Surface Eletromyography Sensing Board

Ayush Chakraborty, Rucha Dave, and Jiayuan Liu

December 18, 2022

1 Introduction

Electromygraphy is a medical tool used to evaluate the health of individual muscles and the nerve cells that dictate their movement. The intention of this project was to design a four-channel surface electromygraphy (sEMG) printed circuit board in order to process real-time electromyographic signals generated from a user's muscle movements. The board was intended to communicate with a user's computer via Bluetooth and Serial to provide flexibility to the physical setup, along with portability for ease of access to EMG information. The full hardware and software implementation can be found here.

2 Schematic Design

When creating the schematic for our EMG board, we researched various circuit designs that fit within both our budget and the scope of the project. We discussed the benefits of each, seeking guidance from our professor, before finalizing the circuit shown in Figure 1 below.

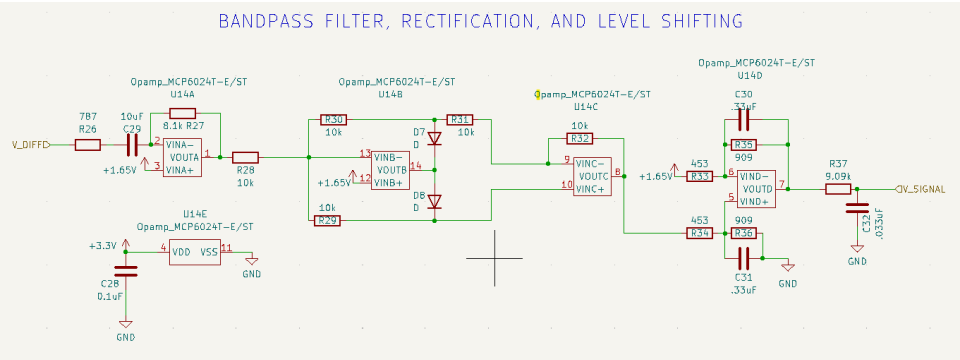

Figure 1: Circuit Diagram of Signal Processing for EMG

However, before beginning to process the signal, we need to use a differential amplifier to amplify the difference between the signals received between the two EMG nodes.

This was done by using an instrumentation amplifier chip (In-Amp), shown in Figure 2.

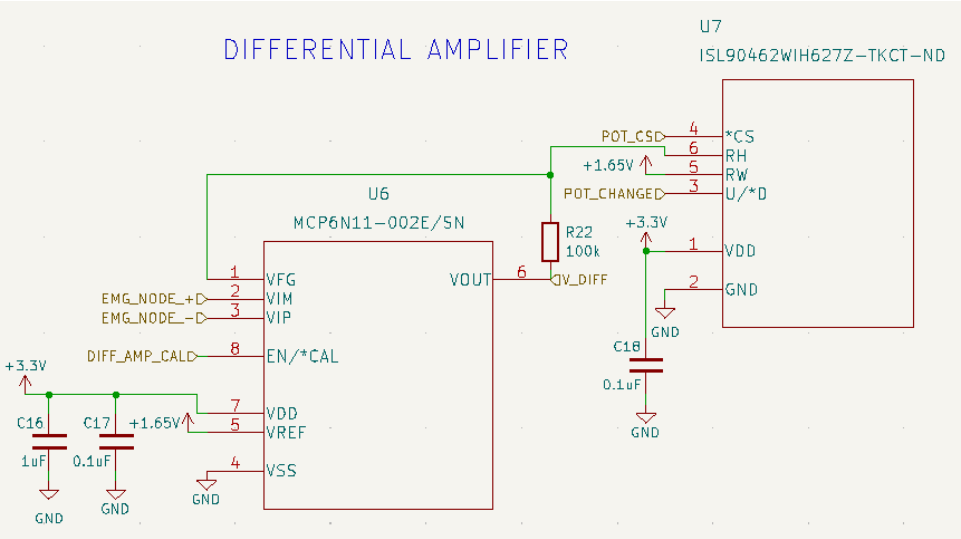

Figure 2: Circuit Diagram for Differential Amplification for EMG Signals

Starting by just considering one channel, our goal was to process the incoming signal to be normalized against a reference, filtered to remove noise, and then modified to represent a signal the computer could easily analyze. This process involves three steps: high pass filter/amplifier, rectifier, and level shifter/low pass filter/amplifier.

