University of Moratuwa
Faculty of Engineering
Department of Electronic & Telecommunication Engineering

BM2210: Biomedical Device Design

Medicore Innovations
Feasibility Report

Group Members

Index No	Name
220005R	Aazir M. A. M.
220562U	Samuditha H. K. P.
220735E	Yashodhara M.H.K.

Contents

1 Introduction

This project introduces an sEMG wrist flexion monitoring system designed specifically for athletes. Using three surface electrodes and a custom electronic circuit, the system captures and processes muscle activity. With data analysis powered by machine learning, it identifies wrist flexion signals and delivers actionable insights. It is compact and portable and helps enhance performance, prevent injuries and support precision sports.

2 Need Statement

"Athletes need a compact and portable solution to effectively monitor wrist muscle health, addressing the limitations of bulky sEMG devices to improve performance and prevent injuries."

3 Proposed Solution

The project features a compact and portable design eliminating the bulky, wired setup of traditional sEMG devices. This innovative design enables continuous tracking of wrist muscle activity during both training sessions and daily activities, ensuring convenience and user comfort. By integrating advanced technology, the device leverages sophisticated signal processing techniques for accurate data collection and employs machine learning algorithms to classify wrist flexion movements and detect abnormalities. This seamless integration ensures uninterrupted performance tracking and reliable health monitoring, making the device a versatile tool for both rehabilitation and fitness applications.

4 Device Details and Methodology

4.1 Schematics

Following are the schematic designs for the electronic circuit of our project.

4.1.1 Power Supply Unit and Charging Circuit

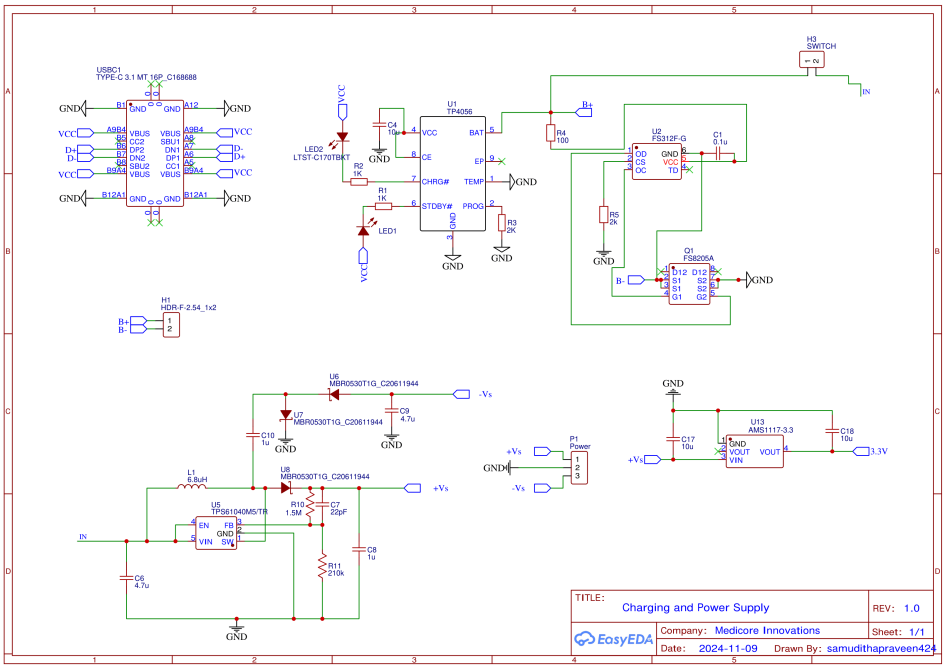


Figure 1: Power Supply Unit

Here we have used a micro USB port to charge the device, which is connected to the LiPo battery through a current-limiting charging circuit. which limits the charging current to 580mA. We designed these schematics using the documentation of the TP4056 LiPo charging module.

Next we have a dual power supply for the Op amp circuitry, which we are using to filter out the noise of the captured emg signal. TPS61040M5/TR Ic and other circuitry provide the needed dual power supply of +8V and -8V.

The next small circuit is a normal regulator circuit which can provide the 3.3V output by getting +8V. 3.3V is needed to power up the esp 32 chip and other modules we are using (example - OLED display)

4.1.2 Filtering Circuit

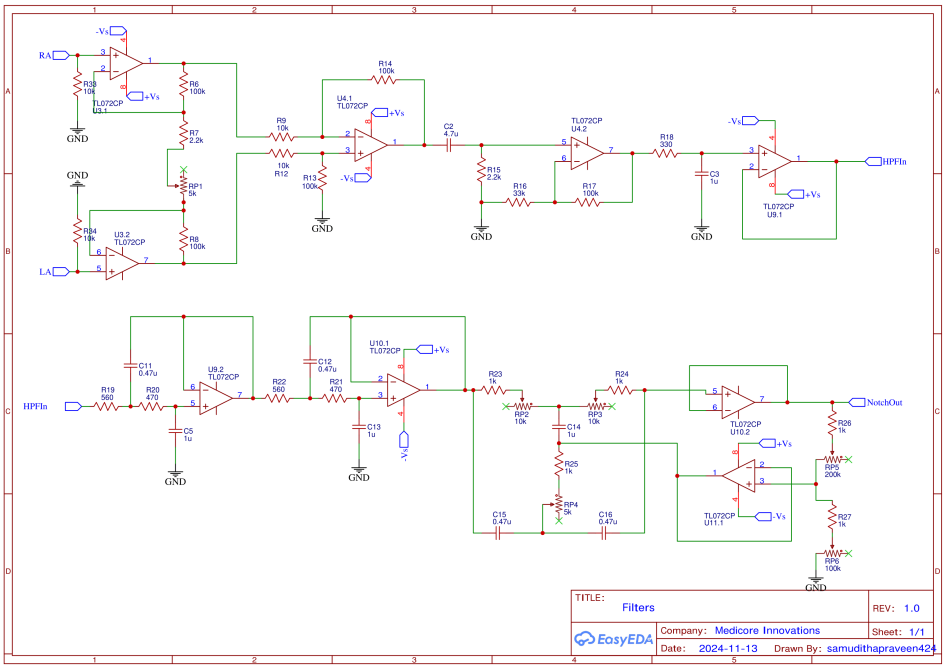


Figure 2: Filtering Circuit

For the filtering process we use three stages

- 1st Order Active High Pass Filter: We use a active high pass filter with a cut-off frequency of 15.4hz. The main purpose of this filter is to remove low frequencies such as the DC components.
- 5th Order Bessel Filter: Here we use a cascaded system of 1st order active low pass filter and two 2nd order active low pass filters. The cut-off frequency is around 450hz to 460hz.
- Notch Filter: In our circuit we use a twin-T notch filter to remove the powerline interfernece of 50hz.We have used five potentiometers to fine tune the notch filter where three of them are used to adjust notch frequency and two of them are used to adjust the required roll-off.

