An EEG-Based Brain-Computer-Interface System

A brain-computer interface system for controlling a simple computer program.

By: Mind-Controller team

AJ Casapulla

Stephen Eaton

Oliver Gambrell

Jayson Herrera

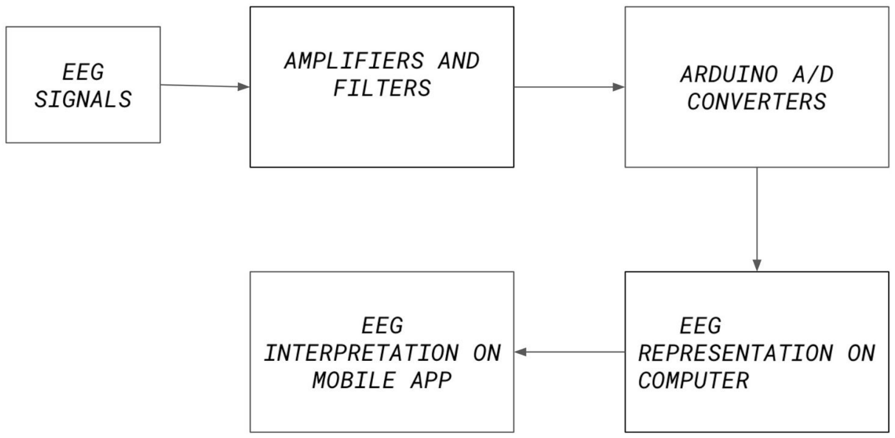
Executive Summary:

The Mind-Controller team will develop a brain-computer interface device that enables the user to control buttons and sliders in a computer program using only their thoughts. Specifically, we will (1) develop an EEG interface to read live brain signals, (2) program a microcontroller to read the signals from the EEG interface, (3) program a machine learning algorithm on the microcontroller to classify the signals, and (4) develop a computer program that uses the classified signals to control buttons and sliders. In the fall 2019 semester we developed a two-probe EEG device that can reproducibly measure blink and heart signals, and have successfully programmed an Arduino due to read live EEG signals and display them on a computer.

Problem Statement

In a more ideal world, information could be accessed more quickly and with less effort to improve problem solving speed. Concerning computers, this could come in the form of thinking of acquiring a particular piece of information or making a calculation and having a computer return the information or calculation by responding to your thought alone. This would be a major time improvement in information acquisition speed and could lead to many problems being solved in less time. To this end, our team has started on what we believe to be a first step towards this far-reaching goal by developing an EEG-based signal classifier for the purpose of controlling buttons and sliders in a computer program. To get down to the specifics, our Mind-Controller team will develop a device that will read differential signals from the brain, amplify them to usable levels, and translate them into commands for a simple program. The signals will be obtained using purchased EEG probes connected to a differential amplifier. This amplifier will filter out unneeded higher frequencies and noise and bring the microvolt level brain waves up to something usable by a microcontroller's analog to digital converter. Once digitized by the microprocessor, the signals will be run through a machine learning algorithm to translate them into usable commands. Finally, the commands will control a simple demonstration program. Presented below is a block diagram representing the project structure.

<u>Block Diagram:</u>


EEG SIGNALS
AMPLIFIERS AND FILTERS
ARDUINO A/D CONVERTERS
EEG REPRESENTATION ON COMPUTER
EEG INTERPRETATION ON MOBILE APP

<u>Customer:</u>

Brain to machine interfaces are of large interest to many areas, ranging from use as game controllers all the way to medical devices. Depending on the complexity of the actions that the Mind-Controller can interpret, it will be marketable to hobbyists, game companies and gamers, hospitals, disabled people, and companies that wish to improve the productivity of their workers, among others.

<u>Team & Responsibilities:</u>

The team consists of four members: AJ Casapulla, Stephen Eaton, Oliver Gambrell, and Jayson Herrera. AJ's primary focus will be on developing the circuitry to amplify the low level brain waves into a usable form. Stephen will mainly develop the code for the microprocessor and demonstration program. Oliver will handle processing the signals to translate them into usable data. Jayson will be contributing to both the hardware and software development, and will create the training data for the neural network.

<u>Objectives & goals:</u>

The overall goal of this project is to develop a device that can reliably interpret a user's intent via brain waves and translate that to an actionable response. Similar devices have been created that use the same theory, such as the NeuroSky MindWave and Uncle Milton Star Wars Force Trainer. Previous Senior Design students have also attempted similar projects in the past, with limited success. We will consider this project a success if we are able to show that our device is reliably reading brain waves which it can then consistently use to perform any assigned action in our demonstration program. If our device is unable to read brain waves, unable to translate them into actionable commands, or inconsistent, we will consider that a failure.

Constraints:

The main constraints of this project are expected to be time and money. After researching EEG probes, the more easily usable dry contact type are fairly expensive. Also, brief research on instrumentation amplifiers used in other DIY EEG type devices can cost over ten dollars. This isn't expected to pose a problem for the initial build, however if we manage to achieve good enough results to add more channels, it will come into play. A possibility to reduce noise levels is to have a custom PCB made for our amplifier circuit. This will take time to make and have shipped to our team for assembly. A possible hidden constraint comes in the form of the assumption that controlling a computer program with EEG to our desired degree of sensitivity is in fact possible. If it is in fact the case that our desired sensitivity is unachievable, our project will be scaled accordingly to accommodate for the limitations. Further work will be proposed at the end of the project to suggest improvements that could be made on the circuit, choice of microcontroller, coding, neural network, program, and alternative methods and modes of signal collection.

Fall Goals:

The team aims to have a prototype amplifier and filter developed and built by the end of the Fall 2019 semester. This will consist of two EEG probes and a single differential amplifier, giving us one channel to verify the design of our amplification and filtering circuitry. The hope is that this will also provide us with the data that we need to begin training our neural network. During this time, we will also be developing a better understanding of the different types of brain waves and how to best measure them. As of the end of this semester, we have met the proposed goals and have begun working on our spring goals by working on the program software and are looking at selecting the appropriate machine learning algorithm to classify the EEG signals.

Spring Goals:

The spring 2020 semester will consist of refining circuitry to obtain better signals and, if finances allow, to increase the number of probes and channels, enabling us to obtain more data. We will continue to train our neural network and will develop our demonstration program. By the end of this semester we expect, at the very least, to be able to demonstrate simple on/off commands.

Approach:

The main idea, as presented in the image below, consists of 4 parts: an EEG cap, an amplifier/filter setup, a microcontroller, and a computer.

More research and simulation is required for the amplifier and filtering circuitry, however the basic idea is that it will use a differential instrumentation amplifier IC to take the low level signals from the probes and amplify them. A series of operational amplifier ICs will then be used for filtering and further amplification if needed. The filtered and amplified signal will be fed into our microprocessors analog to digital converter and sampled at an appropriate rate. The microprocessor will send this data to a computer for processing.

The computer setup consists of two parts: the demonstration software and the neural network. The demonstration software will be written in python, and will look something like the image below. It will consist of 6 actions: ON/OFF, UP, DOWN, LEFT, RIGHT, and a SLIDER function. After training the neural network on Jayson, Jayson will think about turning on or off each button. The arduino will read the signal, the neural network will classify the signal, and the software will convert the signal into an action on one of the buttons. The slider will be handled by having Jayson thinking about moving the slider up and down.

A neural network will be chosen that effectively learns time series data. To train the network, we will have Jayson look at the prompt and think about the actions associated with the buttons. The EEG cap will read these signals, run them through the microprocessor, and put them into the computer. After many repeated sessions of this we will have a trained network that will be used for testing.

Detailed Project Timeline & Schedule:

The projected timeline of our project for this semester is as follows: By the end of October, we would like to have assembled a basic 2-electrode EEG cap and have successfully implemented the amplifier and filter components for basic measurements. By the end of November, we would like to have written the code for the arduino that will read the signals and display them on a computer. By the end of the semester, December, we would like to be able to have consistent readings from the electrodes.

The projected timeline of our project for the following semester is as follows: At the end of February, we would like to have implemented a reliable algorithm that denoises the signal from our 2-electrode cap and have purchased additional electrodes. By the end of March, we would like to have written the demonstration program and have implemented an algorithm that reads the signals and can do very rough classification. By the end of April, we would like to have chosen a neural network and have trained it on Jayson. By the end of the semester, May, we would like to have the completed project that directly reads brain waves from Jayson and converts them directly into actions in the demonstration program.

Challenges:

The main challenges of this project will be: getting consistent readings from the electrodes, filtering the noise from the electrodes, deciding which machine learning algorithm to use to classify the signals, developing an algorithm for signal detection, and understanding the nature of EEG electrode placement and the relationship between thought and signal patterns. After reviewing other devices that are similar to ours, it has become clear that noise will be a large issue with the initial design of our amplifier. Previous teams were unable to overcome this issue and online resources have indicated that a large amount of filtering will need to take place. Using data from these previous teams should help to give a better starting place so that this can be overcome.

There's also a potential that there are clear signals coming in, but our device is unable to translate it into usable commands due to inconsistencies or a lack of patterns. In that event, the issue could potentially be resolved with more data channels, different machine learning algorithms, or modifying where we place the probes.
Our team is unfamiliar with EEG caps and electrodes, but we have experience working with electrical simulation software and circuitry building and testing so we will be able to work on this with a good prior understanding of how to evaluate our results. Skills that we will need to develop include: biofeedback hardware troubleshooting, signal filtering algorithm design, model design methods for neural networks, programming a microcontroller-software interface, and advanced coding libraries.

Design:

The basic idea for the design is shown in the below image (**Fig. D**) and consists of four subsystems: (1) the EEG probes, (2) the amplifier/filter, (3) the microprocessor, and (4) the computer program.

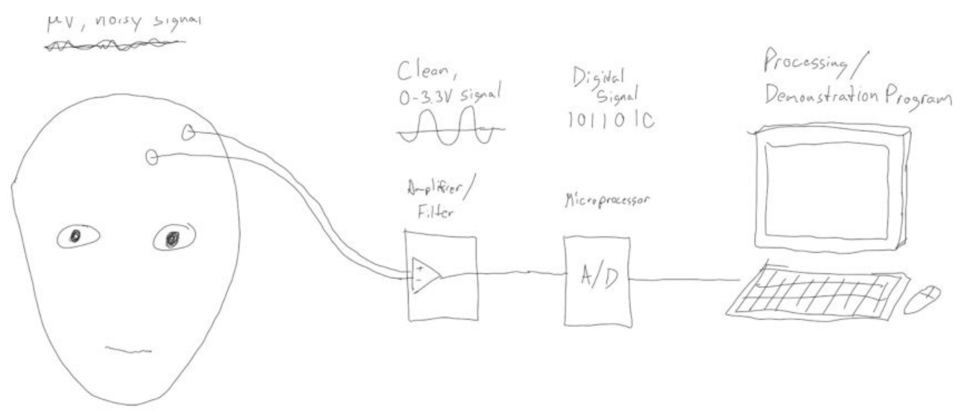


Fig. D: Four subsystems of the project from left to right: (1) the EEG probes, (2) the amplifier/filter, (3) the microprocessor, and (4) the computer program.