1. **High Pass Filter and Amplifier:** The first task after receiving a signal from the sensor was to remove noise that had a low frequency (ex: long term decay in signal). The portion of the circuit in Figure 1 furthest to the left depicts this. The cutoff frequency is given by the equation $f_c = \frac{1}{2\pi RC}$. Based on this, our circuit has a high pass cutoff frequency of 20.2 Hz. This means signals below this frequency are removed. Additionally, this portion of the filter also amplifies the signal by $\frac{8100}{787}$, or about 10 times.

2. **Rectifier:** After low frequency noise was removed from the signal, the signal was rectified to have a reference of 1.65V. This means that, making 1.65V the new "zero" point, all portions of the signal below 1.65V are made "positive". For example, 1.60V would now be 1.70V. The entire signal is now contained between 1.65V and 3.3V without having lost any data. In Figure 1, this is the central portion of the circuit.

3. **Level Shifter, Low Pass Filter, and Amplifier:** Finally, after having modified the signal, it needed to be put through a low pass filter to remove high frequency noise (background nose or outside interference) and further amplified.

Additionally, it also needed to be re-referenced to 0V to make further analysis easier. The right-most portion of Figure 1's circuit depicts this process. Here, the low pass filter has a cutoff frequency of $f_c = \frac{1}{2\pi RC}$ with R = 9.09kΩ and C = 0.033μ F. This makes the cutoff frequency 530 Hz. All frequencies above this are removed from the data. Next, the signal is shifted so that 1.65V moved to 0V. This results in the signal ranging from 0V to 1.65V. Finally, the amplifier is 2x, allowing the signal that originally encompassed only 0 to 1.65V to now double in amplitude and cover the entire range from 0 to 3.3V.

3 LTspice Simulation

In order to validate the proposed signal processing system, we decided to simulate the bandpass filter, rectifier, and level shifter in LTspice. To start, we simulated each individual component individually. When running these isolated tests, we were validating the following:

1. The high pass filter and amplifier has a gain of 10 and its characteristic frequency is around 20 Hz.
2. The rectifier properly rectifies the signal with a 1.65V reference.
3. The level shifter, amplifier, and second order low pass filter has a characteristic frequency of 530 Hz and properly shifts the signal to encompass the full 0.0V to 3.3V measurable range.

Once each individual component was validated, they were integrated together to resemble the actual signal processing module. To test the full signalling process, we inputted a wide range of signals, some out of the expected frequency range and some within that range and examined if the output was as expected. Figure 3 depicts an example input-output for a signal with a frequency of 100 Hz and amplitude of 150 mV (which is the highest expected amplitude being inputted to this module). 100 Hz is within the desired frequency and should be amplified. Figure 4 depicts an example input-output for a signal with a frequency of 0.5 Hz and amplitude of 150 mV. 0.5 Hz is outside of the desired frequency and should be filtered.

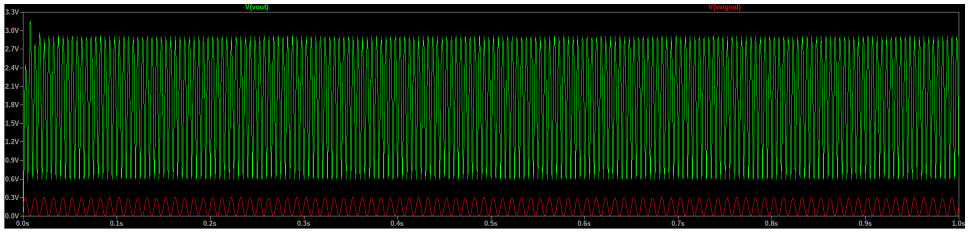

Figure 3: Simulation of signal processing of signal with frequency within the desired range (100 Hz). Green represents the output signal and red represents the input

