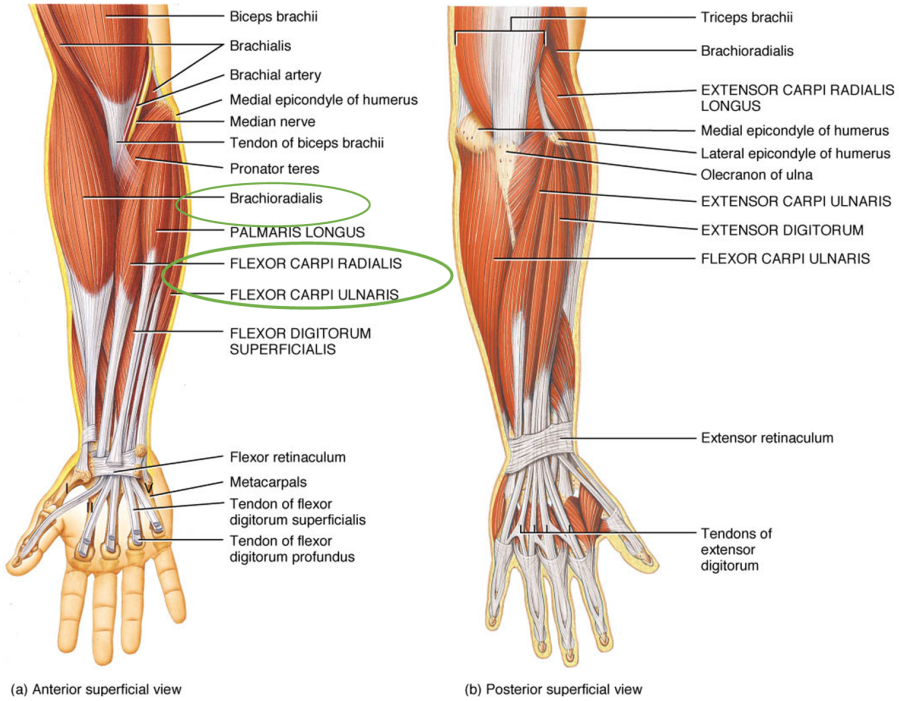

Figure 4.3: Muscles used for signal acquisition

The Brachioradialis muscle is responsible for precise finger movements by maintaining elbow stability and positioning the forearm [2]. Flexor carpi radialis and Flexor carpi ulnaris provide wrist stability and movement which is essential for proper hand positioning and effective grip [2]. Therefore, these muscles contribute collectively to the coordination and control needed for complex hand and finger movements [3].

4.2.2 Gesture selection

For gesture selection we referred to one of the reviewed research papers and recorded the gestures they used, plus some of the gestured we wanted to include. The gestures were recorded using Biopac Student Lab acquisition system. The Biopac allowed us to visualize which gestures had stronger signals. Based on this information we were able to eliminate and select the gestures we would record. At that time because of our delivery setbacks there were only two sensors available so, we recorded the gestures with Biopac to later eliminate and select the appropriate gestures. The muscles we used for this were the Brachioradialis (sensor 2) and the Flexor carpi radialis (sensor 1). Here are our results and please note that the red represents sensor 1, flexor carpi ulnaris and blue represents sensor 2, brachioradialis.

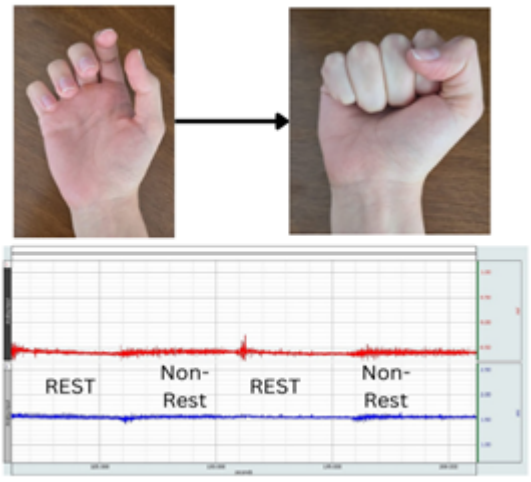

Figure 4.4: Rest to fist

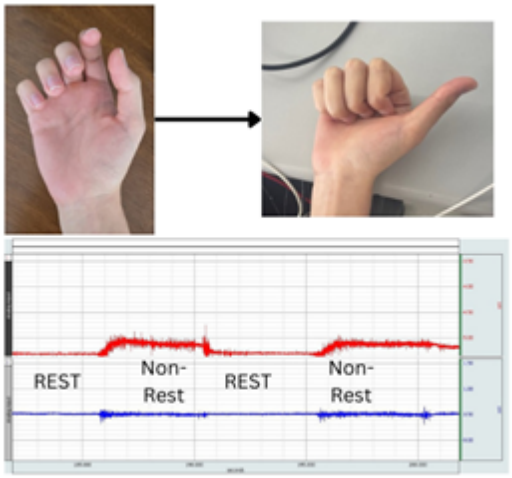

Figure 4.5: Rest to thumb

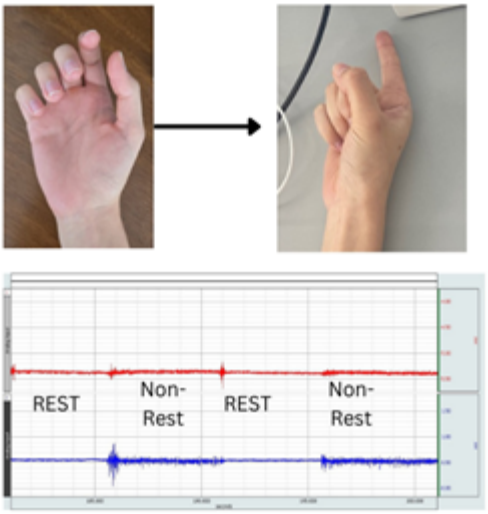

Figure 4.6: Rest to index

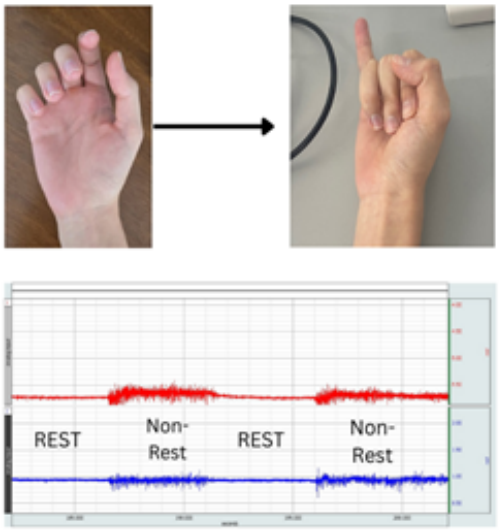

Figure 4.7: Rest to pinky

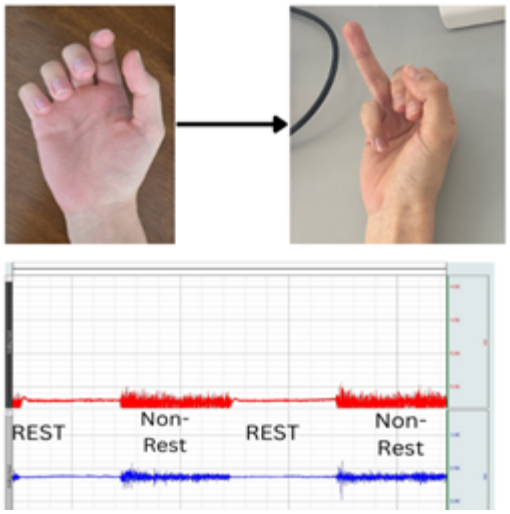

Figure 4.8: Rest to ring

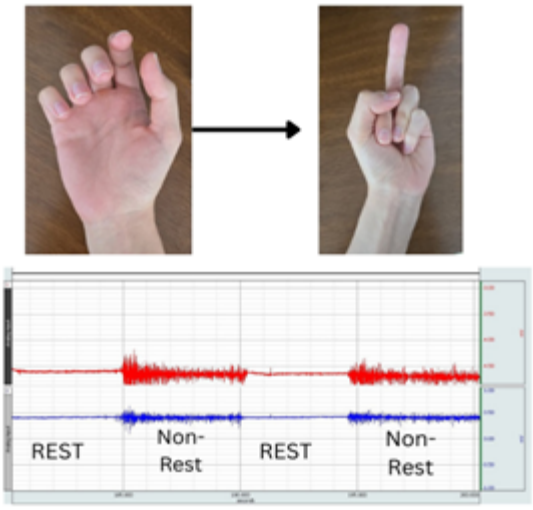

Figure 4.9: Rest to middle

Based on the acquisitions made with Biopac the gestures we selected for our prosthetic hand are; Fist, Thumb Finger, Index Finger, Scissors, Three, Four, Paper, Okay and Finger Gun. Therefore, we have 9 non-rest gestures and 1 rest gesture. The data set used to train the AI model was composed of 8 individual datasets, 4 females and 4 males. The sampling frequency was 1070Hz which gave us a sampling raze of 1100/1300 samples per second. Hence for every 0.03seconds 3 different samples from the 3 different channels was recorded. The structure we chose for the acquisition of the gestures mentioned above was as follows:

- 5s rest followed by 5s non rest. This forms a set.
- 5 repeated sets form 1 round
- 4 rounds will be conducted for each gesture by each subject.
- 10 s interval between rounds
- 3 min rest after 4 rounds

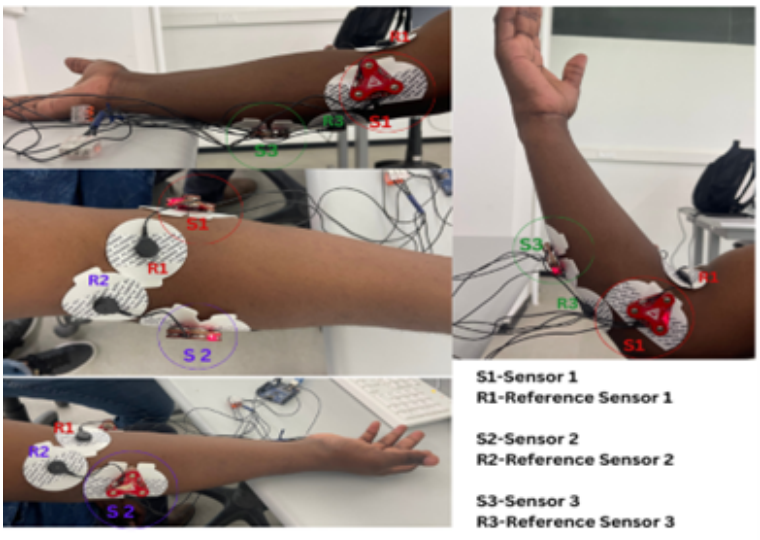

Figure 4.10: Electrode Placements

4.3 AI MODEL

4.3.1 Control configurations

When it comes to controlling prosthetic devices using electromyography (EMG) signals, there are two primary configurations: Direct Control and Pattern Recognition. Each method has its own advantages and limitations, making them suitable for different applications.

4.3.2 Direct control

Direct control uses the magnitude of the EMG signal to execute specific movements in the prosthetic device. EMG signals are inherently random and variable, making it difficult to use raw EMG signals effectively. To overcome this, the Mean Absolute Value (MAV) of the EMG signal is calculated over a set period . If the MAV exceeds a certain threshold, it triggers the prosthetic device to perform a movement. This approach can also incorporate multiple EMG channels from different muscles, allowing for multi-gesture control. The movement's speed can be proportionally controlled by the MAV value, known as proportional control. The simplicity of direct control makes it an attractive option for many commercial prosthetics, as it is relatively straightforward to implement. However, direct control is not without its challenges. It is effective for simple actions like opening and closing a hand, but as more gestures are required, the need for additional EMG channels increases. This adds to the system's complexity, increases the likelihood of hardware failures, and reduces accuracy due to muscle crosstalk, where signals from adjacent muscles interfere with each other. These factors can make the system less reliable overall. Despite these challenges, direct control remains a popular method due to its simplicity and ease of implementation. It is particularly well-suited for prosthetic devices that perform basic, repetitive movements.

4.3.3 Pattern recognition

Pattern recognition offers a more advanced and versatile approach to controlling prosthetics. Instead of relying on the magnitude of the EMG signal, pattern recognition techniques analyze the patterns within the signal to identify specific gestures. This method uses both handcrafted and computer-generated features of the EMG signals, including envelope EMG signals, to control the prosthetic device. Given the complex, time-series nature of EMG signals, recognizing patterns by human observation is impractical. Therefore, feature extraction techniques are used to process the signals. These techniques can be traditional methods or more advanced approaches using artificial neural networks. Once the features are extracted, machine learning algorithms classify the signals into different gesture categories. While pattern recognition is more computationally demanding and algorithmically complex than direct control, it offers several significant advantages. It tends to be more robust, with fewer points of failure and greater resistance to muscle crosstalk. This makes it better suited for handling multiple gestures and providing a more reliable control mechanism for prosthetics. In this project, we will use pattern recognition techniques rather than direct control. This decision is based on the robustness and versatility of pattern recognition, which can handle a broader range of gestures with higher accuracy and reliability. By leveraging these advanced techniques, we aim to develop a more effective and dependable control system for prosthetic devices.

4.4 DATA PREPROCESSING

When working with EMG signals for controlling prosthetic devices, it is essential to preprocess the signals effectively to ensure accurate real-time pattern recognition. This involves several steps, including pre-recording and labeling the signals, windowing the data, and choosing the appropriate feature extraction methods.

4.4.1 Windowing

For real-time pattern recognition, it is impractical to use the entire signal for feature extraction, as this would cause significant delays between the actual movement and the prosthetic device's response. Instead, the signal is divided into smaller segments or windows. This approach reduces the delay and allows for more responsive control of the prosthetic device. Windowing can be done using either an overlapping or non-overlapping (disjoint) windowing scheme. In overlapping windowing, each window overlaps with the previous one by a certain amount, providing a more continuous and detailed analysis of the signal. On the other hand, disjoint windowing uses non-overlapping segments, which is simpler but may miss some details. The choice of windowing scheme depends on the specific requirements of the prosthetic device and the desired trade-off between responsiveness and accuracy. In this project, we will use overlapping windowing to ensure a more detailed analysis of the EMG signals and improve the accuracy of the pattern recognition system.