The flow of information is as follows: Electrical activity associated with brain processes is produced, the EEG probes pick up the signal, the amplifier/filter amplifies the signal to usable levels and filters out the 60Hz noise, the microprocessor receives the modified signal and inputs the signal through a neural network, when the neural network can classify and detect the desired signal it sends a command into the program that manipulates buttons and/or a slider.

For the hardware, we used an AD8429 instrument amplifier along with 2 TL084 quad operational amplifiers and some passive electronic components to amplify and filter our input signals. In addition to the hardware for the circuit we used an arduino due for the microcontroller. The software used for the project was python and C++. Python was used to code the arduino-computer interface and C++ was used to code the circuit-arduino interface.

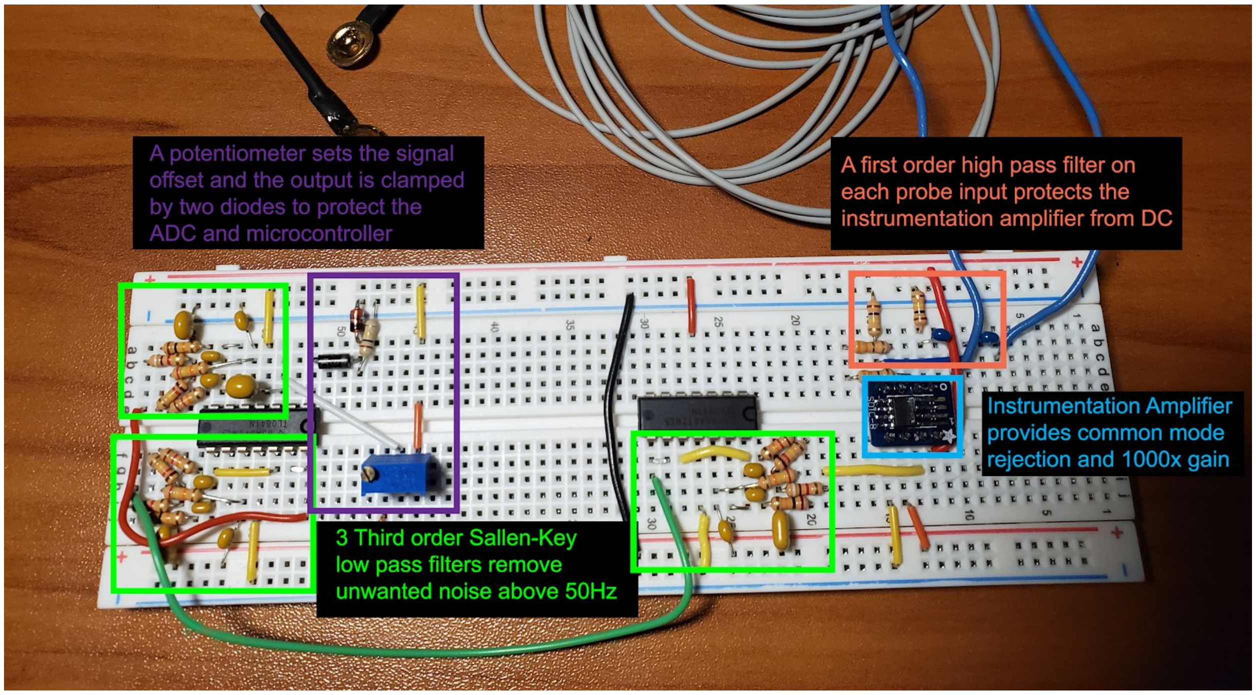
Safety Issues:

Exposure to electrical currents is one of the safety issues when working with an EEG. To avoid harming the person that the EEG is connected there are a handful of things to keep in mind. There should always be a grounding electrode, unless other electrical equipment is attached to the patient (to avoid double-grounding). The equipment should also use a three-prong plug and avoid the use of extension cords.
Source: https://www.ncbi.nlm.nih.gov/books/NBK390355/

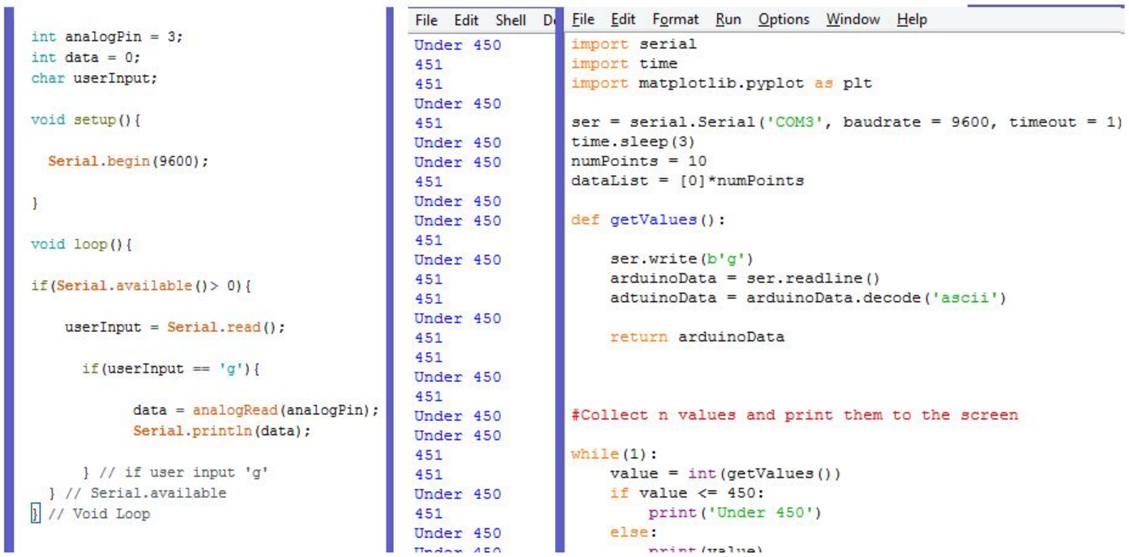
Fall 2019 Results

There were two main results for this semester's work: (1) we constructed an EEG device that reproducibly measures electrical signals from blink and heart activity, and (2) we developed python code that enabled an arduino due to read and display live signal data from the EEG device into the computer.

For the amplifier, while we initially attempted to filter out unwanted noise using a series of low pass, high pass, and notch filters, we found that we could get a reasonable signal using only low pass filtering. Our instrument amplifier requires high pass filtering that removes unwanted DC, so nothing further was required. For the high pass filters, we used a ninth order Sallen-Key filter. This filtering allows us to make out muscle signals and heartbeats however, it's not yet good enough to make out brain signals. 60Hz noise shows through, especially when there isn't a good probe connection. We have begun looking into adding a driven right leg connection, which will attempt to cancel out the noise by outputting an opposite signal onto the subjects body. We also believe that our signals will be improved by making a PCB to house our circuit, rather than a breadboard.


A potentiometer sets the signal offset and the output is clamped by two diodes to protect the ADC and microcontroller
A first order high pass filter on each probe input protects the instrumentation amplifier from DC
Instrumentation Amplifier provides common mode rejection and 1000x gain
3 Third order Sallen-Key low pass filters remove unwanted noise above 50Hz

All code for the arduino was written in python and C++. C++ was used for the arduino interface, and python was used to display and make calculations of values. Six variations of code were developed: code that (1) read the EEG signal as raw bytes, (2) read a single numerical value from the EEG signal, (3) read 10 bytes of EEG signal, (4) calculated the mean from 10 values of the EEG signal (5) continuously read and filtered out values of the EEG signal below a certain threshold (**Fig. O1** below), and (6) read and displayed a live EEG signal on the screen (**Fig. O2** below). The most important of these variations were number five and six. Number six was important because allowed us to visualize the data and verify that we were displaying signals that we could reproduce from repeated measurements. Number five was important because it will form a large basis of our work for the coming spring 2020 semester. Specifically, we will be writing machine learning code that uses this live stream of numerical values as input data and will convert it to commands that will be used to control the program.

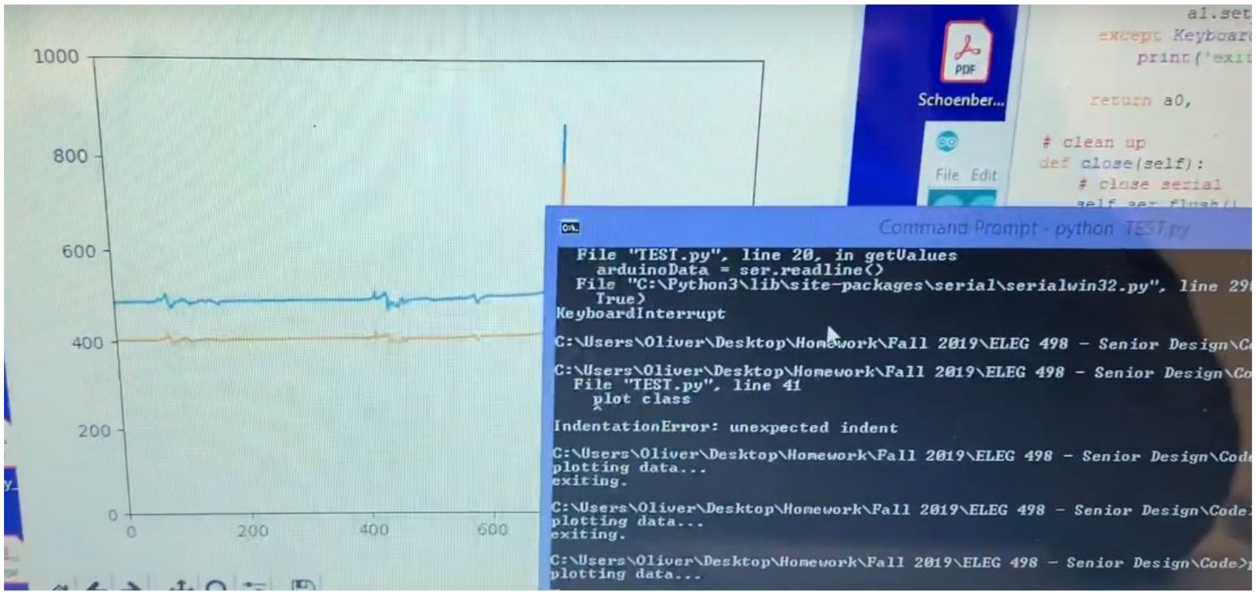


```
int analogPin = 3;
int data = 0;
char userInput;

void setup(){

  Serial.begin(9600);

}

void loop(){

if(Serial.available()> 0){

    userInput = Serial.read();

      if(userInput == 'g'){

            data = analogRead(analogPin);
            Serial.println(data);

      } // if user input 'g'
  } // Serial.available
} // Void Loop
```

```
Under 450
451
451
Under 450
451
Under 450
Under 450
451
Under 450
Under 450
451
Under 450
451
451
Under 450
451
451
Under 450
451
Under 450
Under 450
451
451
Under 450
451
Under 450
```

```
import serial
import time
import matplotlib.pyplot as plt

ser = serial.Serial('COM3', baudrate = 9600, timeout = 1)
time.sleep(3)
numPoints = 10
dataList = [0]*numPoints

def getValues():

    ser.write(b'g')
    arduinoData = ser.readline()
    adtuinoData = arduinoData.decode('ascii')

    return arduinoData


#Collect n values and print them to the screen

while(1):
    value = int(getValues())
    if value <= 450:
        print('Under 450')
    else:
```

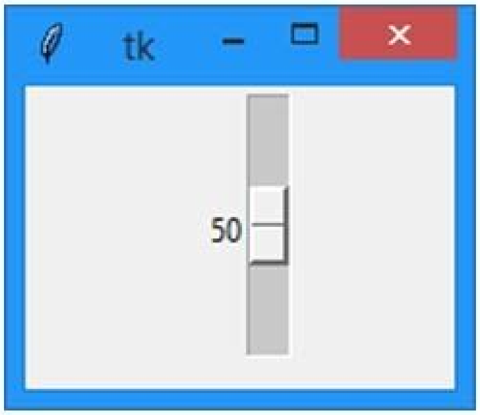
Fig. O1: Live data being read into the arduino and displayed in the middle part of the image. All data that is read below 450 is displayed as “Under 450” while values above that are displayed as is.