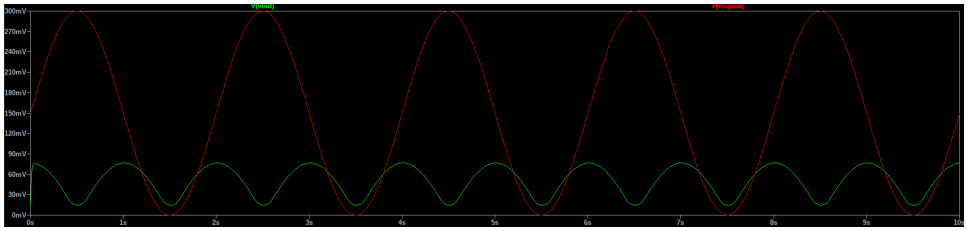

Figure 4: Simulation of signal processing of signal with frequency outside the desired range (0.5 Hz). Green represents the output signal and red represents the input

Since the simulation behaved as expected for several different inputs that account for the signals we were expecting, we were fairly confident in our schematic design, allowing us to move forward to developing the circuit board layout.

4 Additional Parts

Finally, we needed to include numerous additional parts to our board to pull it together for the final purpose. We used multiple bulk and bypass capacitors, a Bluetooth module, RP2040, RP2040 memory device, a potentiometer, etc. These all involved their own circuitry. A complete circuit is given in our GitHub here. A full list of all parts used is provided in Section 6 (Bill of Materials).

5 Layout Design

While developing the layout for our EMG sensing board, we had to design it with the following constraints in mind:

1. Minimize the overall area of the board. This allows for the EMG sensing to be as portable as possible.
2. Allow for easy access to the EMG signal input pin headers to minimize disorderly wiring that could interfere with consistent signalling.

While developing, we had two main iterations to the design: one two layer design and, based on received feedback, one four layer design.

5.1 Iteration 1 - Two Layer Board

For our first iteration, we planned a two layer board, where the top layer was for power and signals, while the bottom layer was for ground and signal. To start, each footprint was organized into separate component groups, which were power management, each of the four signal processing modules, and the micro controller. Then, they were placed relative to each other, adhering to any specifications on each data sheet (ex: bypass

capacitors within 10mm of their corresponding pad, RP2040's memory device placed as close as possible to the RP2040). This was done without traces to easily shift components, while trying to minimize space to adhere to design constraint 1.

Once organized, each individual component group was placed relative to each other. We chose to place the EMG signal input pin headers on the very bottom edge of the board, all next to each other, for easy access. Then, each individual signal processing group was placed near the corresponding pin headers. The power management module was placed right of these signals, allowing for the shortest path between power and the signalling devices that require power. Finally, the micro controller and other corresponding parts were placed in above the signal processing modules, allowing their output to be routed directly to the RP2040. Then, the traces between each component were routed, starting with the 3.3V power source, then the 1.65V power source, and finally each of the signal lines. This led to a rather messy layout, as can be seen in Figure 5.

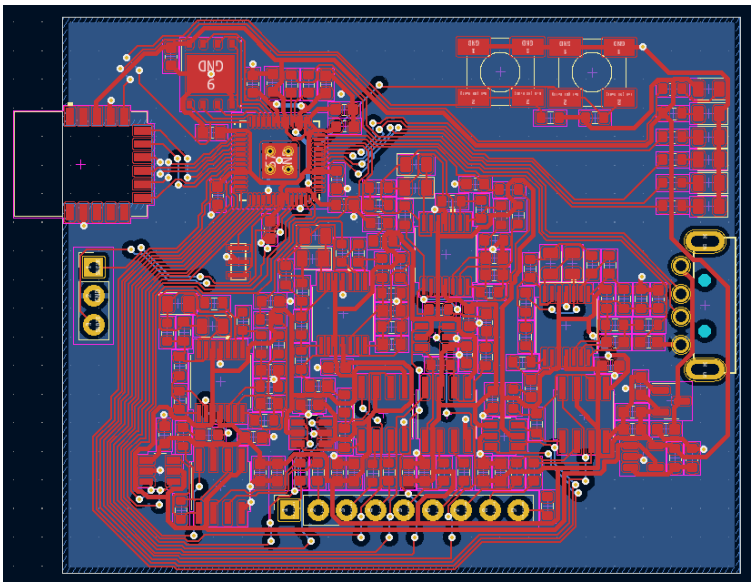

Figure 5: First Iteration of EMG Board Layout.