4.1.3 ESP Sampling Circuit

- The EMG signal is shifted by 1.7 volts using tuned potentiometers to ensure it is within the ADC range of the ESP32.
- The ESP32 samples the shifted signal and transmits it wirelessly to a laptop for further processing.
- A push button is included in the circuit to enable user-controlled transmission of the EMG signal.

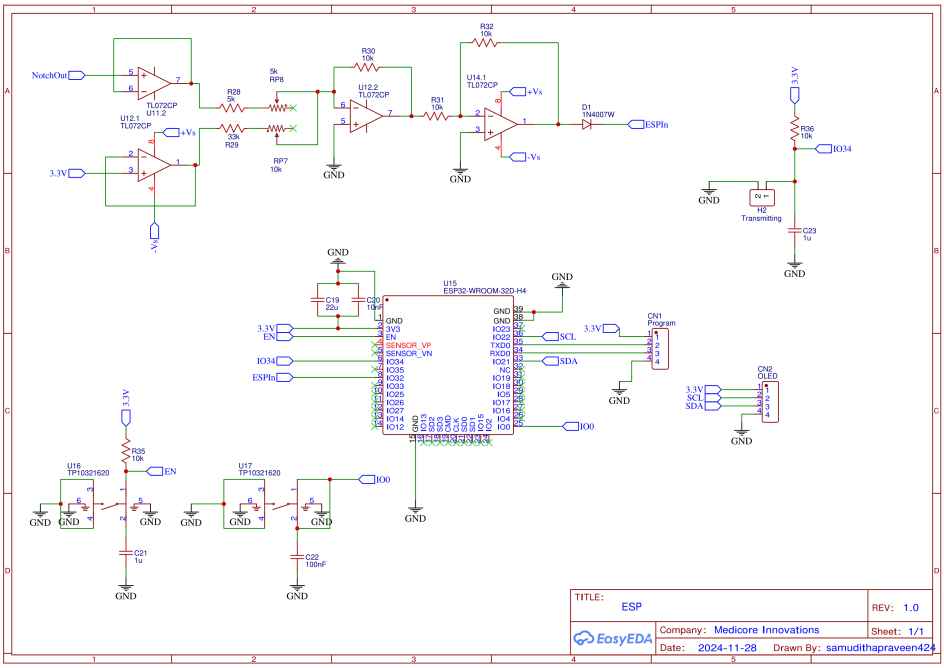


Figure 3: ESP sampling circuit

4.2 PCB designs

To design the printed circuit board (PCB) for our wireless sEMG monitoring device, we utilized EasyEDA software. This tool allowed us to efficiently create and simulate the PCB layout, ensuring optimal design for signal conditioning, sampling, and wireless transmission. EasyEDA's user-friendly interface and extensive library of components made it well-suited for the development of this biomedical application. By leveraging its advanced features, we were able to streamline the PCB design process and ensure the circuit met the requirements of the wireless sEMG monitoring system.

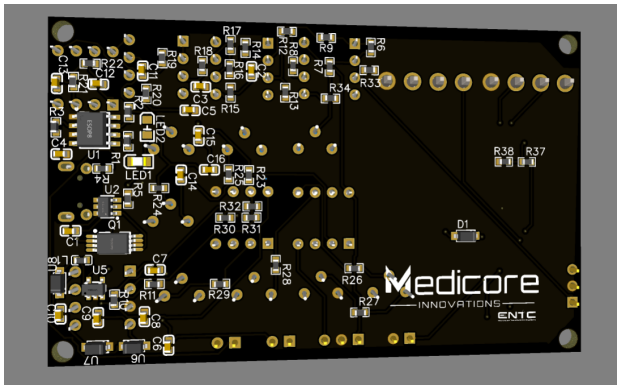


Figure 4: PCB Bottom View

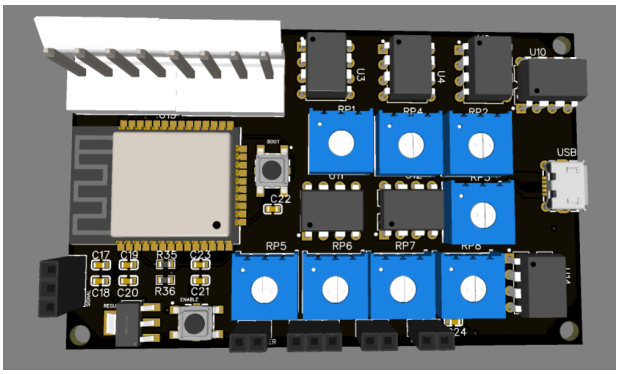


Figure 5: PCB Top View

Since the device is more compact, we designed a 4 layer pcb.

