PATIENT SPECIFIC HAND GESTURE RECOGNITION BASED ON ARTIFICIAL FEED FORWARD NEURAL NETWORK AND EMG

A DISSERTATION-I REPORT

Submitted by

MOHAMMED YASAR ARAFATH A

in partial fulfilment for the award of the degree of

MASTER OF ENGINEERING

in

MECHATRONICS

DEPARTMENT OF PRODUCTION TECHNOLOGY

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY, CHENNAI 600 044.

DECEMBER 2024

ANNA UNIVERSITY, CHENNAI 600 044

BONAFIDE CERTIFICATE

Certified that this Report titled **"PATIENT SPECIFIC HAND GESTURE RECOGNITION BASED ON ARTIFICIAL FEED FORWARD NEURAL NETWORK AND EMG"** is the bonafide work of **MOHAMMED YASAR ARAFATH A (2023608031)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. J. JANCIRANI	**Dr. P. GANESH**
Professor and Head	**Associate Professor and Supervisor**
Department of Production Technology	Department of Production Technology
Madras Institute of Technology	Madras Institute of Technology
Anna University, Chennai – 600 044	Anna University, Chennai – 600 044

ACKNOWLEDGEMENT

It is a pleasure to place on record our profound sense of gratitude to my project supervisor **Dr. P. GANESH**, Associate Professor and project panel members **Dr. S. VIJAYAKUMAR,** Associate Professor, **Dr. P. KARTHIKEYAN,** Assistant Professor (Sr.Gr.), **Dr. C. ARUN PRAKASH,** Assistant Professor, **Mr. R. MATHIYAZHAGAN**, Teaching Fellow and **Mr. S. MOHAMED SHAZULI**, Teaching Fellow and staff members TANII **Mr. A. VINOTH** and **Mr. P. VIGNESH**, for their valuable guidance, advice, motivation and encouragement which helped me immensely to accomplish this project successfully.

I sincerely thank all the Teaching and Non- Teaching staff members, Department of Production Technology, Madras Institute of Technology Campus, for their valuable suggestions and constant support and also thank my friends and family for their valuable suggestions and their help provided during this project.

I express deep sense of gratitude to our Head of the Department and the project coordinator **Dr. J. JANCIRANI**, Professor, Department of Production Technology, Madras Institute of Technology for her valuable and in depth experienced advises which helped me in completion of this project successfully.

I express heartful gratitude to **Dr. K. RAVICHANDRAN**, Dean, Madras Instituteof Technology, Anna University, Chennai for providing me an opportunity to carry out the project work with all the facilities.

MOHAMMED YASAR ARAFATH A

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVATIONS

EMG	Electromyography
VMD	Variational Mode Decomposition
IMF	Intrinsic Mode Function
SNR	Signal-to-Noise Ratio
SD	Standard Deviation
RMS	Root Mean Square
ZC	Zero Crossing
ANN	Artificial Neural Network
STFT	Short-Time Fourier Transform
CWT	Continuous Wavelet Function
PSD	Power Spectral Density
FNN	Feed-forward Neural Network

திட்டப்பணி சுருக்கம்

இந்த திட்டத்தில், ஒரு நோயாளி குறிப்பிட்ட நேரடி கை இயக்கத்தை அடையாளம் காணும் மாதிரியை முன்மொழிகிறது. இந்த மாதிரி, 10 அடுக்குகளுடன் கூடிய க முன்னேற்ற நரம்பியல் நெட்வொர்க் அடிப்படையில் செயல்படுகிறது மற்றும் கைகள் மத்தியில் இருந்து ஒரு மின் மயோ கிராஃபி (EMG) சிக்னல்களைப் பயன்படுத்துகிறது. முன்மொழியப்பட்ட மாதிரி ஆறு மாட்யூல்களை உள்ளடக்கியது: பையோ-அம்ப் பிள்ளையைப் பயன்படுத்தி எக்ஸ் ஜி சிக்னலின் பெறுதல், முன்-செயலாக்கம், பிரிப்பு, அம்சங்கள் எடுக்குதல், வகைப்படுத்தல் மற்றும் குறுக்கு சரிபார்ப்பு. முன்மொழியப்பட்ட மாதிரியில், முன்னேற்ற நரம்பியல் நெட்வொர்க் ஐப் பயன்படுத்தி tripod, பிடிப்பு மற்றும் சக்தி பிடிப்பு போன்ற இயக்கங்களை அடையாளம் காண 87.5% முதல் 93.5% வரை துல்லியமுள்ளது. எடுக்கப்பட்ட அம்சங்கள் நேரம் மற்றும் அதிர்வெண் துறைகளை அடிப்படையாகக் கொண்டவை. இந்த வேலைக்கு முக்கியமான பங்களிப்புகள் (1) ஒப்பீட்டுக் கையாளும் மாதிரி, (2) சிக்னலின் அம்சங்களை எடுக்குவதற்கான செயல்முறை நேரம் மற்றும் அதிர்வெண் துறைகளில் செய்யப்பட்டது.

ABSTRACT

In this project proposed a patient specific real-time hand gesture recognition model. This model works based on artificial feed forward neural network with 10 layers and an electromyography (EMG) signal from the forearm. The proposed model is composed of six modules: acquisition of emg signal using the Bip-Amp EXG pill, pre-processing, segmentation, feature extraction, classification, and cross validation. The proposed model has an accuracy of 87.5% to 93.5% at recognizing gestures like tripod, pinch and power grip by using feed forward neural network. The features extracted are based on both time and frequency domain. The main contributions of this work include (1) a hand gesture recognition model that responds relatively good accuracy, (2) the process of feature extraction of signal was done in both time and frequency domains.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

A prosthetic hand is an artificial device designed to replace the function and appearance of a missing hand. Today, prosthetic hands are equipped with advanced technologies such as myoelectric sensors, which detect muscle signals from the residual limb to control the hand’s movements. These devices aim to restore a range of motion, improve grip strength, and enhance the overall quality of life for individuals who have lost a hand.

The development of prosthetic hands has evolved significantly over the years, driven by advancements in technology and a deeper understanding of human anatomy and biomechanics. Prosthetic hands serve as vital assistive devices for individuals who have experienced upper limb loss due to injury, disease, or congenital conditions. These devices not only restore basic functionality but also enhance the quality of life for users by enabling them to perform daily activities with greater independence and confidence.

Modern prosthetic hands range from simple mechanical devices to sophisticated bionic systems that mimic the complex movements of a natural hand. Traditional prosthetics often lack the dexterity and intuitive control required for nuanced tasks, leading to frustration among users. However, recent innovations have introduced myoelectric prostheses that utilize electromyographic (EMG) signals from residual muscles to control finger movements. This approach allows for more precise and versatile hand functions, enabling users to grasp objects with varying shapes and sizes.

Furthermore, the integration of advanced algorithms enhances the control mechanisms of prosthetic hands, allowing for adaptive grasping strategies that respond to real-time feedback from the environment. For instance, tactile sensors can provide sensory feedback, simulating the sense of touch and enabling users to gauge pressure and texture when handling objects. This sensory integration is crucial for performing delicate tasks without damaging items or dropping them.

As research continues to push the boundaries of what is possible in prosthetic design, the focus is increasingly on creating personalized solutions that cater to individual user needs, which works more specific for them.

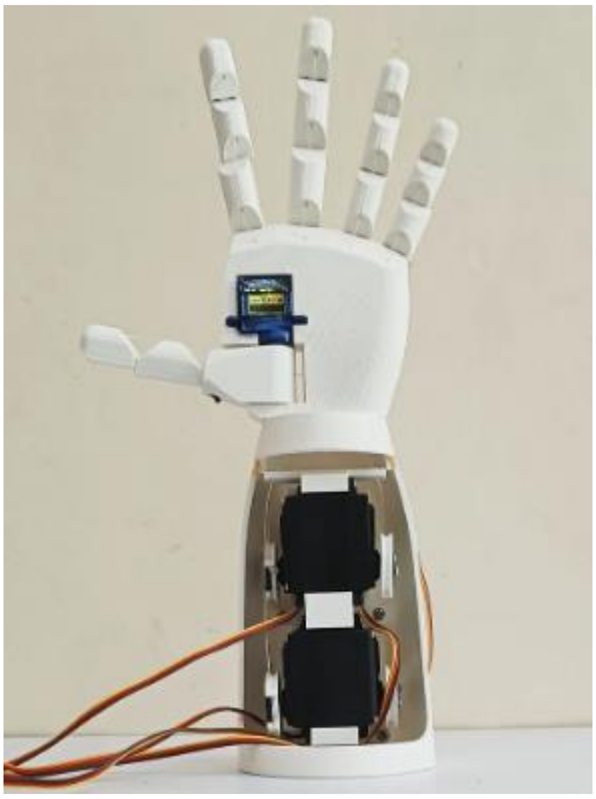


Figure 1.1 Prosthetic Hand

1.2 TYPES OF PROSTHETICS

1. **Body-powered prosthetics:** use cables and harnesses controlled by body movements.
2. **Myoelectric prosthetics:** detect electrical signals from muscle contractions in the residual limb to control movements via motors.
3. **Advanced prosthetics:** are exploring sensory haptic feedback to give users a sense of touch and more natural control.

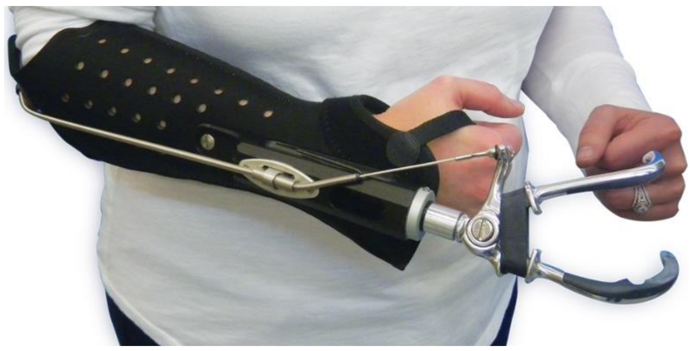


Figure 1.2 Body- Powered Prosthetic Hand

1.3 ELECTROMYOGARPHY(EMG)

Electromyography (EMG) is a technique used to measure the electrical activity of muscles, providing valuable insights into neuromuscular function. It involves placing electrodes on the skin surface or implanting them within muscle tissue to capture electrical signals generated during muscle contractions. EMG signals can be analyzed to understand muscle activation patterns and are increasingly utilized in the control of prosthetic devices, enabling more intuitive and responsive operation. Recent advancements in machine learning and pattern recognition have significantly improved the accuracy of interpreting these signals, allowing for better gesture recognition in prosthetic hands and limbs. Despite these advancements, challenges remain, including signal variability, electrode placement issues, and the need for real-time processing capabilities.

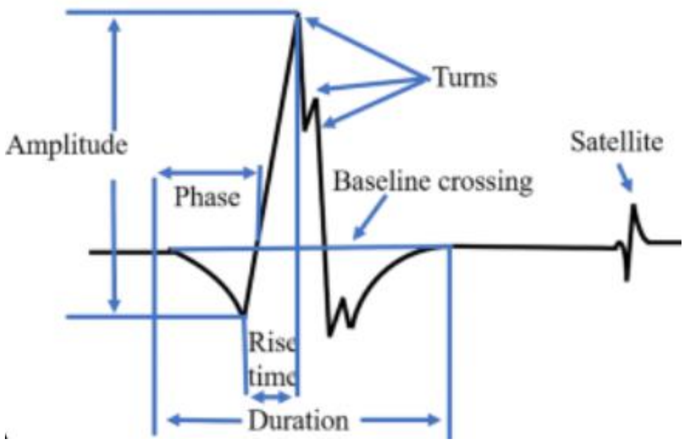


Figure 1.3 EMG Signal

1.4 SENSOR TO RECORD EMG SIGNAL

The BioAmp EXG Pill is a compact, professional-grade analog front-end module designed for biosensing applications, capable of recording high-quality biopotential signals such as ECG (electrocardiography), EMG (electromyography), EOG (electrooculography), and EEG (electroencephalography). This versatile device operates within an input voltage range of 4.5 to 40 V and features an high input impedance making it suitable for various microcontroller. Its small size facilitates easy integration into mobile and space-constrained projects, and it boasts powerful noise rejection capabilities, allowing for reliable signal acquisition even near AC mains supply. The BioAmp EXG Pill comes with all necessary components, including gel electrodes and cables, enabling users to conduct a wide range of biosensing experiments and applications, from health monitoring to innovative control systems in robotics.

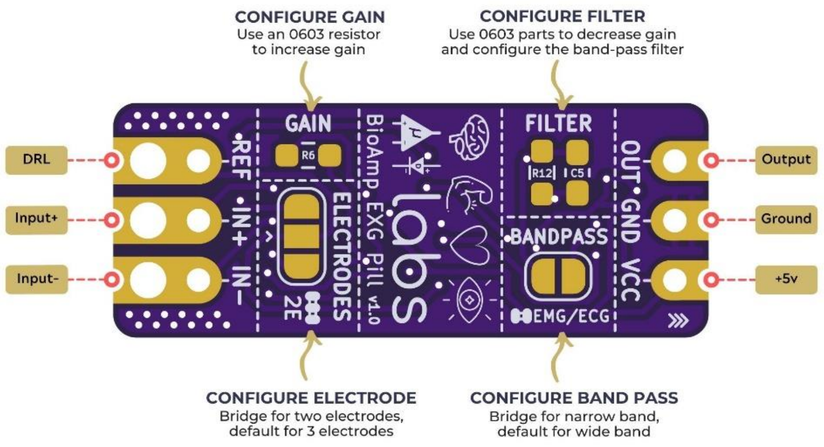

CONFIGURE GAIN
Use an 0603 resistor to increase gain
CONFIGURE FILTER
Use 0603 parts to decrease gain and configure the band-pass filter
DRL
Input+
Input-
Output
Ground
+5v
GAIN
R6
FILTER
R12
C5
BANDPASS
EMG/ECG
REF
IN+
IN-
ELECTRODES
2E
BioAmp EXG Pill v1.0
labs
OUT
GND
VCC
CONFIGURE ELECTRODE
Bridge for two electrodes, default for 3 electrodes
CONFIGURE BAND PASS
Bridge for narrow band, default for wide band

Figure 1.4 BioAmp EXG Pill

1.4.1 Connection to Acquire EMG Signal

1) The two electrodes (Positive and Negative)- should be placed along the muscle selected to be measured.

2) The last electrode (Reference)- should be placed at the non-muscular portion.

3) The pill is powered with 5v, GND pin- to the ground pin and output pin is connected to an analog input pin of your microcontroller. i.e., A0 pin

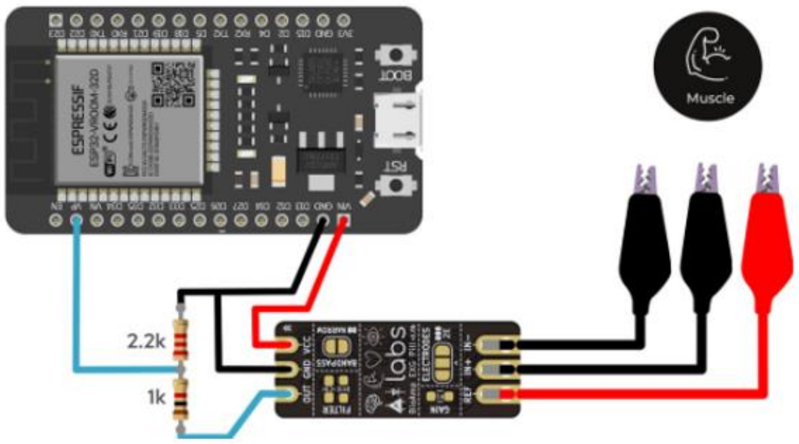


Figure 1.5 Connection for BioAmp EXG Pill to Record EMG

`

CHAPTER 2

LITERATURE REVIEW

2.1 LITERATURE REVIEW ON PROSTHETIC HAND AND ALGORITHM

Karla et al., (2024), proposed an innovative approach to creating a cost-effective prosthetic hand. Utilizing surface electromyography (sEMG) signals, the bionic hand interprets muscle activity to perform various gestures. The design leverages 3D printing technology with low-cost materials like Poly Lactic Acid (PLA), making it customizable and accessible. An 8-channel sEMG sensor captures muscle signals, which are processed using a neural network trained to recognize different hand movements. The system achieves approximately 85% accuracy in gesture recognition, enabling real-time control of the prosthetic hand. The authors highlight challenges such as sensor placement variability and the need for consistent muscle activation, suggesting further refinements for practical applications. Overall, this research demonstrates the potential of integrating advanced technologies in developing functional and affordable prosthetic devices.

Marianne et al., (2023), explores various strategies for enhancing the quality of single-channel electromyography (EMG) signals, which are often compromised by noise and artifacts. A significant focus of the review is on Variational Mode Decomposition (VMD), a technique that effectively decomposes noisy EMG signals into multiple band-limited modes, allowing for targeted noise reduction. VMD has demonstrated superior performance in filtering out common types of noise such as powerline interference, baseline wandering, and white Gaussian noise, particularly through methods like wavelet soft thresholding and iterative interval thresholding. The authors highlight that VMD not only improves the signal-to-noise ratio but also preserves the essential characteristics of the original EMG signal, making it a promising approach for applications in gesture recognition and other fields requiring precise EMG analysis.

Priya et al., (2023), explores the integration of machine learning techniques to enhance the control of prosthetic arms using electromyographic (EMG) signals from the residual limb of amputees. The authors emphasize that traditional myoelectric prostheses often present challenges in usability, including difficulty in controlling the device and high cognitive load for the user. By employing advanced machine learning algorithms, such as real-time prediction learning and adaptive control methods, the study aims to improve the responsiveness and intuitiveness of prosthetic control systems. The review highlights significant findings from existing literature, indicating that adaptive control strategies can significantly reduce the number of switches and total switching time required for operating prosthetic devices, thereby enhancing user experience. Furthermore, the authors discuss various machine learning frameworks that leverage movement synergies between proximal and distal joints to facilitate natural control of multi-degree-of-freedom (DoF) prosthetic arms. This approach not only aims to improve precision and robustness but also seeks to lower the cognitive burden on users during daily activities.