The primary concern was that due to the close proximity of the numerous digital signals being routed from the RP2040 to the digital potentiometers and operational amplifiers, there would be significant cross-talk between the lines, causing logical errors when using the board. This become the basis for the next iteration. Another concern was the rather messy power distribution. Since we had to routed +1.65V and +3.3V throughout the signal processing modules, there were significant crosses between the two lines, causing concern about providing consistent power to each integrated circuit component.

5.2 Iteration 2 - Four Layer Board

For the second layout design, we chose to implement a four year board, where the board stack up from top to bottom was as follows: power and signal, ground and signal, power, ground and signal. The reason for the third layer containing a power layer was so that the +3.3V could be provided solely through this layer, while only +1.65V would be routed through the top layer, leading to a simpler power distribution.

Additionally, the availability of two additional layers meant that the digital signals from the RP2040 could be more evenly distributed and routed away from each other to preserve signal integrity. In terms of component placement, the placement in the two-layer design was only slightly modified to move components closer together since the top layer required less room for traces. This led to a smaller overall board, better adhering to the first design constraint. The updated layout can be seen in Figure 6.

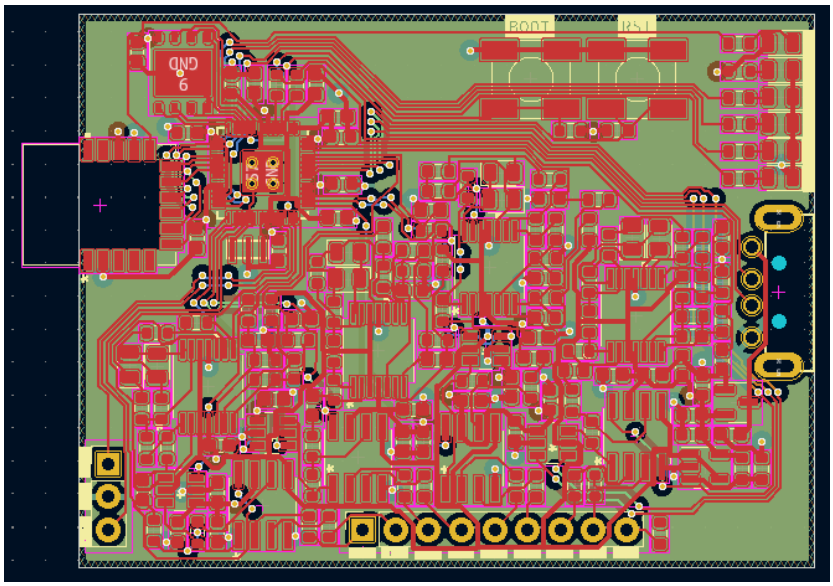

Figure 6: Second Iteration of EMG Board Layout.

While there were significantly more vias in this design, the presence of extra layers allowed for interruptions of the ground plane to be evenly distributed, allowing us to maintain two relatively continuous ground planes. This served as our final layout design for this revision of the board.

6 Bill of Materials

In order to physically create the board, the following components were used (per board):

Component Name	Quantity
0.1μF Capacitor	22
0.33μF Capacitor	8
1μF Capacitor	7
10μF Capacitor	6
0.033μF Capacitor	4
10kΩ Resistor	22
100kΩ Resistor	13
453Ω Resistor	8
909Ω Resistor	8
1kΩ Resistor	6
787Ω Resistor	4
8.1kΩ Resistor	4
9.09kΩ Resistor	4
5.1kΩ Resistor	2
27Ω Resistor	2
Schottky Diode	8
LED	6
Differential Amplifier (MCP6N11)	4
Digital Potentiometer (ISL90462WIH627Z-TKCT-ND)	4
Quad Operational Amplifier (MCP6024)	4
Serial Flash Memory (W25Q128JV)	1
RP2040	1
BLE Module (RN4871)	1
Linear Regulator (MCP1702T)	1
Single Operational Amplifier (MCP6021)	1.555
Ceramic Resonator	1
Buttons	2
USB A Male Connector	1
01x09 Male Pin Header	1
01x03 Male Pin header	1

Table 1: Bill of Materials

The detailed part sheet, include part references and specific part models, can be found here.