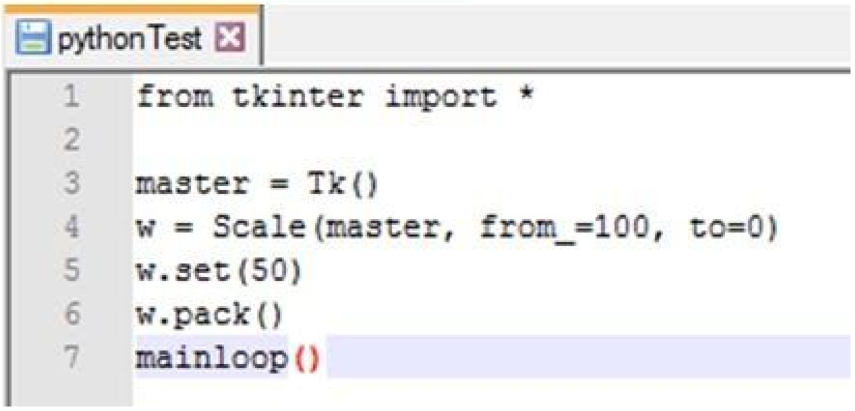


```
  File "TEST.py", line 20, in getValues
    arduinoData = ser.readline()
  File "C:\Python3\lib\site-packages\serial\serialwin32.py", line 29
    True)
KeyboardInterrupt

C:\Users\Oliver\Desktop\Homework\Fall 2019\ELEG 498 - Senior Design\C

C:\Users\Oliver\Desktop\Homework\Fall 2019\ELEG 498 - Senior Design\Co
  File "TEST.py", line 41
    plot class
    ^
IndentationError: unexpected indent

C:\Users\Oliver\Desktop\Homework\Fall 2019\ELEG 498 - Senior Design\Code
plotting data...
exiting.

C:\Users\Oliver\Desktop\Homework\Fall 2019\ELEG 498 - Senior Design\Code
plotting data...
exiting.

C:\Users\Oliver\Desktop\Homework\Fall 2019\ELEG 498 - Senior Design\Code>
plotting data...
```

Fig. O2: The EEG signal is being read live and displayed on the screen.

Concerning the computer program, we have designed and tested a simple slider in python. The slider is currently only responsive to mouse input, and in the coming semester we will modify the code so that it will be able to use the EEG readings as a way to control the slider. We also plan to add buttons where the EEG readings will be used to click the buttons. (Fig: S1).





```
from tkinter import *

master = Tk()
w = Scale(master, from_=100, to=0)
w.set(50)
w.pack()
mainloop()
```

Fig. S1: This is the code for the slider and the result from the code running. The code makes a simple slider that ranges between 0 and 100.

10-20 EEG Electrode Placement

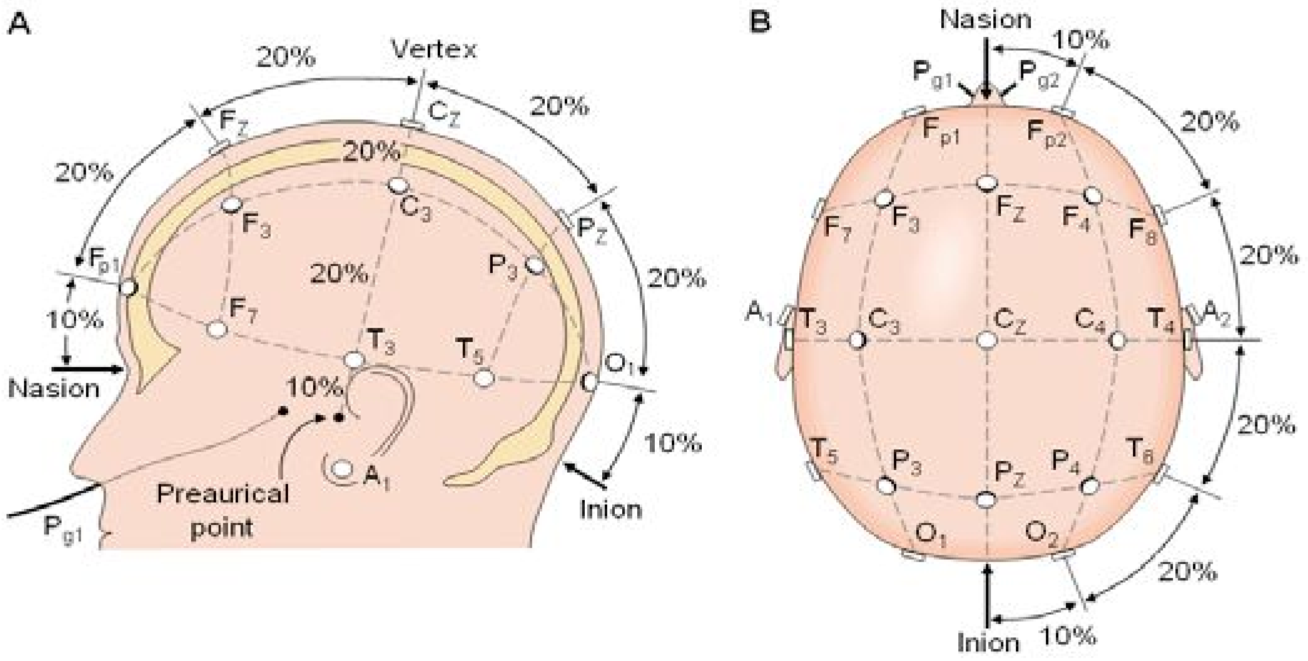
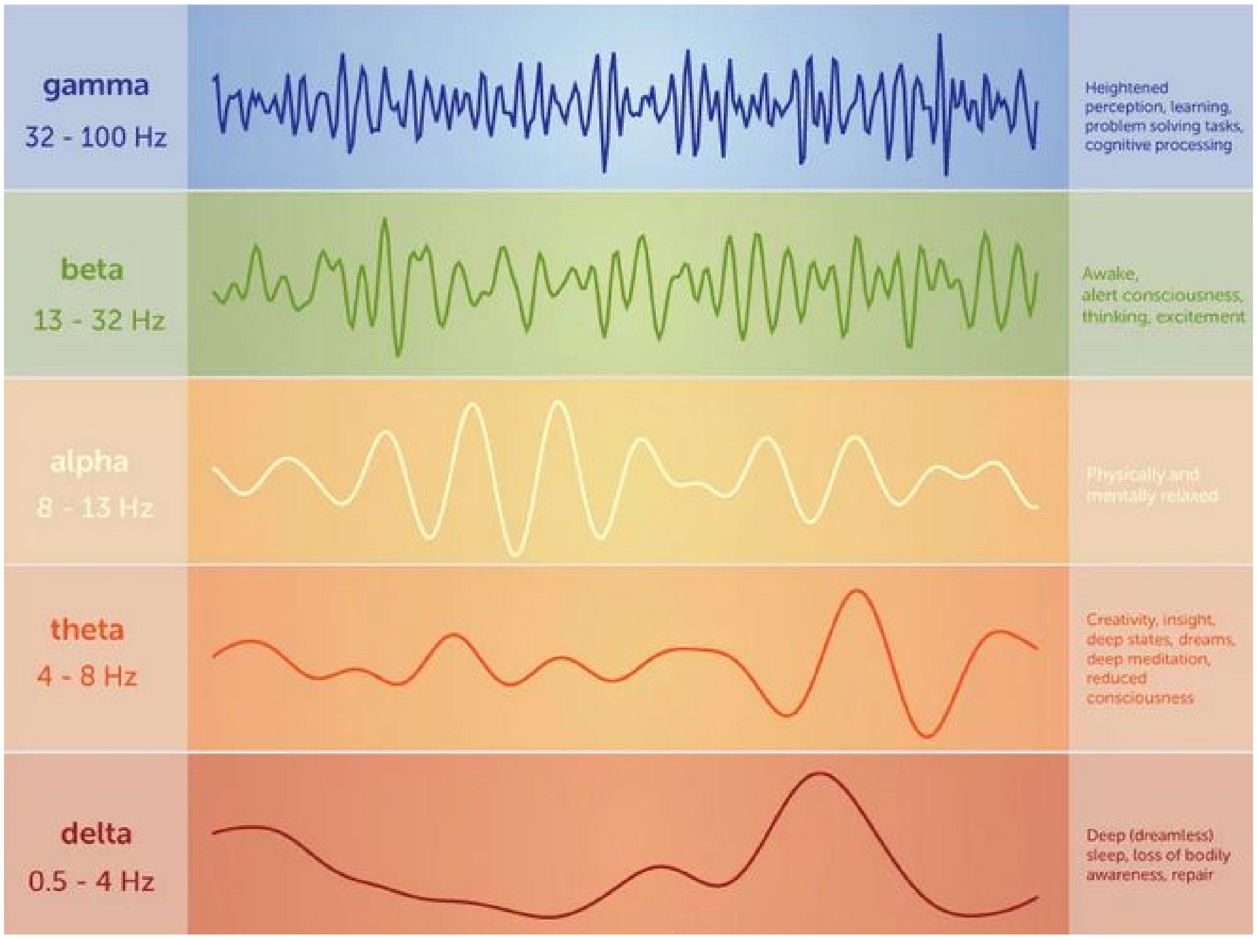


Fig. Electrode placement: The image above shows the proper placement of electrodes according to the 10-20 method.
Source: https://www.ers-education.org/lrmedia/2016/pdf/298830.pdf