Yanchao et al., (2022), explores a prosthetic hand system that enhances adaptive grasping through myoelectric pattern recognition. The proposed system focuses on interpreting electromyographic (EMG) signals to enable the prosthetic hand to adjust its grip based on the user's intended actions. By implementing two closed-loop control loops, the design allows for precise manipulation of various objects, improving the functionality and user experience of the prosthetic device. The authors address existing challenges in prosthetic technology, such as limited adaptability and control accuracy, and demonstrate how their approach effectively overcomes these limitations. The research emphasizes the importance of real-time signal processing and adaptive control strategies, contributing to advancements in prosthetic technology aimed at providing users with a more intuitive and effective tool for everyday tasks.

Wand et al., (2022), investigates the effectiveness of feedforward artificial neural networks (ANNs) in controlling myoelectric prosthetic hands through EMG signals. The authors demonstrate that state-of-the-art deep neural networks, particularly those with multiple hidden layers, significantly outperform traditional linear methods in achieving proportional control of prosthetic movements. By training the ANN on data from both able-bodied subjects and prosthesis users, the study highlights the network's ability to accurately map EMG features to desired hand movements, thereby enhancing the usability and functionality of prosthetic devices. The findings reveal that the architecture of the feedforward network, characterized by its simplicity and direct information flow, contributes to improved performance in controlling simultaneous movements. This research underscores the potential of machine learning approaches, particularly feedforward ANNs, to revolutionize prosthetic technology and improve quality of life for amputees.

Ahmed et al., (2021), emphasize the challenges faced in traditional myoelectric control systems, such as variability in EMG signal patterns and the limitations of existing classification methods. They present a comprehensive overview of various machine learning algorithms that can be applied to improve gesture recognition and control accuracy in prosthetic devices. The paper highlights the importance of feature extraction and selection processes, which are critical for enhancing model performance. Additionally, it addresses the need for real-time processing capabilities to facilitate seamless interaction between users and their prosthetic limbs. The authors conclude by identifying future research directions aimed at integrating more sophisticated machine learning approaches and improving user adaptability to these advanced control systems, ultimately enhancing the functionality and usability of prosthetic arms for amputees.

`

Wei et al., (2021), explores the integration of surface electromyography (sEMG) signals with deep learning techniques to enhance the control of prosthetic hands. It highlights the significant impact of upper limb amputation on quality of life and emphasizes the need for more natural control systems in prosthetics. The authors discuss how sEMG signals can effectively predict motor intentions, which is crucial for prosthetic functionality and review various deep learning methodologies that improve sEMG pattern recognition, covering aspects such as signal acquisition, preprocessing, feature extraction, and classification. The paper also addresses existing challenges in the field, including computational demands and the limitations of current commercial prosthetic devices. Finally, it outlines future research directions aimed at improving gesture recognition accuracy and clinical applicability of these advanced control systems.

Nawadita et al., (2019), provides a thorough examination of the advancements and challenges in utilizing electromyography (EMG) for controlling hand prostheses. The authors highlight the evolution of pattern recognition techniques, emphasizing the critical role of real-time processing in enhancing user experience and functionality. They discuss various methodologies including signal preprocessing, data segmentation, feature extraction, and classification approaches that are essential for accurate gesture recognition. Notably, the paper underscores the importance of increasing the number of EMG channels, as studies have shown that using more channels can significantly improve control accuracy and reduce classification errors. Additionally, the review addresses practical challenges such as user adaptability, system calibration, and the need for intuitive interfaces that facilitate seamless interaction between users and prosthetic devices. The authors also explore future directions for research, advocating for the integration of advanced machine learning algorithms and real-time feedback mechanisms to further enhance the responsiveness and reliability of EMG-based control systems.

`

Marco et al., (2018), works on user specific which use 5 instances for each of the 6 classes so totally 30 samples. EMG signals are recorded for 2 s of measurement. Works based on feed-forward Neural Network (NN) and use low-pass Butterworth filter for pre-processing and for feature extraction use Dynamic Time Warping (DTW). A hand band is developed to record the signals. Research of new algorithms to find automatically the centers of the clusters of each category for the feature extraction module. Another problem to research about is the use of recurrent neural networks for the classification module in order to improve the recognition accuracy using context information.

Rubana et al., (2013), provides a comprehensive overview of the methodologies employed in processing and classifying surface electromyography (sEMG) signals, which are crucial for understanding muscle activation patterns. The authors discuss the fundamental principles of sEMG, including the physiological basis of muscle signals and the importance of accurate data acquisition. They highlight various signal processing techniques such as filtering, wavelet transforms, and feature extraction methods that are essential for enhancing signal quality and reducing noise. A significant focus is placed on the Continuous Wavelet Transform (CWT), which offers a time-frequency representation of sEMG signals, allowing for the analysis of non-stationary signals that exhibit varying frequency components over time. CWT effectively extracts detailed features from sEMG by generating scalograms that visualize energy distribution across different frequencies and time intervals, thereby improving clinical interpretation and classification accuracy. The review emphasizes the role of machine learning algorithms, including support vector machines (SVM), neural networks, and decision trees, in effectively classifying complex sEMG data for applications in rehabilitation, prosthetics, and human-computer interaction. Additionally, the authors address the challenges faced in real-time applications, such as the need for robust algorithms that can adapt to individual users' muscle patterns and the importance of developing intuitive interfaces for better user experience.

`

2.2 RESEARCH GAP

i. Literature Review - Most prosthetic hands require users to undergo extensive training and adaptation, which can be a challenging process.
ii. Need - Extensive training required to learn to control a prosthetic arm using their muscle signals.
iii. Solution – A patient specific prosthetic hand, which works for individual efficiently with less training, as algorithm developed by their specific emg signals.

2.3 OBJECTIVES OF THE PROJECT

The objective of this project is:

i. To develop a patient specific prosthetic hand for controlling the hand movements using pattern recognition by implementing Artificial Neural Network.

CHAPTER 3

METHODOLOGY

3.1 METHODOLOGY FOR PATIENT SPECIFIC HAND GESTURE RECOGNITION BASED ON ARTIFICIAL FEED FORWARD NEURAL NETWORK AND EMG

The process of designing a prosthetic hand involves several critical steps to ensure functionality and user satisfaction. It begins with designing the hand using advanced CAD software and followed by stimulating the designed model to evaluate its performance and functionality before physical production. The outer shell is then manufactured using 3D printing techniques, allowing for lightweight and durable components tailored to the user’s specifications. Following this, the parts are meticulously assembled, ensuring a seamless fit and incorporating features like grips for enhanced functionality. The next step involves adding a controller and actuators that manage different movements like Tripod, Pinch and Power grip. Finally, the prototype under testing and validation and necessary adjustments can be made.

The development of an algorithm for controlling a prosthetic hand using EMG signals involves several key steps to ensure accurate and effective functionality. Initially, collection of EMG data sets is performed from a specific individual, capturing signals during various actions such as tripod grip, pinch grip, and power grip. This diverse dataset serves as the foundation for further processing. The next step is signal processing, where the collected EMG signals undergo denoising to eliminate any noise or artifacts that may interfere with accurate interpretation. Following this, feature extraction is conducted to identify relevant characteristics from the cleaned signals that can effectively represent the intended movements. These features are then subjected to classification, where machine learning algorithms categorize the movements based on the extracted features, allowing the system to recognize different grips. To ensure the reliability and robustness of the classification model, cross-validation is implemented, which involves partitioning the dataset into training and testing subsets to evaluate performance metrics. After validating the algorithm, it is applied to prosthetic hand testing and validation, where the prosthetic device is assessed for its responsiveness and accuracy in mimicking the user's intended movements. Finally, comprehensive documentation of the entire process is maintained to provide insights into methodologies, results, and potential areas for improvement in future iterations of the algorithm.

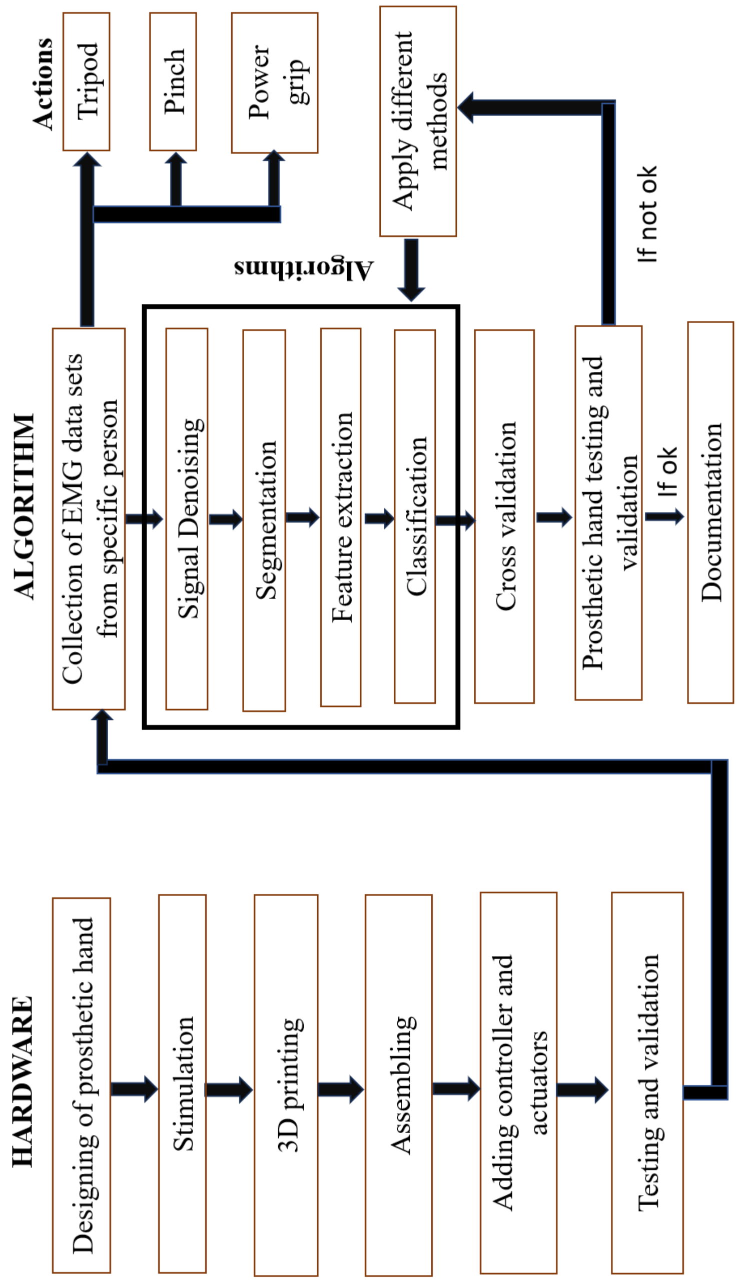


Figure 3.1 Methodology

CHAPTER 4

SIGNAL PROCESSING AND SEGMENTATION

4.1 RECORDING OF THE SIGNAL

To record the EMG signals effectively BioAmp EXG pill with surface gel electrodes were used. And to capture the EMG signal a software called Spike Recorder is used.

Before placing electrode, prepare the skin surface can be cleaned by applying Nuprep Skin Preparation Gel to the designated area. This gel helps remove dead skin cells and dirt. After thoroughly massaging the gel on the skin, followed by wiping the area with a wet wipe for further cleaning. Also make sure the skin surface should free of arm hairs.

Figure 4.1 Skin Preparation Gel And Wipes

After cleaning the area, the electrodes are placed to capture EMG signal. For this project gel electrodes are used for efficient signal acquisition as it have more contact. For this project, a rectangular gel electrode having diameter of 15mm gel area is used. The electrodes should kept clean, if gel in the electrode peeled off or dried, it should be replaced with new electrodes.

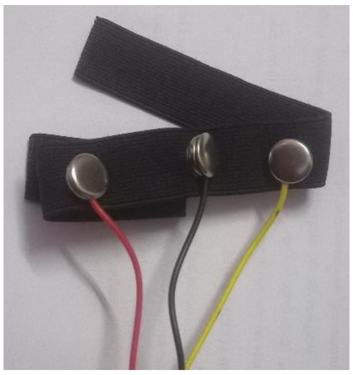


Figure 4.2 Dry Electrode

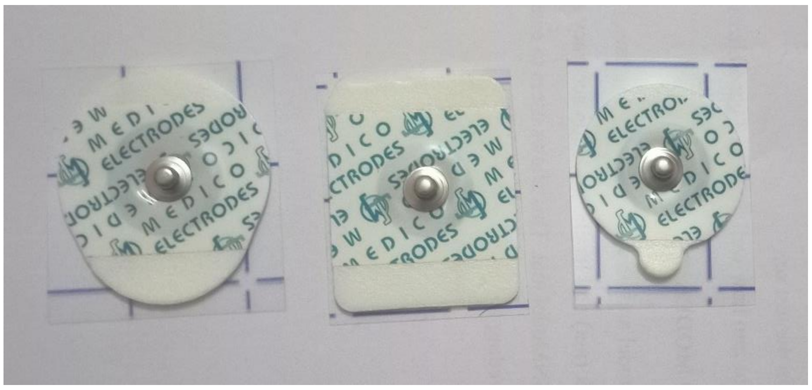


Figure 4.3 Different Sizes of Gel Electrodes

The Bioamp EXG pill is connected with micro-controller and the gel electrode placement as follows,

1) The Positive and Negative electrodes (Red and Black)- should be placed along the muscle selected to be measured. i.e., at forearm region.
2) The Reference electrode (Yellow)- should be placed at the non-muscular portion. i.e., at elbow region.

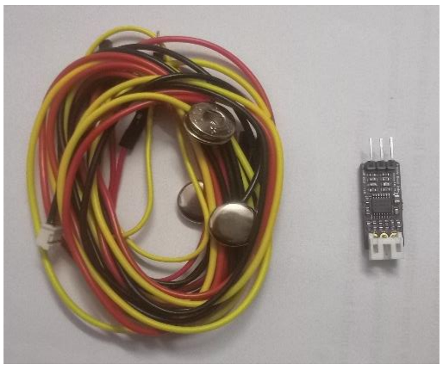


Figure 4.4 BioAmp EXG Pill and Electrode Connectors

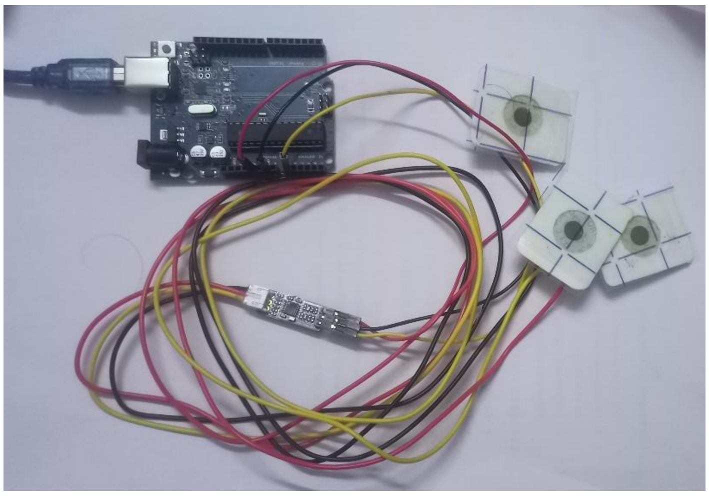


Figure 4.5 BioAmp EXG Pill Connection

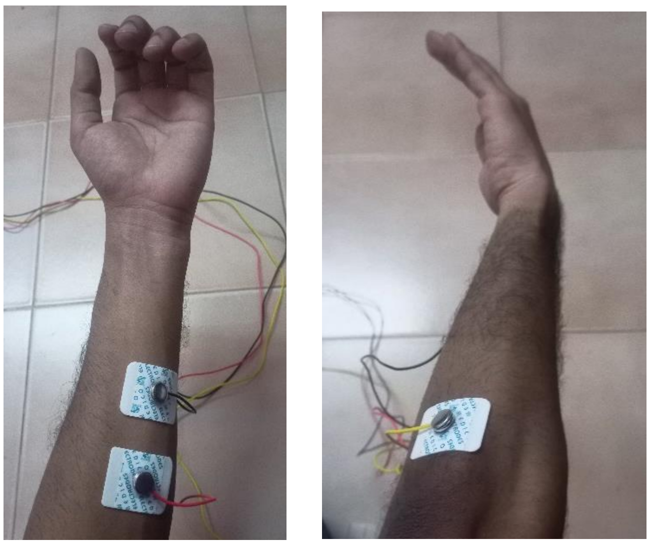
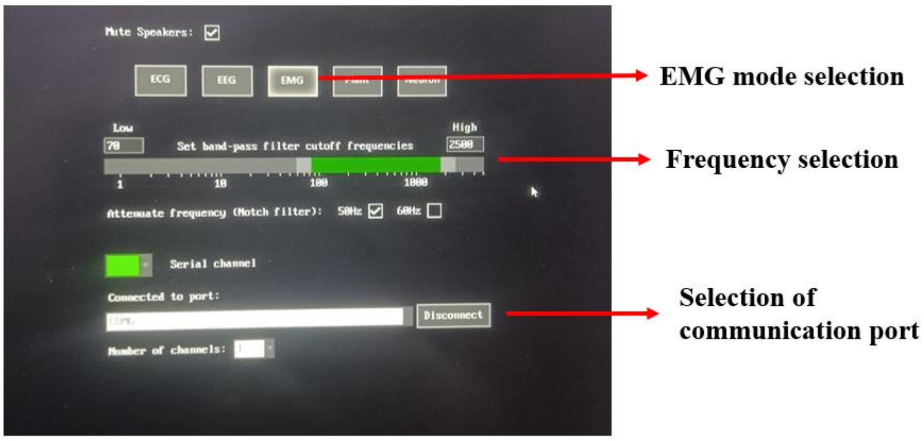


Figure 4.6 Electrodes Placement

Once electrodes placement is done, the micro-controller is configured with Spike Recorder to capture the emg signal as a wave file extension. The frequency is set to allow between 70 Hz to 2500 Hz and corresponding communication port is selected. The EMG signal can recorded by starting the recording button. For one recording, a signal is recorded by repeating the same action i.e., Tripod, Pinch or Power grip for five number of times and the output can be store as wave file extension.