7 Circuit Board Population

We made several attempts before successfully assembled a fully functional PCB board. The biggest challenge in the assembly process came from soldering the RP2040 and the Serial Flash Memory device as they were both QFN packages. For both of the first two boards we populated, we were unable to establish a connection between the RP2040 and a laptop. While debugging the issue, we noticed that the power indicator was flickering

rather than maintaining a constant brightness, leading us to believe that there was an issue with our power distribution on the board. Upon further investigation, we found that the voltage output of the linear regulator that stepped the USB 5V input down to 3.3V was inconsistent, oscillating between 0V and 2.3V. It was also reaching significantly high temperatures. Using a thermal camera, we found that there was another elevated thermal area on the RP2040, suggesting that there was a short between two pads. This problem was resolved by reflowing the board and using a soldering iron to remove any new shorts between pads. This allowed us to establish a connection between the RP2040 and a laptop.

Another issue we encountered was, whenever we flashed code on to the RP2040, it immediately reset and became ready for another flash rather than storing the firmware image. This led us to believe that there was an issue with the connection between the RP2040 and the serial flash memory device. While there were no noticeable shorts between the two devices, we decided to re-heat the connections between the two. After doing so and flashing another image, the device was successfully able to store it, resolving the issue.

The second difficulty came from debugging one of the EMG signal channels which outputs a constant high voltage even without any active signal input. To troubleshoot, we simulated voltage at each pin along that trace with LTspice and compared the simulated voltage to the actual voltage measured with an oscilloscope. Eventually, this problem was resolved by replacing the quad op-amp used in the signal processing module for that channel.

8 Firmware Overview

For this circuit, the firmware, running on the RP2040, was responsible to analyzing and communicating the current reading of each EMG channel. For the first iteration of the firmware architecture, we had two main requirements for the first iteration.

1. Continuously read the input of 4 analog signals (processed EMG signals).
2. Send the read values of each EMG channel, along with a marker to denote which channel it came from, via the USB connection.

To program the RP2040, we chose to use the arduino-pico library, which is a hardware abstraction library for the RP2040 that allows it to be programmed using the Arduino IDE. The firmware was split into two main functions, setup() and loop().

The setup() function begins by initializing 4 indicator LEDs to HIGH. This is to make sure any firmware image flashed to the RP2040 is properly running on the micrcontroller. Then, the USB serial connection is begun as a baud rate of 9600 bits per second.

Code Segment 1

```
int serial_baud_rate = 9600;
int LED_PIN_ONE = 8;
int LED_PIN_TWO = 9;
int LED_PIN_THREE = 10;
int LED_PIN_FOUR = 11;

void setup() {
  pinMode(14, OUTPUT);
  digitalWrite(14, HIGH);
  pinMode(15, OUTPUT);
  digitalWrite(15, HIGH);
  pinMode(18, OUTPUT);
  digitalWrite(18, HIGH);
  pinMode(23, OUTPUT);
  digitalWrite(23, HIGH);
  pinMode(LED_PIN_ONE, OUTPUT);
  digitalWrite(LED_PIN_ONE, HIGH);
  pinMode(LED_PIN_TWO, OUTPUT);
  digitalWrite(LED_PIN_TWO, HIGH);
  pinMode(LED_PIN_THREE, OUTPUT);
  digitalWrite(LED_PIN_THREE, HIGH);
  pinMode(LED_PIN_FOUR, OUTPUT);
  digitalWrite(LED_PIN_FOUR, HIGH);
  Serial.begin(serial_baud_rate);
}
```

Next, the loop() function, which is continuously called, serves the purpose of reading each analog signal from the 4 EMG channels and sending it over serial. The addition of x0000 to each signal is to serve as the marker for which channel the signal came from. This is for the software to differentiate between signals. This means that each message takes is composed of the left most digit indicating the channel number and the next four digits indicated the 10 bit reading from the analog pin (0-1024). Once sent, the serial interface is flushed to ensure that the messages are sent.