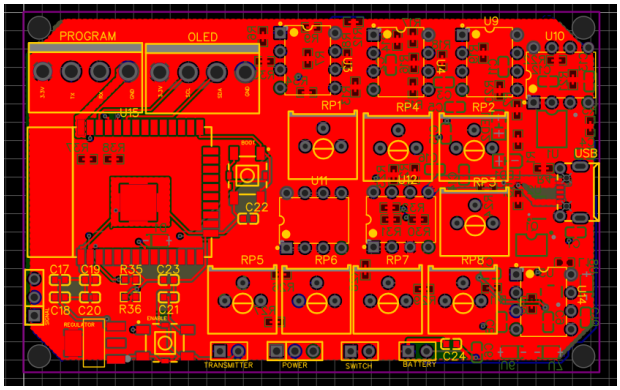


Figure 6: Top Layer of the PCB

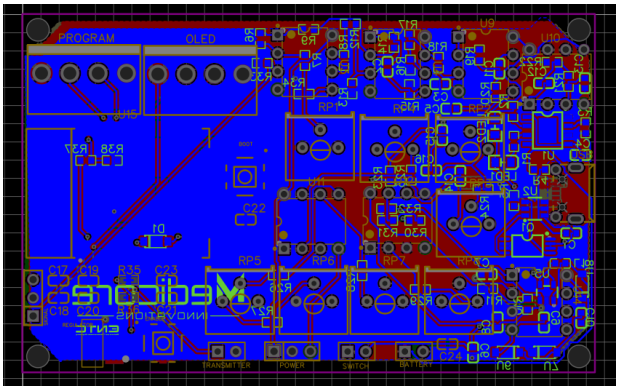


Figure 7: Bottom Layer of the PCB

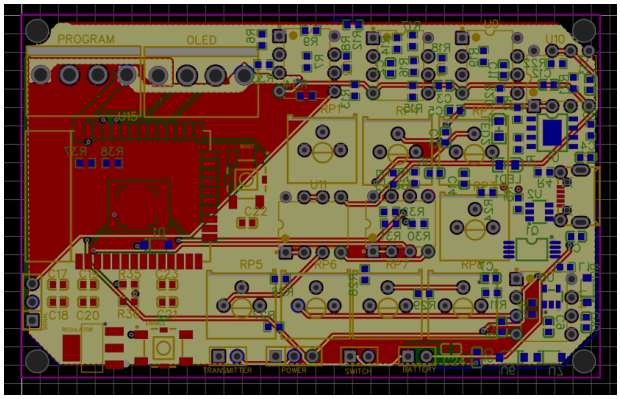


Figure 8: Inner Layer 1 of the PCB

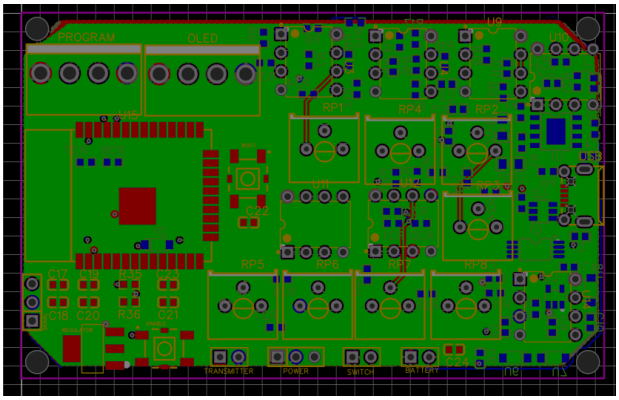


Figure 9: Inner Layer 2 of the PCB

4.3 Machine Learning for Wrist Flexion Classification

The machine learning component of this project involved the classification of wrist flexion signals using a Random Forest classifier.The dataset was taken from `https://archive.ics.uci.edu/ml/datasets/EMG+data+for+gestures`. The dataset was sourced from recorded surface electromyography (sEMG) signals, comprising eight input channels and a class label that represented various wrist flexion movements or states. The dataset included 63,196 samples, each with the following structure:

- Time: Sequential index for each observation.
- Channels: Eight sEMG signal channels (channel1 to channel8) capturing electrical activity during wrist movements.
- Class: Target variable with six classes representing different wrist flexion movements or rest states.

4.3.1 Preprocessing

Before training, the dataset underwent preprocessing steps to ensure its readiness for classification:

- Feature Scaling: Standardized channel data to handle varying scales across features.
- Label Encoding: The target variable (class) was encoded for compatibility with the classifier.
- Data Splitting: The dataset was divided into training (70%) and testing (30%) sets to evaluate model performance.

4.3.2 Model Selection

Random Forest was selected as the classifier due to its robustness against overfitting and its capability to handle high-dimensional data effectively. The model was configured as follows:

- Number of Trees (n_estimators): Number of trees used in the forest.
- Maximum Depth: Limited to optimize computational efficiency while maintaining accuracy.
- Feature Selection: During training, random subsets of features were considered at each split to introduce diversity among trees.

4.3.3 Training and Testing

The Random Forest model was trained on the sEMG dataset using the training set. Each tree in the forest independently classified the data, and the final prediction was made using majority voting across all trees.

4.3.4 Evaluation

The model was evaluated on the test set using metrics such as accuracy, precision, recall, confusion matrix and F1-score. It demonstrated strong classification performance, indicating its effectiveness in distinguishing between different wrist flexion states.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\Nilani Thushanthika> & "C:/Users/Nilani Thushanthi
Accuracy: 0.98

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     12436
           1       0.98      0.97      0.98      6523

    accuracy                           0.98     18959
   macro avg       0.98      0.98      0.98     18959
weighted avg       0.98      0.98      0.98     18959


Confusion Matrix:
 [[12283   153]
 [  170  6353]]
F1 Score: 0.98
Model saved as 'random_forest_updated_model.joblib'
PS C:\Users\Nilani Thushanthika>
```

Figure 10: model evaluation

4.3.5 Graphical User Interface

Graphical User Interface (GUI) is implemented using Streamlit to facilitate data visualization for analyzing wrist flexion signals. This intuitive and user-friendly GUI will serve as the central hub for managing classification, and display processes. Following are the tests done using different datasets and and displaying it in the UI.

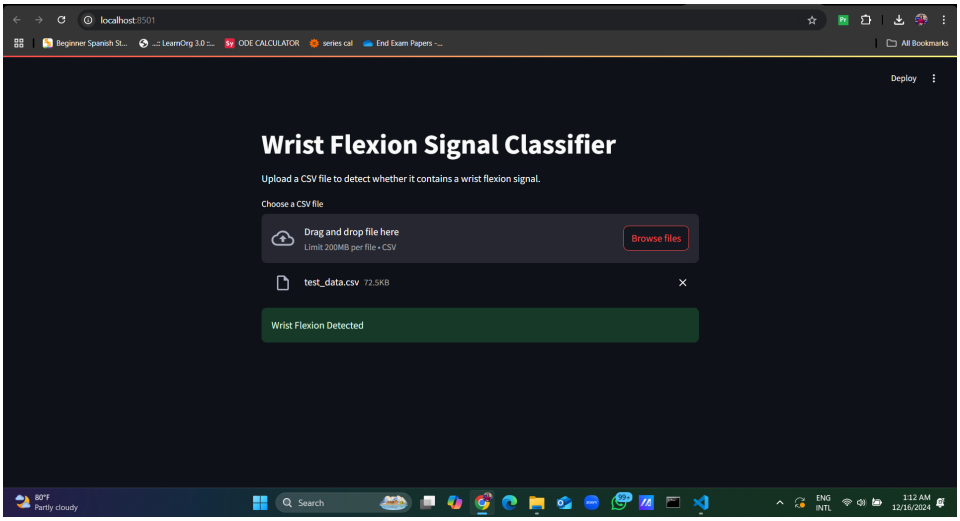


Figure 11: Wrist Flexion Dataset

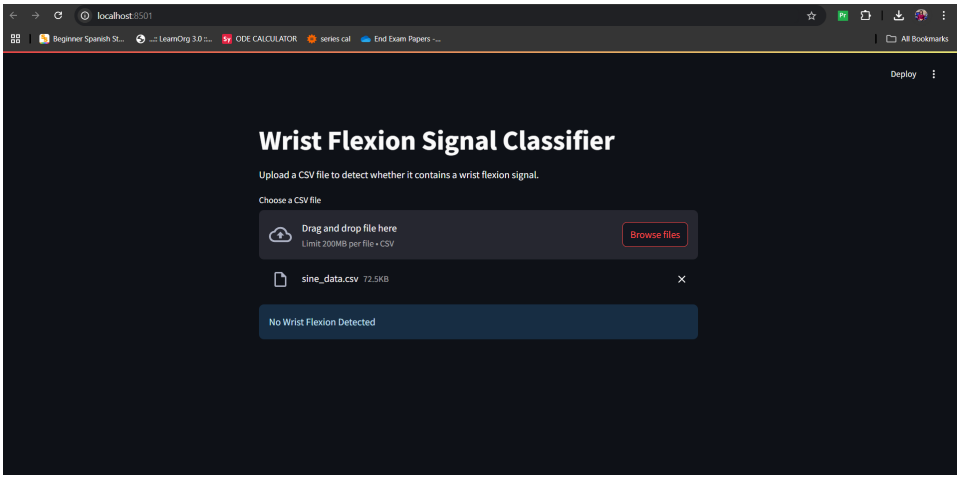


Figure 12: No Wrist Flexion Dataset

4.3.6 Codes

- Model

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
    ↪ confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler
from scipy.signal import welch
import matplotlib.pyplot as plt
import joblib

# Load the dataset
data = pd.read_csv('D:\sem 3\BM device design\ecg\ML part\code testing\
    ↪ converted_data.csv')

# Feature Engineering: Extract Statistical and Frequency Features
def extract_features(row):
    features = {}

    # Statistical features
```

```
    features['mean'] = row.mean()
    features['std'] = row.std()
    features['max'] = row.max()
    features['min'] = row.min()
    features['range'] = row.max() - row.min()

    # Frequency-domain features (adjust nperseg for short signals)
    nperseg = min(len(row), 8)  # Use a maximum of 8 or the length of the row
    freqs, psd = welch(row, fs=100, nperseg=nperseg)
    features['dominant_freq'] = freqs[np.argmax(psd)] if len(freqs) > 0 else 0
    features['power'] = np.sum(psd) if len(psd) > 0 else 0

    return pd.Series(features)

# Apply feature extraction for each sample (row-wise)
X = data.drop(columns=['channel4']).apply(lambda row: extract_features(row), axis=1)

# Ensure target variable matches the number of rows in X
y = (data['channel4'] > 0).astype(int)  # Binary classification

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Initialize and train the classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print(f"F1 Score: {f1:.2f}")

# Visualize feature importances
feature_importances = clf.feature_importances_
plt.barh(X.columns, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Feature Names')
plt.title('Feature Importance in Random Forest')
plt.show()

# Save the model
file_path = "D:\\sem 3\\BM device design\\ecg\\ML part\\code testing\\random_forest_updated_model.joblib"
joblib.dump(clf, file_path)
print("Model saved as 'random_forest_updated_model.joblib'")
```