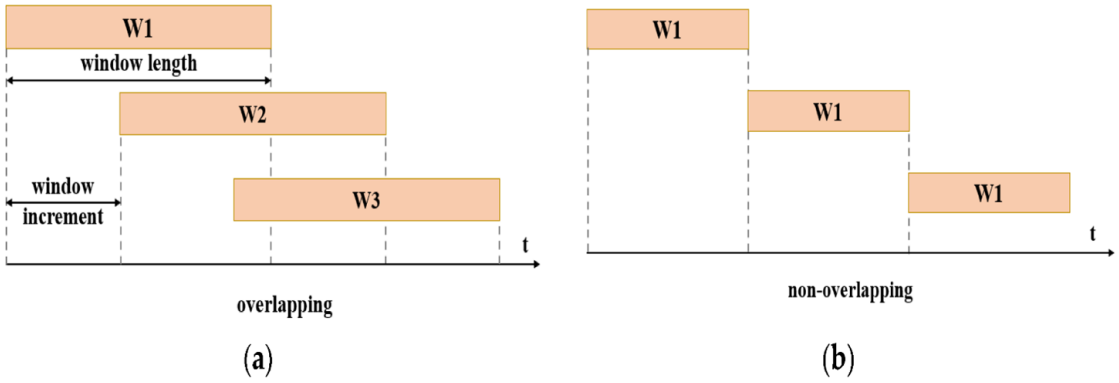

Figure 4.11: Overlapping and non-overlapping windows

Research has shown that overlapping windowing schemes generally produce better classification accuracy than disjoint windowing schemes, although they are more computationally intensive and can introduce delays in the process (Englehart & Hudgins, 2003). However, the classification accuracy also depends

on other factors, such as window size, the number of training samples, and the parameters of the classifier used (Khushaba, 2012). Given these considerations, this project uses an overlapping windowing scheme to balance accuracy and computational efficiency.

4.4.2 Threshold function

To differentiate between rest and non-rest states in EMG signals, we employed the Mean Absolute Value (MAV) of each windowed segment as a key feature. The MAV is a widely used metric in EMG signal processing due to its simplicity and effectiveness in representing the signal's overall activity level. It is calculated by taking the average of the absolute values of the signal's amplitudes within a given window. This method effectively captures the intensity of muscle activity, making it a reliable indicator of whether a muscle is at rest or active.

- **Youden's J Statistic:**

Youden's J Statistic is a straightforward method to determine the optimal threshold for distinguishing between two classes, such as rest and non-rest gestures. It is calculated by adding the sensitivity (true positive rate) and specificity (true negative rate) and then subtracting 1: **Youden's J=Sensitivity+Specificity-1**

- **Receiver Operating Characteristic (ROC) Curve Analysis**

To establish a threshold for MAV that accurately distinguishes between rest and non-rest states, we utilized the Receiver Operating Characteristic (ROC) curve. The ROC curve is a graphical representation that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It plots the True Positive Rate (sensitivity) against the False Positive Rate (1-specificity) for different threshold values. By analyzing the ROC curve, we can identify

the optimal threshold that balances sensitivity and specificity, maximizing the classifier's accuracy.

In this context, the ROC curve complements Youden's J Statistic by providing a visual method to evaluate the classifier's performance across various thresholds. The decision to use the ROC curve and Youden's J Statistic was driven by their ability to provide a comprehensive evaluation of the classifier's performance across various thresholds. This approach allowed us to select a threshold that minimizes the rate of false positives (incorrectly classifying rest as non-rest) and false negatives (incorrectly classifying non-rest as rest). As a result, the chosen MAV threshold ensures a robust and reliable differentiation between rest and non-rest states, enhancing the overall accuracy and reliability of our EMG signal classification system.

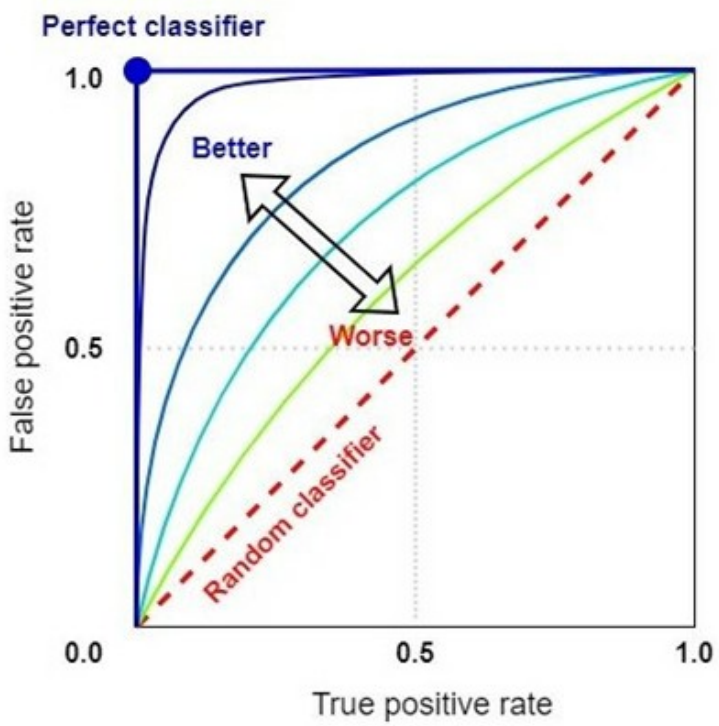

Figure 4.12: Overlapping and non-overlapping windows

4.4.3 Feature extraction

Extracting features from envelope EMG signals is a crucial step in interpreting and classifying the gestures effectively. We utilized several features to effectively capture the characteristics of EMG signals. These features include time-domain features such as Slope Sign Changes (SSC), Skewness, Log Detector, Temporal Moment, Hjorth Mobility and Complexity, and Waveform Length (WL). Additionally, we used Mean Frequency (MNF), a frequency-domain feature. Together, these features provide a comprehensive representation of the signal's statistical and temporal properties, which are essential for accurate gesture recognition. Below is a detailed explanation of these features: 1. 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). Where x_k is the current sample value in the EMG signal at index k, x_{k-1} is the

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

Figure 4.13: Slope Sign Changes (SSC) Formula

previous sample value in the EMG signal at index $k-1$, x_{k+1} is the next sample value in the EMG signal at index $k+1$, and ε is the threshold value to determine significant slope changes, used to filter out noise.

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

$$Skewness = \frac{\frac{1}{N}\sum_{i=1}^{N}(X_i - \mu)^2}{\left(\frac{1}{N}\sum_{i=1}^{N}(X_i - \mu)^2\right)^{\frac{3}{2}}} \tag{4.1}$$

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

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

$$\text{logDetector} = e^{\frac{1}{N}\sum_{i=1}^{N}\log(|X_i|)} \tag{4.2}$$

4. 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).

$$TM = \left|\frac{1}{N}\sum_{i=1}^{N} X_i^{\text{order}}\right| \tag{4.3}$$

5. 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.

$$\text{Mobility} = \sqrt{\frac{\text{var}\left(\frac{\partial y(t)}{\partial t}\right)}{\text{var}(y(t))}} \tag{4.4}$$

Hjorth complexity represents the change in frequency

$$\text{Complexity} = \frac{\text{Mobility}\left(\frac{\partial y(t)}{\partial t}\right)}{\text{Mobility}\,(y(t))} \tag{4.5}$$

6. Waveform Length (WL): Waveform Length (WL) measures the total length of the waveform over time, providing an indication of the waveform's complexity.

$$WL = \sum |\text{diff}(x)| \tag{4.6}$$

7. Mean Frequency (MNF): Mean Frequency (MNF) is the average frequency of the signal's power spectrum, indicating the dominant frequency component in the signal.

$$f_{\text{mean}} = \frac{\sum_{i=0}^{n} I_i \cdot f_i}{\sum_{i=0}^{n} I_i} \tag{4.7}$$

Where I_i represents the power spectrum's amplitude at a specific frequency component, f_i is the frequency value of the i-th component in the power spectrum, and n The total number of frequency components in the signal's power spectrum.

4.4.4 feature normalization

Feature normalization is a crucial step in preprocessing data, especially for machine learning models. It ensures that all features contribute equally to the model's performance. In this section, we discuss three common normalization techniques: Min-Max Scaler and StandardScaler.

1. Min-Max Scaler : Min-Max Scaler transforms features by scaling each feature to a given range, usually between 0 and 1. This is achieved by subtracting the minimum value of each feature and then dividing by the range (maximum value minus minimum value). The formula for Min-Max Scaling is:

$$x_{\text{scaled}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{4.8}$$

The primary advantage of Min-Max Scaler is that it preserves the relationships between data points, making it useful when features have different units and ranges. However, one of its disadvantages is its sensitivity to outliers because it uses the minimum and maximum values of the feature set. In this project, we used Min-Max Scaler for the K-Nearest Neighbors (KNN) algorithm to ensure all features contributed equally to the distance calculations used by KNN.

2. StandardScaler : StandardScaler standardizes features by removing the mean and scaling to unit variance, meaning each feature will have a mean of 0 and a standard deviation of 1. The formula is:

$$z = \frac{x_i - \mu}{\sigma} \tag{4.9}$$

where μ is the mean and σ is the standard deviation of the feature. The advantages of StandardScaler include standardizing the data, which is essential for algorithms that assume normally distributed input data, and being useful when the features have different units or scales. However, it can still be affected by

outliers, although not as much as Min-Max Scaler. In this project, we used StandardScaler for the Artificial Neural Network (ANN) model. This approach ensured that each feature had equal importance and contributed equally to the performance of our classification model. Proper normalization helped improve the convergence and accuracy of our ANN model, ultimately leading to better gesture recognition accuracy.

4.5 MODELS

4.5.1 K-nearest neighbors (knn)

K-Nearest Neighbors (kNN) is a straightforward and intuitive algorithm used for classification tasks. It operates on the principle of proximity, assuming that similar data points exist close to each other. The core idea of kNN is to classify a new data point based on the 'k' closest data points from the training set. Here's how it works: 1. Neighbors (k): The parameter 'k' refers to the number of nearest neighbors to consider when making a classification decision. For instance, if k=3, the algorithm looks at the three closest data points. If two of these points belong to class A and one belongs to class B, the new data point is classified as class A through a majority vote mechanism. 2. Distance Metric: The Euclidean distance is typically used to measure the distance between data points. It calculates the straight-line distance between two points in Euclidean space, which is given by:

$$|X - Y| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.10}$$

where X and Y are two points in n-dimensional space.

3. Weights:

- Uniform: Each of the k neighbors is given equal weight in the voting process.
- Distance: Neighbors closer to the new data point are given more weight, making their votes

In this project, we utilized the Min-Max Scaler to normalize the data before applying kNN. Normalization ensures that all features contribute equally to the distance calculations, preventing features with larger ranges from dominating the distance metric and thereby improving classification accuracy. kNN was chosen for its simplicity and effectiveness in capturing proximity-based relationships between different gestures in the EMG signal data. Despite its computational intensity, which requires storing all training data and calculating distances during classification, kNN's non-parametric nature makes no assumptions about data distribution, making it versatile for various tasks.

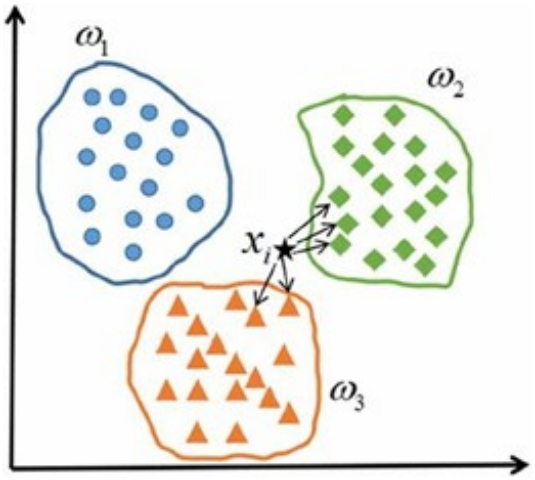

Figure 4.14: Visual Representation of K-Nearest Neighbors (kNN) Classification

4.5.2 artificial neural network (ann):

Artificial Neural Networks (ANNs) are inspired by the biological neural networks found in human brains. They consist of layers of interconnected nodes (neurons), where each connection has an associated weight. In our project, we designed the ANN with specific architectural choices to achieve optimal performance. 1. Hidden Layers and Neurons:

Hidden Layers: These are intermediate layers between the input and output layers. They perform computations to extract features from the input data. The number of hidden layers and the number of neurons in each layer can be adjusted to control the model's capacity to learn complex patterns. Neurons: Each hidden layer consists of neurons, where the number of neurons can affect the model's ability to capture different levels of abstraction.

ReLU (Rectified Linear Unit): This activation function is used in the hidden layers. It introduces non-linearity to the model, enabling it to learn complex patterns. The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{4.11}$$

Softmax: Used in the output layer for classification tasks. It converts the raw output scores into probabilities, making it easier to interpret the model's predictions. The Softmax function is defined as:

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{4.12}$$

where z_j is the input to the i-th neuron, and K is the number of classes

3. Dropout: Dropout is a regularization technique used to prevent overfitting. During training, a fraction of the neurons is randomly set to zero at each iteration, which helps the model to generalize better to new data. This is controlled by a parameter called dropout rate.

4. Batch Normalization: This technique normalizes the inputs of each layer to have a mean of 0 and a standard deviation of 1. It stabilizes the learning process and allows for higher learning rates, leading to faster convergence.

5. Epochs and Batch Size: The number of times the entire training dataset is passed through the network. More epochs can improve the model's accuracy but may also lead to overfitting if not controlled.

6. Optimizer and Learning Rate: We used optimizers like Adam or Stochastic Gradient Descent (SGD) to update the model weights. Adam is popular due to its adaptive learning rate, which adjusts the learning rate during training for faster convergence.

7. Loss Function: The loss function measures the difference between the predicted output and the actual target. For classification tasks, we commonly use categorical cross-entropy as the loss function.

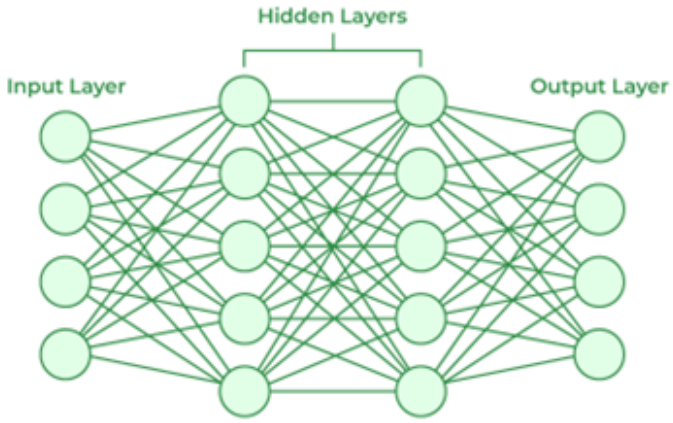

Figure 4.15: Architecture of an Artificial Neural Network (ANN)

In this project StandardScaler is used to normalize the input features before feeding them into the ANN. Standardization ensures that each feature has a mean of 0 and a standard deviation of 1, which helps in training efficiency and model performance. The ANN's architecture was designed to capture the complex patterns in the EMG signal data, leading to high classification accuracy for gesture recognition. The flexibility and power of ANNs made them an ideal choice for this task, despite the computational demands. The results from the ANN were highly promising, showing the potential of neural networks in accurately classifying gestures based on EMG signals.