Code Segment 2

```
void loop() {
  Serial.println(analogRead(A0) + 10000);
  Serial.println(analogRead(A1) + 20000);
  Serial.println(analogRead(A2) + 30000);
  Serial.println(analogRead(A3) + 40000);
  Serial.flush();
  delay(.5);
}
```

9 Software Overview

The first iteration of the software, which would interface with the EMG sensing board via serial, was split into two main functions: initialize_serial() and process_serial_input().

The initialize_serial() was designed to establish the serial connection with the hardware. This is done using python's serial library, where we specified a USB port, serial baud rate, and timeout in order to establish the connection.

Code Segment 3

```
def initialize_serial():
    arduino_port = "/dev/ttyACM0"
    baud_rate = 9600
    serial_port = serial.Serial(arduino_port, baud_rate, timeout=1)
    return serial_port
```

Once established, the process_serial_input() is called in order to continuously process data on the serial line and dynamically updated a graph with the received data. Each received line is decoded, then the first digit is split off to determine which EMG channel the data should be appended to. Then, the analog reading is extracted and appended to a list of received values for the channel. Additionally, we record the time that each message was received.. The received time was used instead of the time that the message was processed on the microcontroller in order to minimize the amount of data being sent via the serial line, allowing for a higher rate of EMG values. The data processing is nested in a try-except statement due to errors that arise from bitloss in serial communication.

Code Segment 4

```
while True:
    try:
        curr_data = int(serial_port.readline().decode())
        data[curr_data//10000][1].append(curr_data%10000)
        data[curr_data//10000][0].append(time.perf_counter()-
            start_time)
    except (UnicodeDecodeError, ValueError, KeyError):
        continue
```

In order to plot the data, a graph with four subplots, one for each channel, is initialized. Then, each sub-graph is assigned a line to store its data. On every iteration of the while loop, those lines are updated based on the received data. Finally, the x axis is set to visualize the most recent ten seconds of data.

Code Segment 3

```
plt.ion()
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax4 = fig.add_subplot(224)

line1, = ax1.plot(data[1][0], data[1][1], 'k-')
line2, = ax2.plot(data[2][0], data[2][1], 'g-')
line3, = ax3.plot(data[3][0], data[3][1], 'b-')
line4, = ax4.plot(data[4][0], data[4][1], 'r-')

while True:
    # Process serial input
    line1.set_xdata(data[1][0])
    line1.set_ydata(data[1][1])
    line2.set_xdata(data[2][0])
    line2.set_ydata(data[2][1])
    line3.set_xdata(data[3][0])
    line3.set_ydata(data[3][1])
    line4.set_xdata(data[4][0])
    line4.set_ydata(data[4][1])
```

```
    ax1.set_xbound(lower=data[1][0][-1]-10, upper=data[1][0][-1]+10)
    ax2.set_xbound(lower=data[1][0][-1]-10, upper=data[1][0][-1]+10)
    ax3.set_xbound(lower=data[1][0][-1]-10, upper=data[1][0][-1]+10)
    ax4.set_xbound(lower=data[1][0][-1]-10, upper=data[1][0][-1]+10)

    fig.canvas.draw()
    fig.canvas.flush_events()
```

An example plot is shown in Figure 7.