- Classification on a Specific Dataset

```
import joblib
import numpy as np
import pandas as pd
from scipy.stats import skew, kurtosis  # Import skew and kurtosis functions

# Load the pre-trained model
clf = joblib.load("D:\\sem 3\\BM device design\\ecg\\ML part\\code testing\\
    ↪ random_forest_updated_model.joblib")

# Function to extract 9 features (example with 9 features)
def extract_features(raw_signal):
    # Use only the features included during training
    mean_val = np.mean(raw_signal)
    std_val = np.std(raw_signal)
    max_val = np.max(raw_signal)
    min_val = np.min(raw_signal)
    range_val = max_val - min_val
    median_val = np.median(raw_signal)
    rms_val = np.sqrt(np.mean(raw_signal**2))

    # Combine only these 7 features (to match the training script)
    features = [mean_val, std_val, max_val, min_val, range_val, median_val,
        ↪ rms_val]

    return features

def classify_signal_from_csv(csv_file):
    # Load the CSV data
    data = pd.read_csv(csv_file)

    # Extract the Amplitude column as the raw signal
    raw_signal = data['Amplitude'].values  # Ensure the column name matches
        ↪ your dataset

    # Extract features from the raw signal
    features = extract_features(raw_signal)

    # Reshape the features to match the input format for the classifier (1
        ↪ sample with multiple features)
    features = np.array(features).reshape(1, -1)

    # Predict the class (1 for flexion, 0 for non-flexion)
    prediction = clf.predict(features)

    # Output the prediction for wrist flexion or not
    if prediction == 1:
        print("Wrist Flexion Detected")
    else:
        print("No Flexion")


# Example usage:
csv_file = "D:\sem 3\BM device design\ecg\ML part\code testing\
    ↪ sine_wave_data.csv"  # Provide the path to your CSV file
classify_signal_from_csv(csv_file)
```

- GUI

```
import streamlit as st
import pandas as pd
import numpy as np
from scipy.stats import skew, kurtosis
import joblib

# Load the pre-trained model
clf = joblib.load("D:\\sem 3\\BM device design\\ecg\\ML part\\code testing\\random_forest_updated_model.joblib")

# Function to extract 7 features (consistent with training)
def extract_features(raw_signal):
    mean_val = np.mean(raw_signal)
    std_val = np.std(raw_signal)
    max_val = np.max(raw_signal)
    min_val = np.min(raw_signal)
    range_val = max_val - min_val
    median_val = np.median(raw_signal)
    rms_val = np.sqrt(np.mean(raw_signal**2))

    # Combine the 7 features
    features = [mean_val, std_val, max_val, min_val, range_val, median_val, rms_val]
    return features

# Streamlit interface
st.title("Wrist Flexion Signal Classifier")
st.write("Upload a CSV file to detect whether it contains a wrist flexion signal.")

# File uploader
uploaded_file = st.file_uploader("Choose a CSV file", type="csv")

if uploaded_file is not None:
    # Load the CSV data
    data = pd.read_csv(uploaded_file)

    # Ensure the CSV contains an 'Amplitude' column
    if 'Amplitude' not in data.columns:
        st.error("The uploaded CSV must contain an 'Amplitude' column.")
    else:
        # Extract the Amplitude column as the raw signal
        raw_signal = data['Amplitude'].values

        # Extract features from the raw signal
        features = extract_features(raw_signal)
        features = np.array(features).reshape(1, -1)

        # Predict the class
        prediction = clf.predict(features)[0]

        # Display the result
        if prediction == 1:
            st.success("Wrist Flexion Detected")
        else:
            st.info("No Wrist Flexion Detected")


#streamlit run "d:/sem 3/BM device design/ecg/ML part/code testing/app_updated.py"
```

4.4 Wireless Transmission

We used an ESP32 chip to transmit the EMG signal data to the user's computer. Then a python coded program was used to fetch the data and plot the EMG signal. At the same time an output data file is created for the use of the ML model. Also the ESP32 code is integrated such that necessary messages and battery level can be displayed in the OLED display.