4.6 VOTING MECHANISM

In the final stages of our project, we implemented a technique known as Majority Voting to enhance the robustness and reliability of our gesture recognition system. Majority voting is a method where the system considers a series of predictions over a specified number of windows or time frames and determines the final output based on the most frequently predicted class. This approach helps to mitigate the effects of noise and transient misclassifications, leading to more stable and accurate gesture recognition. After training our models and evaluating their accuracy, we applied Majority Voting to the sequence of predictions. Instead of relying on a single prediction for each window of data, we maintained a queue of recent predictions. As new data windows were processed, their predicted gestures were added to the queue. Once the queue reached a predefined size, we identified the most common gesture within the queue and used this as the final output. This Majority Voting mechanism proved to be particularly effective in our real-time application. By aggregating multiple predictions, the system could smooth out any short-term inaccuracies and provide a more consistent and reliable output. This approach is crucial in practical applications where noise and variability in the input signals can lead to occasional misclassifications. The Majority Voting strategy ensured that the overall system perfor-

mance remained robust, even in the presence of such challenges. However, this increased robustness comes with the penalty of extended delay in the decision process, as more predictions are required before making a final decision.

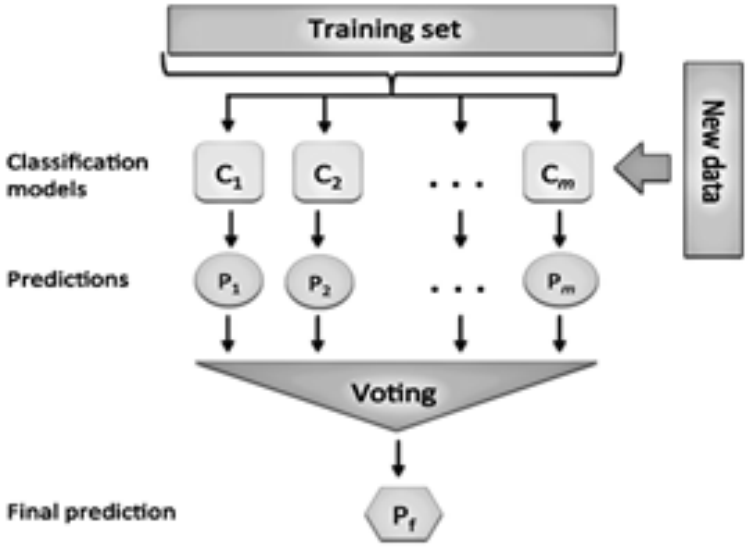

Figure 4.16: Majority Voting Mechanism for Model Aggregation

4.7 EVALUATION METRICS

Evaluating the performance of our gesture recognition system is crucial to understand its effectiveness and reliability. Several metrics were employed to assess the models, providing a comprehensive view of their performance. 1. Accuracy: Accuracy is one of the most straightforward evaluation metrics and represents the proportion of correctly classified instances among the total instances. It is calculated as:

$$\text{Accuracy} = \frac{\text{Number of correct Predictions}}{\text{Total number of predictions}} \tag{4.13}$$

Accuracy provides a general sense of how well the model is performing, but it can be misleading in cases where class imbalance exists. Therefore, while accuracy is useful, it should be considered alongside other metrics. Precision,

Recall, and F1-Score Precision, recall, and F1-score are more nuanced metrics that provide deeper insights into the model's performance:

- Precision

measures the proportion of true positive predictions among all positive predictions. It indicates how many of the predicted gestures were correctly classified. Precision is calculated as:

$$\text{Precision} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Positive (FP)}} \tag{4.14}$$

- Recall (Sensitivity or True Positive Rate)

measures the proportion of true positive predictions among all actual positives. It shows how many of the actual gestures were correctly identified. Recall is calculated as:

$$\text{Recall} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Negative (FN)}} \tag{4.15}$$

- F1-Score

is the harmonic mean of precision and recall, providing a single metric that balances both. It is particularly useful when the class distribution is uneven. F1-score is calculated as:

$$F1 \text{ score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.16}$$

These metrics are critical in understanding how well the model handles each class, especially in identifying which gestures it predicts accurately and where it struggles. Confusion Matrix

The confusion matrix offers a detailed breakdown of the model's predictions. It is a table that shows the true positive, false positive, true negative, and false

negative counts for each class. This matrix helps in understanding the types of errors the model is making and which classes are being misclassified. The confusion matrix is particularly valuable in multi-class classification problems, such as ours, where understanding the performance across all gestures is essential.

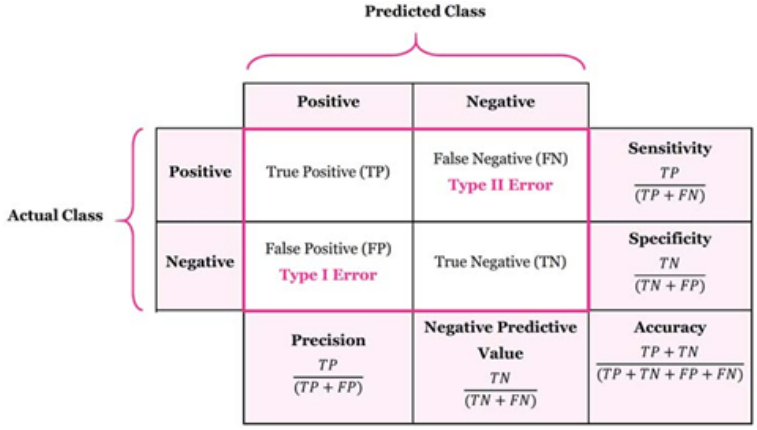

Figure 4.17: Structure of a Confusion Matrix and Associated Metrics

By employing these evaluation metrics, we were able to thoroughly assess the performance of our gesture recognition models. Each metric provided unique insights into different aspects of the model's behavior, allowing us to fine-tune and optimize our system for the best possible performance. The combination of these metrics ensured that our evaluation was comprehensive and that the final model was both accurate and reliable for real-time applications.

4.8 SOFTWARE AND LIBRARIES UTILIZED

In the development and implementation of this project, a variety of software tools and libraries were crucial. The primary programming language used was Python, selected for its extensive support in machine learning and data analysis.

Python

served as the backbone of the project, enabling seamless integration of various libraries and tools, providing a robust environment for coding, testing, and deploying machine learning models. NumPy played a critical role in numerical

computations, offering support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. It was essential for efficient handling and manipulation of numerical data.

Pandas was extensively used for data manipulation and analysis. This powerful library facilitated tasks such as data cleaning, transformation, and visualization, enabling efficient handling and processing of the EMG signal data.

Scikit-learn was the primary library for machine learning tasks. It provided tools for data preprocessing, model training, evaluation, and feature selection. Techniques like Recursive Feature Elimination (RFE), mutual information for feature selection, and cross-validation were implemented using this library. Models such as Decision Tree Classifier and K-Nearest Neighbors (kNN) were also built and evaluated with scikit-learn.

Joblib was used for saving and loading Python objects, such as the scalers used in preprocessing the data. This allowed for efficient persistence of models and preprocessing steps.

TensorFlow and Keras were employed for building and training the Artificial Neural Network (ANN) model. TensorFlow provided the backend for executing the computations, while Keras offered a high-level interface for defining and training the neural network. The optimizers used included Adam and Stochastic Gradient Descent (SGD).

TensorFlow Lite was utilized to convert the trained ANN model into a format suitable for deployment on the Raspberry Pi. This lightweight version of TensorFlow enabled efficient model inference on the resource-constrained environment of the Raspberry Pi.

SciPy was used for scientific and technical computing. It provided functions for signal processing, essential for tasks like filtering the EMG signals. The library's welch method was particularly useful for frequency analysis, and the skew function helped in calculating the skewness of the signal data.

Matplotlib was used for creating visualizations. It helped in plotting graphs

and charts to analyze the data and visualize the model's performance. These tools and libraries collectively contributed to the project's success, providing the necessary functionalities to handle data, build and train models, evaluate performance, and deploy the solution on a hardware platform. Each library played a distinct role in different stages of the project, from data preprocessing and feature selection to model training and deployment.

4.9 DESIGN AND IMPLEMENTATION

In this project, data collection was a critical first step. Our dataset contains a total of 4,891,104 samples, gathered at a sampling frequency of 1070 Hz. We gathered EMG signal data from a total of eight participants, comprising four females and four males. The participants were asked to perform three specific gestures:

- Fist
- Paper
- Okay

Each gesture was repeated in four rounds, with each round consisting of five repetitions of the gesture. Each repetition lasted for five seconds, followed by a five-second rest period. This structure ensured a consistent and ample dataset for training and testing our machine learning models. Given the sampling frequency of 1070 Hz, each five-second repetition produced 5350 samples. The high sampling rate allowed for capturing detailed EMG signal patterns, which are essential for accurate gesture classification.

To prepare the data for analysis, we employed a windowing technique. Windowing involves segmenting the continuous EMG signal into smaller, manageable segments, or windows, which can be analyzed individually. We experimented with various window sizes: 150 ms, 200 ms, 250 ms, and 300 ms. Ad-

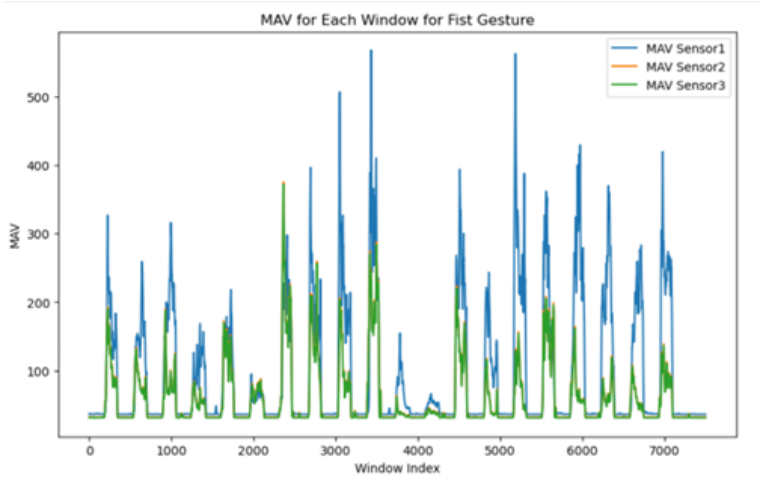

Figure 4.18: Mean Absolute Value (MAV) for Each Window for Fist Gesture

ditionally, we used overlapping windows to ensure a smoother and more continuous analysis. The overlap for each window size was set to create a delay of 15 ms, 20 ms, 25 ms, and 30 ms, respectively. This approach enabled us to determine the optimal window size for our models, balancing between capturing enough signal information and maintaining computational efficiency.

For each window created, we calculated its Mean Absolute Value (MAV). This value was crucial in determining a threshold that could separate between rest and non-rest states. To establish the optimal threshold, we employed Youden's J Statistic and Receiver Operating Characteristic (ROC) curve analysis. These methods helped identify the threshold that maximized the classifier's accuracy by balancing sensitivity and specificity. The chosen threshold ensured a robust differentiation between rest and non-rest states, enhancing the overall accuracy and reliability of our EMG signal classification system.

The Fig.4.23 shows the Mean Absolute Value (MAV) for each window of the "Fist" gesture and rest periods. The x-axis is the window index, and the y-axis is the MAV. Lines represent MAV values for Sensor1, Sensor2, and Sensor3. Peaks indicate the "Fist" gesture, while low values show rest periods. This separation helps determine the threshold to distinguish between gesture and rest states.

To identify this threshold, we employed the Receiver Operating Characteristic (ROC) curve. The dataset was split into two parts:

- The first part was used to find the best threshold using the ROC curve.
- The second part was used to test the effectiveness of this threshold.

After determining the optimal threshold using the ROC curve analysis, the next step involves comparing each calculated Mean Absolute Value (MAV) with this threshold. This process helps in distinguishing between active gestures and rest periods in real-time.

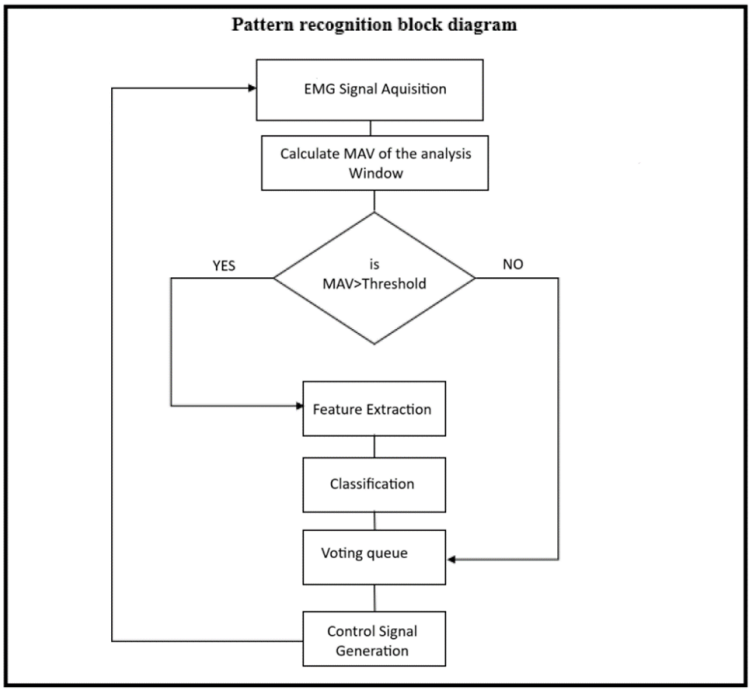

Figure 4.19: Pattern recognition pipeline

As depicted in the block diagram:

- EMG Signal Acquisition: The EMG signals are continuously acquired from the sensors placed on the muscles.

- MAV Calculation: For each analysis window, the MAV is calculated to quantify the signal's intensity.
- Threshold Comparison: The MAV of the current window is compared against the pre-determined threshold.

o If MAV ¿ Threshold: This indicates an active gesture. The system proceeds with:

- Feature Extraction: Extract relevant features from the EMG signal within the window.
- Feature Scaling: The extracted features are scaled to ensure they are on a similar scale, which helps in improving the performance of the AI model.
- Classification: Use the trained AI model to classify the scaled features and predict the gesture.
- Voting Queue: The predicted gesture is added to the voting queue to contribute to the final decision.
- Voting Queue: The 'Rest' label is added to the voting queue without performing feature extraction and classification.
- Control Signal Generation: The voting mechanism, which employs Majority Voting, determines the most frequent label in the voting queue. This label is then used to generate the appropriate control signal for the prosthetic device. The voting queue length we decided to use is 150, meaning that after 150 predictions , we determine the final prediction based on the most frequent label in this queue.