We must use proper electrode placement to obtain clear and compelling brain signals. Consequently, these signals will be filtered, amplified, and interpret to perform a desired action. The 10-20 Electrode placement is a standardized method used internationally. Also, it ensures that electrodes are equally spaced, and the arrangement is proportional to the person's head. Furthermore, this method will cover all regions of the brain, Frontal, Temporal, Central, Parietal, and Occipital. Because of our limited budget, we will not be able to cover all areas of the brain. Therefore, we will only focus on 2-3 areas, the frontal, central, and occipital regions. These regions will allow us to analyze the different brainwaves' activities. There are five different types of human brainwaves, Gamma, which ranges from 32-100Hz, Beta 13-32Hz, Alpha 8-13Hz, Theta 4-8Hz, Delta 0.5-4Hz. For our sole purpose, we will mainly focus on Beta, and Alpha waves, which are the most commonly produced. Beta waves are associated with normal waking consciousness, and Alpha waves are associated with deep relaxation.


gamma
32 - 100 Hz
Heightened perception, learning, problem solving tasks, cognitive processing
beta
13 - 32 Hz
Awake, alert consciousness, thinking, excitement
alpha
8 - 13 Hz
Physically and mentally relaxed
theta
4 - 8 Hz
Creativity, insight, deep states, dreams, deep meditation, reduced consciousness
delta
0.5 - 4 Hz
Deep (dreamless) sleep, loss of bodily awareness, repair

<u>Fig. Wave frequencies</u>
<u>Source</u>:
https://choosemuse.com/blog/a-deep-dive-into-brainwaves-brainwave-frequencies-explained-2/

Although we were not able to record any brain activity this semester, we were able to produce heart and muscle signals with two probes. Ultimately, we will have a working EEG cap with approximately 6-8 probes.

Standards

There are no IEEE standards on how to build and operate an EEG. However, there are medical guidelines that we should follow for reliability. Some of the things we need to consider are equipment, electrodes, and recordings.

Equipment

In order to acquire useful readings, we need to cover as many parts of the head as possible and simultaneously record the data. Furthermore, to obtain data where the most usual and unusual EEG patterns occur, it is recommended to use a minimum of 16 channels. However, we will not be able to comply with this as we have a restricted budget. Regardless, we should be able to extract useful data with six to eight channels. This will just require extensive testing and data analysis.

Electrodes

It is essential to ensure that electrodes do not have any damages, and do not attenuate signals between 0.5 and 50 Hz. They suggest using silver or gold disk electrodes with a good conductive gel. Furthermore, EEG recording equipment should not exceed impedances above 10 k Ohms and should not be under 100 Ohms as it can be an indication of a short circuit. With this in mind, electrode impedances should be checked during recordings.

Recordings

Proper calibrations should be made at the beginning of every EEG recording. The recordings should account for eye movement, periods for when the eyes are open, and closed need to be recorded to examine the effects on the data. Finally, a low-frequency filter setting above 1 Hz to attenuate slow-wave artifacts should not be used. This can cause the loss of data in the Delta range. Similarly, high-frequency filters setting should not be lower than 70 Hz as it can cause spikes in the data.

Spring 2020 Results

A few changes were made to the structure of the project: (1.) We discovered an IC that was designed for EEGs and decided to switch to using that. (2.) The interface was changed from working with live data to working with pre-recorded data. Specifically, pre-recorded data was fed into a neural network which identified the signal type and passed a command into the program. (3.) The slider code was changed to remove the colored box, and a button was added that displays the words either “on” or “off”. The slider and button was also changed from using a mouse to change the slider, to using a keyboard input, and then it was made to select a random number that would determine what command was executed. (4.)We used Solidworks to design an enclosure to house the Arduino Due and PCB. Later we were able to use AJ's printer to produce a 3D printed model.

<u>**Network-Program Interface**</u> : (Oliver)

The interface went through two iterations during the Spring 2020 semester: (1.) A version that used Bollinger Bands to classify live data, and (2.) a version that used neural networks to classify pre-recorded data.The Bollinger interface is made up of five components, as depicted in **fig. OF1** below.

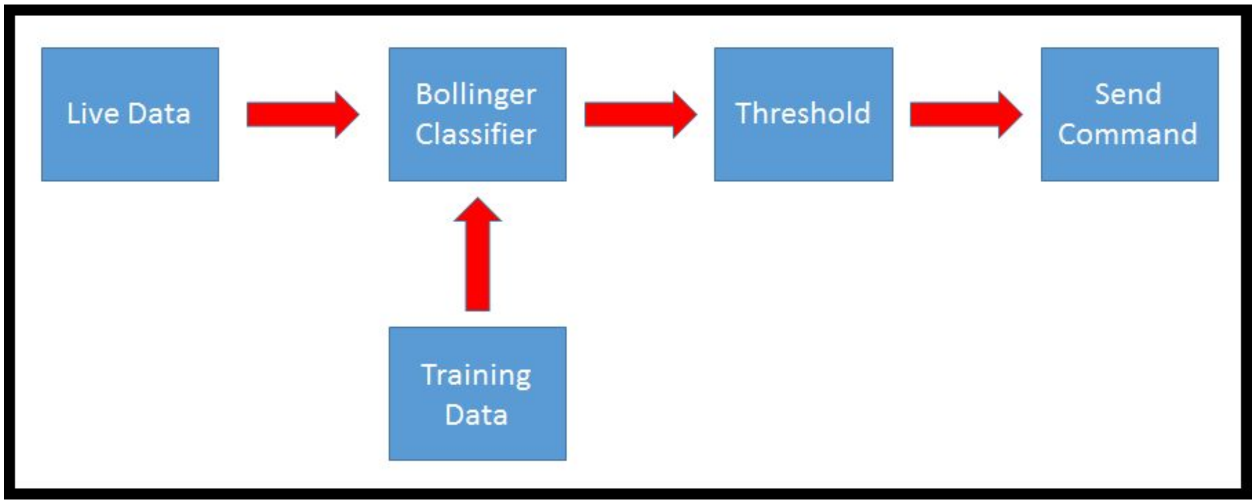


Fig. OF1 - Bollinger interface components.

The data flow is as follows: Live data was to be read into the Bollinger Classifier, which was then thresholded and converted to a command that the program could use. The Bollinger Classifier is modeled after the concept of Bollinger Bands, which takes a single-channel time series and returns two related time-series signals called the “upper” and “lower” bands. The upper and lower bands are the signals that are two standard deviations above and below the moving average of the data, respectively. All data is of the form eight x N, where eight is the number of channels and N is the predetermined length of the signal. Since the program had three commands (“Button”, “Up”, and “Down”), we had to learn 8 x 3 = 24 Bollinger bands, one for each of the eight channels in each command (see **Fig. OF2**).

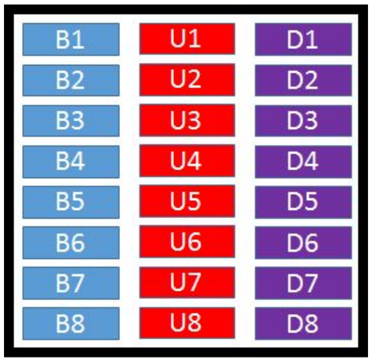


Fig. OF2 - Signal graphic.

A deep neural network would have been chosen to learn the bands based on the pre-recorded training data, but due to the limitations of the semester, the structure of the interface had to be changed. Although the aforementioned interface was abandoned, it will be presented here for full discussion of the work done over the semester.

Presented in **Fig. OF3** is an example of what the interface would look like.

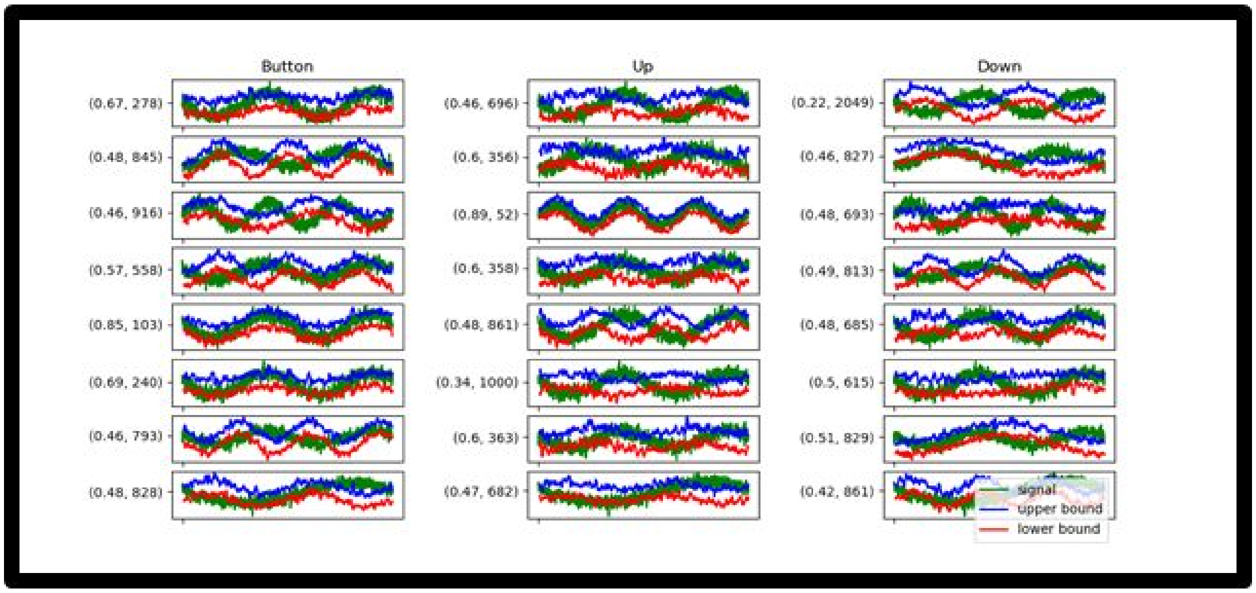


Fig. OF3 - Bollinger interface tests.

The n-th row represents the n-th channel while the column represents the command. Blue lines represent the upper bounds while the red lines represent the lower bounds for each channel-command pair's Bollinger Band. The green lines represent a sample signal while the 2-tuple to the left of each Bollinger Band represents the score match of the signal against the Bollinger Band (The score match is part of the "Threshold" component in **Fig. OF1).** The score was to be calculated in two steps: (1.) Count the number of times the test signal is within the Bollinger Bands and divide by then length of the signal, and (2.) sum the total amount by which the signal is outside of the bands by (a.) subtracting the upper band value from the signal if the signal is greater than the upper band at that point, or (b.) subtract the signal value from the lower band value if the signal value is less than the lower band value at that point. The next step in the plan was to use a secondary neural network to learn the space of appropriate match scores. The inputs to this network would have been the collection of match score 2-tuple and the output would have been a command to the program. Due to the limitations of the semester, we were unable to collect live data so the interface had to be restructured.