4.4.1 ESP32 Code

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include <WebServer.h>
#include <math.h>

// Define OLED display dimensions
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SSD1306_I2C_ADDRESS 0x3C // in Adafruit_SSD1306.h
// Define OLED reset pin (set to -1 if not used)
#define OLED_RESET    -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Battery level pin and variables
#define BATTERY_PIN 35
#define BUTTON_PIN 15
int batteryLevel = 0;
bool isRecording = false;

// Wi-Fi Access Point credentials
const char* ssid = "Medicore";
const char* password = "12345678";

// Create an HTTP server instance
WebServer server(80);

// EMG signal sampling parameters
const int sampleCount = 10000;  // Number of samples to transmit (time duration)
const int analogPin = 34;     // Analog input pin
const int offset = 2;
float analogValues[sampleCount];  // Array to hold sampled analog values

// Function to collect analog readings
void collectAnalogReadings() {
  for (int i = 0; i < sampleCount; i++) {
    analogValues[i] = (analogRead(analogPin) * 3.3) / 4095.0 - offset; // Convert
       ↪  ADC value to voltage
    delayMicroseconds(500); // 100 kHz sampling rate
  }
}

// Function to handle root page requests
void handleRoot() {
  collectAnalogReadings(); // Collect EMG signal samples from analog pin

  String jsonResponse = "[";

  // Convert sampled analog data to JSON format
  for (int i = 0; i < sampleCount; i++) {
    jsonResponse += String(analogValues[i]);
    if (i < sampleCount - 1) {
      jsonResponse += ",";
```

```
    }
  }
  jsonResponse += "]";

  // Send JSON response
  server.send(200, "application/json", jsonResponse);
}

void setup() {
  // Initialize serial communication
  Serial.begin(115200);

  // Start Wi-Fi as Access Point
  WiFi.softAP(ssid, password);
  Serial.println("Wi-Fi Access Point started");
  Serial.print("IP Address: ");
  Serial.println(WiFi.softAPIP());

  // Configure HTTP server routes
  server.on("/", handleRoot);

  // Start the HTTP server
  server.begin();
  Serial.println("HTTP server started");

  // Initialize OLED display
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }

  // Set up button pin
  pinMode(BUTTON_PIN, INPUT_PULLUP);

  // Clear the display buffer
  display.clearDisplay();

  // Display the welcome screen
  //displayWelcomeScreen();
  delay(3000); // Wait for 3 seconds

  // Clear the screen for the next message
  display.clearDisplay();
  display.display();
}

void loop() {
  // // Read battery level
  // batteryLevel = analogRead(BATTERY_PIN);

  // // Check if the button is pressed
  // if (digitalRead(BUTTON_PIN) == LOW && !isRecording) {
  //   isRecording = true;

  //   server.handleClient();

  //   displayRecordingScreen();
  //   delay(3000); // Simulate recording time
  //   displayDoneScreen();
  //   delay(2000); // Wait before turning off the display
  //   display.clearDisplay();
  //   display.display();
  //   isRecording = false;
```

```
  // } else if (!isRecording) {
  //   // Display the message and battery indicator
  //   displayMessageWithBattery("Click the button to start recording");
  displayWelcomeScreen();
  delay(1000);
  }



void displayWelcomeScreen() {
  // Display welcome message
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 20);
  display.setTextSize(2);
  display.println("WiEMG");
  display.setTextSize(1);
  display.setCursor(50, 45);
  display.println("by Medicore ");

  // Update the display with the buffer
  display.display();
}

void displayMessageWithBattery(const char* message) {
  // Clear the display buffer
  display.clearDisplay();

  // Display battery level indicator
  displayBatteryIndicator();

  // Display the main message
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 30);
  display.println(message);

  // Update the display with the buffer
  display.display();
}

void displayRecordingScreen() {
  // Clear the display buffer
  display.clearDisplay();

  // Display "Recording..." message
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 30);
  display.println("Recording......");

  // Update the display with the buffer
  display.display();
}

void displayDoneScreen() {
  // Clear the display buffer
  display.clearDisplay();

  // Display "Done" message
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 30);
```

```
  display.println("Done");

  // Update the display with the buffer
  display.display();
}

void displayBatteryIndicator() {
  // Map battery level to a percentage
  int batteryPercent = map(batteryLevel, 0, 4095, 0, 100);

  // Draw the battery outline
  display.drawRect(110, 5, 16, 8, SSD1306_WHITE);
  display.drawRect(126, 7, 2, 4, SSD1306_WHITE); // Battery positive terminal

  // Fill the battery level
  int fillWidth = map(batteryPercent, 0, 100, 0, 14);
  display.fillRect(112, 7, fillWidth, 4, SSD1306_WHITE);

  // Display the percentage value (optional)
  display.setCursor(90, 5);
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.print(batteryPercent);
  display.print("%");
}
```

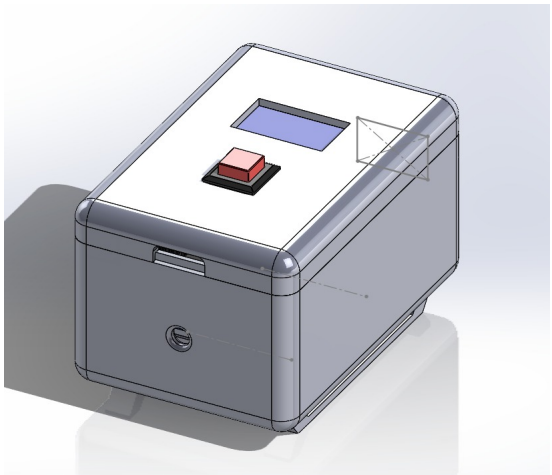
4.4.2 Python Code

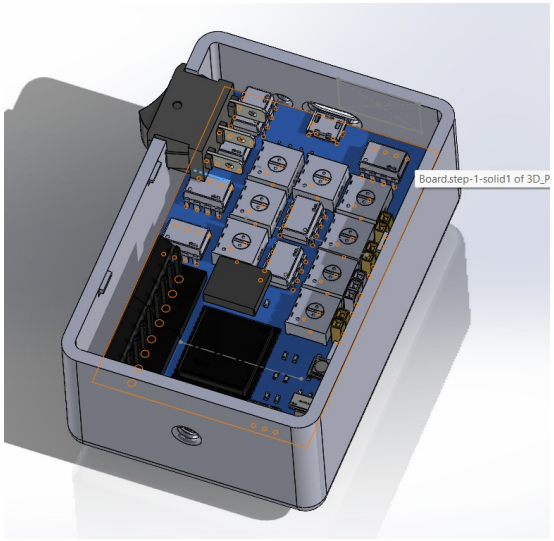
```
import requests
import matplotlib.pyplot as plt
import numpy as np
import csv
from scipy.signal import iirnotch, filtfilt
import pandas as pd