4.10 DESIGN AND CONTROL

4.10.1 Control mechanisms

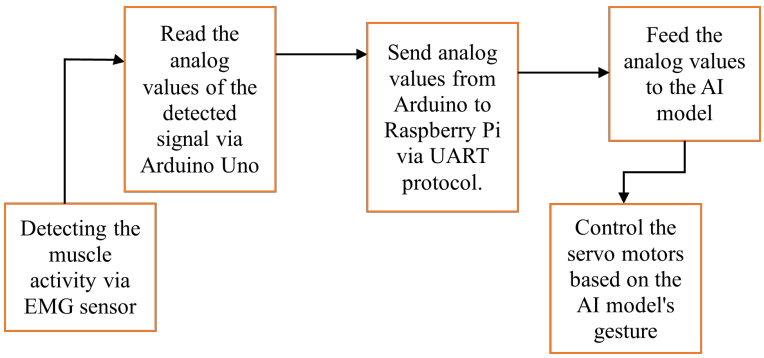

Figure 4.20: Control mechanisms

The design and control mechanism of the prosthetic hand divided into two parts:

- 3D design
- Actuators and controllers

4.10.2 3d design

The 3D design of the prosthetic hand, based on a Thing verse open-source model, utilizes a tendon-driven mechanism to control finger movements. This mechanism mimics the natural operation of human tendons and muscles, allowing for both independent finger movement and preset gestures. Here's a detailed explanation of the design and its components:

4.10.2.1 Functionality

Tendon-Driven Mechanism: The fingers are controlled by tendons, similar to how human fingers are moved by tendons connected to muscles. Fishing lines are used as tendons. These lines are chosen for their ability to withstand high tension, necessary for forceful and precise finger movements. The tendons are connected to actuators that pull or release them, thereby moving the fingers. This setup allows for fine control over each finger's position.

Finger Movement: Each finger can move independently, providing a high degree of dexterity. This feature is crucial for performing tasks that require precision and coordination. The design includes preset gestures, enabling the hand to perform common movements automatically, which can be particularly useful for repetitive tasks or specific functional grips.

Structural Components: The prosthetic hand is made using ABS (Acrylonitrile Butadiene Styrene), a plastic known for its high strength, light weight, heat resistance, and ductility. The components are printed using a Cubicon Single Plus 3D printer. This printer is capable of producing high-quality, durable parts necessary for the reliable operation of the prosthetic hand. The fishing lines used as tendons in the prosthetic hand must possess certain characteristics to ensure optimal performance: The lines must be able to withstand significant force without snapping. This is essential for controlling the fingers with precision and strength. The lines should have minimal stretch to ensure accurate and consistent finger movements. Stretchy lines could lead to imprecise control and reduced functionality. The lines must be lightweight to minimize the load on the actuators. Lighter lines require less force to move, resulting in more efficient finger control.

Durability: The tendons need to endure repeated use without degrading, ensuring the longevity of the prosthetic hand. Flexibility: While strong, the lines must also be flexible enough to move smoothly through the hand's structure, allowing for fluid finger movements.

4.10.2.2 Thingiverse hand design and assembly

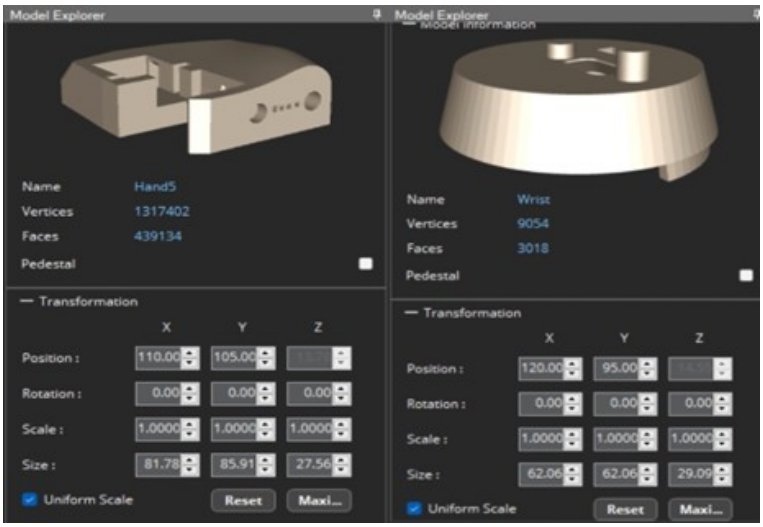

Figure 4.21: Palm & Wrist design

Mounting the Wrist Connector: The wrist unit usually has a universal connector or specific adapter that fits into the base of the palm. This can be a threaded connector, a snap-on mechanism, or a dovetail slide. Then Ensure the connector is aligned correctly with the palm's base. Use screws, bolts, or a locking mechanism to secure the wrist connector to the palm. This step is crucial to ensure that the palm does not detach during use.

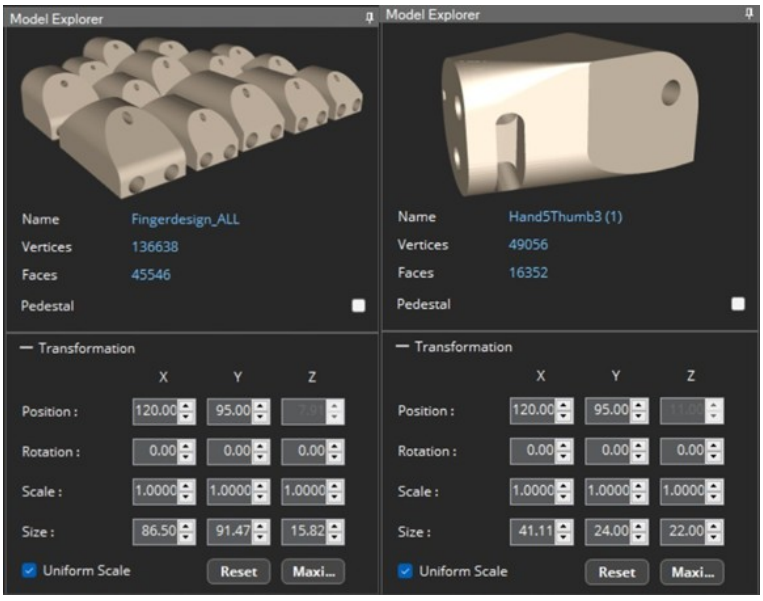

Figure 4.22: Fingers & Thumb design

The prosthetic hand design utilizes flexible paracord to attach the fingers and thumb to the palm, allowing for smooth bending and unbending at the joints. The paracord provides the necessary flexibility for a full range of motion in the fingers and thumb, while fishing lines act as tendons, connecting these digits to the hand's actuators. These fishing lines, capable of handling up to 5kg of force, enable precise and independent control of each finger and the thumb. This setup mimics natural hand movements, combining the flexibility of paracord for joint connections with the tensile strength of fishing lines for effective actuation, resulting in a lifelike and functional prosthetic hand that is durable and easy to maintain.

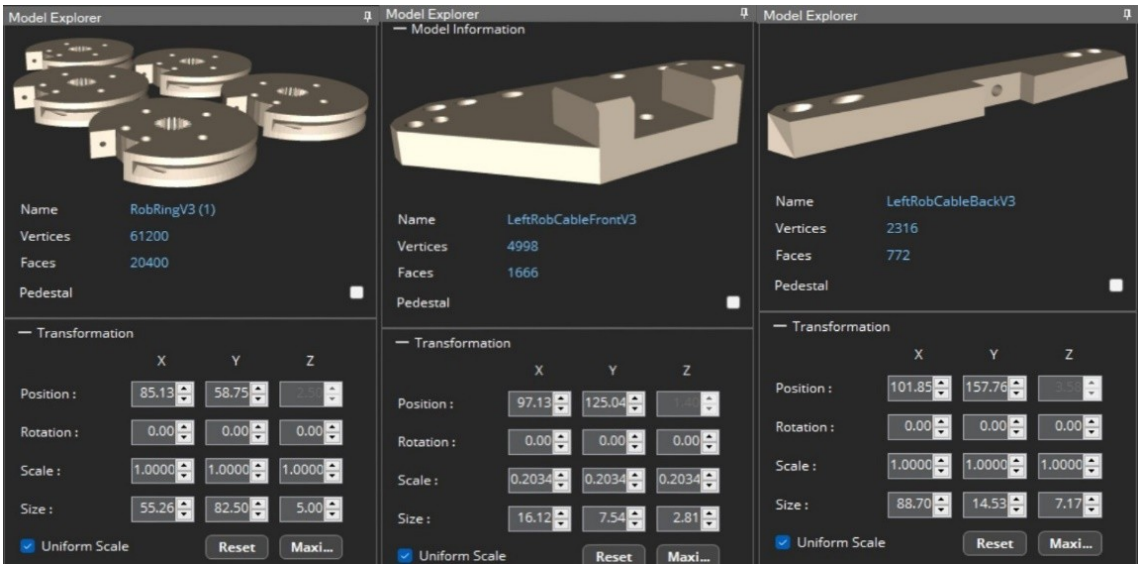

Figure 4.23: Fishing Lines pulleys

Design and Structure: Circular Design: The circular design of servo horns ensures an even distribution of force. This shape helps in minimizing stress concentrations and wear on specific points, contributing to a smoother and more reliable operation over time. Material: Typically made from lightweight and durable materials such as aluminum or reinforced plastic, servo horns are designed to withstand the repetitive forces exerted during the movement of the prosthetic hand. Attachment to Servo Motor: Mounting Holes: The servo horn is attached to the servo motor shaft using screws or a spline fitting, which ensures a secure connection. The mounting holes in the servo horn align with those on the servo motor shaft. Alignment and Calibration: Precise alignment is essential to ensure that the servo horn can transfer the motor's rotational motion accurately to the mechanical linkages or tendons. Calibration is performed to synchronize the horn's position with the intended neutral position of the hand.

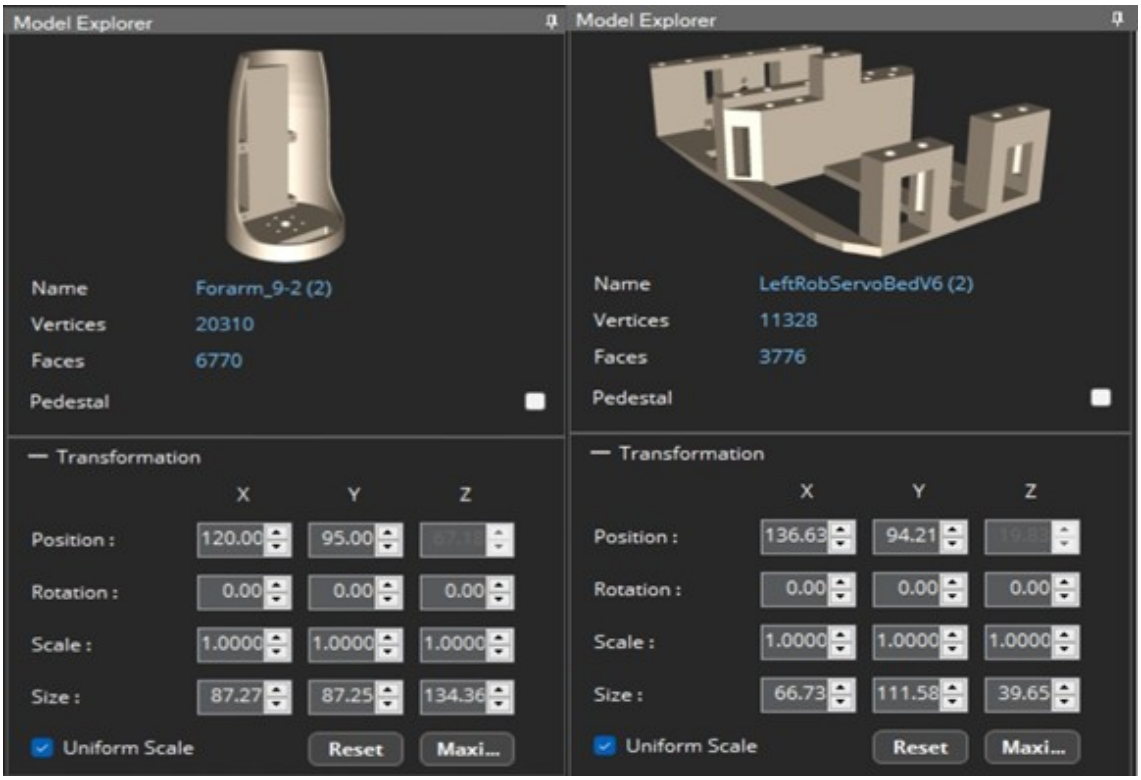

Figure 4.24: Forearm and Servo holder design

The forearm housing is typically custom-designed to accommodate the servo motors, servo horns, and cable channels. This ensures a compact and efficient layout that maximizes the available space. Servo motor holders are typically constructed from plastic materials such as ABS or PLA . These materials provide the necessary strength to support the servo motors under load. Secure Mounting: Servo motors are placed into the servo motor holder and mounted onto the forearm using screws, providing a stable and secure attachment. The brackets are designed to hold the motors firmly in place, preventing any unwanted movement. Mounting: The servo motor holder is integrated with a base that is designed to fit the internal structure of the forearm. This base is strategically placed to provide maximum stability and support to the servo motors and the overall structure of the holder.

Screw Attachment: Stable Connection: The servo motor holder is attached to the forearm using screws that pass through holder base and into the forearm structure. This creates a stable and secure connection that can withstand the forces generated during the operation of the prosthetic hand. The precise fit of the servo motors within the holder, combined with the secure attachment of the holder to the forearm, ensures that the motors remain firmly in place during use. This rigidity is essential for accurate and reliable operation.

Print settings:
Printer brand: Cubicon single
Printer: Cubicon single plus
Infill: 30

Parts to print without supporters:

- Servo horns
- Fingers
- Palm
- Servo holder

Parts to print with supporters:

- Forearm
- Wrist
- Hand thumb

Note: To ensure the prosthetic hand components print correctly and without damage, supports must be added to the parts that's need supports. These supports are especially important for parts like forearm, wrist and thumb connections.