Mute Speakers: ✓

ECG | EEG | EMG

Low 70 Set band-pass filter cutoff frequencies High 2500

Attenuate frequency (Notch filter): 50Hz ✓ 60Hz ☐

Serial channel

Connected to port:

Disconnect

Number of channels: 1

Figure 4.7 Spike Recorder Settings

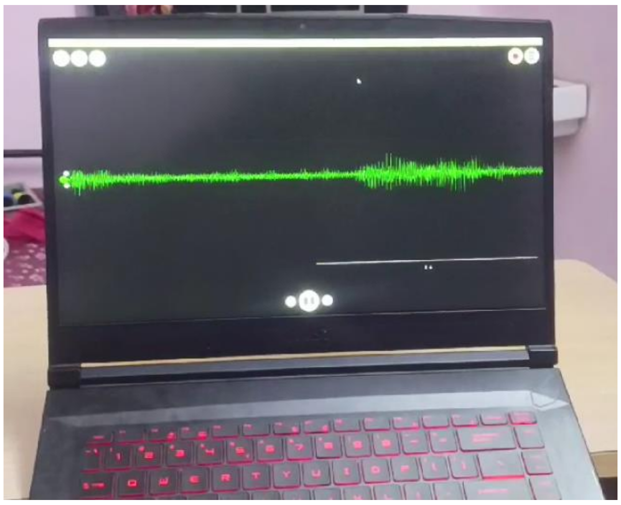


Figure 4.8 EMG Signal Recording using Spike Recorder

Recording of EMG signal were done on daily basis during morning, afternoon and night. The actions that performed are Tripod, Pinch and Power grip.

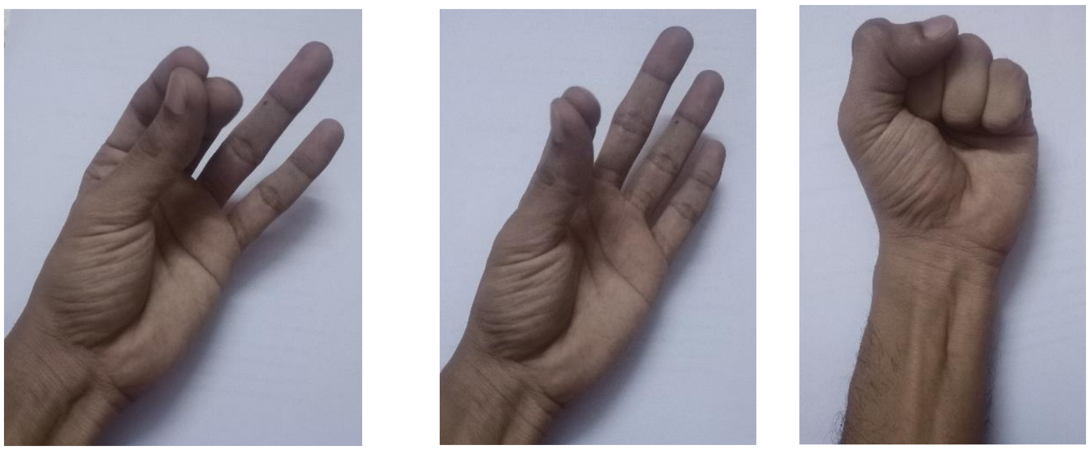


Figure 4.9 Tripod, Pinch and Power Grip Actions

4.2 GENERATION OF GRAPHICAL DATA

I. The signal recorded using spike recorder can be saved as wave file extension.

II. To visualize and to process the recorded emg signal, the .wav file is converted into graph as a matlab figure having time in x-axis and corresponding amplitude in y-axis using Matlab software.

4.2.1 Steps Involved in Generating Graphical Data

The step involved in generating graphical data from wave file as follows,

1) Read the EMG data and sample rate using ‘audioread’.
2) Create a time vector based on the EMG data length and sample rate.
3) Create a new figure window for each audio file.
4) Plot the EMG data against the time vector and label the axes and add a grid for better visualization.
5) Save and plot as a MATLAB figure file (.fig) extension for visualization.

Outputs

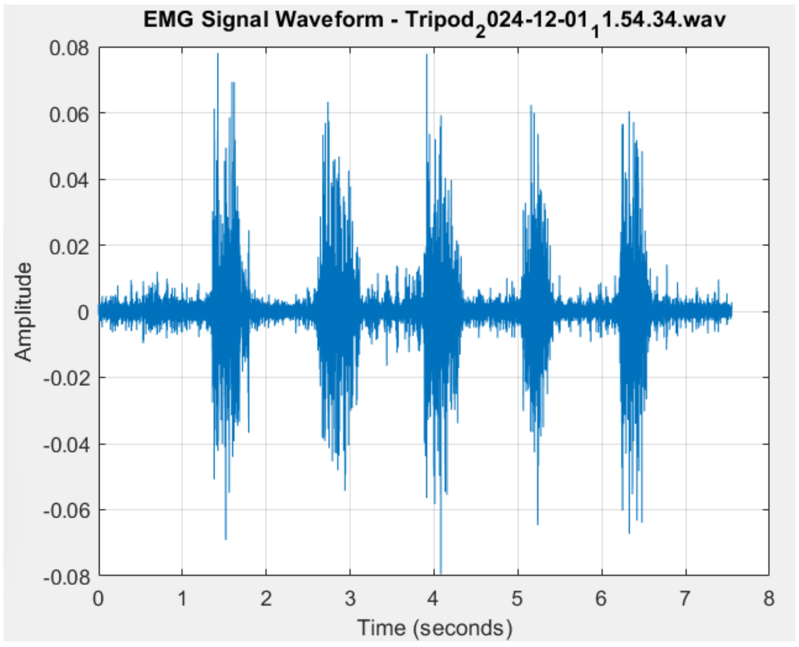


Figure 4.10 Graph output for Tripod Action

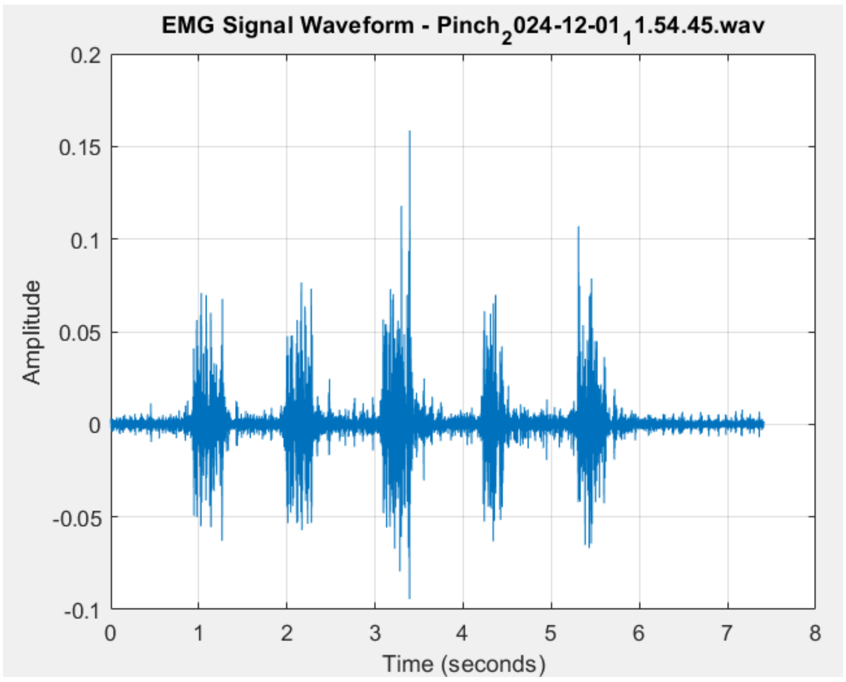


Figure 4.11 Graph output for Pinch Action

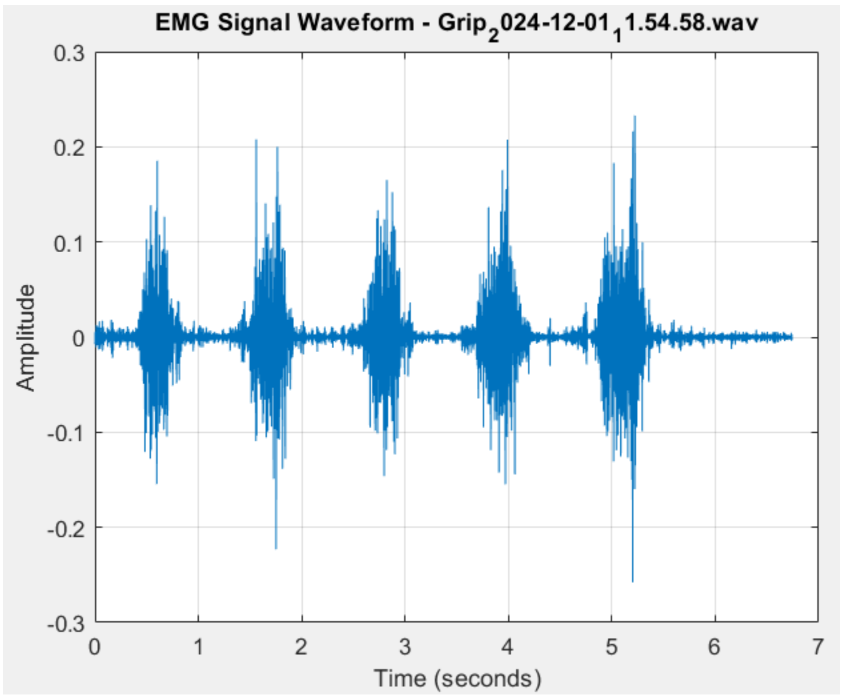


Figure 4.12 Graph output for Power Grip Action

4.3 GENERATION OF EXCEL DATA

I. The signal recorded using spike recorder can be saved as wave file extension.

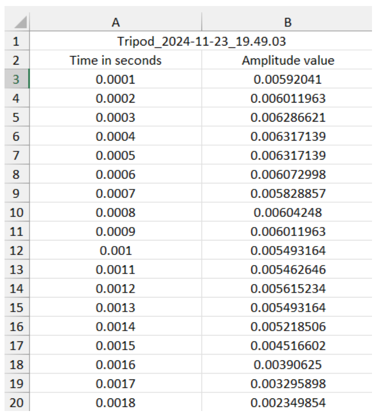
II. The time and the amplitude values are converted into excel data using Matlab and used for further analysis.

4.3.1 Steps Involved in Generating Excel Data

The steps involved in generating graphical data from wave file as follows,

1) Start an instance of Excel using ActiveX.
2) Create a table containing time and amplitude values.
3) Read the EMG data and sample rate using ‘audioread’.
4) Create a time vector based on the EMG data length and sample rate.
5) Create a table containing time and amplitude values.
6) Create an output filename by replacing the ‘.wav’ extension with ‘.xlsx’.
7) Open the newly created Excel workbook.
8) Select the first sheet in the workbook.
9) Merge cells A1 and B1, then insert the filename (without extension) into A1.
10) Add custom axis labels: "Time in seconds" in A2 and "Amplitude value" in B2 and store respective values.
11) Save the file.

Output



	A	B
1	Tripod_2024-11-23_19.49.03	
2	Time in seconds	Amplitude value
3	0.0001	0.00592041
4	0.0002	0.006011963
5	0.0003	0.006286621
6	0.0004	0.006317139
7	0.0005	0.006317139
8	0.0006	0.006072998
9	0.0007	0.005828857
10	0.0008	0.00604248
11	0.0009	0.006011963
12	0.001	0.005493164
13	0.0011	0.005462646
14	0.0012	0.005615234
15	0.0013	0.005493164
16	0.0014	0.005218506
17	0.0015	0.004516602
18	0.0016	0.00390625
19	0.0017	0.003295898
20	0.0018	0.002349854

Figure 4.13 Excel Output for Tripod Action

4.4 SIGNAL DENOISING

i. The recorded signal has to denoised (remove of artifacts) for improved analysis.

ii. To remove noise from the emg signal used **Variational Mode Decomposition (VMD)** to decompose the noisy EMG signal into 5 Intrinsic Mode Functions (IMFs).

iii. Variational Mode Decomposition (VMD) in denoising EMG signals is more effective in separating the signal into distinct Intrinsic Mode Functions (IMFs) without mode mixing.

iv. It can handle multiple types of noise, such as power line interference, white Gaussian noise, and baseline wandering, all of which often contaminate EMG signals.

4.4.1 Variational Mode Decomposition

Variational Mode Decomposition (VMD) is a signal processing technique used to decompose a complex signal into a set of intrinsic mode functions (IMFs). It is particularly useful for analyzing non-stationary and nonlinear signals, such as electromyographic (EMG) signals, which can contain various frequency components and noise.

1) Objective Function:

The goal of VMD is to minimize the following variational problem:

$$\min \sum_{k=1}^{K} \left\| \frac{\partial}{\partial t} \left(x(t) - \sum_{k=1}^{K} u_k(t) \right) \right\|^2 + \alpha \sum_{k=1}^{K} \| u_k(t) \|^2 \quad \ldots..(4.1)$$

where:

x(t) is the original signal.

uk(t) are the IMFs.

K is the number of modes.

α is a regularization parameter that controls the trade-off between fidelity to the original signal and smoothness of the modes.

2) Mode Constraints:

Each mode u k (t) is constrained to be band-limited around a center frequency

$$u_k(t) = A_k e^{j(2\pi f_k t + \phi_k)} \quad \text{.....(4.2)}$$

uk(t) are the Intrinsic Mode Functions

4.4.2 Steps Involved to Perform Variational Mode Decomposition

The steps involved to perform variational mode decomposition from wave file as follows,

1) Read the EMG data and sample rate from the WAV file.
2) Perform Variational Mode Decomposition (VMD) to the EMG signal to extract Intrinsic Mode Functions (IMFs).
3) Generate a time vector based on the length of the EMG signal and its sample rate for plotting.
4) Create and open a new figure window for visualizing the extracted IMFs.
5) Loop through each IMF and plot it in a separate subplot with appropriate titles and labels.
6) Save the figure as a MATLAB .fig file in the same directory.

Output

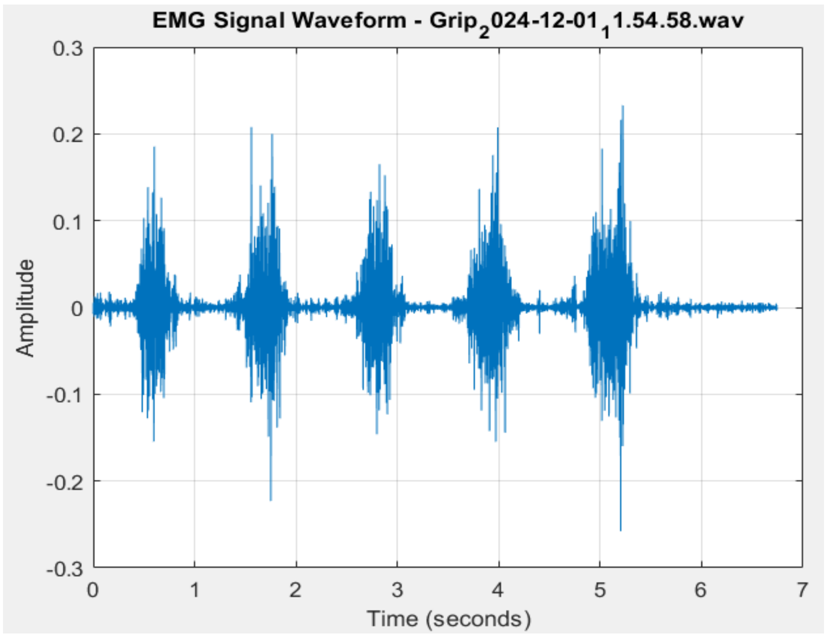


Figure 4.14 Raw Signal of Grip action

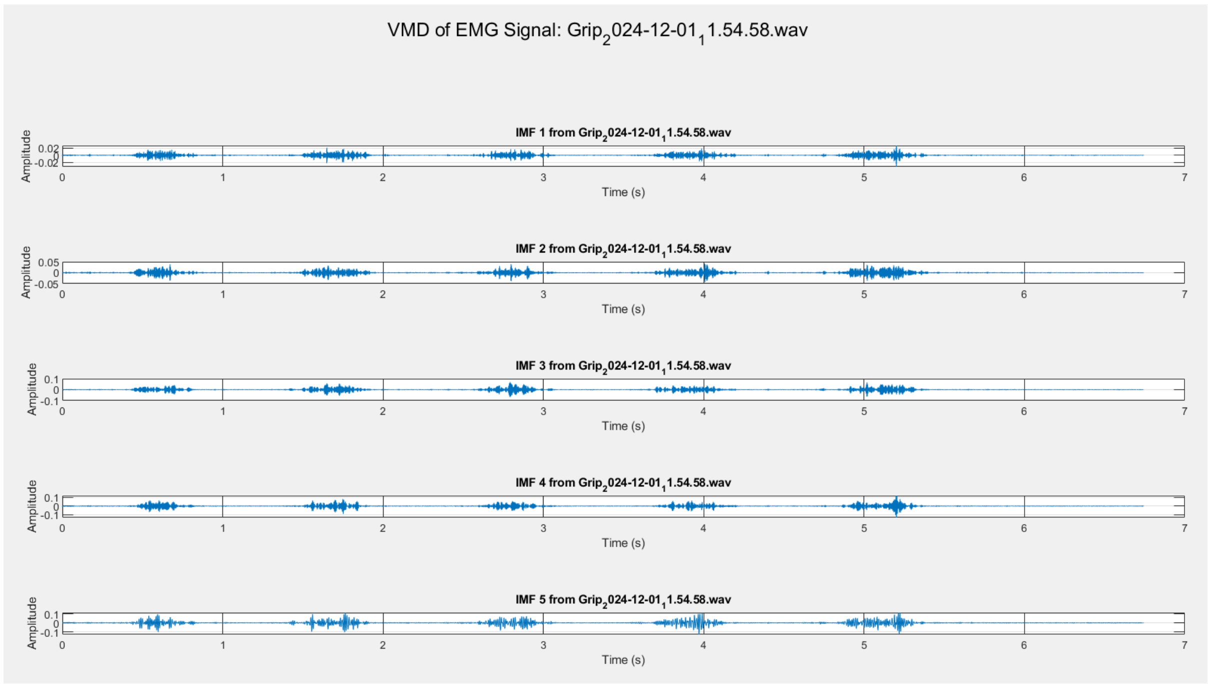


Figure 4.15 VMD Denoised Signal of Grip Action

4.5 SIGNAL-TO-NOISE RATIO

To process further, the best denoised signal have to be selected. It can be done by calculating Signal to Noise ratio for every IMF output and choose the best one from it.