# ESP32 IP address
ESP32_IP = "http://192.168.4.1/"

# Function to fetch EMG data from ESP32
def fetch_emg_signal():
    try:
        response = requests.get(ESP32_IP)
        response.raise_for_status()  # Raise an error for bad status codes
        emg_signal_data = response.json()  # Parse JSON response
        return emg_signal_data
    except requests.RequestException as e:
        print(f"Error fetching data from ESP32: {e}")
        return []

# Save sine wave data to a CSV file
def save_to_csv(time, amplitude, filename="emg_signal_data.csv"):
    try:
        with open(filename, mode="w", newline="") as file:
            writer = csv.writer(file)
            writer.writerow(["Time (s)", "Amplitude"])
            writer.writerows(zip(time, amplitude))
        print(f"Sine wave data saved to {filename}")
    except IOError as e:
        print(f"Error saving to CSV file: {e}")

# Apply a notch filter to remove 50 Hz noise
def apply_notch_filter(signal, fs, f0=50, Q=5):
```

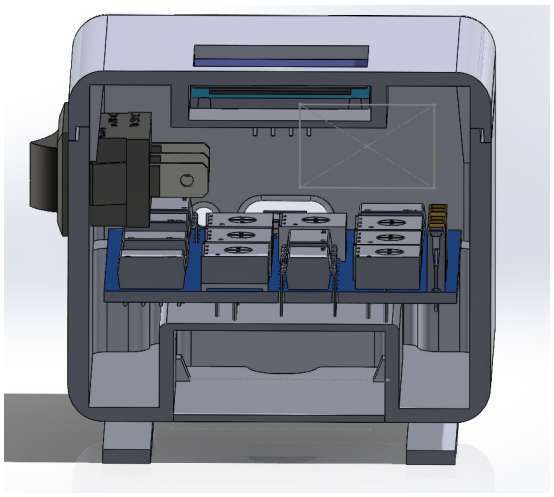
```
    b, a = iirnotch(f0 / (fs / 2), Q)
    filtered_signal = filtfilt(b, a, signal)
    return filtered_signal

# Fetch sine wave data
emg_signal = fetch_emg_signal()

# Check if data is valid
if emg_signal:
    # Generate time axis for plotting
    sample_count = len(emg_signal)
    sampling_rate = 2000  # Adjusted sampling rate in Hz (e.g., 2 kHz)
    time = np.linspace(0, sample_count / sampling_rate, sample_count)

    # Save data to a CSV file
    save_to_csv(time, emg_signal)

    # Apply a 50 Hz notch filter
    emg_signal_fil = np.array(emg_signal)
    filtered_emg_signal = apply_notch_filter(emg_signal_fil, sampling_rate)