4.10.3 Actuators and controllers

Raspberry Pi – Arduino was used to implement the system. This method allows the combining of computing power / wireless capabilities of Raspberry Pi with more versatile input/output capabilities of a Arduino Uno. Arduino will receive the signals and send it to the AI model in raspberry pi 4 . while raspberry pi will receive the acquired signals form Arduino and control the servo motors accordingly. Using universal Asynchronous Reception and Transmission (UART) serial communication protocol, Raspberry and Arduino will communicate with each other via USB ports to send the EMG signals and control the servo motors according to the input signal detected by the Myoware 2.0 sensor. According to the mentioned information about the components and its specification the final circuit diagram as follows:

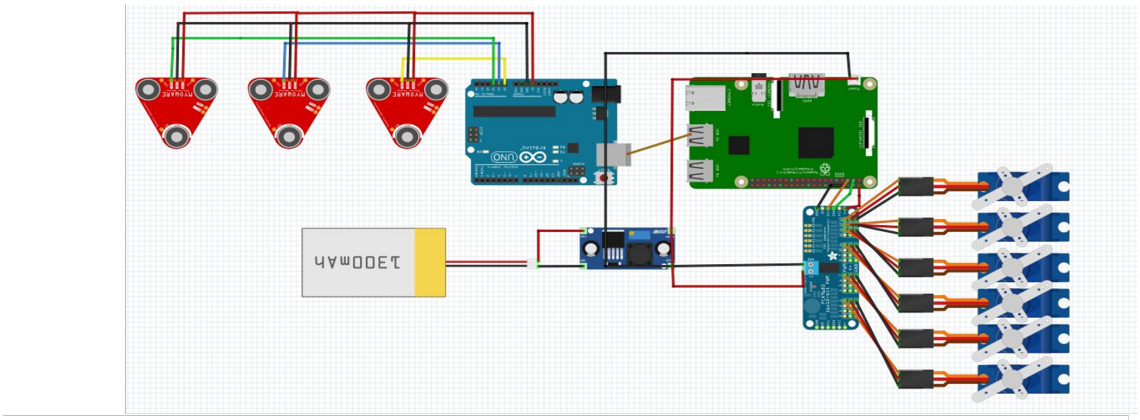

Figure 4.25: Circuit diagram

4.10.3.1 Actuators

To control the fingers of the prosthetic hand 5-Servo motors were used. An additional servo motor will be used to control the Metacarpal joint of the thumb finger. The chosen model of the servo motor is Tower pro MG995 Metal gear servo with the following specifications:

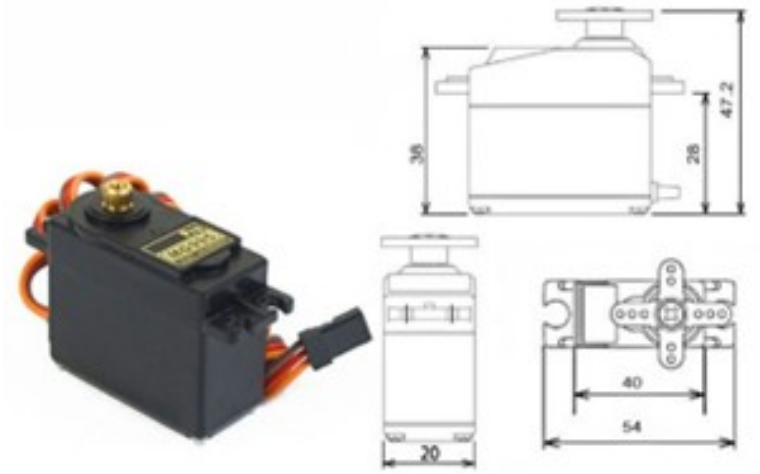

Figure 4.26: MG995 Tower pro Servo motor

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

Table 4.1: Motor Specifications

4.10.3.2 Controllers

he servo motor delivers adequate torque and speed while remaining compact enough to fit on the arm base. All servos were controlled using 12-bit 16 channel PWM/ (PCA9685) Which connects the microcontroller via I2C communication board Interface. The PCA9685 is a 16-channel, 12-bit PWM driver that can control up to 16 servos with a single board. It uses the I2C communication protocol to communicate with the microcontroller, allowing for precise control of the servo motors. The PCA9685 provides a simple and efficient way to control multiple servos simultaneously, making it ideal for applications like prosthetic hand control.

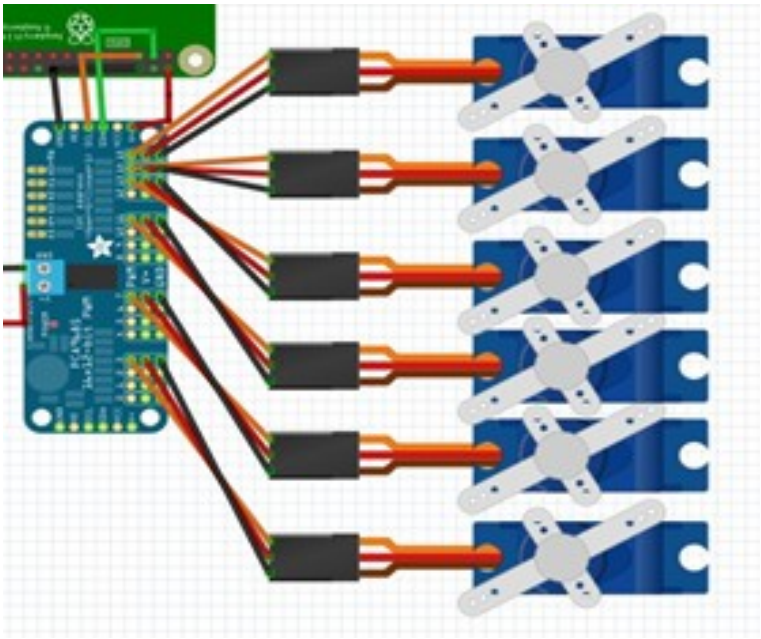

Figure 4.27: PCA 9685 servo driver

4.10.3.3 EMG Sensors

For the signal acquisition, 3 Myoware 2.0 sensors with dry electrodes were used. The Myoware 2.0 sensor is used to detect the muscle activity of the hand then will collect the data and amplify it with the internal amplifier integrated in the sensor. To acquire EMG signals three channels are used. Envelope output of each sensor is going to be used to extract the signals. The outputs of the sensors

Figure 4.28: Myoware sensor

are interfaced with the microcontroller as follows:

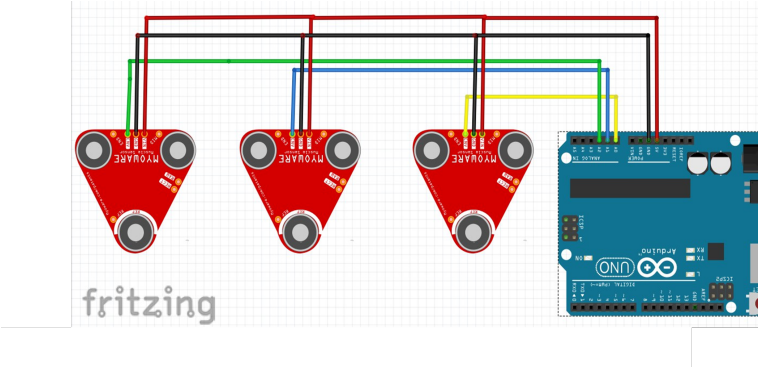

Figure 4.29: Configuration of Myoware sensor with Arduino uno

4.10.3.4 Voltage and current consumption of the system

Component	Voltage	Current
Arduino Uno	5V	200 mA
Raspberry Pi 4GB	5V	3 A
PCA9685 I2C Board	5V	10 mA per pin
DC-DC Buck Converter	Up to 40V	3A maximum
MG995 Servo Motor	4.8-7.2 V	1.5 A at 6V
Myoware 2.0	3.3-5V	9 mA

Table 4.2: Component Specifications

Arduino uno is powered through USB cable connected to the raspberry pi. Myoware sensors is powered via the 5 V pin of the Arduino. Raspberry pi requires power delivery from a power due to its high consuming of current (3A) as well as the servo motors. Raspberry pi maximum input voltage is 5 V and 6V for each servo motor, For the prosthetic arm to be portable the system is powered via LiPo battery .The input voltage for the I2C board and is controlled via DC-DC adjustable buck converter, this converter precisely controls the input voltage into it and give us the required output voltage.

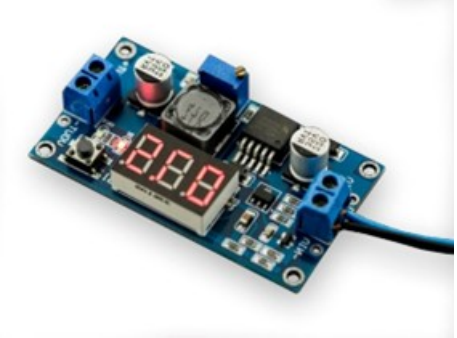

Figure 4.30: LM2596 DC-DC Buck Converter

4.10.3.5 Power consumption

Power consumption of the servo motors:

Operating Voltage: 6 V

Operating Current: 3.5 A (stall current) per motor

The supply current of the servo motor on average is 2.5 A.

For 6V supply:

$$P = V \times I = 6V \times 2.5A = 12.5\,W \text{ per servo.} \tag{4.17}$$

Total Power Consumption for 6 Servo Motors:

$$P_{\text{total}} = 6 \times P_{\text{servo}} = 6 \times 12.5\,W = 75\,W. \tag{4.18}$$

Total Current Consumption:

$$I_{\text{total}} = 6 \times I = 6 \times 2.5\,A = 15\,A. \tag{4.19}$$

Power consumption of a PCA9685 I2C board:

Total power consumption of servos:

$$P_{\text{total\_servos}} = 5 \times P_{\text{servo}} = 5 \times 12.5\,W = 62.5\,W. \tag{4.20}$$

Assume the PCA9685 board consumes 100 mA at 5V:

$$P_{\text{PCA9685}} = 5V \times 0.1A = 0.5\,W. \tag{4.21}$$

Power consumption of MG90S Micro Servo Motor:

Voltage: 6V (Let's assume 5V for calculation)

Current: 0.22A (running), 0.75A (stall)

An average of 0.5 A.

For 5V supply:

$$P = V \times I = 5V \times 0.5A = 2.5\,W. \tag{4.22}$$

Power consumption of Raspberry Pi 4:

Average current of 1.5 A.

$$P = 5V \times 1.5A = 7.5\,W \tag{4.23}$$

Power consumption of Arduino Uno:

$$P = 5V \times 38mA = 190\,mW \tag{4.24}$$

Power consumption of Myoware 2.0 sensor:

Operating Voltage: 5V

Current Consumption: 250pA – 1nA = 0.00125 A.

At 5V:

$$P = 5V \times 0.009A = 0.045\,W \text{ per sensor.} \tag{4.25}$$

Total current consumption for 3 sensors:

$$I_{\text{total}} = 3 \times I = 3 \times 0.009A = 0.027A. \tag{4.26}$$

For a total of three sensors:

$$P_{\text{Total}} = 0.045W \times 3 = 0.135W. \tag{4.27}$$

Component	Power consumption (W)	Current (A)
5 MG995 Servo Motors	62.5	12.5
1 MG90S Micro Servo	2.5	0.5
PCA9685 I2C Board	0.5	0.1
Raspberry Pi 4	7.5	1.5
Arduino Uno	0.19	0.05
Three Myoware 2.0 Sensors	0.135	0.027
Total	**73.32**	**14.677**

Table 4.3: Power Consumption of system

4.10.3.6 Battery calculations

Components and Their Specifications:

MG995 Servo Motors:

- Voltage: 6V
- Current: 1.5A (operating), 3.5A (stall)
- Average operating current: 2.5A each
- Total current for 5 servos:

$$2.5A \times 5 = 12.5A$$

MG90S Micro Servo Motor:

- Voltage: 6V
- Current: 0.5A

Total current drawn by 6 servo motors:

$$12.5A + 0.5A = 13A$$

PCA9685 I2C Board:

- Voltage: 5V
- Current: 0.1A

Raspberry Pi 4:

- Voltage: 5V
- Current: 600mA to 3A
- Average current: 1.5A

Arduino Uno:

- Voltage: 5V
- Current: 0.05A

Three Myoware 2.0 Sensors:

- Voltage: 5V
- Current: 9mA each
- Total for 3 sensors:

$$9mA \times 3 = 27mA$$

Total Current Draw:

$$I_{\text{Total}} = 13A + 0.1A + 1.5A + 0.05A + 0.027A = 14.677A$$

Battery Capacity Calculation:
Battery Capacity (Wh):

$$\text{Power}(W) \times \text{Runtime}(\text{hours}) = 73.385W \times 24\,\text{hours} = 1761.24\,\text{Wh}$$

Battery Capacity (Ah):

$$\frac{\text{Wh}}{7.5} = \frac{1761.24}{7.5} = 234.832\,\text{Ah}$$

Therefore, we will need a battery with a capacity of approximately **234.8 Ah** at 7.5V to run all these components for 24 hours.

CHAPTER FIVE
RESULTS AND DISCUSSION

5.1 DATA SET AND DATA VALIDATION

After detecting the strong signals by using biopac and determining the gestures that will be used, we started recording with myoware 2.0 and discover undesired values. In order to compare both raw and envelope datasets to BIOPAC's we plotted the normalization of the datasets. This formula sets the variables between 1 and 0.

$$\text{Normalized} = \frac{X - X_{\text{minimum}}}{X_{\text{maximum}} - X_{\text{minimum}}}$$

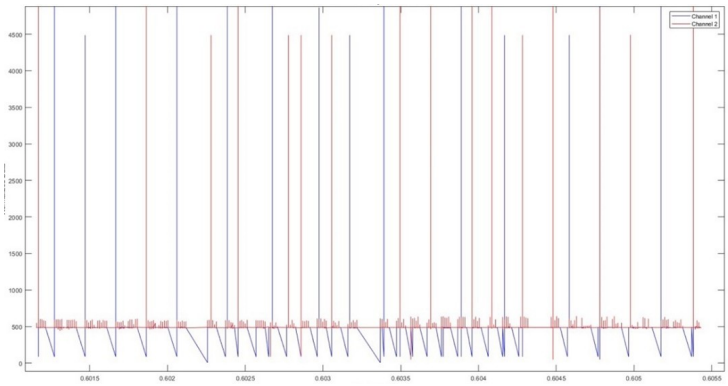

Figure 5.1: Myoware Normalized Data 200s Rest-Fist

What was noticed from this was that; there were still high peaks, undesired values in the datasets, and the sampling rates and frequency of BIOPAC was different from the Myoware 2.0. As well as we noticed a lot of irregularities with the myoware data, these were values as high as 4500. The second method to check our signals was root mean square to calculate the average power of

the signals and to analyze the amplitude of the EMG signals to see if there is a difference and patterns between rest and non-rest. This method was more efficient since we were dealing with raw data. We took 5s and 10s recording of each gesture and plotted the rms for 50 samples per gesture.We had to again adjust the y limits of each sensor for better comparison. These processes had to be repeated again once the third sensor arrived. These plots can be found in appendix 3.

$$\text{RMS} = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{N}}$$

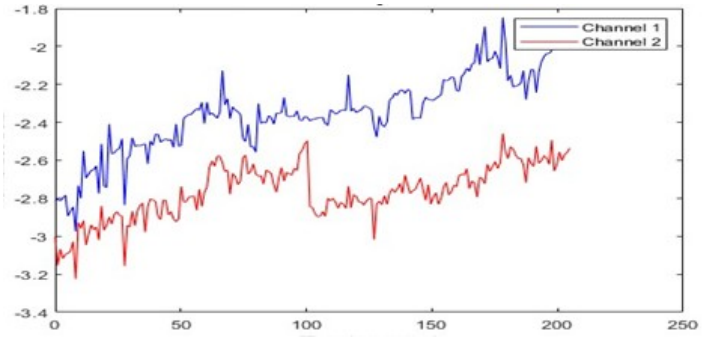

Figure 5.2: Normalized data,200s rest-fist

While doing these, values that shouldn't exist got detected. To eliminate them, a range was used (100¡EMG value¡1024) and for the unusable values, the program took the previous value. The MATLAB code that was used to plot these graphs and the graphs can be found in appendix 4. After the examinations of the plots, it was seen that there are no specific patterns, and while the dataset was being acquired, some time needed for the graphs to smoothen from gesture to non-gesture. We tried recording rest for 5 seconds and during non-rest's third second returning back to rest. This resulted in much better, consistent plots found in appendix 5 and a much better accuracy on the AI model. We also concluded that it was easier to notice the difference between different signals of different gestures when using envelope data.