Signal-to-Noise Ratio (SNR) is a measure that quantifies the level of a desired signal compared to the level of background noise. It is expressed as:

$$SNR = Psignal / Pnoise \qquad \text{…..(3)}$$

SNR is typically measured in decibels (dB), with higher values indicating a clearer and more distinguishable signal from noise.

4.4.2 Steps Involved to Perform Signal-To-Noise Ratio

The steps involved to perform Signal-To-Noise Ratio as follows,

1) Read the EMG data and sample rate from the MATLAB figure files using a file dialog.
2) Create a cell array to store SNR values for each figure being processed.
3) Retrieve all axes handles from the opened figure, which represent different Intrinsic Mode Functions (IMFs).
4) Prepare an array to store SNR values for each IMF in the current figure and initial values for tracking the best SNR, its index, and corresponding IMF data.
5) Specify the sampling frequency (Fs) 10000 hz relevant to the data being analyzed
6) Get the Y data from the first line object within each axis, assuming it represents an IMF.
7) Compute the power of the signal by calculating the mean of its squares.
8) Threshold is set to 5% (0.05) to identify noise and calculate its power based on values below this threshold and compute the Signal-to-Noise Ratio (SNR) in decibels (db).
9) Record and check if the current IMF has a higher SNR than previously recorded and update best SNR variables accordingly.

10) Save this plot as a new .fig file in the output directory and store the calculated SNR values for this figure in the designated cell array.

Output

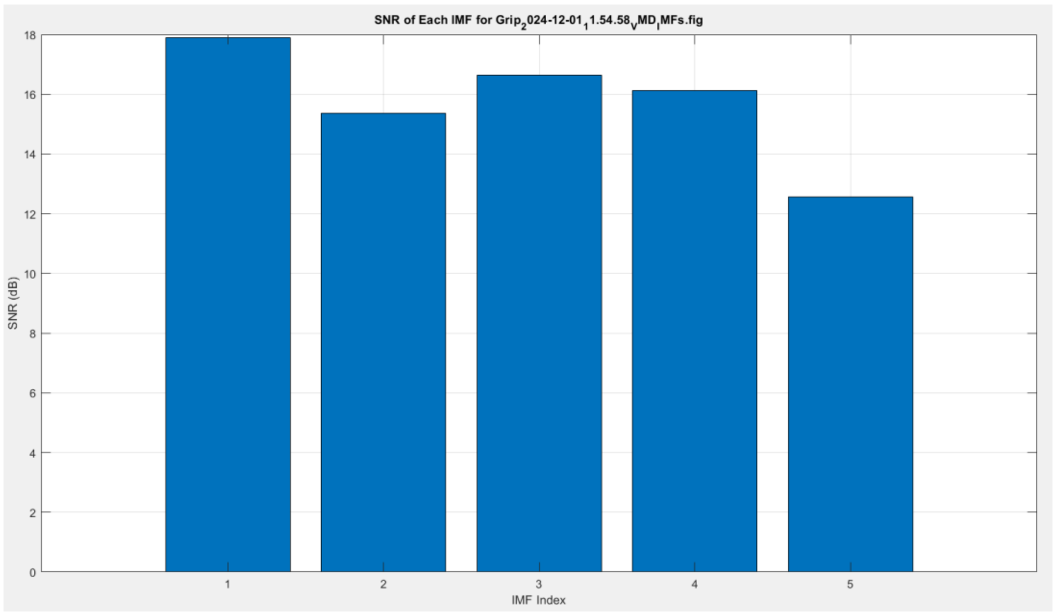


Figure 4.16 SNR Output for Grip Action

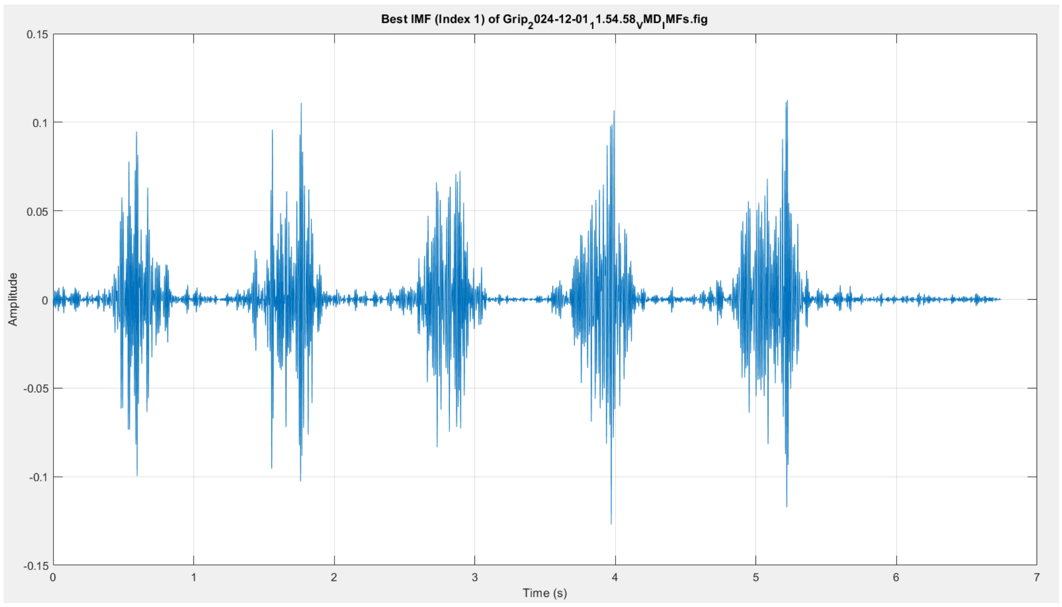


Figure 4.18 Best Signal Based on SNR Output

4.6 SEGMENTATION

During recording the emg signal using Bioamp EXG pill, in each trial the participant performed a tripod, pinch and grip action repeatedly for about five times simultaneously. For feature extraction process each signal have to be processed separately to get valuable information. i.e.,features from it. Hence the signal has to be segmented based on the timing. In this process, segmenting timing are altered based on visualization of recorded signal.

4.6.1 Steps Involved to Perform Segmentation

The steps involved to perform Signal-To-Noise Ratio as follows,

1) Read the EMG data and sample rate from the MATLAB figure files using a file dialog.
2) Loop through selected figure files and open the figure without displaying it on the screen.
3) Retrieve axes and line objects to access plotted data and get time (X) and EMG amplitude (Y) data from the first line object.
4) Perform segmenting based on the time intervals of the EMG data.
5) Convert time values to indices, ensuring they are valid, and extract corresponding data segments.
6) Create and save plots for each segment with appropriate titles and labels.