    # Plot the original and filtered sine wave
    plt.figure(figsize=(12, 6))
    plt.plot(time[5000:5500], emg_signal[5000:5500], label='Original Sine Wave',
        ↪ color='red', alpha=0.6)
    plt.plot(time[5000:5500], filtered_emg_signal[5000:5500], label='Filtered
        ↪ Sine Wave (50 Hz Removed)', color='blue', linewidth=1)
    plt.title("Sine Wave from ESP32 (Before and After Filtering)")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.grid()
    plt.legend()
    plt.show()
else:
    print("No data received from ESP32.")
```

4.5 Enclosure

This is the Solidworks design of our Enclosure




Board.step-1-solid1 of 3D_P



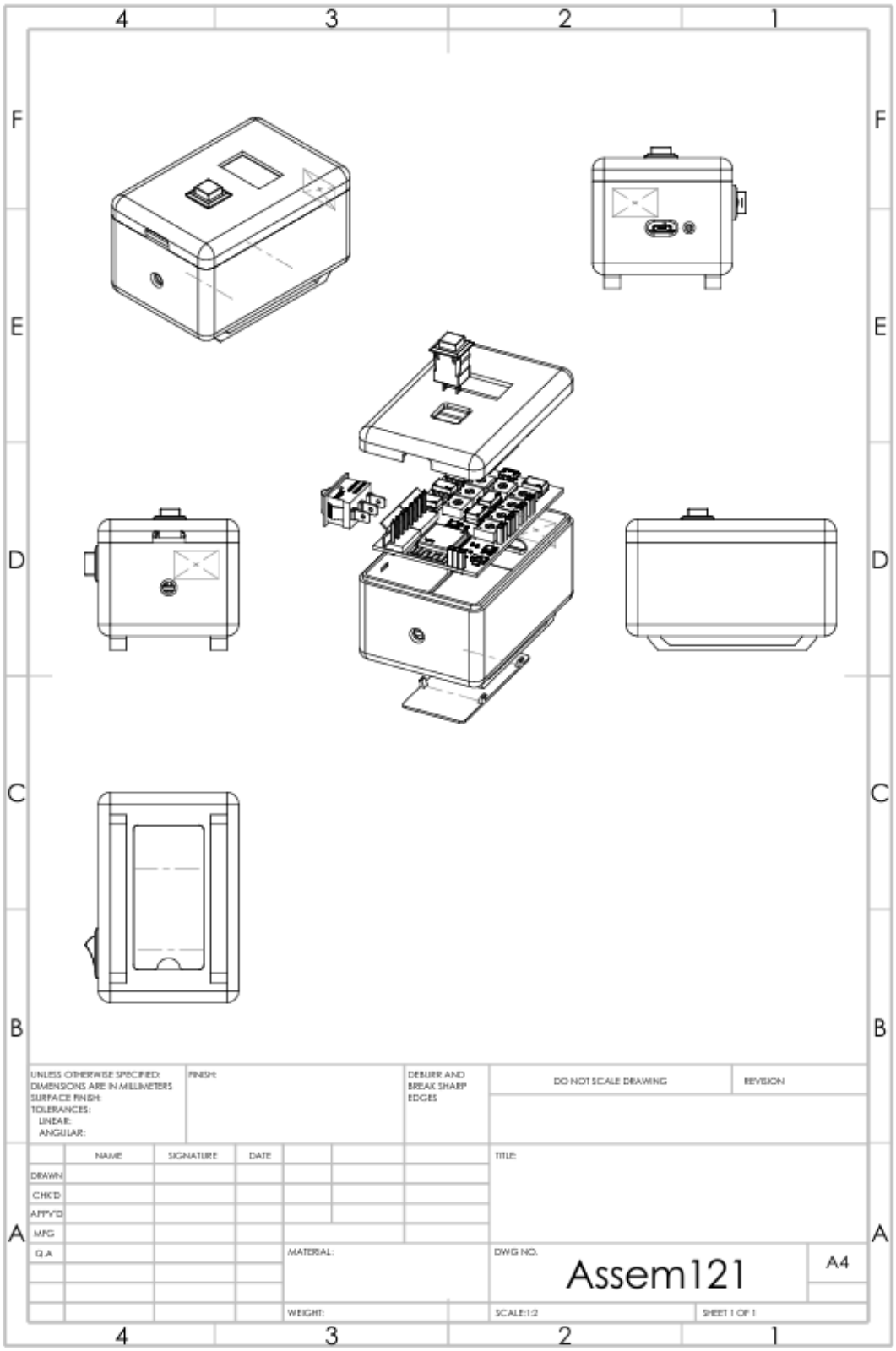


Figure 13: The Drawing File of our enclosure

5 Simulation Results

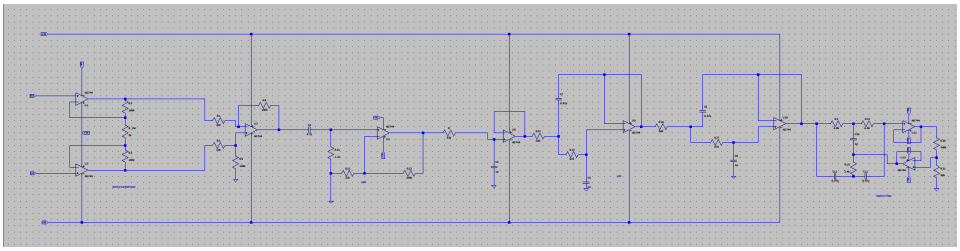


Figure 14: LT Spice simulation circuit

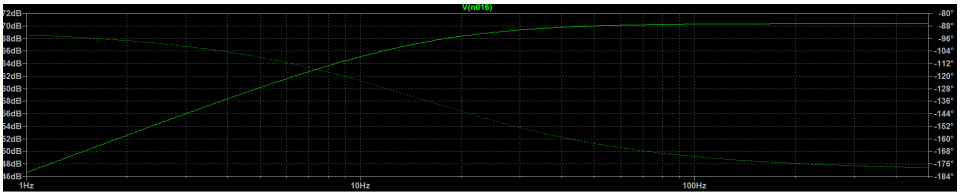


Figure 15: Frequency Response after the High Pass filter

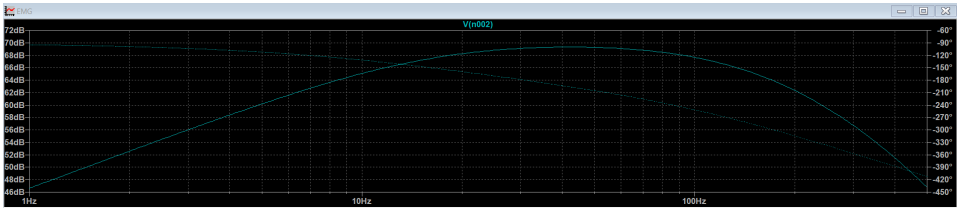


Figure 16: Frequency Response after the Low Pass filter

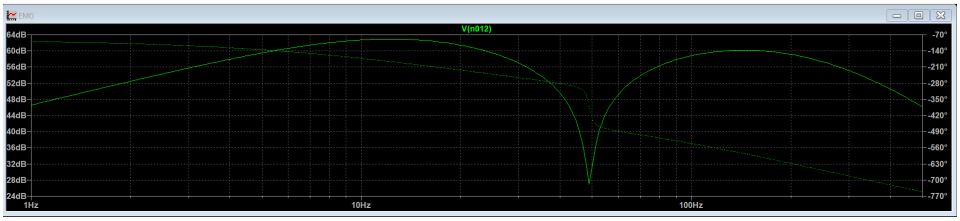


Figure 17: Frequency Response after the Notch filter

6 Test Results

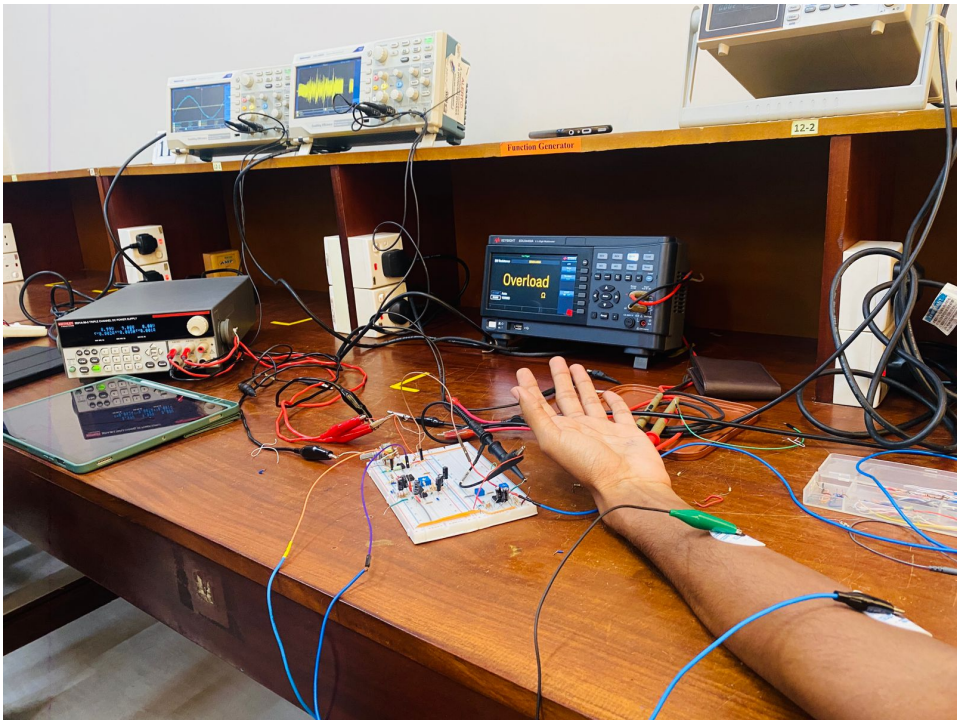


Figure 18: Bredboard Testing of the Filters

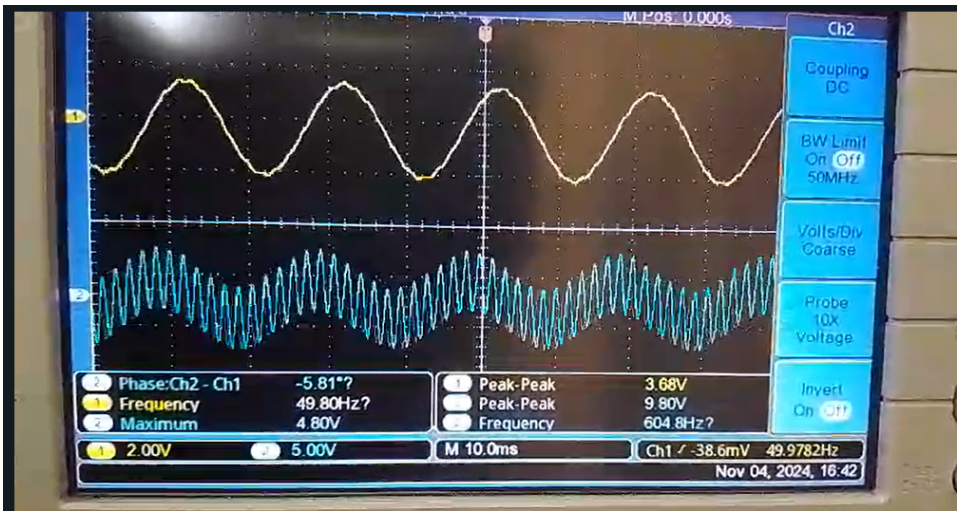


Figure 19: Clipping Off the Higher Frequency components

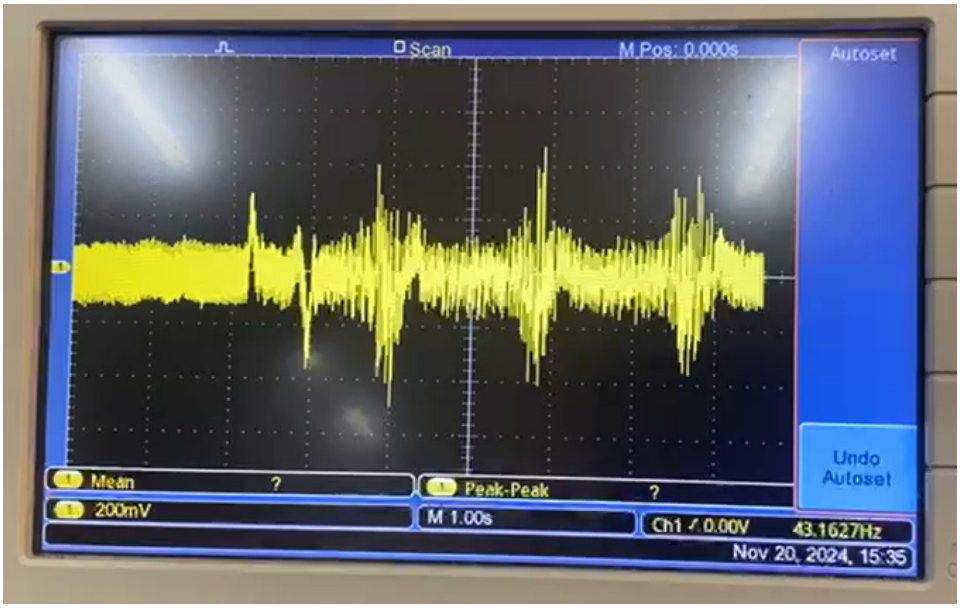


Figure 20: Filtered sEMG Signal on the Breadboard

7 Possible Regulatory Pathway

The sEMG wrist flexion monitoring device for athletes is likely to be classified under the U.S. FDA regulations based on its intended use and risk profile. Given its focus on monitoring muscle activity and providing actionable feedback, the device falls under one of the following categories:

- Electromyograph (EMG) Device (21 CFR 882.1870):
  - Class: Class II
  - Purpose: Evaluates neuromuscular function.
  - Pathway: Requires a 510(k) premarket notification, demonstrating substantial equivalence to an existing FDA-cleared device.
- Biofeedback Device (21 CFR 890.1375):
  - Class: Class II
  - Purpose: Improves physical function through biofeedback.
  - Pathway: Also requires a 510(k) premarket notification, with additional clinical data if novel claims are made.

If we marketed strictly for non-medical purposes (e.g., performance monitoring for athletes without diagnostic claims), the device may not require FDA clearance but must adhere to consumer device standards. Further analysis of the device's intended claims and features would determine the exact classification and regulatory requirements.

8 Future Improvements

8.1 ML Model Improvement for Movement Classification

Currently, the machine learning model classifies wrist flexion and extension. For future improvements, we can enhance the model to classify all types of wrist movements, including:

- Wrist Extension
- Wrist Flexion
- Radial Deviations

- Ulnar Deviations

To achieve this, consider:

- Expanding the dataset to include a broader range of wrist movements to ensure the model can learn distinct patterns for each type.