5.2 AI MODEL

5.2.1 Threshold

In this section, we present the results of our analysis and discuss the implications of our findings. Initially, we collected datasets from our acquisition system and labeled them simultaneously, marking each segment as either a gesture or a rest state in real-time. This approach helped us effectively distinguish between different states for optimizing our gesture recognition model. To analyze the data, we applied a windowing technique to segment the continuous EMG signal into smaller, analyzable windows. We experimented with window sizes of 150 ms, 200 ms, 250 ms, and 300 ms to determine which size provided the most accurate results for our model. For each window, we calculated its Mean Absolute Value (MAV), which was crucial in establishing a threshold to differentiate between rest and non-rest states. We employed Youden's J statistic to determine the optimal threshold. Youden's J statistic is used to find the point on the ROC curve that maximizes the difference between the true positive rate and false positive rate, effectively optimizing the balance between sensitivity and specificity. The ROC curve for a 300 ms window size is shown below, where the x-axis represents the False Positive Rate and the y-axis represents the True Positive Rate. Using our labeled datasets, we identified True Positives (TP) as instances where a gesture-labeled window was correctly classified as a gesture, and False Positives (FP) as instances where a rest-labeled window was incorrectly classified as a gesture. The optimal threshold value, indicated on the ROC curve, was selected by identifying the point with the maximum Youden's J statistic, thereby maximizing the true positive rate while minimizing the false positive rate. This allows for an effective separation of gestures from rest periods, leading to improved classification accuracy in our gesture recognition system. The curve illustrates the threshold's performance in distinguishing between rest and non-rest states, highlighting the optimal value that enhances overall model accu-

racy. By calculating the accuracy and confusion matrix on the second part of the

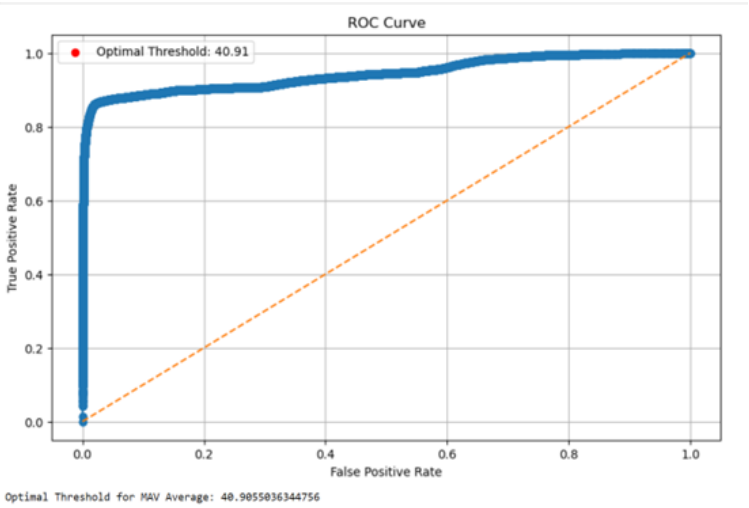

Figure 5.3: ROC Curve for Optimal Threshold Determination

dataset, we were able to validate whether the chosen threshold was effective in distinguishing between rest and non-rest states. The confusion matrix provides a detailed breakdown of the model's performance, showing the number of true positives, true negatives, false positives, and false negatives. This information helps evaluate the model's accuracy, precision, recall, and F1 score, providing valuable insights into its classification capabilities. The accuracy of the model was calculated by dividing the sum of true positives and true negatives by the total number of samples. This metric indicates the overall correctness of the model's predictions, reflecting its ability to classify gestures accurately.

Window size (ms)	Threshold	Accuracy	Mean Cross-Validation Accuracy
150	41.03	0.9369	0.9312
200	40.90	0.9284	0.9227
250	40.86	0.9309	0.9250
300	40.90	0.9332	0.9269

Table 5.1: Summary of Threshold, Accuracy, and Cross-Validation Accuracy for Different Window Sizes

The table summarizes the results of different window sizes (150 ms, 200 ms, 250 ms, and 300 ms) used for segmenting EMG signals. For each window size, the threshold value determined through ROC curve analysis, the validation set accuracy, and the mean cross-validation accuracy are presented. The table shows that varying window sizes affect the accuracy and generalizability of the model, with window sizes of 150 ms and 300 ms yielding the highest validation set accuracies of 0.9369 and 0.9332, respectively, and mean cross-validation accuracies of 0.9312 and 0.9269. These results help in selecting the optimal window size for accurate and efficient gesture recognition.

5.2.1.1 Feature Extraction Delays

Window length	**150ms**	**200ms**	**250ms**	**300ms**
Delay	8.7721 ms	9.1026 ms	10.1303 ms	9.6674 ms

Table 5.2: Feature Extraction Delays

According to these measurements, these delays indicate the time taken for the feature extraction process for each window length. We cannot definitively choose a single window length based solely on the current analysis, these delay measurements will help us refine our decision in future work. Balancing both recognition accuracy and processing delay is crucial for efficient and responsive real-time gesture recognition systems.

5.2.1.2 Models

5.2.1.3 K-Nearest Neighbors Classifier (KNN):

For 150ms Window Size: The kNN classifier was evaluated using different parameter settings to determine the best configuration. The cross-validation accuracies for each parameter setting are listed below: The best kNN cross-validation accuracy for the 150ms window size was 0.8848. The test accuracy for this configuration was 0.9063.

Index	Parameter Setting	Cross-Validation Accuracy
1	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'uniform'}	0.8848
2	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'distance'}	0.8848
3	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}	0.8491
4	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}	0.8598
5	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'uniform'}	0.8277
6	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}	0.8483

Figure 5.4: kNN Cross-Validation Accuracies for 150ms Window Size

For 200ms Window Size: The kNN classifier was evaluated using different parameter settings for the 200ms window size. The cross-validation accuracies are listed below: The best kNN cross-validation accuracy for the 200ms window

Index	Parameter Setting	Cross-Validation Accuracy
1	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'uniform'}	0.9101
2	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'distance'}	0.9101
3	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}	0.8753
4	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}	0.8849
5	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'uniform'}	0.8538
6	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}	0.8725

Figure 5.5: kNN Cross-Validation Accuracies for 200ms Window Size

size was 0.9101. The test accuracy for this configuration was 0.9304.

For 250ms Window Size: The kNN classifier was evaluated using different parameter settings for the 250ms window size. The cross-validation accuracies are listed below: The best kNN cross-validation accuracy for the 250ms window

Index	Parameter Setting	Cross-Validation Accuracy
1	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'uniform'}	0.9205
2	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'distance'}	0.9205
3	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}	0.8883
4	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}	0.8969
5	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'uniform'}	0.8655
6	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}	0.8852

Figure 5.6: kNN Cross-Validation Accuracies for 250ms Window Size

size was 0.9205. The test accuracy for this configuration was 0.9390.

For 300ms Window Size: The kNN classifier was evaluated using different parameter settings for the 300ms window size. The cross-validation accuracies are listed below: The best kNN cross-validation accuracy for the 300ms window

Index	Parameter Setting	Cross-Validation Accuracy
1	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'uniform'}	0.9233
2	{'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'distance'}	0.9233
3	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}	0.8932
4	{'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}	0.9014
5	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'uniform'}	0.8702
6	{'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}	0.8883

Figure 5.7: kNN Cross-Validation Accuracies for 300ms Window Size

size was 0.9233. The test accuracy for this configuration was 0.9440.

Confusion Matrices

The confusion matrices for each window size are shown below, illustrating the performance of the kNN model in predicting the gestures:

Best Parameters: The best parameters found for the kNN model across all window sizes were:

- Metric: Euclidean
- Number of Neighbors: 1
- Weights: Uniform

Classifier accuracies for each window length were calculated and plotted below : Principal Component Analysis (PCA) was also applied in an attempt to re-

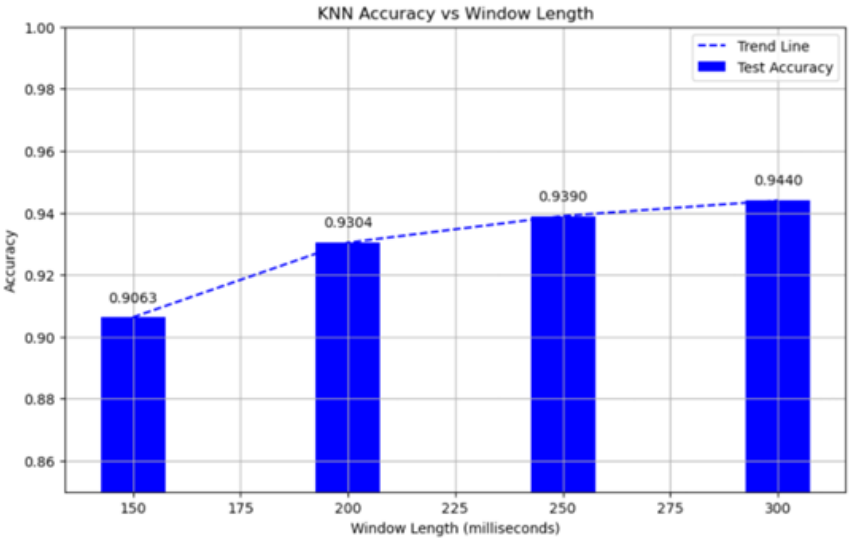

Figure 5.8: kNN Accuracy vs Window Length

duce the dimensionality of the feature set and improve the model performance. However, it was observed that using PCA actually led to a decrease in accuracy. Therefore, PCA was not utilized in the final model configuration.

5.2.1.4 Artificial Neural Network (ANN) Model:

Parameters and Setup

For the ANN model, we experimented with various configurations to optimize the model's performance. The key parameters included:

- Number of Hidden Layers: We used 4 hidden layers.
- Neurons per Layer: Each hidden layer had 128 neurons.
- Activation Function: The ReLU activation function was used for the hidden layers, and Softmax was used for the output layer.
- Dropout: To prevent overfitting, a dropout rate of 20% was applied after each hidden layer.
- Batch Normalization: Applied after each hidden layer to normalize the inputs of each layer.
- Epochs: The model was trained for 300 epochs.
- Batch Size: We used a batch size of 1000.
- Optimizer: Adam optimizer with a learning rate of 0.001.
- Loss Function: Categorical cross-entropy.

Principal Component Analysis (PCA) was also applied to reduce the dimensionality of the feature set before feeding them into the ANN. However, it was observed that the use of PCA resulted in a lower accuracy. Consequently, PCA was not utilized in the final ANN model configuration.

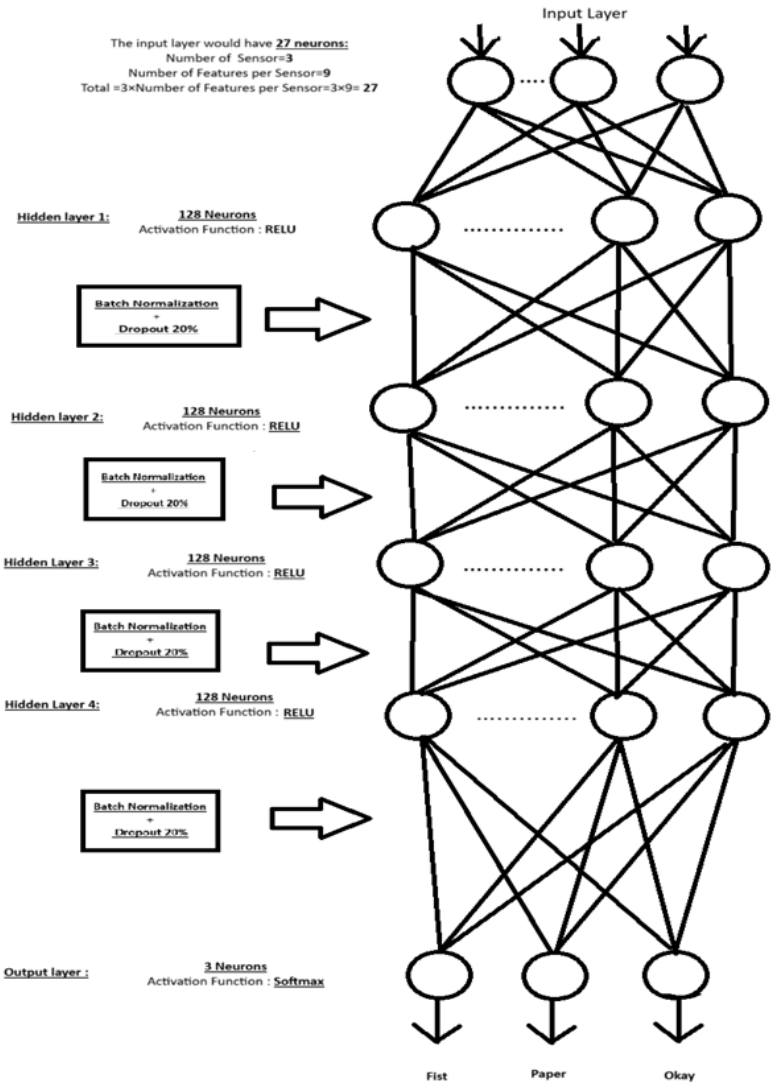

Figure 5.9: Artificial Neural Network Structure for EMG Signal Classification

Results by Window Size

For 150 ms:

- Test Accuracy: 0.8773
- • Classification Report:

	precision	recall	f1-score	support
0	0.94	0.91	0.93	11501
1	0.84	0.86	0.85	9936
2	0.85	0.85	0.85	11251
accuracy			0.88	32688
macro avg	0.88	0.88	0.88	32688
weighted avg	0.88	0.88	0.88	32688

For 200 ms:

- Test Accuracy: 0.9048
- • Classification Report:

	precision	recall	f1-score	support
0	0.95	0.94	0.95	8727
1	0.87	0.89	0.88	7640
2	0.89	0.88	0.88	8685
accuracy			0.90	25052
macro avg	0.90	0.90	0.90	25052
weighted avg	0.91	0.90	0.90	25052

For 250 ms:

- Test Accuracy: 0.9192
- Classification Report:

	precision	recall	f1-score	support
0	0.95	0.95	0.95	7200
1	0.90	0.90	0.90	6163
2	0.90	0.90	0.90	7002
accuracy			0.92	20365
macro avg	0.92	0.92	0.92	20365
weighted avg	0.92	0.92	0.92	20365

For 300 ms:

- Test Accuracy: 0.9375
- Classification Report:

	precision	recall	f1-score	support
0	0.97	0.96	0.96	5713
1	0.91	0.94	0.92	5153
2	0.93	0.92	0.93	5757
accuracy			0.94	16623
macro avg	0.94	0.94	0.94	16623
weighted avg	0.94	0.94	0.94	16623

These results illustrate how different window sizes impacted the ANN model's accuracy and performance metrics. The 300 ms window size yielded the highest test accuracy, indicating its effectiveness in capturing the necessary features for gesture classification.