Output

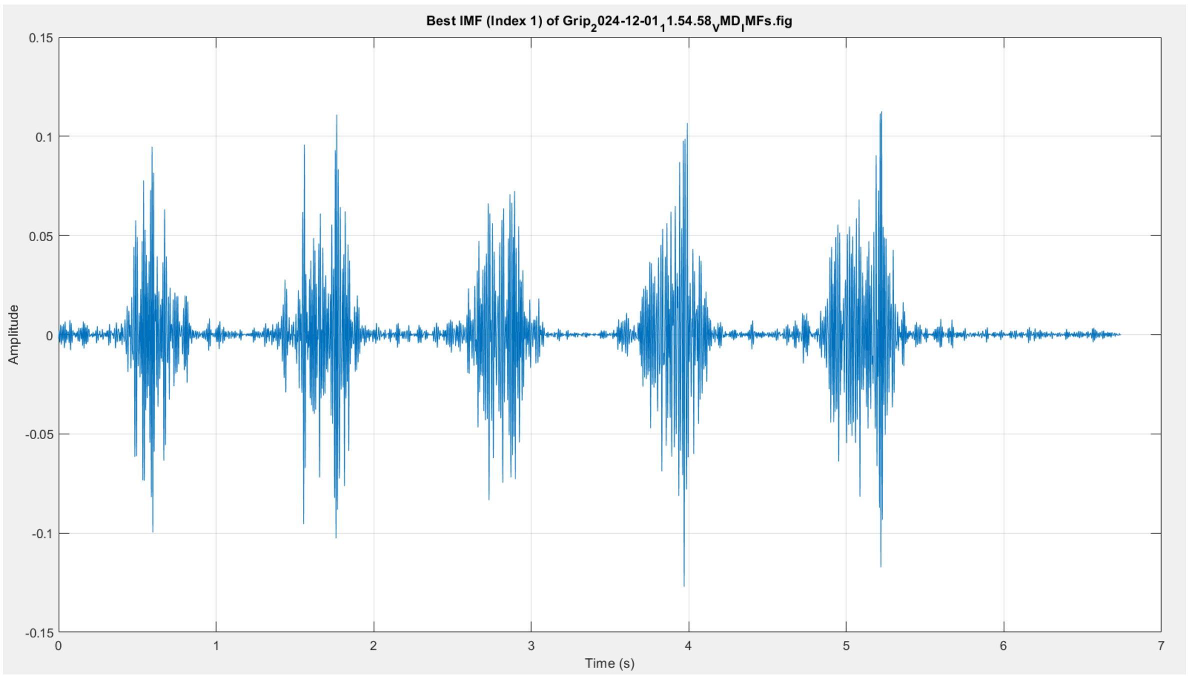


Figure 4.19 Best Signal for Grip Action

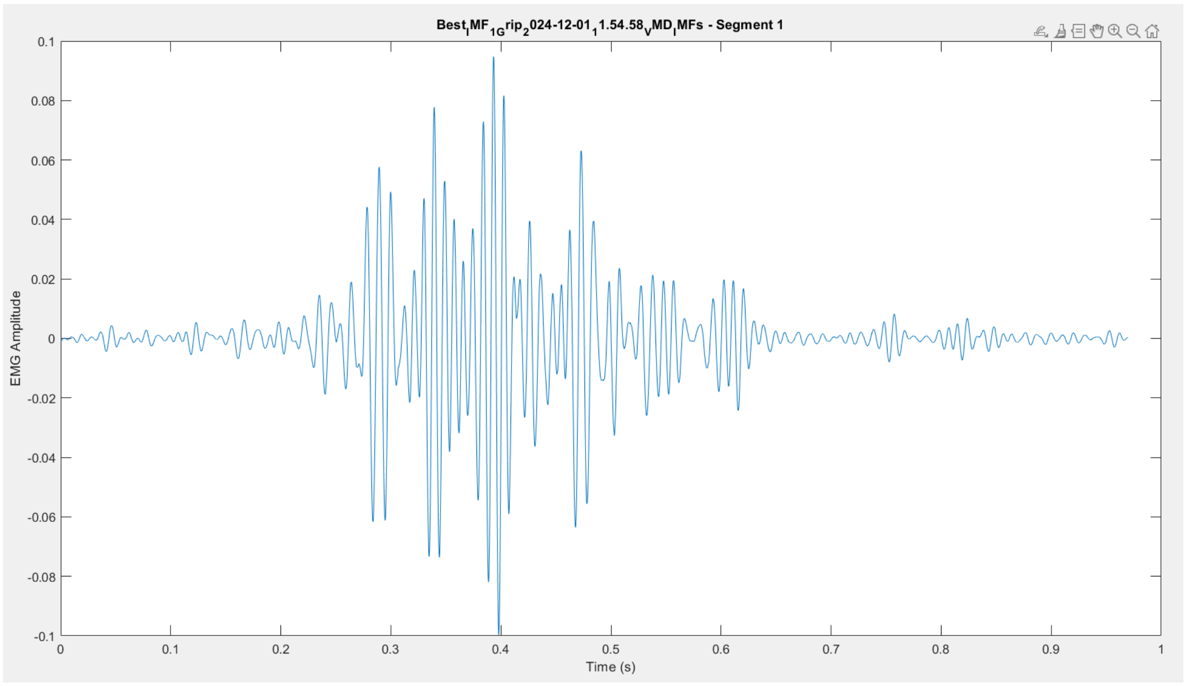


Figure 4.20 Segment 1 of Grip Action

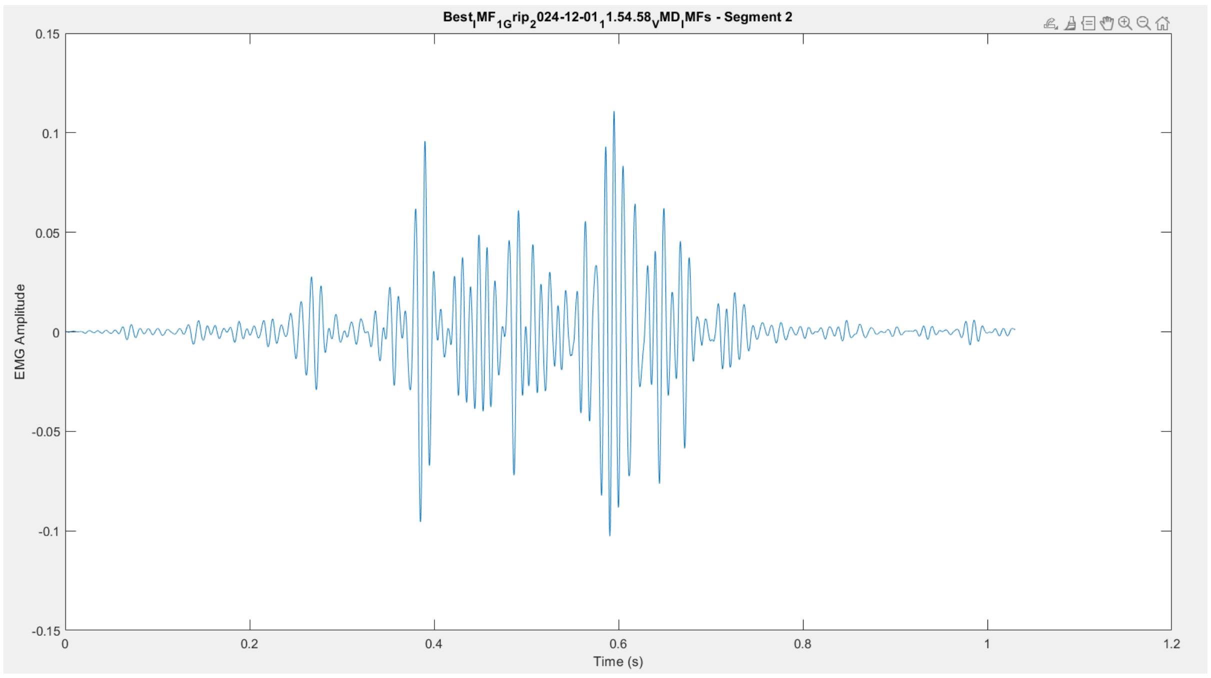


Figure 4.21 Segment 2 of Grip Action

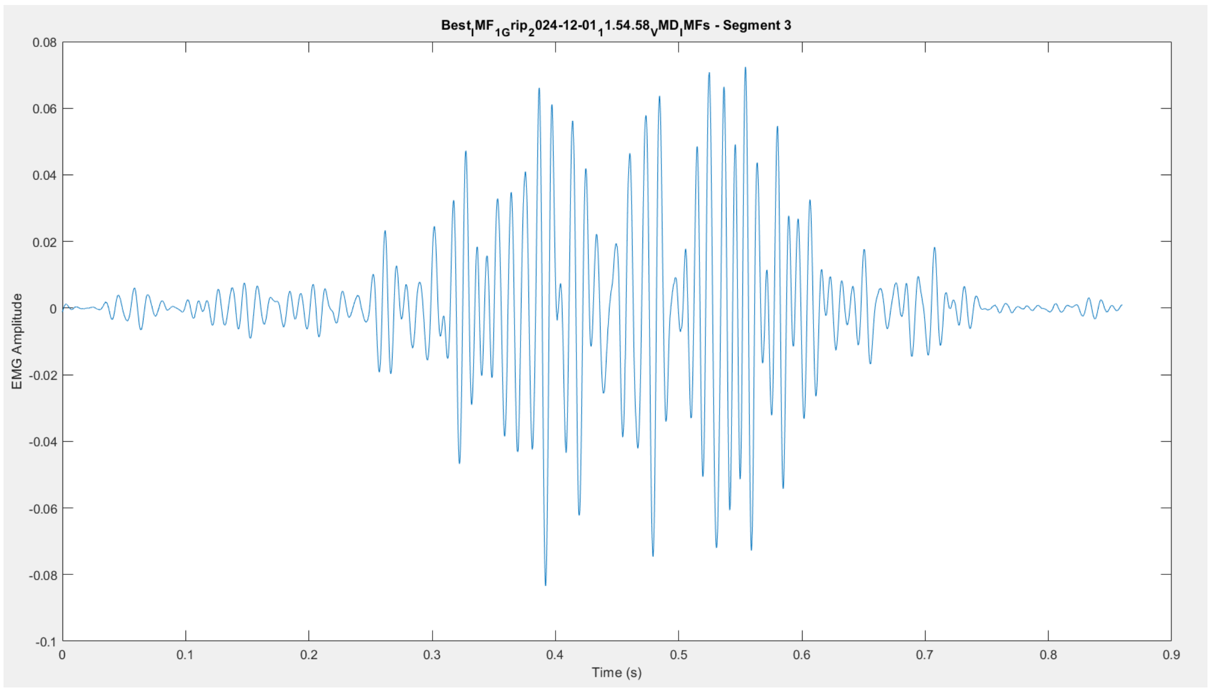


Figure 4.22 Segment 3 of Grip Action

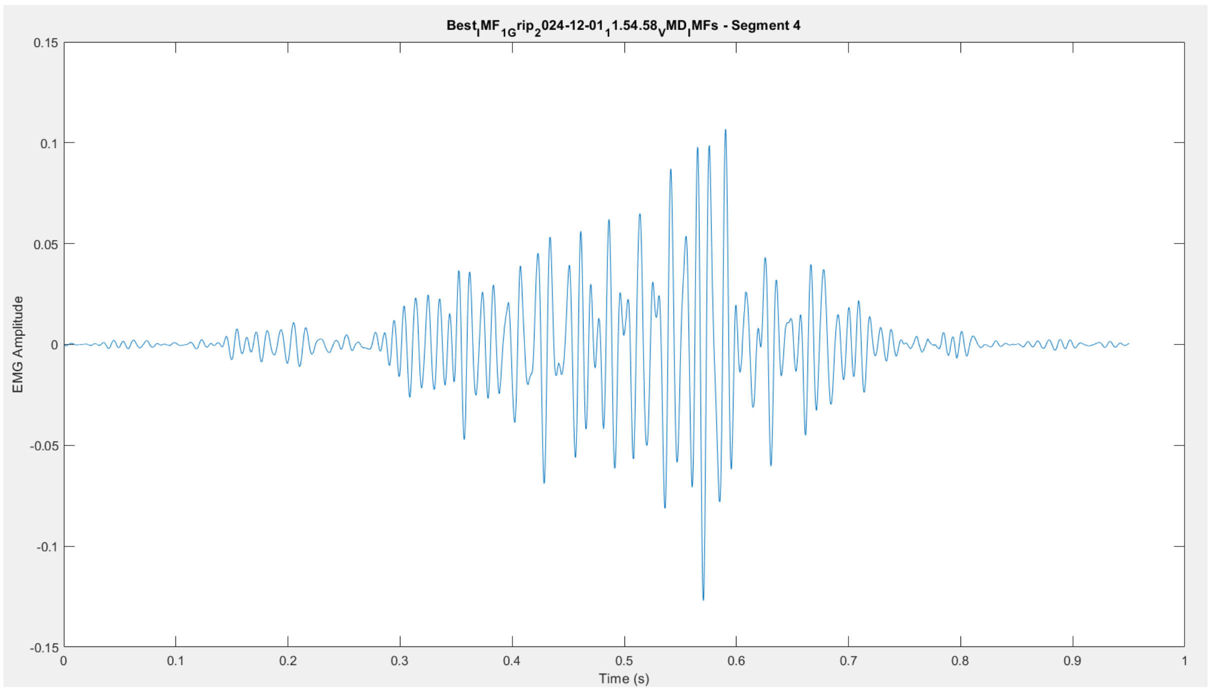


Figure 4.23 Segment 4 of Grip Action

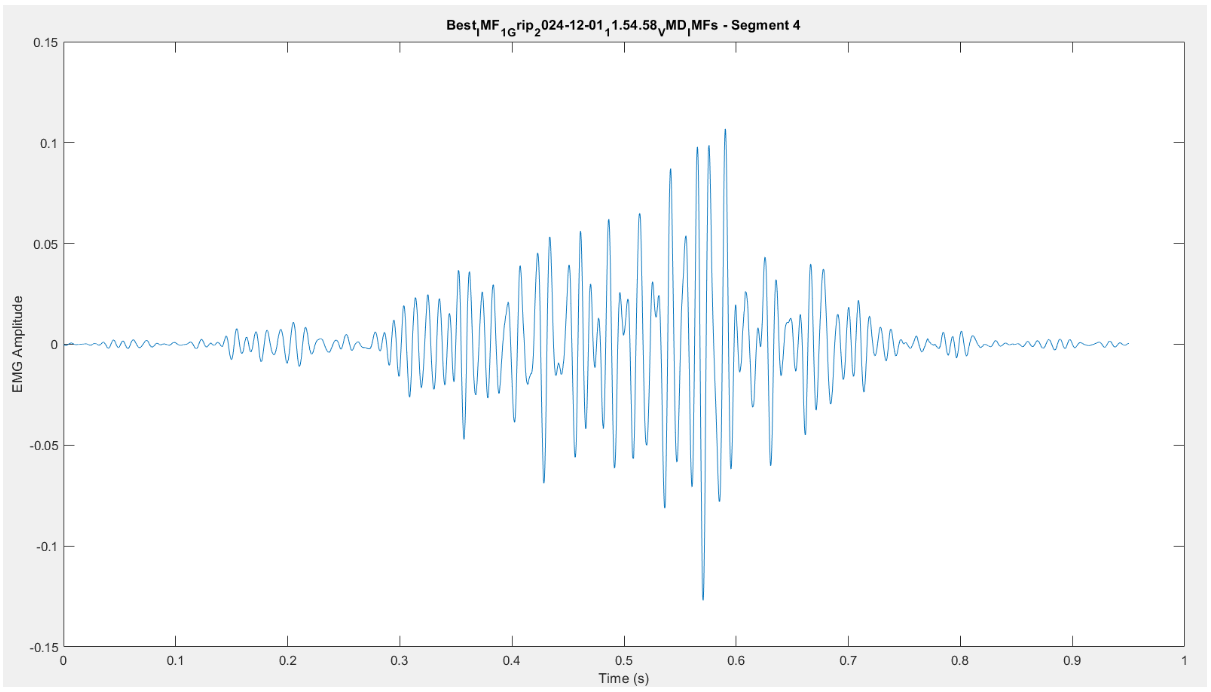


Figure 4.24 Segment 5 of Grip Action

CHAPTER 5

FEATURE EXTRACTION AND CLASSIFICATION

5.1 FEATURE EXTRACTION

Feature extraction is a critical process in machine learning and data analysis that involves transforming raw data into a set of informative features. These features represent the essential characteristics or attributes of the data, making it easier for machine learning algorithms to analyze and learn from the information.

The features computed from the processed emg signal can be done based on time-domain and frequency domains. For better results in this project used both the two domains of feature extraction were done.

5.1.1 Types of Features Extracted

The following are the types of features that extracted from each domain.

i. **Time-Domain Features:** Mean value, standard deviation, root mean square (RMS), and zero crossings.
ii. **Frequency-Domain Features:** Mean power spectral density using Short-Time Fourier Transform (STFT).
iii. **Wavelet Transform Features:** Energy computed from continuous wavelet transform (CWT).

5.1.2 Steps Involved in Feature Extraction Process

The following are the steps that involved during feature extraction process.

i. Selection of multiple segmented matlab figure files.
ii. Padding features with zero for consistency based on the highest x axis time value.
iii. Determine label based on filename content. i.e., Tripod, Pinch, Grip.
iv. Extract feature based on time-domain, frequency-domain and time-frequency domain.
v. Store features and corresponding label as Matlab data.

vi. Prepare data for classification.

vii. Store features and label to excel files.

5.2 TIME DOMAIN ANALYSIS

Time domain feature extraction involves examining the characteristics of a signal as it varies over time.

The following are the features that extracted based on time-domain.
1) Mean Value:

i. The mean value of the EMG signal represents the average amplitude of the signal over the observation period.

ii. Mathematical Expression:

$$\text{Mean} = \frac{1}{N}\sum_{i=1}^{N} x[i] \qquad \text{.....(5.1)}$$

where,

N is the number of samples and

x[i] represents the EMG signal values.

i. A higher mean value may indicate stronger muscle activation, while a lower mean suggests less activity.

2) Standard Deviation (STD):

i. The standard deviation quantifies the amount of variation or dispersion in the EMG signal.

ii. Mathematical Expression:

$$\text{STD} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x[i] - \text{Mean})^2} \qquad \text{.....(5.2)}$$

where,

N is the number of samples and

x[i] represents the EMG signal values.

iii. A larger standard deviation indicates greater variability in muscle activation, which may be relevant for distinguishing between different types of movements or actions.

3) Root Mean Square (RMS):

i. The RMS value provides a measure of the effective value of the EMG signal, accounting for both positive and negative amplitudes.

ii. Mathematical Expression:

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x[i]^2} \quad \text{.....(5.3)}$$

iii. RMS is particularly useful for assessing overall muscle activity levels. Higher RMS values indicate stronger muscle contractions.

4) Zero Crossings:

i. Zero crossings refer to the number of times the EMG signal crosses the zero amplitude line.

ii. Mathematical Expression:

$$\text{Zero Crossings} = \sum_{i=1}^{N-1} \text{sign}(x[i]) \neq \text{sign}(x[i+1]) \quad \text{.....(5.4)}$$

iii. The count of zero crossings can provide insight into the frequency and nature of muscle contractions. A higher number of zero crossings may indicate more rapid changes in muscle activation.

Output

S.No	Sample	Label	Mean Value	Standard Deviation	Root Mean Square	Zero Crossings
1	Grip 1	2	9.6924408576E-07	1.1010132786E-02	1.1009553394E-02	1.9800000000E+02
2	Grip 2	2	1.4974127255E-06	1.0212114617E-02	1.0211433987E-02	1.4900000000E+02
3	Grip 3	2	-4.6929469454E-06	1.3146512167E-02	1.3145739749E-02	1.7700000000E+02
4	Grip 4	2	-7.0070876697E-06	1.3593407159E-02	1.3592670253E-02	1.9300000000E+02
5	Grip 5	2	3.5790141062E-06	1.4244699796E-02	1.4243862397E-02	1.7900000000E+02
6	Tripod 1	1	-6.7084899276E-07	3.2620911777E-03	3.2619281543E-03	3.7900000000E+02
7	Tripod 2	1	-9.1753375836E-07	4.7998940844E-03	4.7996118503E-03	3.4000000000E+02
8	Tripod 3	1	2.1041695668E-07	4.7973055838E-03	4.7970234189E-03	3.3400000000E+02
9	Tripod 4	1	-1.0610964209E-06	4.5858640421E-03	4.5856094158E-03	3.5400000000E+02
10	Tripod 5	1	3.7372996676E-06	5.1096661225E-03	5.1093836426E-03	3.3700000000E+02

Table 5.1 Time-Domain Feature Output

5.3 FREQUENCY DOMAIN ANALYSIS

Frequency involves transforming the time-domain representation of a signal into its frequency components, enabling the analysis of how energy is distributed across different frequencies. This approach is essential for understanding muscle activity patterns, detecting fatigue, and classifying different movements.

5.3.1 Short-Time Fourier Transform (STFT)

The STFT works by dividing a longer signal into shorter segments, or windows, and then computing the Fourier transform for each segment. This allows for the examination of how the frequency content of the signal changes over time. The key steps involved in computing the STFT are:

i. Windowing: The signal is multiplied by a window function (e.g., Hamming or rectangular window) to isolate a segment of the signal. This windowing helps to minimize edge effects that can distort the Fourier transform results.

ii. Fourier Transform: The Fourier transform is computed for each windowed segment, providing a representation of the frequency content within that specific time frame.

iii. Spectrogram Generation: The results from each segment can be visualized in a spectrogram, which displays frequency on one axis, time on another, and amplitude or power as color intensity.

The Mathematical Representation of the STFT of a continuous-time signal x(t) can be mathematically expressed as:

$$X(t,f)=\int_{-\infty}^{\infty}x(\tau)w(t-\tau)e^{-j2\pi f\tau}d\tau \quad \text{…..(5.5)}$$

where,

w(t−τ) is the window function centered at time

f represents frequency,

j is the imaginary unit.

1) Power Spectral Density (PSD):

i. The power spectral density quantifies how power (or variance) of a signal is distributed with respect to frequency.

ii. Mathematical Expression for power spectral density,

$$P(f)=\frac{1}{N}|X(f)|^2 \quad \text{…..(5.6)}$$

where,

X(f) is the Fourier transform of the signal and

N is the number of samples.

iii. PSD provides insight into which frequencies are most prominent in the EMG signal, helping identify dominant muscle activation patterns.

2) Mean Power Spectral Density (Mean PSD):

i. Mean PSD is calculated as the average power across all frequencies.

ii. This feature summarizes the overall energy distribution and can be useful for comparing different muscle activities or conditions.

5.3.2 Wavelet Transform Features

Wavelet Transform is a powerful tool for analyzing signals, particularly non-stationary signals like electromyographic (EMG) data. It provides a time-frequency representation that captures both the temporal and frequency characteristics of the signal.

5.3.3 Continuous Wavelet Transform

i. The Continuous Wavelet Transform decomposes a signal into wavelets, which are localized oscillatory functions that can be stretched or compressed to analyze different frequency components at various time points.

ii. Mathematical Representation:

The CWT of a signal x(t) can be expressed as:

$$W(a,b) = \int_{-\infty}^{\infty} x(t)\psi^*\left(\frac{t-b}{a}\right) dt \quad \text{....(5.7)}$$

where,

W(a,b) is the wavelet coefficient at scale a and translation b.

ψ(t) is the mother wavelet function.

a controls the scale (frequency), and

b controls the translation (time).

Output

S. No	**Sample**	**Label**	**Energy 1**	**Energy 2**	**Energy 3**	**Mean PSD 1**	**Mean PSD 2**	**Mean PSD 3**
1	Grip 1	2	4.5649754308E-08	8.5070981032E-08	1.5123170048E-07	9.9271233953E-07	9.9419154009E-07	9.9132598732E-07
2	Grip 2	2	4.3884267200E-08	8.4012918917E-08	1.5158935774E-07	3.0368183375E-07	3.0585050384E-07	3.0559305979E-07
3	Grip 3	2	4.6337020488E-08	8.7565416444E-08	1.5749984803E-07	6.3433122037E-07	6.4797389902E-07	6.5493614707E-07
4	Grip 4	2	4.9555827419E-08	9.2540097090E-08	1.6415433720E-07	7.2858708824E-07	7.1743283069E-07	7.0700541396E-07
5	Grip 5	2	4.5503636443E-08	8.5431907658E-08	1.5192695624E-07	2.2901493239E-07	2.3020693394E-07	2.2940326439E-07
6	Tripod 1	1	4.2367444984E-08	7.7730460810E-08	1.3492554857E-07	1.9590899888E-08	1.9791357657E-08	2.0395471854E-08
7	Tripod 2	1	4.0753551007E-08	7.5733870280E-08	1.3272212717E-07	9.1082671113E-08	9.1846660855E-08	9.2627345306E-08
8	Tripod 3	1	3.5726930227E-08	6.7636380955E-08	1.2117696463E-07	9.9788880973E-08	1.0100987893E-07	1.0166846548E-07
9	Tripod 4	1	3.9036414483E-08	7.2763006410E-08	1.2958467238E-07	2.3206863871E-08	2.2367835514E-08	2.1140632637E-08
10	Tripod 5	1	4.3628395316E-08	7.9009288894E-08	1.3743976661E-07	3.9224893467E-08	3.8829991770E-08	3.8637580241E-08