The second iteration of the interface is presented in **Fig. OF4.**

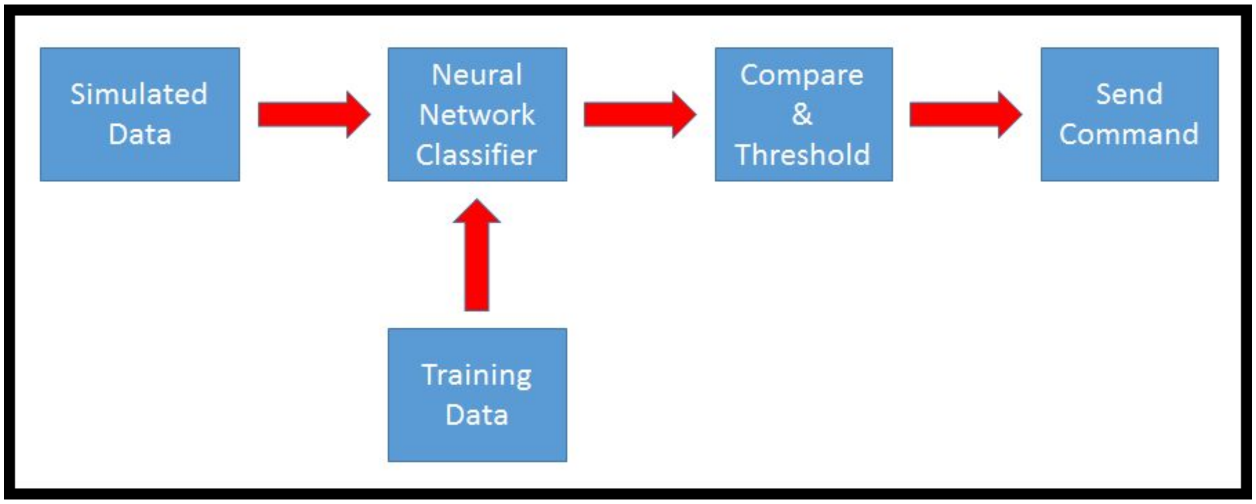


Fig. OF4 - Interface second iteration diagram.

The data flow of the second interface is as follows: Prerecorded data is fed into the Neural Network Classifier, which is compared and thresholded, and then converted to an output for the program. Since we were not able to collect data, it was instead simulated and the resulting data was used in training and testing of the network.

After visual observation of EEG data, sample data was simulated in a three step process: (1.) Generate a number of spikes, (2.) generate a number mountains, (3.) Generate random noise, (4.) generate trend lines, and (4.) add the spikes, mountains, and trend lines to the random noise to generate data samples. Spikes were generated as a sum of exponentials of length 1000 units and placed in specific locations depending on which of the 24 bands was chosen. (**Fig. OF5)** Mountain data was generated by sums of three to seven exponentials, representing associated non-spike neural data present in EEG signals (**Fig. OF6**). Trend lines, noise data, and the final sum data are presented in **Fig. OF7** to represent the final data put into the network.

To train the network, the value of N (the predetermined length of the signal) was set to 40,000 and data was reshaped to the form (1, 8 * N). From here, data and labels were generated and the network was trained. The architecture of the network was a deep neural network with seven layers with nodes of form 2^(8-k) where k is the kth layer of the network for k = 1, 2, …, 7. After training, the network correctly classified signals around 66% of the time. This result can be contributed to two things: (1.) The way the data was created, and (2.) the choice of network. The data simulation was created from visual inspection of EEG data without looking to see if there has been other work on generating more reliable EEG data. The locations that the spikes were placed were chosen by splitting up a length 40000 array into 8 random segments and populating the segments with a number of spikes depending on which location was chosen. This level of stability of repeated signals may not exist in real neural data and may require alternative means of handling. A network that may have better suited the data include convolutional or LSTM models, as they may be more efficient for dealing with time series data of this origin.

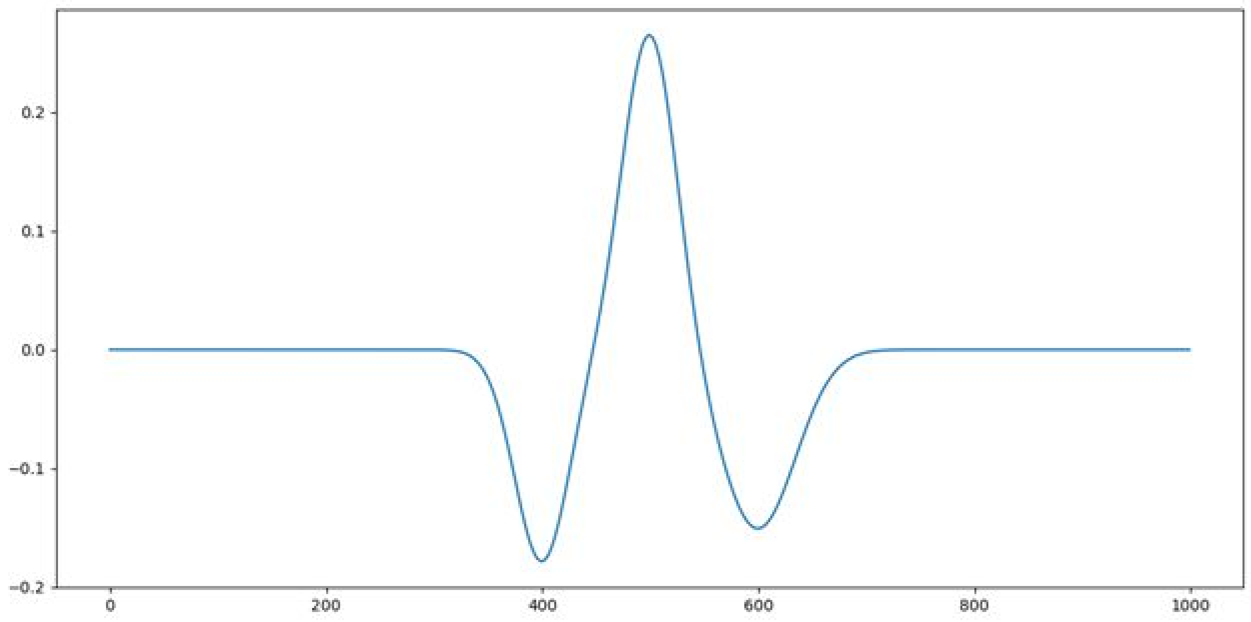


Fig. OF5 - Spike data sample visual.

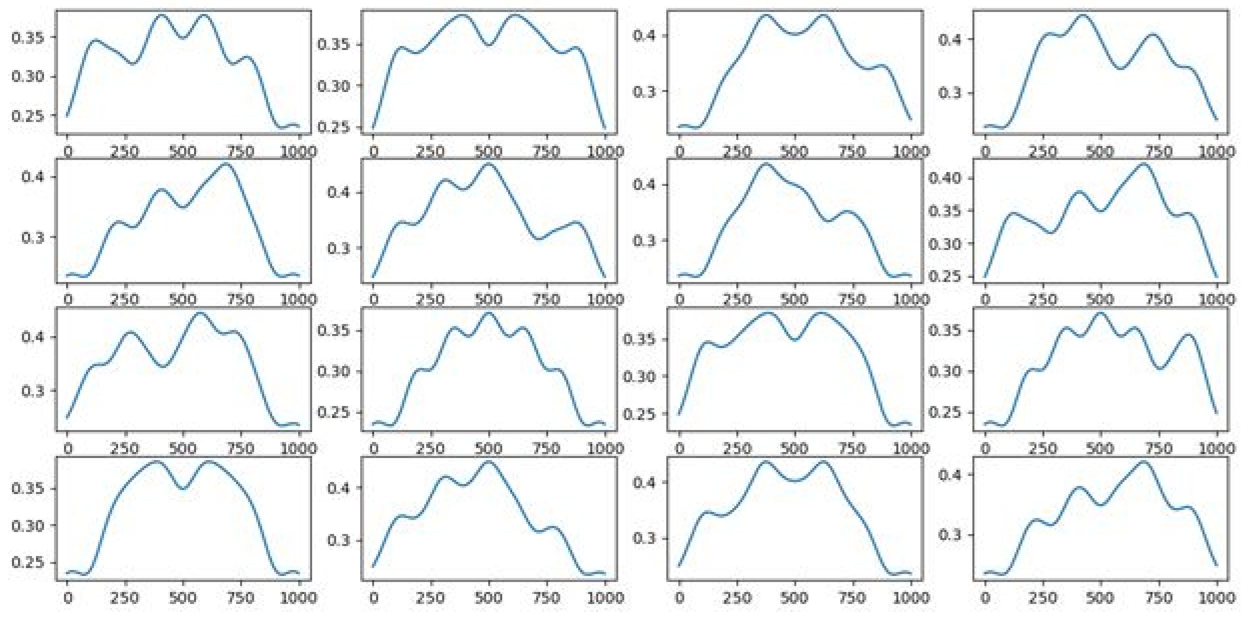


Fig. OF6 - Mountain data sample visual.