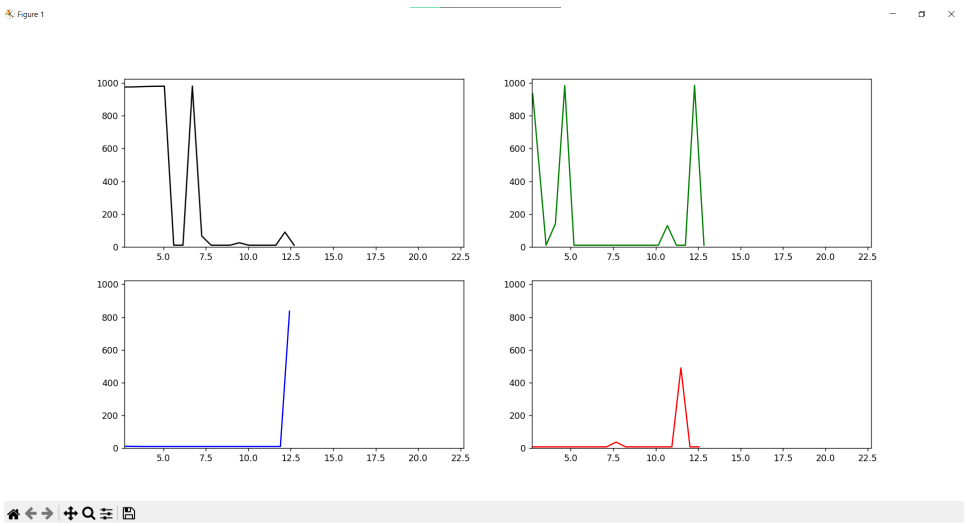

Figure 7: Sample visualization generated by a user's laptop while running an EMG trial.

10 Results

With the current hardware, we have received inconsistent results in processing and analyzing EMG data from the sensing board. At times, we are able to properly receive data. This is demonstrated in our one-channel demo. However, there are issues with running all four channels working synchronously. During some tests, all four channels can be properly monitored for a brief period of time, but, in most tests, only one or two channels yield expected results while the others constantly remain undetectable. The full demo of the current state of the system can be found here.

10.1 Limitations

Possible causes of the inconsistent functioning of all channel include:

1. Misplacement of the EMG electrodes on the muscle. If the electrodes are placed improperly, they may pick up the same signal, causing their difference to be always be 0, leading to no signal.

2. Inadequate contact between the EMG electrode and the muscle. If proper contact isn't established with the muscle, then the signal will be weaker, or not detectable at all, leading to improper results.

3. Delay in serial communication. We found that, at times, there is a delay between a physical movement and the visualization of that response. We believe this is either because of how the serial communication is implemented or a physical limit to how fast serial communication can be established.

These problems might have resulted in the board not being able to accurately detect every muscle movement on all four channels.

11 Reflection

11.1 Ayush

This project was an excellent learning experience for me. It was my first hardware project that wasn't firmly scaffolded by external design constraints, allowing me to explore new avenues in electrical engineering. This included learning a lot about setting my own design constraints based on a specific function I want to implement. The main challenge I faced during this project was learning how to debug an embedded system. Numerous times throughout the process, I wasn't sure if there was a hardware or software issue causing unintended results, which forced me to learn new debugging tools to isolate the problem to one or the other and act accordingly.

Additionally, this project taught me the value of implementing features to de-risk certain aspects of the project in order to maximize the chance of meeting an intended deliverable. For example, we wanted to establish a bluetooth connection between the sensing board and a computer, but due to time constraints, were not able to implement this. Thankfully, we had de-risked the bluetooth communication by also including a serial interface to receive data, allowing us to still extract and visualize the processed EMG signal.

11.2 Rucha

Coming into this class, I had a very minimal understanding of how circuit components that we had learned about in ISIM could be brought together to accomplish a goal. Through this final project, I was really able to delve deep into different filters and amplifier designs to understand the basics of how a signal could be processed to reach the final goal. This project not only helped me strengthen my conceptual understanding of

the different circuit portions change a signal, but also gave me a chance to mathematically step my way through the circuit and analyze it. I really enjoyed this project and believe I learned a lot from the process.

11.3 Jiayuan

I learned a lot from the technical aspect of this project, especially in designing signal processing circuit with operational amplifiers, board population, and board debugging. The main challenge I met in the circuit designing stage was analyzing the different filters we researched. Some of the confusions I had in the schematics designing stage were cleared later while debugging the board with an oscilloscope, as I physically measure the voltage at each pin of the filters. Overall, I found this project challenging and engaging.

12 Next Steps

To address the problems we had with inconsistent signal measurement, our next step is to try using one electrode as the reference while placing the other electrode on the center of a muscle, instead of placing a electrodes pair on one muscle. To develop more meaningful features of our board, we can first improve our software for faster data reading and add on algorithm for muscle movement classification. This project can potentially become a interactive game for user to exercise different part muscles!