Table 5.2 Frequency-Domain Feature Output

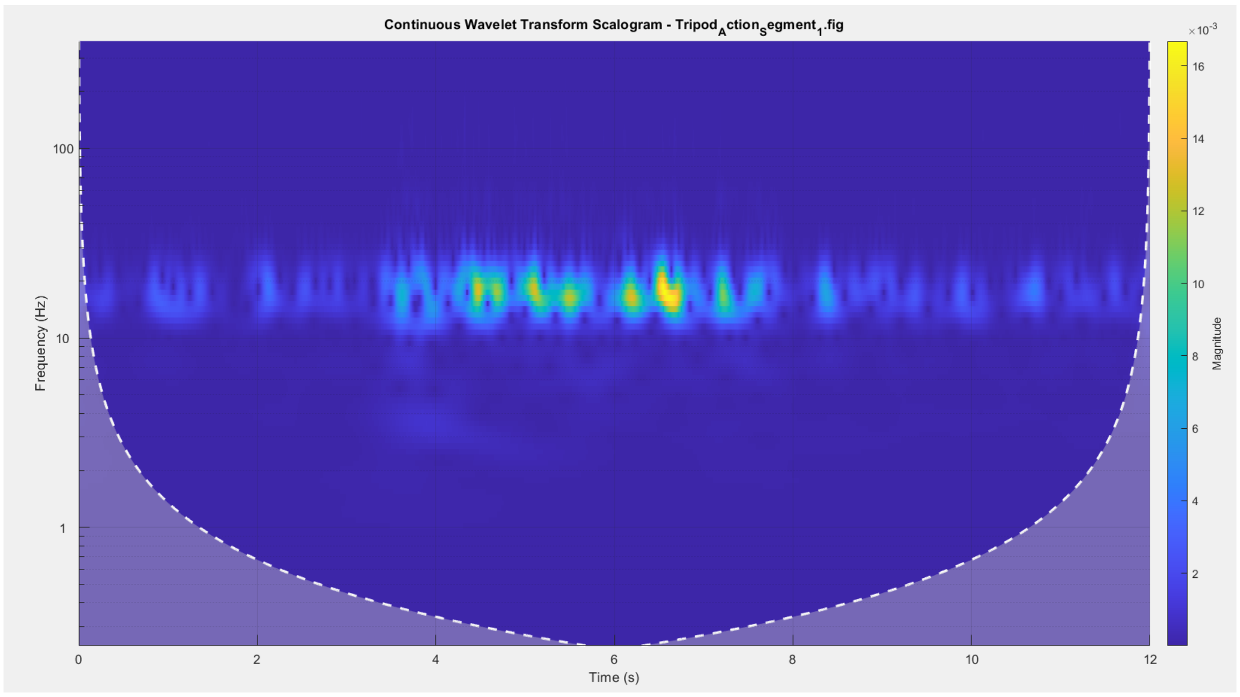


Figure 5.1 CWT Scalogram of Segment 1 of Tripod Action

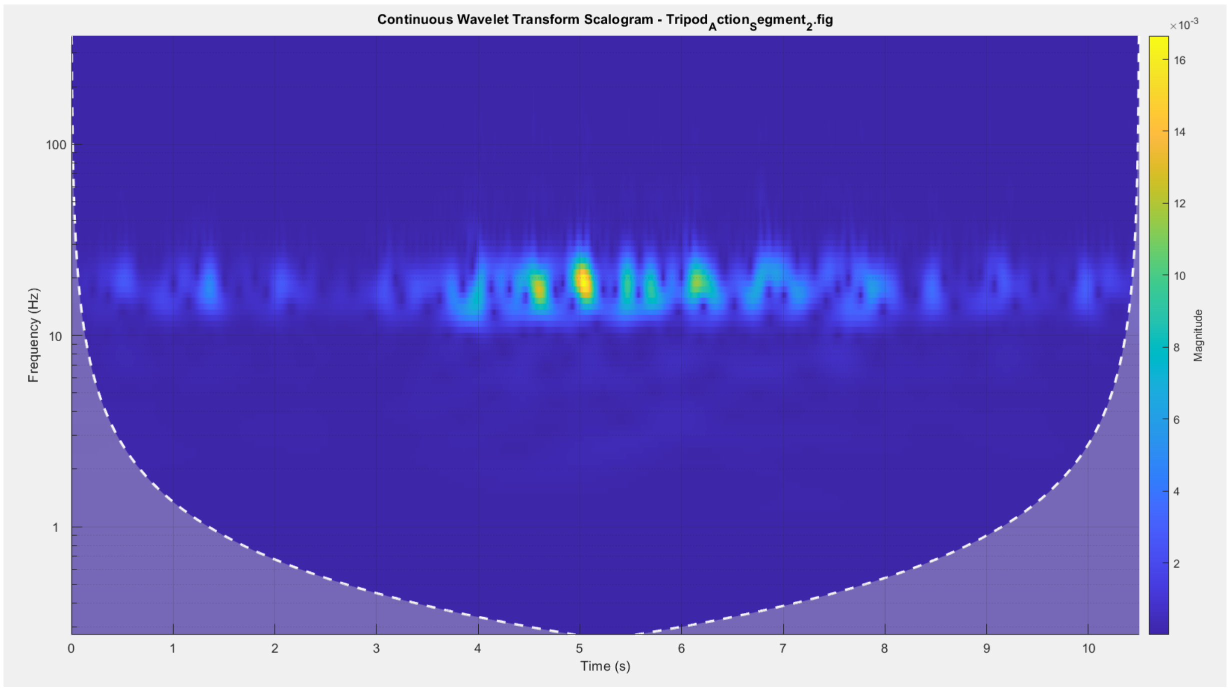


Figure 5.2 CWT Scalogram of segment 2 of Tripod

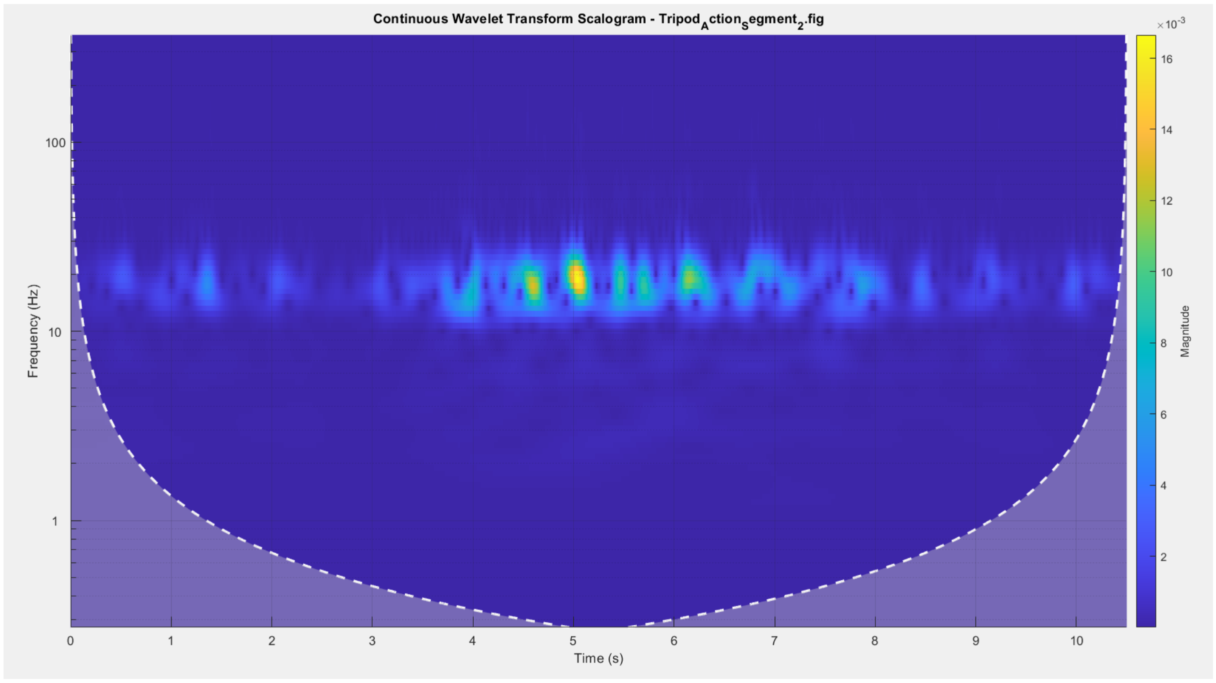


Figure 5.3 CWT Scalogram of segment 3 of Tripod

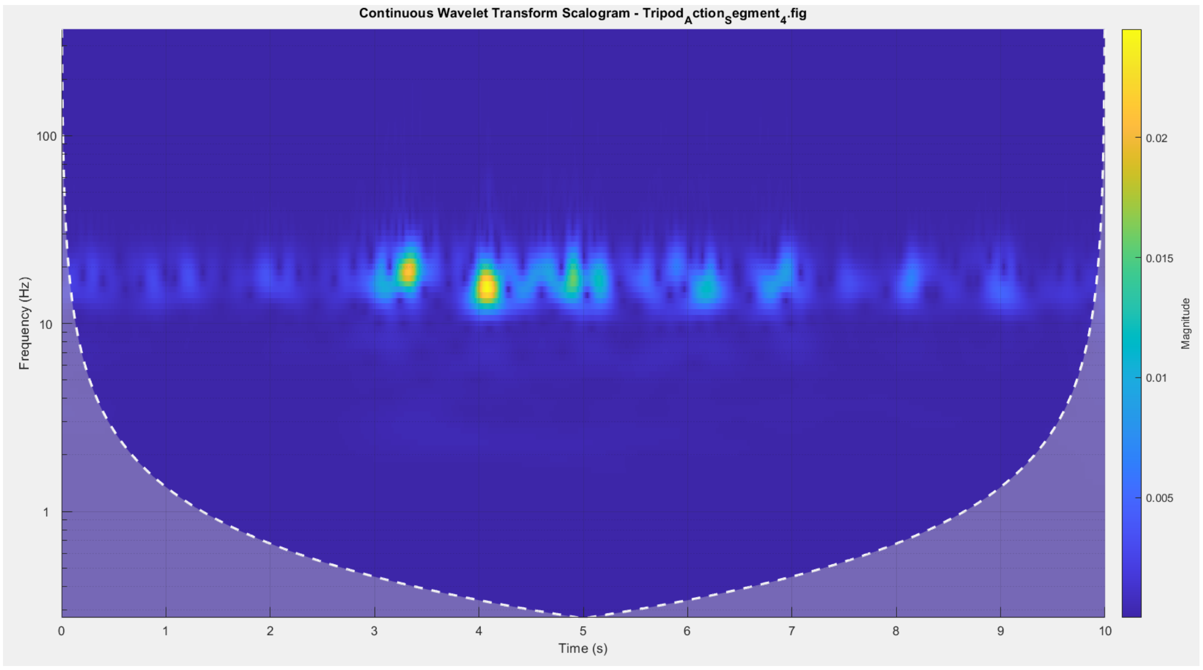


Figure 5.4 CWT Scalogram of segment 4 of Tripod

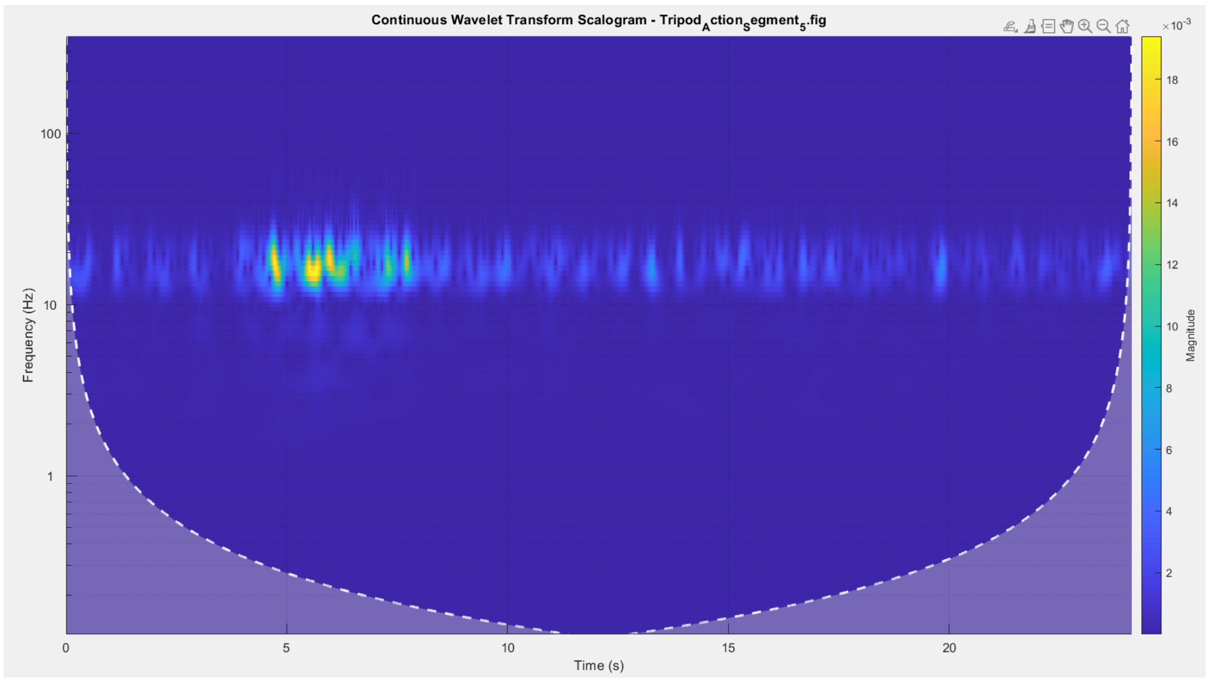


Figure 5.5 CWT Scalogram of segment 5 of Tripod

5.4 OTHER IMPORTANT STEPS INVOLVE IN FEATURE EXTRACTION

5.4.1 Padding

The variability in length of EMG signal can lead to issues when trying to compile the features into a single dataset for classification or analysis. Padding ensures that all feature vectors have the same length, allowing them to be combined seamlessly.

After all features have been extracted and stored in the 'allFeatures' structure, a loop iterates over each field. If the length of the feature vector is less than maximum feature length, zeros are appended to the end of the vector until it matches max feature length.

5.4.2 Labelling

The labeling process is essential for supervised learning tasks, where the model requires both input features and their corresponding labels to learn from the data.

The labelling can be done based on the filename of the MATLAB figure files.

i. **Identification:** It allows for the identification of different actions represented by the EMG signals, such as "Tripod" labelled as 1 and "Grip" labelled as 2.

ii. **Data Organization:** Proper labeling helps in organizing and categorizing data for analysis and visualization.

5.4.3 Data Saving

The extracted feature can be saved as both Matlab data file and as Excel file.

1) MATLAB Format:

i. Extracted features can be saved as Matlab data (.mat) file format.

ii. Retains all variable types and structures, enabling complex data manipulations within MATLAB.

iii. Facilitates quick loading of large datasets without loss of information.

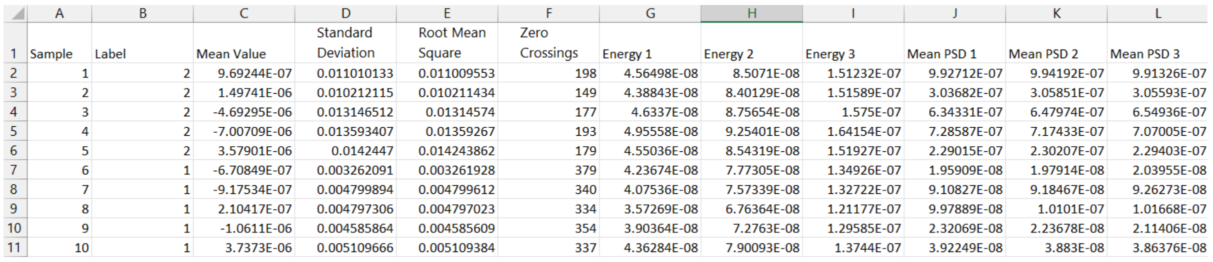
2) Excel Format:

i. Extracted features can be saved as Excel data (.xlsx) file format.

ii. Provides accessibility for users who may not be familiar with MATLAB.

iii. Allows for straightforward visual analysis and sharing of results with stakeholders or collaborators.

Output



	A	B	C	D	E	F	G	H	I	J	K	L
1	Sample	Label	Mean Value	Standard Deviation	Root Mean Square	Zero Crossings	Energy 1	Energy 2	Energy 3	Mean PSD 1	Mean PSD 2	Mean PSD 3
2	1	2	9.69244E-07	0.011010133	0.011009553	198	4.56498E-08	8.5071E-08	1.51232E-07	9.92712E-07	9.94192E-07	9.91326E-07
3	2	2	1.49741E-06	0.010212115	0.010211434	149	4.38843E-08	8.40129E-08	1.51589E-07	3.03682E-07	3.05851E-07	3.05593E-07
4	3	2	-4.69295E-06	0.013146512	0.01314574	177	4.6337E-08	8.75654E-08	1.575E-07	6.34331E-07	6.47974E-07	6.54936E-07
5	4	2	-7.00709E-06	0.013593407	0.01359267	193	4.95558E-08	9.25401E-08	1.64154E-07	7.28587E-07	7.17433E-07	7.07005E-07
6	5	2	3.57901E-06	0.0142447	0.014243862	179	4.55036E-08	8.54319E-08	1.51927E-07	2.29015E-07	2.30207E-07	2.29403E-07
7	6	1	-6.70849E-07	0.003262091	0.003261928	379	4.23674E-08	7.77305E-08	1.34926E-07	1.95909E-08	1.97914E-08	2.03955E-08
8	7	1	-9.17534E-07	0.004799894	0.004799612	340	4.07536E-08	7.57339E-08	1.32722E-07	9.10827E-08	9.18467E-08	9.26273E-08
9	8	1	2.10417E-07	0.004797306	0.004797023	334	3.57269E-08	6.76364E-08	1.21177E-07	9.97889E-08	1.0101E-07	1.01668E-07
10	9	1	-1.0611E-06	0.004585864	0.004585609	354	3.90364E-08	7.2763E-08	1.29585E-07	2.32069E-08	2.23678E-08	2.11406E-08
11	10	1	3.7373E-06	0.005109666	0.005109384	337	4.36284E-08	7.90093E-08	1.3744E-07	3.92249E-08	3.883E-08	3.86376E-08

Figure 5.6 Excel Output of Extracted Features

5.5 CLASSIFICATION

Classification in machine learning is a supervised learning technique used to categorize input data into predefined classes or groups based on training data. The primary goal of classification is to predict the correct label for new, unseen data by learning from a labeled dataset, where each input is associated with a corresponding output label

For the classification purpose, feed-forward neural network classifier is used. A feedforward neural network (FNN) is a type of artificial neural network where the flow of information moves in one direction—from the input layer, through any hidden layers, and finally to the output layer—without any cycles or loops. This architecture is characterized by its simplicity and effectiveness in various applications, including classification, regression, and pattern recognition tasks. Each layer in a feedforward network consists of interconnected nodes (neurons), where each connection has an associated weight that is adjusted during the training process to minimize the error between predicted and actual outputs. Feedforward networks are trained using backpropagation, a method that adjusts weights based on the error calculated at the output. They are widely used due to their ability to approximate functions effectively and their foundational role in more complex architectures like convolutional and recurrent neural networks.

The structure of a feedforward neural network typically includes:

1) **Input Layer:** This layer receives the input data and passes it to the next layer without performing any computations.
2) **Hidden Layers:** One or more layers that process the input data through weighted connections. These layers apply activation functions to introduce non-linearity, enabling the network to learn complex relationships.
3) **Output Layer:** This layer produces the final output of the network, which can vary in size depending on the specific task (e.g., binary classification may have one output neuron, while multi-class classification may have multiple).