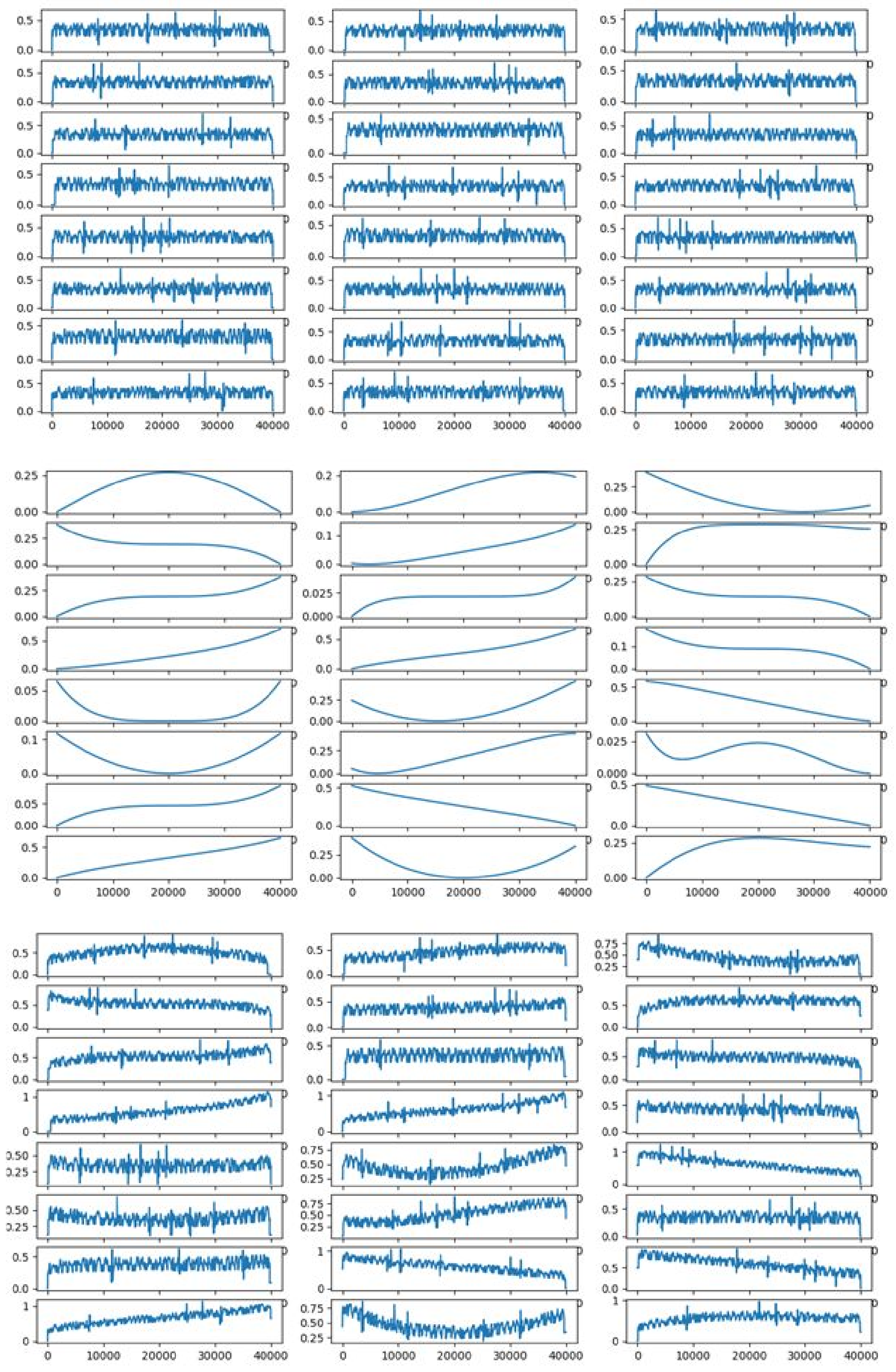
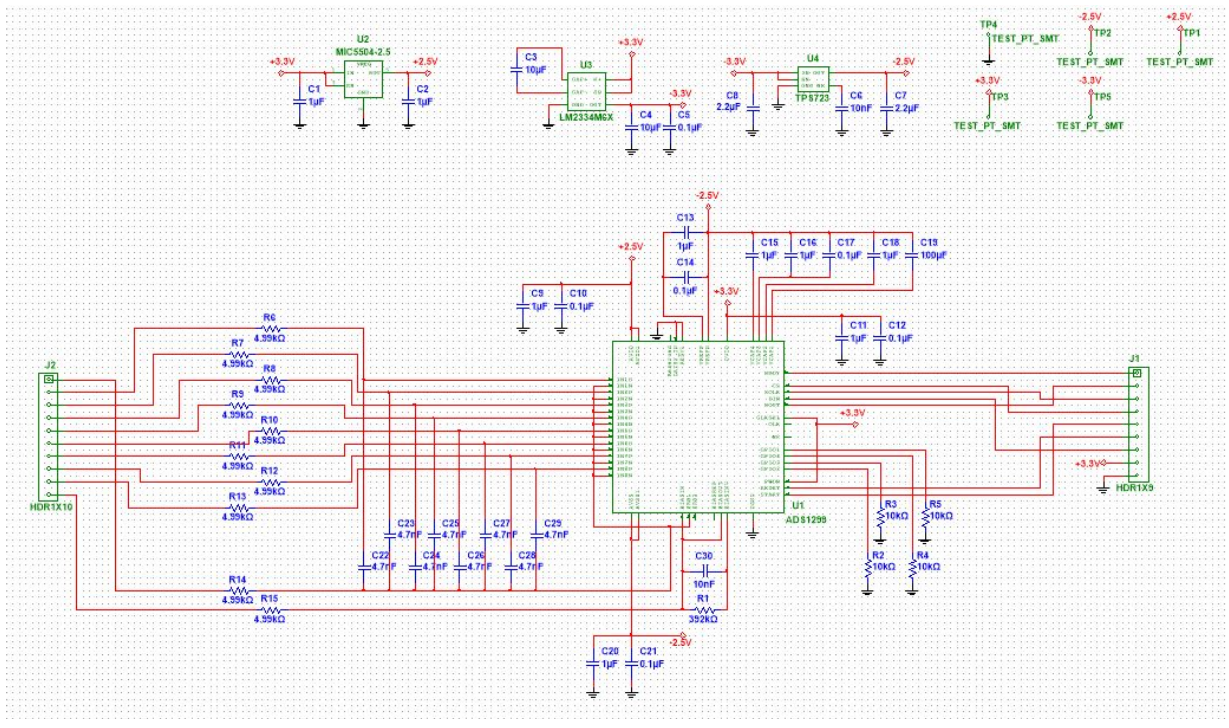


Fig. OF7 - Noise, Trend, and Final Signal Data

<u>Hardware & Firmware</u> : (AJ)

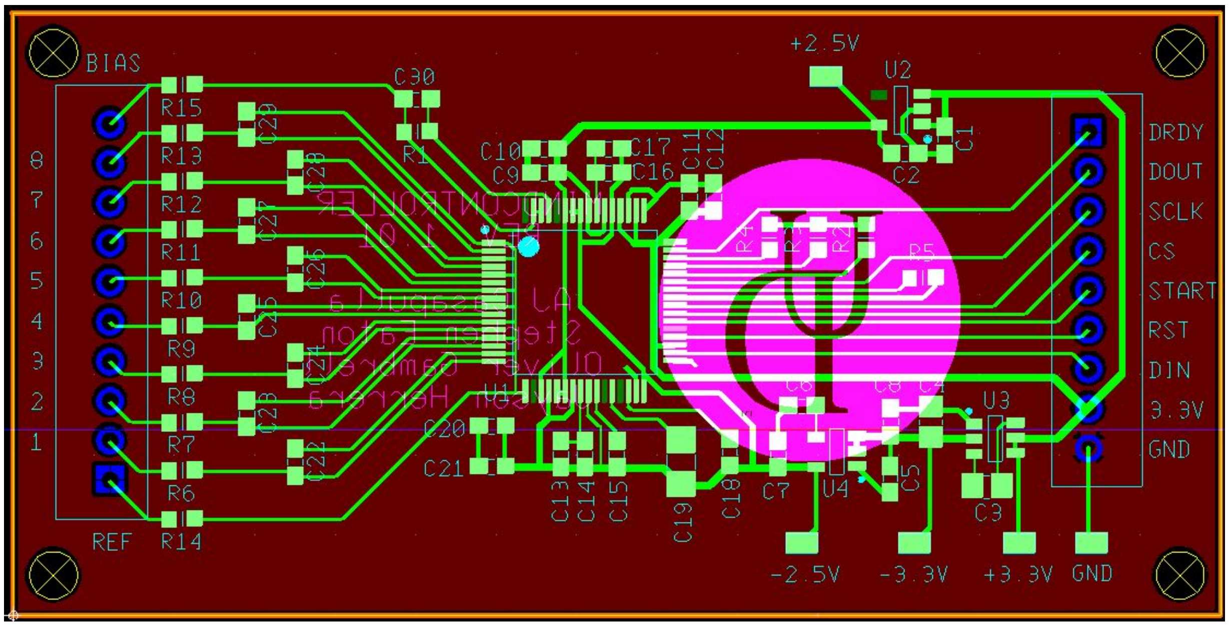
The new IC we found was the ADS1299. This IC has 8 differential amplifiers, each attached to a 12 bit ADC, and a patient bias amplifier. Because it could so readily replace and enhance our existing hardware, we quickly decided to abandon our efforts from last semester and begin working with this new chip.

Using examples in the datasheet, TI's evaluation board, and the OpenBCI project as references, we developed a schematic for the ADS1299 with it set up in a referential montage mode (the inverting sides of each amplifier were attached to the reference). The ADS would communicate with a microcontroller via an SPI bus. The microcontroller would supply +3.3V which our board would convert into the +2.5V and -2.5V required by the ADS.



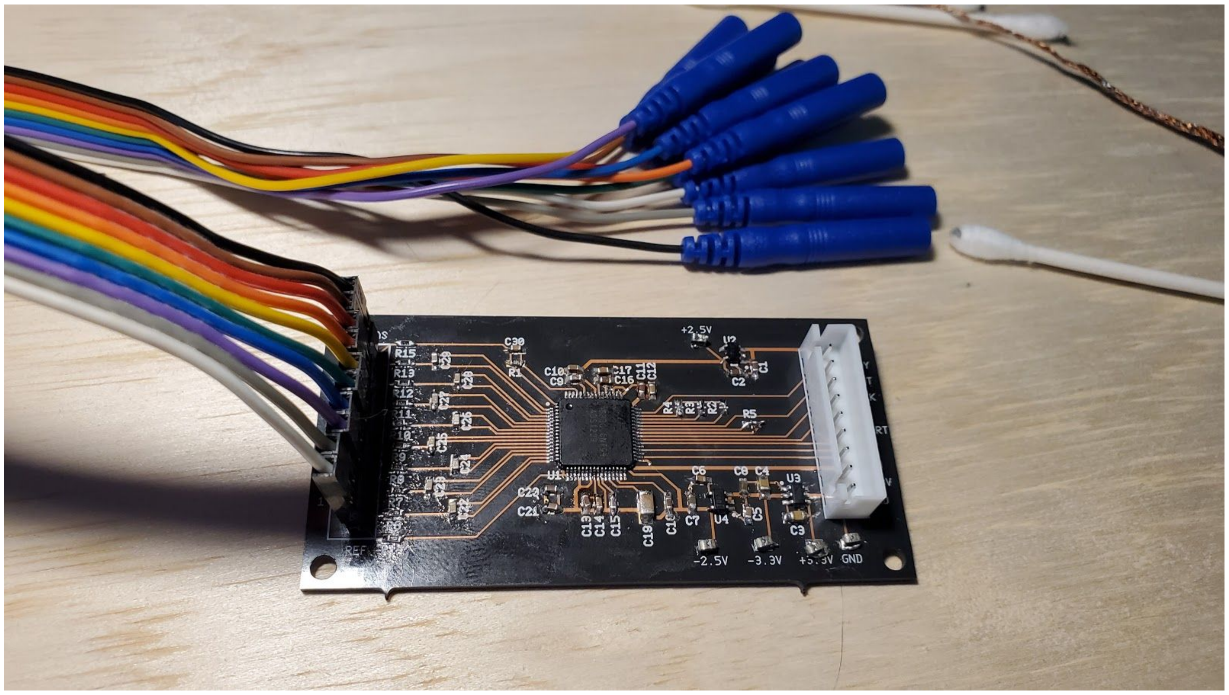
MindController schematic

Once the schematic was complete, it was laid out, gerber files generated, and the board and bill of materials ordered.