The accuracies for each window length were calculated and plotted below :

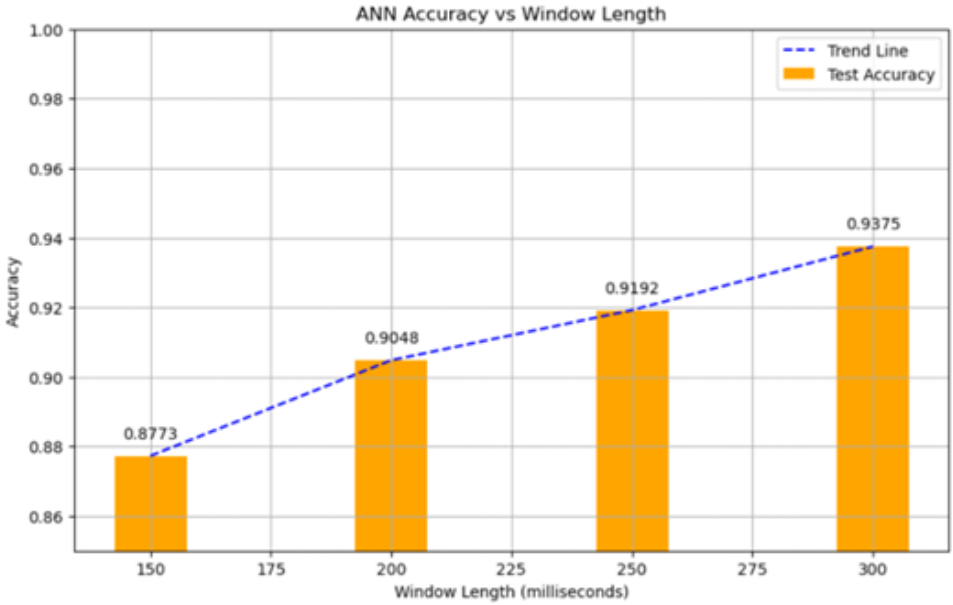

Figure 5.10: ANN Accuracy vs Window Length

5.2.1.5 Comparison of kNN and ANN Models

In this section, we compare the performance of kNN and ANN models across different window lengths. The accuracies of both models for window lengths of 150ms, 200ms, 250ms, and 300ms are illustrated in the bar chart below. The following observations can be made from the chart:

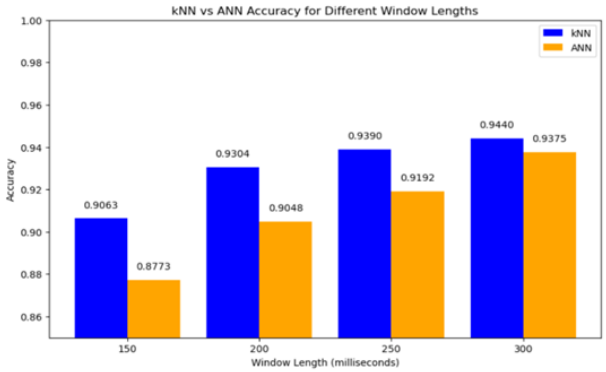

Figure 5.11: kNN vs ANN Accuracy for Different Window Lengths

- For the 150ms window length, the kNN model achieved an accuracy of 90.63%, while the ANN model achieved 87.73%.
- At the 200ms window length, kNN performed slightly better with an accuracy of 93.04% compared to ANN's 90.48%.
- For the 250ms window length, kNN reached 93.90% accuracy, surpassing the ANN model, which achieved 91.92%.
- At the 300ms window length, kNN continued to perform better with an accuracy of 94.40%, while ANN achieved 93.75%.

These results indicate that the kNN model generally outperforms the ANN model across the tested window lengths. However, these accuracy results alone do not justify the selection of the kNN model for our final implementation. The critical factor in our decision-making process is the prediction and feature extraction delays, which will be analyzed in the subsequent section. Our analysis will ensure that we consider not only the accuracy but also the computational efficiency and real-time applicability of both models. This comprehensive evaluation will guide us to choose the most suitable model for our application.

5.2.1.6 5.2.3.4 Prediction Delay Comparison between kNN and ANN Models

The bar chart below illustrates the prediction delay for k-Nearest Neighbors (kNN) and Artificial Neural Network (ANN) models across different window lengths (150 ms, 200 ms, 250 ms, and 300 ms)

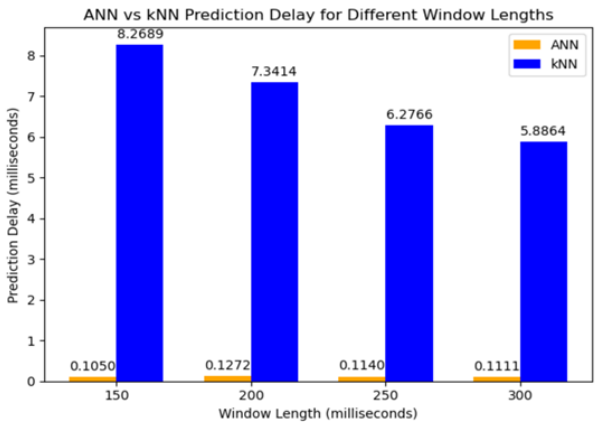

Figure 5.12: ANN vs kNN Prediction Delay for different Window Lengths

5.2.1.7 Key Observations

The ANN model exhibits significantly lower prediction delays compared to the kNN model for all window lengths. At 150 ms, the ANN prediction delay is 0.1050 ms. At 200 ms, the prediction delay slightly increases to 0.1272 ms. For 250 ms and 300 ms window lengths, the ANN model shows prediction delays of 0.1140 ms and 0.1111 ms, respectively. Conversely, the kNN model demonstrates higher prediction delays across all window lengths. At 150 ms, the kNN prediction delay is 8.2689 ms, which is significantly higher than that of the ANN model. The delay reduces as the window length increases: 7.3414 ms at 200 ms, 6.2766 ms at 250 ms, and 5.8864 ms at 300 ms. The ANN model consistently outperforms the kNN model in terms of prediction delay, showcasing much faster prediction times across all tested window lengths. This result suggests that while kNN might offer competitive accuracy, its prediction speed

is notably slower than that of the ANN model. These findings emphasize the importance of considering both accuracy and prediction delay when selecting the optimal model for real-time applications. Additionally, the use of TensorFlow Lite for deploying the ANN model demonstrates its efficiency and suitability for edge devices such as the Raspberry Pi, enabling low-latency predictions in resource-constrained environments. This is particularly useful for applications requiring real-time processing and quick response times.

Voting:

To smooth class decisions, postprocessing techniques were employed for window lengths of 150ms, 200ms, 250ms, and 300ms. Longer window lengths were less likely to overwhelm actuators due to increased prediction delays.

Majority Voting: Feature extraction and prediction delays were consistent across different window sizes, so classifier accuracies determined the number of decisions used in Majority Voting for each classification algorithm. To ensure robust decision-making, we used 150 votes. Both Feedforward ANN and kNN classifiers demonstrated high accuracy levels. Feedforward ANN was preferred for its accuracy and shorter prediction delay. The table below shows suitable classifiers, window sizes, accuracy per vote, and total delay combinations.

Window length (ms)	Number of Votes	ANN Accuracy	kNN Accuracy	ANN Total Delay (ms)	kNN Total Delay (ms)
150.0	150.0	0.8773	0.9063	23832.0	25056.0
200.0	150.0	0.9048	0.9304	31384.5	32466.0
250.0	150.0	0.9192	0.939	39036.0	39961.5
300.0	150.0	0.9375	0.944	46467.0	47334.0

Figure 5.13: Classifier Accuracy and Total Delay for Different Window Lengths

The kNN model generally exhibited higher accuracy compared to the ANN model across all window lengths. For the 300ms window length, the accuracy difference between kNN (0.9440) and ANN (0.9375) is minimal, indicating both models perform well at this window length. The ANN model consistently demonstrated lower total delays compared to the kNN model at all window lengths. For the 300ms window, the ANN model had a total delay of 46467.0 ms, which is significantly lower than the kNN model's total delay of 47334.0 ms. The ANN model with a 300ms window length offers a high accuracy of 0.9375, which is very close to the highest accuracy observed in this study. Lower total delays with the ANN model enhance real-time performance, crucial for applications requiring quick responses. Additionally, ANN models, especially with TensorFlow Lite, are well-suited for deployment on edge devices like the Raspberry Pi, which is advantageous for embedded system applications. Based on these findings, the ANN model with a 300ms window length is selected as the optimal model due to its balance of high accuracy and low total delay, coupled with its suitability for real-time applications and edge device deployment.

CHAPTER SIX
CONCLUSION & FUTURE WORK

In conclusion, we have achieved our aim in which an AI Prosthetic hand through surface electromyography has been developed. Our analysis revealed that the Artificial Neural Network (ANN) model with a 300ms window length is optimal for gesture recognition due to its high accuracy of 93.75Countless engineering challenges had to be analysed and solved in order to achieve our aim. Due to time constraints as full-time students it is needless to say that many things could have been improved in this project to improve the Prosthetic hand's functionality and features, to name a few: - Flex sensors to enable grabbing objects - Eliminating the Arduino Uno and finding a more efficient microprocessor - Size Factor of Servo motors used - Accuracy of the system - More customizable design to fit patient's limb

To conclude, our team is more confident than ever tackling the engineering challenges with making a prosthetic hand. Since the team has completed the project, a lot of ideas were realized and a sense of being comfortable developing this project has grown on us.

APPENDIX

Code Block 6.0.1: Arduino Code For EMG Sampling

```
#include <MyoWare.h>

const long samples = 1000000;  // number of samples

// MyoWare class object
MyoWare myoware;

void setup()
{
  Serial.begin(115200);
  while(!Serial);
  Serial.println('Sampling Test');
  Serial.println('------------------------------');
                                                  // when a
central is connected
}

void loop()
{
  // initialize variables
  int testValue = 0;

  digitalWrite(myoware.getStatusLEDPin(), HIGH);
  Serial.println('Start Time:\t' + String(micros()) + '
microsec');


  unsigned long startMicros = micros(); // variable for
starting time in microseconds
  for (long i = 0; i < samples; i++)
  {

    channel1 = analogRead(A0); //Raw  Sensor 1
    delayMicroseconds(5); //Used to wait for ADC to settle
    channel2 = analogRead(A1); //Raw Sensor 2
    delayMicroseconds(5);
```

```
            channel3 = analogRead(A2); //Raw Sensor 3

            sprintf(buffer, '%d,%d,%d', channel1, channel2,
channel3);
            delayMicroseconds(290);

              }
        const unsigned long endMicros = micros();

        Serial.println('Finish Time:\t' + String(endMicros) + '
microsec');
        digitalWrite(myoware.getStatusLEDPin(), LOW);

        const double secondsPerSample = (endMicros - startMicros)
 / samples;

        Serial.println('Sampling Rate:\t' + String((1.0 /
secondsPerSample) * 1000000) + ' Hz');
        Serial.println('------------------------------');
        delay(1000);
      }
```

The system was then adjusted to 1000Hz

Code Block 6.0.2: Arduino Code For EMG Sampling

```
      char buffer[40]; //Buffer used to store all the values for
sprintF Function

      void setup() {
        Serial.begin(115200); //Fast Serial Communication
Established
      }

      int channel1;
```

```
8         int channel2;
9         int channel3;
10        int endTime;
11
12        void loop() {
13          if (Serial.available() > 0) {
14            String command = Serial.readStringUntil('\n');
15            command.trim(); //Checks for Start Signal from Python
    program
16            if (command == 'START') {
17              recordData();
18            }
19          }
20        }
21
22        void recordData() {
23          endTime = millis() + 5000;
24          while (millis() < endTime) {
25            channel1 = analogRead(A0); //Raw  Sensor 1
26            delayMicroseconds(5); //Used to wait for ADC to settle
27            channel2 = analogRead(A1); //Raw Sensor 2
28            delayMicroseconds(5);
29            channel3 = analogRead(A2); //Raw Sensor 3
30
31            sprintf(buffer, '%d,%d,%d', channel1, channel2,
    channel3);
32            delayMicroseconds(290);
33          }
34          Serial.println('STOP'); // Stop command sent to Python
    Program to signal finishing the sampling of 5 secs
35
36
37
38
```