- Exploring more advanced machine learning algorithms (e.g., deep learning techniques like convolutional neural networks) that may improve accuracy, especially if the movements are subtle.
- Feature engineering: Identify additional relevant features (e.g., movement speed, angle, or time duration) to improve classification accuracy.

8.2 Using More Electrodes to Enhance the Circuit

To enhance the quality and accuracy of the readings, the use of more electrodes can provide more data points and improve the model's ability to classify different movements. For example:

- Adding additional electrodes around the wrist or forearm could capture more muscle activity patterns.
- Improving electrode placement for better signal accuracy and to reduce noise.
- Upgrading the circuit: Make adjustments to handle more electrodes and ensure reliable signal processing (e.g., improving the analog-to-digital conversion, enhancing the signal amplification).

By increasing the number of electrodes, we can improve the granularity of the data, which could lead to more accurate movement classifications. Additionally, ensuring proper circuit design and signal processing techniques will be crucial to handle the increased complexity.

8.3 Optimizing the Notch Filter for Improved Noise Removal

To enhance signal clarity and reduce noise interference, optimizing the notch filter is a promising future improvement. For example:

- Fine-tuning the notch filter's frequency response to target specific noise frequencies more effectively, such as those caused by powerline interference (50 Hz).
- Upgrading the circuit: Incorporate higher-quality components and advanced filtering techniques to minimize signal distortion and preserve the integrity of the original sEMG signal.
- Evaluating digital filtering methods as an alternative or complement to the analog notch filter to enhance overall signal processing capabilities.

8.4 Utilizing MATLAB for Enhanced Data Transmission and Machine Learning Integration

Integrating MATLAB into the workflow can streamline data handling and improve machine learning capabilities. For example:

- Employing MATLAB's robust communication tools to enable seamless data transmission between the device and the computer.
- Using MATLAB's machine learning toolboxes to develop and fine-tune advanced models for classifying wrist movements with higher accuracy.
- Leveraging MATLAB for data preprocessing, such as filtering, feature extraction, and dimensionality reduction, to enhance model performance.
- Simplifying end-to-end analysis by integrating MATLAB scripts for training, testing, and deploying machine learning models directly onto the device.

9 Individual Contributuions

Index No.	**Name**	**Contribution**
220005R	Aazir M. A. M.	Circuit design, Enclosure Design, Wireless Transmission
220562U	Samuditha H. K. P.	Circuit design, PCB Design, Testing and Debuugging of breadboard and PCB
220735E	Yashodhara.M.H.K.	Circuit design, Machine Learning, GUI Implementation using Streamlit, Testing and Debugging of breadboard and PCB