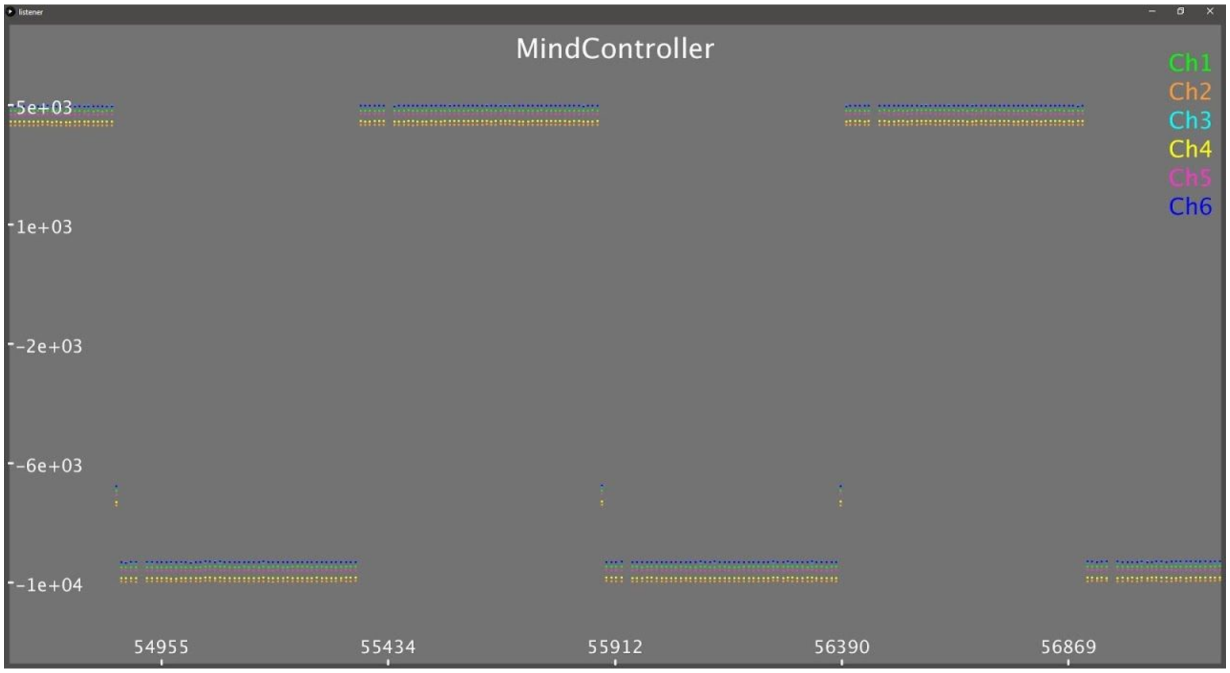
MindController PCB layout

We had some more trouble than expected assembling and troubleshooting the PCA, resulting in lost time. Unmelted solder paste sporadically shorted the power lines to ground but, luckily, caused no lasting damage.



MindController assembled

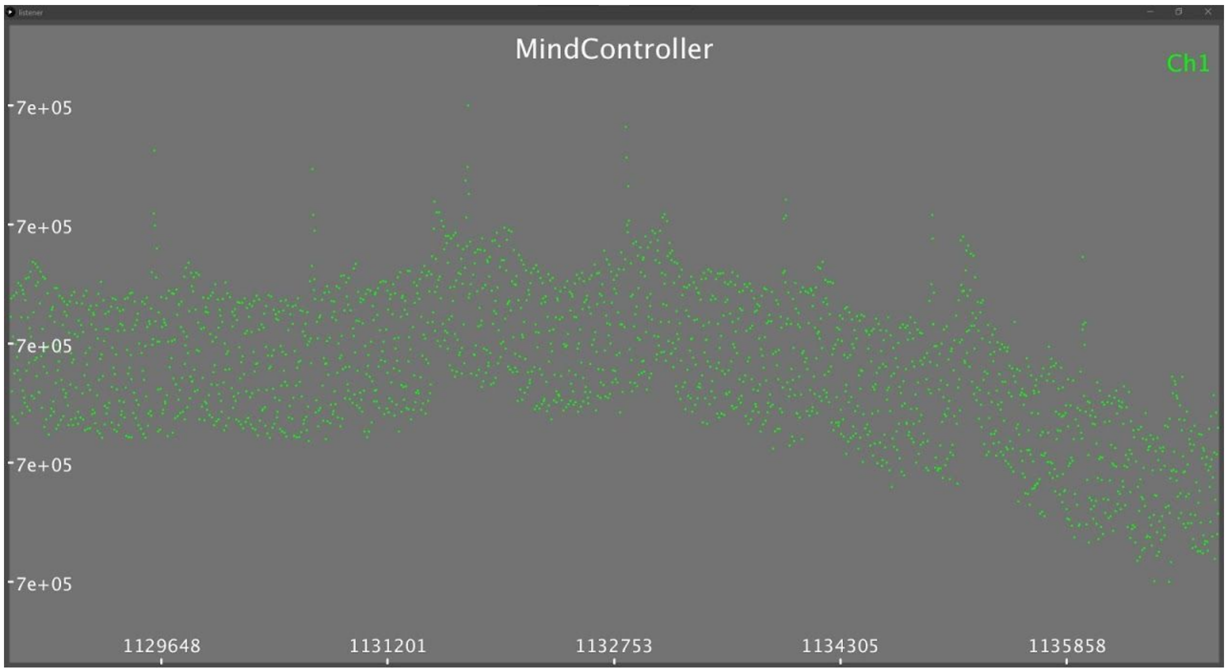
Because the ADS has all of the analog to digital converters built in, we were able to use another 3.3V logic level microcontroller, the ESP8266 to communicate with MindController. The Arduino Due we had planned to use wasn't on hand due to the separation of materials after the COVID-19 shutdown. We were able to write a program that ties all channels to ground to measure noise and offset, then ties them to an onboard square-wave generator to ensure they worked as expected.



Onboard square wave test signal

We used the Arduino library Plotter to quickly display the data. This turned out to be troublesome, but due to a lack of time, we weren't able to put together a better solution. The plotter was only able to display 6 channels at once. It automatically resized the window, making it difficult to get an idea of peak to peak changes. The Y scale markings weren't precise enough to be of any use.

After verifying that the ADS was working as expected, we modified the program to read in real data from the attached probes. We were able to make out heartbeats and verify that the bias drive was improving the readings, but lacked the time needed to further dig into the data and use it.



Heartbeats read using Mindcontroller

3D Printed Enclosure : (Jayson)

Using Solidwork, we were able to design an enclosure that housed the Arduino Due and PCB. The team member assigned to this project had limited experience using this type of software. Therefore, it took time and effort for the individual to learn CAD.

The first step was to figure out the dimensions of the Arduino Due. When searching the web, we were not able to find the specific measurements of the Due. Instead, the only information available was the surface dimensions of the Arduino Mega 2560, which uses the same PCB. We could have used this to design the enclosure's surface, but we needed the details of the USB, power cord, and pins.

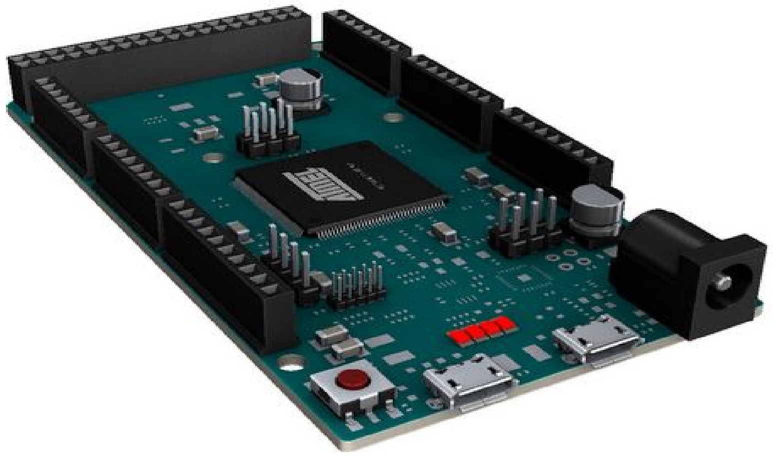


Fig. Arduino Due- An Arduino Due 3D model used from https://grabcad.com/

After further investigation, we discovered that we were able to import a 3D model of the Arduino into Solidworks and design the enclosure around it for a much more accurate design. Using GrabCad, we were able to perform this action.

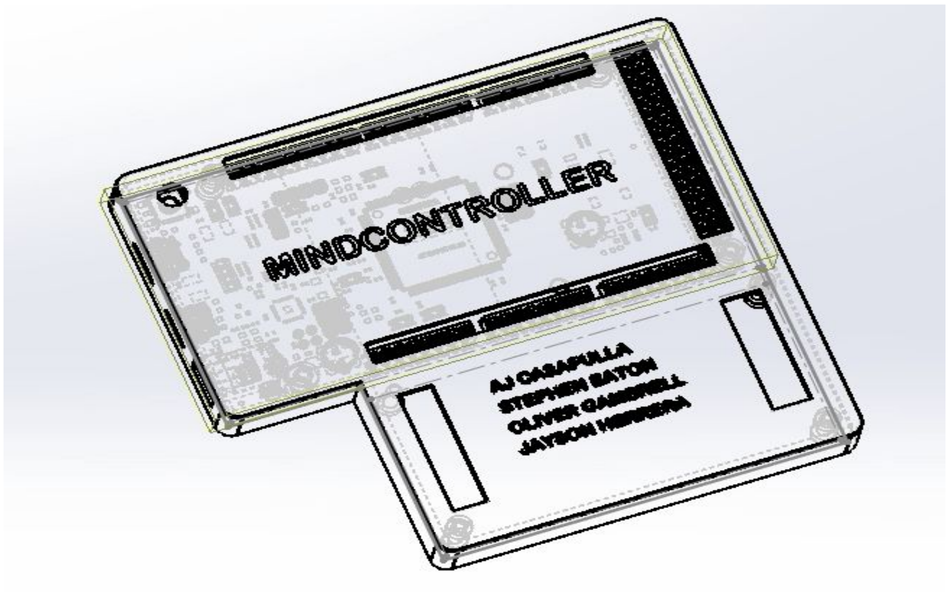


Fig. Solidwork- Enclosure designed around the 3D model.

Once the design was finalized, with a personal 3D printer, we were able to manufacture the enclosure.