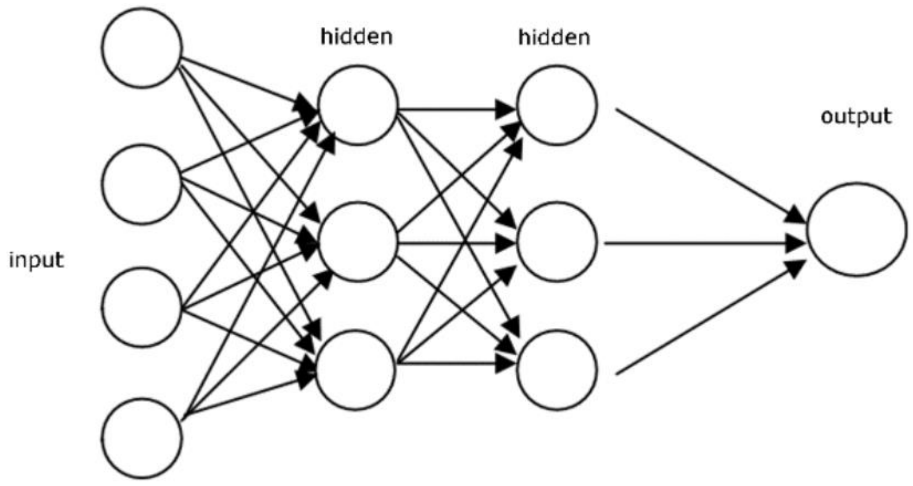


Figure 5.7 Feed forward Neural Network

5.5.1 Key Aspects of Classification

1. **Supervised Learning:** Classification relies on labeled training data that includes both input features and their corresponding class labels. This allows the model to learn the relationship between the features and the classes.
2. **Model Training:** During the training phase, classification algorithms analyze the training data to identify patterns and characteristics that distinguish different classes. Once trained, these models can classify new data points based on the learned patterns.
3. **Evaluation:** After training, the model's performance is typically evaluated using a separate test dataset to ensure it generalizes well to unseen data. Metrics such as accuracy, precision, and recall are often used to assess performance.

5.6 STEPS INVOLVED IN CLASSIFICATION PROCESS

5.6.1 Creating Data Sets

1) Data Splitting

i. To calculate the total number of samples available for training and testing.
ii. To ensure that the training and testing sets are representative and randomly selected.

iii. The randperm function generates a random permutation of integers from 1 to NumSamples. This randomization helps avoid any bias that might occur if samples were ordered in a specific way.

2) Define Split Point

i. To determine how many samples will be allocated to the training set.
ii. The split point is calculated as 80% of the total number of samples. This means that approximately 80% of the data will be used for training, while the remaining 20% will be reserved for testing.

3) Create Training and Testing Sets

i. To create separate datasets for training and testing based on the shuffled indices.
ii. TrainIndices selects indices for the training set (first 80%).
iii. TestIndices selects indices for the testing set (remaining 20%).
iv. The corresponding features and labels are then extracted into TrainFeatures, TrainLabels, TestFeatures, and TestLabels.

5.6.2 ANN Definition and Training

1) Define ANN Structure

i. To define the architecture of the ANN.
ii. A feedforward neural network is created with one hidden layer containing ten neurons (hiddenLayerSize). This structure allows for learning complex patterns in the data.

2) Train the ANN

i. To train the neural network using the training dataset.
ii. The train function is called to fit the network to the training data. Note that both trainFeatures and trainLabels are transposed (') to match MATLAB's expected input format (features as columns).

5.6.3 Model Evaluation

i. To evaluate how well the trained model performs on unseen data.

ii. The trained network predicts labels for the test features by passing them through the network (net).

iii. The predicted outputs are rounded to convert them into discrete class labels (e.g., from probabilities to binary classes).

5.6.4 Accuracy Calculation

i. To assess the performance of the classifier on the test set.

ii. The accuracy is calculated as the proportion of correctly predicted labels compared to actual labels in the test set.

iii. This metric provides insight into how well the model generalizes to new data.

5.6.5 Saving the Trained Model

i. To save both the trained model and its accuracy for future use.

ii. The trained neural network (net) and its classification accuracy are saved into a .mat file named ‘trainedANNModel.mat’. This allows users to easily load and use this model later without needing to retrain it.

Output

The ouput of classification is saved as matlab data format under 'trainedANNModel.mat' file with output accuracy of about 87.5% to 93.5%. This trained model can further used for real time implementation purpose.

5.7 K-FOLD CROSS VALIDATION

Once classification is done, the trained model and the classifier under cross-validation process to understand the performance of the classification. K-fold cross-validation is a widely used technique in machine learning for assessing the performance of predictive models. This method involves partitioning the dataset into k equal-sized subsets, known as "folds." The primary goal of k-fold cross-validation is to ensure that each observation in the dataset has the opportunity to be used for both training and validation, thereby providing a more reliable estimate of the model's ability to generalize to unseen data.

The process begins by randomly shuffling the dataset and then dividing it into k folds. For each iteration, one fold is reserved as the validation set while the remaining k−1 folds are used for training the model. This procedure is repeated k times, with each fold serving as the validation set exactly once. The performance metrics obtained from each iteration are then averaged to produce a single estimation of model accuracy.

K-fold cross-validation is particularly beneficial because it reduces variability in performance estimates that can occur from a single train-test split. Common choices for k include 5 or 10, as these values have been empirically shown to provide a good balance between bias and variance in model evaluation. By utilizing this method, practitioners can gain insights into how well their models are likely to perform on new, unseen data, making it an essential tool in the model selection and validation process

5.7.1 Steps Involved in K-fold Cross Validation

1) **Data Loading:** The function begins by loading the prepared dataset, which contains features and corresponding labels. This data is typically stored in a .mat file format.
2) **Initialization:** It determines the total number of samples in the dataset and randomly shuffles the indices of these samples to ensure that each fold is representative of the overall dataset. The size of each fold is calculated based on the total number of samples divided by the number of folds, k.
3) **Loop Through Folds:** The function enters a loop that iterates over each of the k folds. In each iteration:
    i. It defines the indices for the validation set, which consists of one fold, and identifies the remaining indices for the training set.
    ii. Using these indices, it creates separate training and validation datasets for features and labels.
4) **Model Training:** The function then loads a pre-trained ANN model from a specified file. It trains this model using the training dataset created in the current fold iteration.

5) **Model Validation:** After training, the model is validated using the validation dataset. The predicted labels for the validation set are generated by passing the validation features through the trained network.
6) **Accuracy Calculation:** The accuracy for the current fold is computed by comparing the predicted labels against the actual labels from the validation set. This accuracy is stored in an array for later analysis.
7) **Display Results:** After processing all folds, the function calculates and displays the average accuracy across all folds, providing an overall assessment of the model's performance.

Output

After executing the k-fold cross validation in MATLAB and the output is as follows:

```
>> kFoldCrossValidation(10)

    Fold 1 Accuracy: 93.75%

    Fold 2 Accuracy: 100%

    Fold 3 Accuracy: 87.5%

    Fold 4 Accuracy: 100%

    Fold 5 Accuracy: 87.5%

    Fold 6 Accuracy: 81.25%

    Fold 7 Accuracy: 93.75%

    Fold 8 Accuracy: 93.75%

    Fold 9 Accuracy: 100%

    Fold 10 Accuracy: 87.5%

    Average Accuracy across 10 folds: 92.5%

    >>
```

CHAPTER 9

RESULT AND DISCUSSION

An algorithm for patient specific hand gesture recognition based on artificial feed forward neural network and emg has been successfully developed, and the following conclusions have been drawn:

- The Electromyography signal is recorded from a person using Bioamp EXG pill and spike recorder during morning, afternoon and night as a sequence for hand movements like Tripod, pinch and Power grip, showing difference in their amplitudes.

- The recorded raw signal contains noises, so denoised using Variable Mode Decomposition method (VMD) was performed to eliminate the noise, such as power line interference, white Gaussian noise, and baseline wandering, which are all often contaminate EMG signals.

- The output of VMD have five Intrinsic Mode Function (IMF) outputs and the best IMF output is selected by means calculating signal-to-noise ratio. From the observation it shows out of five IMF indices, IMF index 1 and 2 showing high signal to noise ratio mostly when noise threshold is set to be 5%.

- Segmented signal undergoes feature extraction processes under time and frequency domains and features such as Mean value, Standard deviation (SD), Root Mean Square (RMS), Zero Crossing (ZC) are obtained from time domain and Mean Power spectral Density (PSD), Energy in frequency bands are obtained from frequency domain using Short-Time Fourier Transform (STFT) and Continuous Wavelet Transform (CWT).

- Extracted features undergoes classification process by means of Feed forward Neural Network (FNN) with training and testing set restricted to 80% and 20%, showing accuracy of about 87.5% to 93.5%.

- When the feature extraction done with CWT alone with minimal data, the result of accuracy is around 33%. But when the data increased to 160 numbers and using both time and frequency domain features the accuracy increased notably.

- Trained model undergoes K-fold cross validation with the same data and showing average accuracy of 92.5%

WORK SCHEDULE FOR PHASE II

- Design and development of hardware- Prosthetic hand.
- Implementation of trained algorithm model in hardware.
- Testing and Validation.

CHAPTER 9

CONCLUSION

By implementing the Artificial Feed-forward Neural Network for the features that extracted using both time and frequency domains, from the VMD denoised emg signal and shows accuracy about 87.5% to 93.5% to recognize different hand gestures like tripod, pinch and power grip. In future, the trained ANN model have to implemented in real time prosthetics.

REFERENCES

1. Ahmed W. Shehata, Heather E. Williams, Jacqueline S. Hebert, and Patrick M. Pilarski. (2021). "Machine Learning for the Control of Prosthetic Arms Using Electromyographic Signals for Improved Performance." IEEE 10.1109.

2. Avilés-Mendoza, K.; Gaibor-León, N.G.; Asanza, V.; Lorente-Leyva, L.L.; Peluffo-Ordóñez, D.H. (2023). A 3D Printed, Bionic Hand Powered by EMG Signals and Controlled by an Online Neural Network. Biomimetics, 8, 255.

3. Benalcázar, M. E., Zea, J. A., Jaramillo, A. G., Anchundia, C. E., Zambrano, P., & Segura, M. (2018). Real-time hand gesture recognition based on artificial feed-forward neural networks and EMG. EURASIP Journal on Advances in Signal Processing, 2018(1), Article 12.

4. Boyer, M.; Bouyer, L.; Roy, J.-S.; Campeau-Lecours, A. (2023). Reducing Noise, Artifacts and Interference in Single-Channel EMG Signals: A Review. Sensors, 23, 2927.

5. Kocejko, T., Zubowicz, T., Weglerski, R., & Ruminski, J. (2018). Design aspects of a low-cost prosthetic arm for people with severe movement disabilities. Journal of Biomedical Engineering and Medical Devices, 1(1), Article 3.

6. Li W., Shi P., and Yu H. (2021). Gesture Recognition Using Surface Electromyography and Deep Learning for Prostheses Hand: State-of-the-Art, Challenges, and Future. Frontiers in Neuroscience, 15:621885.

7. Parajuli, N., Sreenivasan, N., Bifulco, P., Cesarelli, M., Savino, S., Niola, V., Esposito, D., Hamilton, T. J., Gunawardana, U., Gargiulo, G. D., & Naik, G. R. (2019). Real-time EMG based pattern recognition control for hand prostheses: A review on existing methods, challenges and future implementation. MPDI-Sensors, 19(20), 4596.

8. Priya, E., & Shyamkumar, M. (2023). A machine learning approach to control a prosthetic arm via signals from residual limb: A boon for amputees. International Journal of Engineering Research and Applications, 13(3), 45-52.

9. Rubana H. Chowdhury et al. (2013). Surface Electromyography Signal Processing and Classification Techniques. Sensors MDPI, 13(12), 12431-12466.

10. Wand, M., Kirstoffersen, M., Franzke, A.W., & Schmidhuber, J. (2022). Analysis of Neural Network based Proportional Myoelectric Hand Prosthesis Control. IEEE Transactions on Biomedical Engineering, 69(7), 2283-2293.

11. Wang Y.; Tian Y.; She H.; Jiang Y.; Yokoi H.; Liu Y. (2022). Design of an Effective Prosthetic Hand System for Adaptive Grasping with the Control of Myoelectric Pattern Recognition Approach. Micromachines, 13(219).

APPENDIX

A1 MATLAB CODE FOR GENERATING GRAPHICAL DATA

The Matlab code for generating graphical data from wave file as follows

```
% Specify the filenames (use full paths if not in the current directory)
filenames = { ...
   'Filename1.wav', ... % Replace with your actual file paths
   'Filename2.wav', ...            % Add more files as needed
   'Filename3.wav' ...
};

% Loop through each file
for i = 1:length(filenames)
   filename = filenames{i}; % Get the current filename

   % Read the WAV file
   [y, Fs] = audioread(filename); % y is the audio data, Fs is the sample rate

   % Create a time vector based on the length of y and sample rate Fs
   t = (0:length(y)-1) / Fs;

   % Create a new figure window for each file
   figure;
   plot(t, y); % Plot time vs. EMG data
```

```
    xlabel('Time (seconds)'); % Label for x-axis
    ylabel('Amplitude'); % Label for y-axis
    title(['EMG Signal Waveform - ' filename]); % Title of the plot with filename
    grid on; % Add a grid for better visualization

    % If stereo, plot both channels
    if size(y, 2) == 2
        hold on; % Hold the current plot
        plot(t, y(:,2)); % Plot the second channel if exists
        legend('Channel 1', 'Channel 2'); % Add legend for clarity
    end

end
```

A2 MATLAB CODE FOR CREATING EXCEL DATA

```
% Specify the filenames (use full paths if not in current directory)
filenames = { ...
    'Tripod_2024-11-23_19.49.03.wav', ... % Replace with your actual file paths
    'Pinch_2024-11-23_19.50.14.wav', ...  % Add more files as needed
    'Grip_2024-11-23_19.50.28.wav' ...
};

% Initialize Excel application outside the loop
excelApp = actxserver('Excel.Application'); % Start Excel application
```

```
excelApp.Visible = false; % Make Excel invisible (set to true to see it)


% Loop through each file
for i = 1:length(filenames)
    filename = filenames{i}; % Get the current filename


    % Read the WAV file
    [y, Fs] = audioread(filename); % y is the audio data, Fs is the sample rate


    % Create a time vector based on the length of y and sample rate Fs
    t = (0:length(y)-1) / Fs;


    % Prepare data for Excel
    % If stereo, use only one channel (e.g., left channel)
    if size(y, 2) == 2
        y = y(:, 1); % Select first channel
    end


    data = table(t', y); % Create a table with time and amplitude


    % Generate output filename by replacing .wav with .xlsx
    [~, name, ~] = fileparts(filename); % Get the base name without extension
    outputFilename = [name '.xlsx']; % Create output filename
    % Write to Excel
    writetable(data, outputFilename); % Write table to Excel

    % Open Excel and add custom headers
    try
        workbook = excelApp.Workbooks.Open(fullfile(pwd, outputFilename)); % Open
the workbook
        sheet = workbook.Sheets.Item(1); % Select the first sheet

        % Add filename in A1 and merge A1 and B1
        sheet.Range('A1:B1').Merge(); % Merge cells A1 and B1
        sheet.Range('A1').Value = name;  % Add filename without prefix

        % Add custom axis labels in specific cells
        sheet.Range('A2').Value = 'Time in seconds';  % X-axis label
        sheet.Range('B2').Value = 'Amplitude value';  % Y-axis label

        % Save changes and close workbook
        workbook.Save();
        workbook.Close(false);

    catch ME
```

```
        disp(['An error occurred while accessing Excel for ' name ':']);
        disp(ME.message);
    end

end

% Quit Excel application after processing all files
excelApp.Quit();
delete(excelApp); % Clean up COM server

disp('Data written to Excel files with custom axis labels and filenames for all specified
WAV files.');
```

A3 MATLAB CODE FOR VARIATIONAL MODE DECOMPOSITION

The MATLAB code to perform Variational Mode Decomposition as follows:

```
% Open a file dialog to select multiple WAV files
[filenames, path] = uigetfile('*.wav', 'Select WAV Files', 'MultiSelect', 'on');

% Check if the user canceled the dialog
if isequal(filenames, 0)
    disp('User canceled the file selection.');
else
    % If only one file is selected, convert it to a cell array for consistency
    if ischar(filenames)
```

```
        filenames = {filenames}; % Convert to cell array
    end

    % Loop through each selected file
    for i = 1:length(filenames)
        filename = fullfile(path, filenames{i}); % Get the full path of the current
filename

        % Read the WAV file
        [emgSignal, Fs] = audioread(filename); % emgSignal is the audio data, Fs is the
sample rate

        % Perform Variational Mode Decomposition
        [imf, residual] = vmd(emgSignal, 'NumIMF', 5); % Adjust NumIMF as needed

        % Time vector for plotting
        t = (0:length(emgSignal)-1) / Fs;

        % Plotting only the IMFs
        figure;
        numIMFs = size(imf, 2); % Number of IMFs
        for j = 1:numIMFs
            subplot(numIMFs, 1, j);
            plot(t, imf(:, j));
            title(['IMF ' num2str(j) ' from ' filenames{i}]);
            xlabel('Time (s)');
            ylabel('Amplitude');
            grid on;
        end

        % Adjust layout for better visibility
        sgtitle(['VMD of EMG Signal: ' filenames{i}]); % Overall title for the figure

        % Save the figure as a .fig file with the same name as the original file
        [~, name, ~] = fileparts(filenames{i}); % Get the base name without extension
        savefig(gcf, fullfile(path, [name '_VMD_IMFs.fig'])); % Save as .fig file with
full path

        close(gcf); % Close figure after saving to avoid cluttering workspace
    end
end
```

A4 MATLAB CODE FOR SELECTING BEST IMF:

The MATLAB code to perform SNR and to select best IMF as follows:

```
% To select multiple MATLAB figure files
[filenames, pathname] = uigetfile('*.fig', 'Select MATLAB Figure Files', 'MultiSelect',
'on');
```

```
if isequal(filenames, 0)
    disp('User canceled the file selection.');
    return; % Exit if no files are selected
end

% Ensure filenames is a cell array
if ischar(filenames)  % If only one file is selected, convert to cell array
    filenames = {filenames};
end

% Create a new folder for saving SNR outputs
outputDir = fullfile(pathname, 'SNR outputs'); % Define the output directory path
if ~exist(outputDir, 'dir')  % Check if the directory exists
    mkdir(outputDir);  % Create the directory if it does not exist
end

% Initialize an array to store SNR values for all figures
allSNRValues = cell(length(filenames), 1); % Cell array to hold SNR values for each
figure

% Loop through each selected figure file
for f = 1:length(filenames)
    % Construct the full path to the selected figure file
    figFilePath = fullfile(pathname, filenames{f});


    % Open the specified figure
    hFig = openfig(figFilePath); % Open figure


    % Get all axes in the figure
    axesHandles = findall(hFig, 'Type', 'axes');


    % Initialize an array to store SNR values for this figure
    numIMFs = length(axesHandles); % Assuming each axis corresponds to an IMF
    snrValues = zeros(numIMFs, 1);


    bestSNR = -Inf; % Initialize best SNR
    bestIMFIndex = 0; % Initialize index of best IMF
    bestIMFData = []; % Initialize variable to store best IMF data


    % Define your sampling frequency (Fs) here; adjust as necessary based on your
data
    Fs = 10000; % Example: Set this to your actual sampling frequency in Hz


    % Loop through each axis (IMF)
    for k = 1:numIMFs
        % Get the current axis handle
    currentAxis = axesHandles(k);

    % Get the data from the current axis (assuming it's a line plot)
    lineHandles = findall(currentAxis, 'Type', 'line');

    if ~isempty(lineHandles)
        % Extract Y data from the first line object in the current axis
        currentIMF = get(lineHandles(1), 'YData');

        % Calculate signal power (mean of squares)
        signalPower = mean(currentIMF.^2);

        % Estimate noise: Here we define a threshold for noise estimation
        threshold = 0.05 * max(abs(currentIMF)); % Adjust threshold as needed
        noise = currentIMF(abs(currentIMF) < threshold); % Values below threshold
considered as noise

        % Calculate power of the estimated noise (mean of squares)
        noisePower = mean(noise.^2);

        % Calculate SNR in dB
        if noisePower > 0  % Prevent division by zero
            snrValues(k) = 10 * log10(signalPower / noisePower);
        else
            snrValues(k) = Inf; % Assign infinity if there's no noise
        end

        % Debugging output for signal and noise powers
        fprintf('Signal Power for IMF %d in file "%s": %.6f\n', k, filenames{f},
signalPower);
        fprintf('Noise Power for IMF %d in file "%s": %.6f\n', k, filenames{f},
noisePower);

        % Display the SNR value for the current IMF
        fprintf('SNR for IMF %d in file "%s": %.2f dB\n', k, filenames{f},
snrValues(k));

        % Check if this is the best IMF based on SNR
        if snrValues(k) > bestSNR
            bestSNR = snrValues(k);
            bestIMFIndex = k;
            bestIMFData = currentIMF; % Store the best IMF data
        end

    else
        fprintf('No line data found for IMF %d in file "%s".\n', k, filenames{f});
    end
    end

    % Display which IMF is best based on SNR
    fprintf('Best IMF is IMF %d with SNR: %.2f dB\n', bestIMFIndex, bestSNR);

    % Create a new figure for the best IMF and plot it using time on X-axis
    t_bestIMF = (0:length(bestIMFData)-1) / Fs; % Time vector in seconds

    bestIMFFigure = figure;
    plot(t_bestIMF, bestIMFData); % Use time vector for X-axis instead of sample
indices
    title(['Best IMF (Index ' num2str(bestIMFIndex) ') of ' filenames{f}]);
    xlabel('Time (s)'); % Change label to Time (seconds)
    ylabel('Amplitude');
    grid on;

    % Save this best IMF plot as a new .fig file with modified name in output directory
    bestIMFFigName = fullfile(outputDir, ['Best_IMF_' num2str(bestIMFIndex) '_'
filenames{f}]);
    savefig(bestIMFFigure, bestIMFFigName);

    % Store SNR values for this figure in the cell array
    allSNRValues{f} = snrValues;
    % Plotting SNR values for visualization per figure and save as .fig file in output
directory
    snrFigure = figure;
    bar(snrValues);
    title(['SNR of Each IMF for ', filenames{f}]);
    xlabel('IMF Index');
    ylabel('SNR (dB)');
    grid on;

    % Save the SNR plot as a .fig file in output directory with modified name
    snrFigName = fullfile(outputDir, [filenames{f}(1:end-4) '_SNR.fig']);  % Remove
.fig extension and add _SNR
    savefig(snrFigure, snrFigName);  % Save as .fig
end
disp('Figures saved successfully in "SNR outputs" directory.');
% Close all figures at the end of processing.
close all;
```

A5 MATLAB CODE FOR SEGMENTATION:

The MATLAB code to perform segmentation as follows:

```
function main()
    % Step 1: Load multiple .fig files invisibly
    [filenames, pathname] = uigetfile('*.fig', 'Select MATLAB Figure Files',
'MultiSelect', 'on');
```

```
if isequal(filenames, 0)
    disp('User canceled the file selection.');
    return; % Exit if no files are selected
end

% Ensure filenames is a cell array
if ischar(filenames)  % If only one file is selected, convert to cell array
    filenames = {filenames};
end

% Create a new folder for saving segments
outputDir = fullfile(pathname, 'Segments'); % Define the output directory path
if ~exist(outputDir, 'dir')  % Check if the directory exists
    mkdir(outputDir);  % Create the directory if it does not exist
end

% Loop through each selected figure file
for f = 1:length(filenames)
    % Construct the full path to the selected figure file
    figFilePath = fullfile(pathname, filenames{f});

    % Open figure invisibly
    hFig = openfig(figFilePath, 'invisible');
```

```
% Step 2: Get axes and line objects
axesHandles = findall(hFig, 'Type', 'axes'); % Find all axes in the figure

if isempty(axesHandles)
    error('No axes found in the selected figure: %s', filenames{f});
end

lineHandles = findall(axesHandles(1), 'Type', 'line');

if isempty(lineHandles)
    error('No line objects found in the selected axes of figure: %s', filenames{f});
end

% Step 3: Extract X and Y data from the first line object
xData = get(lineHandles(1), 'XData'); % Time data (in seconds)
yData = get(lineHandles(1), 'YData'); % EMG amplitude data

% Close the figure after extracting data
close(hFig);

% Step 4: Define time intervals for segmentation (in seconds)
segmentTimes = [
        0.1008, 0.7402;   % Segment 1: from 5s to 15s
        0.9808, 1.7005;   % Segment 2: from 16s to 27s
        2.03, 2.6801;   % Segment 3: from 33s to 43s
        3.007, 3.6783;   % Segment 4: from 45s to 54s
        4, 4.6397;   % Segment 5: from 59s to 68s
    ];

    % Get base name for saving figures
    [~, baseName, ~] = fileparts(filenames{f}); % Extract base name without
extension

    % Validate indices and extract segments
    for i = 1:size(segmentTimes, 1)
        startTime = segmentTimes(i, 1);
        endTime = segmentTimes(i, 2);

        % Convert time values to indices based on xData (assuming xData is in
seconds)
        startIdx = find(xData >= startTime, 1);
        endIdx = find(xData <= endTime, 1, 'last');

        % Check for valid indices
        if isempty(startIdx) || isempty(endIdx) || startIdx < 1 || endIdx > length(yData)
|| startIdx >= endIdx
```

```
            error('Invalid segment times for segment %d in file "%s": [%d s, %d s]', i,
filenames{f}, startTime, endTime);
        end

        % Extract the segment
        segmentX = xData(startIdx:endIdx);
        segmentY = yData(startIdx:endIdx);

        % Adjust x-axis values to start from zero for each segment
        adjustedX = segmentX - segmentX(1);

        % Plot the segment with adjusted x-axis values
        figure;
        plot(adjustedX, segmentY);
        title([baseName ' - Segment ' num2str(i)]); % Title with base name
        xlabel('Time (s)');
        ylabel('EMG Amplitude');

        % Save the figure as .fig file in "Segments" folder with same base name +
segmented index
        saveas(gcf, fullfile(outputDir, [baseName '_Segmented' num2str(i) '.fig']));

        close(gcf); % Close the figure after saving
        end


    end


    disp('Segments saved as separate graphs successfully in "Segments" folder.');
end
```

A6 MATLAB CODE FOR FEATURE EXTRACTION:

The MATLAB code to perform feature extraction as follows:

```
function extractAndSaveEMGFeatures()
    % To select multiple MATLAB figure files
    [filenames, pathname] = uigetfile('*.fig', 'Select MATLAB Figure Files',
'MultiSelect', 'on');
    if isequal(filenames, 0)
        disp('User canceled the file selection.');
        return; % Exit if no files are selected
    end

    % Ensure filenames is a cell array
    if ischar(filenames)  % If only one file is selected, convert to cell array
        filenames = {filenames};
    end

    % Initialize a structure to hold all features for each file
    allFeatures = struct();
    maxFeatureLength = 0; % Initialize variable to track maximum feature length

    % Loop through each selected figure file
    for k = 1:length(filenames)
        % Construct the full path to the selected figure file
        figFilePath = fullfile(pathname, filenames{k});

        % Open the specified figure
        hFig = openfig(figFilePath); % Open figure

        % Get all axes in the figure
        axesHandles = findall(hFig, 'Type', 'axes');

        if ~isempty(axesHandles)
            currentAxis = axesHandles(1); % Get the first axis handle
            lineHandles = findall(currentAxis, 'Type', 'line');

            if ~isempty(lineHandles)
                % Extract X (time) and Y (amplitude) data from the first line object in the
current axis
                timeData = get(lineHandles(1), 'XData');
                filteredEMG = get(lineHandles(1), 'YData');
```

```
fs = 10000; % Sampling frequency in Hz (adjust as necessary)


% Extract features from the EMG signal
features = extractEMGFeatures(filteredEMG, fs);


% Update maximum feature length
maxFeatureLength = max(maxFeatureLength, length(features));


% Sanitize filename for use as a field name
fieldName = strrep(filenames{k}, '.fig', '');
fieldName = regexprep(fieldName, '[^\w]', '_');


% Determine label based on filename content
if contains(filenames{k}, 'Tripod', 'IgnoreCase', true)
   label = 1;
elseif contains(filenames{k}, 'Grip', 'IgnoreCase', true)
   label = 2;
else
   label = NaN;
   warning(['No valid action found in filename: ', filenames{k}]);
end
```

```
            % Store features and corresponding label in allFeatures structure with
sanitized field name
            allFeatures.(fieldName).features = features;
            allFeatures.(fieldName).label = label;

         else
            error(['No line data found in the selected figure: ' filenames{k}]);
         end
      else
         error(['No axes found in the selected figure: ' filenames{k}]);
      end

      if ishghandle(hFig)
         close(hFig);
      end

   end

   % Pad features to ensure consistent length based on maxFeatureLength
   fieldNames = fieldnames(allFeatures);
   for i = 1:length(fieldNames)
      actionName = fieldNames{i};
```

```
        features = allFeatures.(actionName).features;

        % Pad with zeros to match maxFeatureLength
        if length(features) < maxFeatureLength
            features(end+1:maxFeatureLength) = 0;
        elseif length(features) > maxFeatureLength
            features = features(1:maxFeatureLength);
        end

        allFeatures.(actionName).features = features; % Update padded features back
into structure
    end

    % Save all extracted features to a .mat file for further processing
    save(fullfile(pathname, 'ExtractedFeatures.mat'), 'allFeatures');

    disp('All features extracted and saved successfully in ''ExtractedFeatures.mat''.');

    % Step 2: Prepare Data for Classification
    featureList = [];
    labelList = [];

    fieldNames = fieldnames(allFeatures);
```

```
for i = 1:length(fieldNames)
    actionName = fieldNames{i};

    % Extract features and label
    features = allFeatures.(actionName).features;

    % Check dimensions of the current feature vector
    disp(['Size of features for ', actionName, ':']);
    disp(size(features));  % Display size of current feature vector

    % Ensure features are in matrix form (row vector)
    if isrow(features)
        features = features';  % Transpose to make it a column vector if necessary
    end

    label = allFeatures.(actionName).label;

    % Append only if dimensions match or handle inconsistencies accordingly
    if isempty(featureList)
        featureList = features';  % Initialize with the first feature set (as a row)
        labelList = [label];       % Initialize labels
    else
        if size(features, 1) == size(featureList, 2)  % Check if dimensions match
```

```
            featureList = [featureList; features'];  % Append as a new row
            labelList = [labelList; label];
        else
            warning(['Feature dimensions do not match for ', actionName]);
        end
    end
end

% Check dimensions before saving
disp('Size of featureList:');
disp(size(featureList));  % Should show [num_samples, num_features]

disp('Size of labelList:');
disp(size(labelList));     % Should show [num_samples, 1]

% Save Prepared Data for ANN Training as .mat file and Excel file
save(fullfile(pathname, 'PreparedData.mat'), 'featureList', 'labelList');

%% New Code: Save Features and Labels to Excel File
featureTable = array2table(featureList);   % Convert feature list to table format
featureTable.Labels = labelList;           % Add labels as a new column in the table

% Define output Excel file name and path
```

```
    excelFilePath = fullfile(pathname, '7PreparedData.xlsx');


    writetable(featureTable, excelFilePath);   % Write table to Excel file
    disp(['Feature data saved successfully in "', excelFilePath, '".']);
end

function features = extractEMGFeatures(filteredEMG, fs)
   features = [];

   % Time Domain Features
   meanValue = mean(filteredEMG);
   stdValue = std(filteredEMG);
   rmsValue = sqrt(mean(filteredEMG.^2));
   zeroCrossings = sum(diff(sign(filteredEMG)) ~= 0);

   features = [features; meanValue; stdValue; rmsValue; zeroCrossings];

   % Wavelet Transform Features using CWT
   [wt, f_wavelet] = cwt(filteredEMG, 'amor', fs);

   energyWavelet = sum(abs(wt).^2, 2);
   features = [features; energyWavelet];
```

```
  % Frequency Domain Features using Short-Time Fourier Transform (STFT)
  windowSize = round(fs * 0.1);
  overlapSize = round(windowSize * 0.5);

  FFTLength = max(1024, windowSize);

  [S, F, T] = stft(filteredEMG, fs, 'Window', hamming(windowSize), ...
              'OverlapLength', overlapSize, 'FFTLength', FFTLength);

  powerSpectralDensity = abs(S).^2;
  meanPSD = mean(powerSpectralDensity, 2);
  features = [features; meanPSD];

  return;
end
```

A7 MATLAB CODE FOR CLASSIFICATION:

The MATLAB code to perform classification as follows:

```
function trainANNClassifier()
   % Load Prepared Data from .mat file
   data = load('PreparedData.mat');

   featureList = data.featureList;
```

```
labelList = data.labelList;


% Check dimensions of featureList and labelList

disp('Size of featureList:');

disp(size(featureList));  % Should show [num_samples, num_features]


disp('Size of labelList:');

disp(size(labelList));     % Should show [num_samples, 1]


%% Step 1: Split Data into Training and Testing Sets

numSamples = size(featureList, 1); % Total number of samples


% Randomly shuffle the indices

indices = randperm(numSamples);


% Define split point (e.g., 80% for training)

splitPoint = round(0.8 * numSamples); % For an 80/20 split


% Create training and testing sets

trainIndices = indices(1:splitPoint);

testIndices = indices(splitPoint+1:end);


trainFeatures = featureList(trainIndices, :);
```

```
trainLabels = labelList(trainIndices);


testFeatures = featureList(testIndices, :);

testLabels = labelList(testIndices);


disp('Training Features Size:');

disp(size(trainFeatures));

disp('Training Labels Size:');

disp(size(trainLabels));


disp('Testing Features Size:');

disp(size(testFeatures));

disp('Testing Labels Size:');

disp(size(testLabels));


%% Step 2: Define and Train the ANN

hiddenLayerSize = 10;

net = feedforwardnet(hiddenLayerSize);


% Train the network using transposed trainFeatures and trainLabels

net = train(net, trainFeatures', trainLabels');


%% Step 3: Evaluate the Model on Test Set
```

```
    predictedLabels = net(testFeatures');
    predictedLabels = round(predictedLabels);

    %% Step 4: Calculate Accuracy (optional)
    accuracy = sum(predictedLabels' == testLabels) / length(testLabels);
    disp(['Classification Accuracy on Test Set: ', num2str(accuracy * 100), '%']);

    %% Step 5: Save the Trained Model
    save('trainedANNModel.mat', 'net', 'accuracy'); % Save the network and accuracy
end

% Example usage:
% Call this function to train your ANN classifier.
% trainANNClassifier();
```

A MATLAB CODE FOR K-FOLD CROSS-VALIDATION

The MATLAB code to perform K-fold cross-validation as follows:

```
function kFoldCrossValidation(k)
    % Load Prepared Data from .mat file
    data = load('PreparedData.mat');

    featureList = data.featureList;
    labelList = data.labelList;
```

```
% Check dimensions of featureList and labelList
numSamples = size(featureList, 1); % Total number of samples
indices = randperm(numSamples); % Randomly shuffle indices
foldSize = floor(numSamples / k); % Size of each fold
accuracies = zeros(k, 1); % Array to store accuracies for each fold

for i = 1:k
   % Define validation and training indices
   valIndices = indices((i-1)*foldSize + 1:min(i*foldSize, numSamples));
   trainIndices = setdiff(indices, valIndices);

   % Create training and validation sets
   trainFeatures = featureList(trainIndices, :);
   trainLabels = labelList(trainIndices);

   valFeatures = featureList(valIndices, :);
   valLabels = labelList(valIndices);

   % Load the trained ANN model (assumes it's saved as 'trainedANNModel.mat')
   loadedData = load('trainedANNModel.mat');
   net = loadedData.net; % Load the trained network

   % Train the network using transposed trainFeatures and trainLabels
```

```
        net = train(net, trainFeatures', trainLabels');


        % Validate the model
        predictedLabels = net(valFeatures');
        predictedLabels = round(predictedLabels);


        % Calculate accuracy for this fold
        accuracies(i) = sum(predictedLabels' == valLabels) / length(valLabels);


        disp(['Fold ', num2str(i), ' Accuracy: ', num2str(accuracies(i) * 100), '%']);
    end


    % Calculate average accuracy across all folds
    avgAccuracy = mean(accuracies);
    disp(['Average Accuracy across ', num2str(k), ' folds: ', num2str(avgAccuracy *
100), '%']);
end
```