APPENDIX II

Code Block 6.0.3: EMG Data Collection Script

```
import serial
import time
import csv
from time import sleep
# Set the serial port and baud rate
SERIAL_PORT = 'COM4'
BAUD_RATE = 115200
# Initialize the serial connection
ser = serial.Serial(SERIAL_PORT, BAUD_RATE)
class colors:
    RESET = '\033[0m'
    RED = '\033[91m'
    GREEN = '\033[92m'
    # Function to get user input for person ID and gender
def get_user_info():
    person_id = input('Enter the Person ID: ')
    gender = input('Enter the Gender (Male/Female): ')
    return person_id, gender
    # Initialize the error counter
    error_count = 0
    # Define a function to record data for a specified duration
    with a stopwatch
def record_period(duration, label, round_number, gesture_name,
    writer, person_id, gender):
    global error_count
    print(f'{colors.GREEN}Start in 2 seconds{colors.RESET}')
    sleep(2)  # Give time to prepare before recording starts
    print(f'{colors.RED}Recording {label} for {gesture_name}, Round
    {round_number}.{colors.RESET}')
    # Send start command to Arduino
    ser.write(b'START\n')

    start_time = time.time()  # Record the start time
    while True:
    line = ser.readline().decode('utf-8').strip()
    if line == 'STOP':
```

```
34          break
35      if line:
36          sensorValues = line.split(',')
37          if len(sensorValues) >= 3:
38              try:
39                  sensorValue1 = int(sensorValues[0])
40                  sensorValue2 = int(sensorValues[1])
41                  sensorValue3 = int(sensorValues[2])
42      elapsed_time = time.time() - start_time  # Calculate elapsed
    time
43      elapsed_time_str = f'{elapsed_time:.6f}'  # Format elapsed time
     to 6 decimal places for microseconds
44      writer.writerow([elapsed_time_str, sensorValue1, sensorValue2,
    sensorValue3, label, round_number, gesture_name, person_id,
    gender])
45      except ValueError:
46                      print('Error parsing sensor values: Skipping
    this line.')
47                      error_count += 1  # Increment the error counter
48              else:
49                  print('Incomplete data received, skipping...')
50          else:
51              print('Empty line received, skipping...')
52  def main():
53  global error_count
54  # Open a CSV file to store the data
55  person_id, gender = get_user_info()
56  with open('emg_data.csv', 'w', newline='') as csvfile:
57  writer = csv.writer(csvfile)
58  writer.writerow(['Elapsed Time (s)', 'Sensor1', 'Sensor2', 'Sensor3
    ', 'Label', 'Round', 'Gesture', 'Person ID', 'Gender'])
59  gestures = ['Fist','Thumb Finger','Index Finger','Scissors','Paper'
    ,'Three','Four','Okay','Finger Gun']
60  GESTURE_DURATION = 5  # duration for each gesture recording in
    seconds
61  SHORT_REST = 10  # short rest between rounds in seconds
62
63  for gesture_name in gestures:
64  input(f'Press Enter to start sequence for {gesture_name} or Ctrl+C
    to exit.')
```

```
for round_number in range(1, 5):
for _ in range(5):
    record_period(GESTURE_DURATION, 'Rest', round_number, 'Rest',
    writer, person_id, gender)
        record_period(GESTURE_DURATION, 'Non-Rest', round_number,
    gesture_name, writer, person_id, gender)
            if round_number < 4:
            print(f'Short rest of {SHORT_REST} seconds.')
            sleep(SHORT_REST)

            if gesture_name \!= gestures[-1]:
                print('Long rest of 3 minutes until next gesture.')
                sleep(120)
                print('1 min Remaining')
            sleep(60)

print('Data collection complete for all gestures.Processing data.')
print('.....')
print('Rename CSV File for current patient so the data doesn't get
    overwritten with the next patient's data')
print(f'Total parsing errors encountered: {error_count}')
if _name_ == '_main_':
 main()


```

APPENDIX III

Code Block 6.0.4: Data Normalization

```
myo=table2array(emg);

myoware_data2=myo(206185:220515,2);
myoware_data3=myo(206185:220515,3);
myoware_data1=myo(206185:220515,1);


  for j=1:length(myoware_data2)
      if (myoware_data2(j)>1024||myoware_data2(j)<100)
          myoware_data2(j)=myoware_data2(j-1);

      end
      if (myoware_data1(j)>1024||myoware_data1(j)<100)
          myoware_data1(j)=myoware_data1(j-1);

      end

       if (myoware_data3(j)>1024||myoware_data3(j)<100)
          myoware_data3(j)=myoware_data3(j-1);

      end
  end

  segment_length=50;

  numsegments1=floor(length(myoware_data1)/(segment_length)); %
   calculates the number of segments

  rmsvalues1=zeros(1,numsegments1) % array to store rms values for
   each segment

  %calculates the rms for each segment
  for i = 1:numsegments1

      startIndex = (i - 1) * (segment_length) + 1;
      endIndex = startIndex + segment_length - 1;
```

```
35        segment_Data1= myoware_data1(startIndex:endIndex);
36        rmsvalues1(i) = rms(segment_Data1);
37    end
38
39
40    figure;
41
42    plot(rmsvalues1)
43    xlabel('Time(samples)')
44    ylabel('rms')
45    title('RMS values of 50 samples for 1 round fist-rest -sensor 1')
46 %%
47    numsegments2=floor(length(myoware_data2)/(segment_length)); %
       calculates the number of segments
48  %%numsegments1=2504635;
49
50 rmsvalues2=zeros(1,numsegments2) % array to store rms values for
      each segment
51
52 %calculates the rms for each segment
53 for t = 1:numsegments2
54
55     startIndex2 = (t - 1) * (segment_length) + 1;
56     endIndex2 = startIndex2 + segment_length - 1;
57     segment_Data2= myoware_data2(startIndex2:endIndex2);
58     rmsvalues2(t) = rms(segment_Data2);
59 end
60
61
62 figure;
63
64 plot(rmsvalues2,'r')
65 xlabel('Time(samples)')
66 ylabel('rms')
67 title('RMS values of 50 samples for 10s Finger Gun-sensor2')
68 %%
69
70 numsegments3=floor(length(myoware_data3)/(segment_length)); %
      calculates the number of segments
71
```

```
rmsvalues3=zeros(1,numsegments3) % array to store rms values for
    each segment

%calculates the rms for each segment
for k = 1:numsegments3

    startIndex3 = (k - 1) * (segment_length) + 1;
    endIndex3 = startIndex3 + segment_length - 1;
    segment_Data3= myoware_data3(startIndex3:endIndex3);
    rmsvalues3(k) = rms(segment_Data3);
end


figure;

plot(rmsvalues3,'g')
xlabel('Time(samples)')
ylabel('rms')
title('RMS values of 50 samples for 10s Finger Gun -sensor3')


```

APPENDIX IV

Code Block 6.0.5: AI MODEL

```
        import serial
import time
import numpy as np
import pandas as pd
from scipy.signal import welch
from scipy.stats import skew
import tensorflow as tf
import joblib
from sklearn.preprocessing import LabelEncoder
from collections import deque
import RPi.GPIO as GPIO
from time import sleep
from adafruit_servokit import ServoKit

SERIAL_PORT = '/dev/ttyUSB0'
BAUD_RATE = 115200

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
COMMAND_PIN = 21
GPIO.setup(COMMAND_PIN, GPIO.OUT)

ser = serial.Serial(SERIAL_PORT, BAUD_RATE)
time.sleep(2)

interpreter = tf.lite.Interpreter(model_path='/home/picap/Desktop/
    m1.tflite')
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
scaler = joblib.load(r'/home/picap/Desktop/s1.pkl')

vote_queue = deque(maxlen=100)
label_encoder = LabelEncoder()
unique_labels = ['Rest', 'Fist', 'Paper', 'Okay']
label_encoder.fit(unique_labels)
```

```
last_voted_prediction = None

buffer = []
fs = 1070  # Sampling frequency
window_length = int(0.30 * fs)  # 300 ms window
increment = int(0.030 * fs)  # 30 ms increment
threshold = 40.90
ThumbFinger = 0
IndexFinger = 1
MiddleFinger = 2
RingFinger = 3
PinkyFinger = 4
Thumb_Extension = 5
kit = ServoKit(channels=8)
dl = 4
for i in range(5):
    kit.servo[i].set_pulse_width_range(500, 2500)
kit.servo[5].set_pulse_width_range(500, 2400)

def RestGesture():
    kit.servo[ThumbFinger].angle = 30
    kit.servo[IndexFinger].angle = 30
    kit.servo[MiddleFinger].angle = 30
    kit.servo[RingFinger].angle = 30
    kit.servo[PinkyFinger].angle = 30
    kit.servo[Thumb_Extension]=0
    time.sleep(1)

def FistGesture():
    kit.servo[ThumbFinger].angle = 180
    kit.servo[IndexFinger].angle = 180
    kit.servo[MiddleFinger].angle = 180
    kit.servo[RingFinger].angle = 180
    kit.servo[PinkyFinger].angle = 180
    kit.servo[Thumb_Extension]= 90
    time.sleep(dl)
    RestGesture()

def PaperGesture():
    kit.servo[ThumbFinger].angle = 0
```

```
76      kit.servo[IndexFinger].angle = 0
77      kit.servo[MiddleFinger].angle = 0
78      kit.servo[RingFinger].angle = 0
79      kit.servo[PinkyFinger].angle = 0
80      kit.servo[ Thumb_Extension]= 90
81      time.sleep(dl)
82      RestGesture()
83
84  def OkayGesture():
85      kit.servo[ThumbFinger].angle = 180
86      kit.servo[IndexFinger].angle = 180
87      kit.servo[MiddleFinger].angle = 0
88      kit.servo[RingFinger].angle = 0
89      kit.servo[PinkyFinger].angle = 0
90      kit.servo[ThumbExtension].angle = 0
91      time.sleep(dl)
92      RestGesture()
93
94  def calculate_mav(segment):
95      mav_sensor1 = np.mean(np.abs(segment[:, 0]))
96      mav_sensor2 = np.mean(np.abs(segment[:, 1]))
97      mav_sensor3 = np.mean(np.abs(segment[:, 2]))
98      return (mav_sensor1 + mav_sensor2 + mav_sensor3) / 3
99
100 def calculate_features(segment):
101     features = {}
102
103     ssc_threshold = 0.01
104     features['SSC Sensor1'] = ((np.diff(segment[:, 0][:-1]) * np.
    diff(segment[:, 0][1:]) < 0) &
105                                (np.abs(np.diff(segment[:, 0][:-1]) -
    np.diff(segment[:, 0][1:])) >= ssc_threshold)).sum()
106     features['SSC Sensor2'] = ((np.diff(segment[:, 1][:-1]) * np.
    diff(segment[:, 1][1:]) < 0) &
107                                (np.abs(np.diff(segment[:, 1][:-1]) -
    np.diff(segment[:, 1][1:])) >= ssc_threshold)).sum()
108     features['SSC Sensor3'] = ((np.diff(segment[:, 2][:-1]) * np.
    diff(segment[:, 2][1:]) < 0) &
109                                (np.abs(np.diff(segment[:, 2][:-1]) -
    np.diff(segment[:, 2][1:])) >= ssc_threshold)).sum()
```

```
110
111     features['RMS Sensor1'] = np.sqrt(np.mean(segment[:, 0]**2))
112     features['RMS Sensor2'] = np.sqrt(np.mean(segment[:, 1]**2))
113     features['RMS Sensor3'] = np.sqrt(np.mean(segment[:, 2]**2))
114
115     features['WL Sensor1'] = np.sum(np.abs(np.diff(segment[:, 0])))
116     features['WL Sensor2'] = np.sum(np.abs(np.diff(segment[:, 1])))
117     features['WL Sensor3'] = np.sum(np.abs(np.diff(segment[:, 2])))
118
119     features['Skewness Sensor1'] = skew(segment[:, 0])
120     features['Skewness Sensor2'] = skew(segment[:, 1])
121     features['Skewness Sensor3'] = skew(segment[:, 2])
122
123     features['Log Detector Sensor1'] = np.exp(np.mean(np.log(np.abs
    (segment[:, 0]) + 1e-10)))
124     features['Log Detector Sensor2'] = np.exp(np.mean(np.log(np.abs
    (segment[:, 1]) + 1e-10)))
125     features['Log Detector Sensor3'] = np.exp(np.mean(np.log(np.abs
    (segment[:, 2]) + 1e-10)))
126
127     features['TM4 Sensor1'] = np.mean((segment[:, 0] - np.mean(
    segment[:, 0]))**4)
128     features['TM4 Sensor2'] = np.mean((segment[:, 1] - np.mean(
    segment[:, 1]))**4)
129     features['TM4 Sensor3'] = np.mean((segment[:, 2] - np.mean(
    segment[:, 2]))**4)
130
131     def hjorth_params(signal):
132         first_deriv = np.diff(signal)
133         second_deriv = np.diff(first_deriv)
134         var_zero = np.var(signal)
135         var_d1 = np.var(first_deriv)
136         var_d2 = np.var(second_deriv)
137         activity = var_zero
138         mobility = np.sqrt(var_d1 / var_zero)
139         complexity = np.sqrt(var_d2 / var_d1) / mobility
140         return mobility, complexity
141
142     features['Hjorth Mobility Sensor1'], features['Hjorth
    Complexity Sensor1'] = hjorth_params(segment[:, 0])
```

```
    features['Hjorth Mobility Sensor2'], features['Hjorth Complexity Sensor2'] = hjorth_params(segment[:, 1])
    features['Hjorth Mobility Sensor3'], features['Hjorth Complexity Sensor3'] = hjorth_params(segment[:, 2])

    def frequency_features(signal, fs):
        f, Pxx = welch(signal, fs, nperseg=min(256, len(signal)))
        total_power = np.sum(Pxx)
        mean_freq = np.sum(f * Pxx) / total_power if total_power != 0 else 0
        return mean_freq

    features['MNF Sensor1'] = frequency_features(segment[:, 0], fs)
    features['MNF Sensor2'] = frequency_features(segment[:, 1], fs)
    features['MNF Sensor3'] = frequency_features(segment[:, 2], fs)

    return features

print('System initialized')
while True:
    try:
        line = ser.readline().decode('utf-8').strip()
        if line == 'STOP':
            continue
        if line:
            sample = list(map(int, line.split(',')))
            buffer.append(sample)

        if len(buffer) >= window_length:
            segment_data = np.array(buffer[:window_length])
            buffer = buffer[increment:]

            mav_average = calculate_mav(segment_data)

            if mav_average < threshold:
                pred_label = 'Rest'
            else:
                features = calculate_features(segment_data)
                X_test = pd.DataFrame([features]).values
                X_test_scaled = scaler.transform(X_test)
```

```
180
181                 interpreter.set_tensor(input_details[0]['index'],
    X_test_scaled.astype(np.float32))
182                 interpreter.invoke()
183                 output_data = interpreter.get_tensor(output_details
    [0]['index'])
184                 y_pred_class = np.argmax(output_data, axis=1)[0]
185
186                 pred_label = label_encoder.inverse_transform([
    y_pred_class])[0]
187
188             print(f'Predicted Gesture: {pred_label}')
189
190             if pred_label == 'Rest':
191                 RestGesture()
192             elif pred_label == 'Fist':
193                 FistGesture()
194             elif pred_label == 'Paper':
195                 PaperGesture()
196             elif pred_label == 'Okay':
197                 OkayGesture()
198
199             GPIO.output(COMMAND_PIN, GPIO.LOW)
200
201     except KeyboardInterrupt:
202         break
203     except Exception as e:
204         print(f'Error: {e}')
205
206 ser.close()
207 GPIO.cleanup()
208
209
```