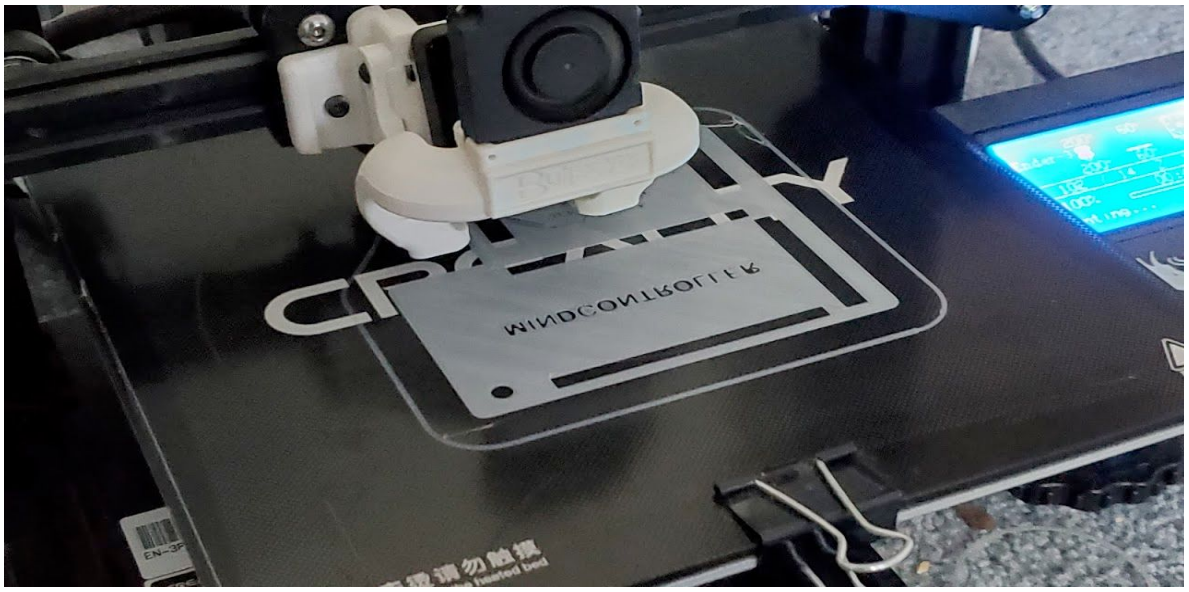
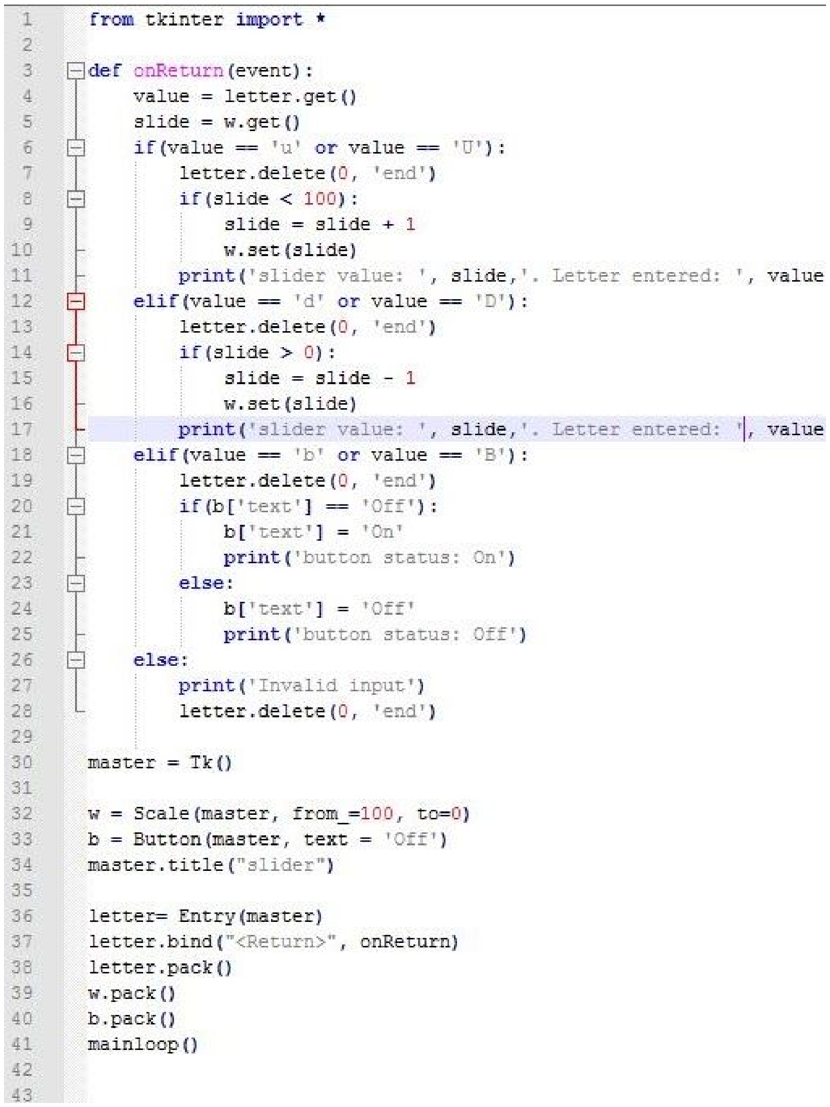


Fig. 3D Printer- Enclosure being printed.

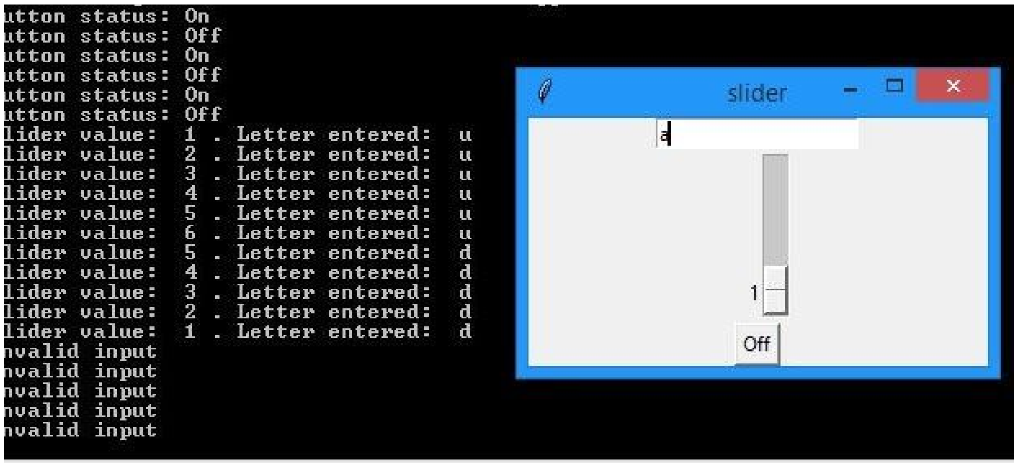
<u>Program</u> : (Stephen)

The colored box was removed because the color wasn't switching as the value was altered on the slider. The code was then updated to have a button, and control both the slider and the button by using different keyboard entries from the user (**Fig. slideKeyboard**). The output of the code, shown in **Fig. slideKeyboardOutput**, shows what character was entered and what was done to either the slider or the button. If a "d" or "u" is entered, the slider will either decrease or increase. If a "b" is entered, then the button will switch between "on" and "off". If a different character is entered then the code prints out "invalid input" and both the slider and the button remain unchanged.



```
from tkinter import *

def onReturn(event):
    value = letter.get()
    slide = w.get()
    if(value == 'u' or value == 'U'):
        letter.delete(0, 'end')
        if(slide < 100):
            slide = slide + 1
            w.set(slide)
        print('slider value: ', slide,'. Letter entered: ', value
    elif(value == 'd' or value == 'D'):
        letter.delete(0, 'end')
        if(slide > 0):
            slide = slide - 1
            w.set(slide)
        print('slider value: ', slide,'. Letter entered: ', value
    elif(value == 'b' or value == 'B'):
        letter.delete(0, 'end')
        if(b['text'] == 'Off'):
            b['text'] = 'On'
            print('button status: On')
        else:
            b['text'] = 'Off'
            print('button status: Off')
    else:
        print('Invalid input')
        letter.delete(0, 'end')

master = Tk()

w = Scale(master, from_=100, to=0)
b = Button(master, text = 'Off')
master.title("slider")

letter= Entry(master)
letter.bind("<Return>", onReturn)
letter.pack()
w.pack()
b.pack()
mainloop()
```

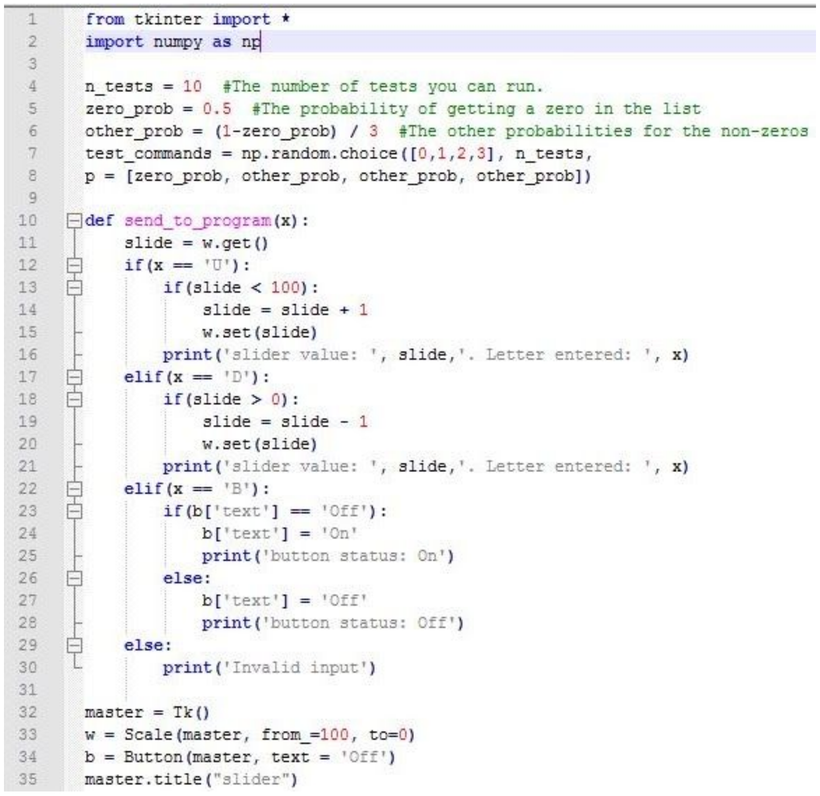
Fig. slideKeyboard: the code for the slider and button that uses the keyboard for controls.



```
utton status: On
utton status: Off
utton status: On
utton status: Off
utton status: On
utton status: Off
lider value:  1 . Letter entered:  u
lider value:  2 . Letter entered:  u
lider value:  3 . Letter entered:  u
lider value:  4 . Letter entered:  u
lider value:  5 . Letter entered:  u
lider value:  6 . Letter entered:  u
lider value:  5 . Letter entered:  d
lider value:  4 . Letter entered:  d
lider value:  3 . Letter entered:  d
lider value:  2 . Letter entered:  d
lider value:  1 . Letter entered:  d
nvalid input
nvalid input
nvalid input
nvalid input
nvalid input
```

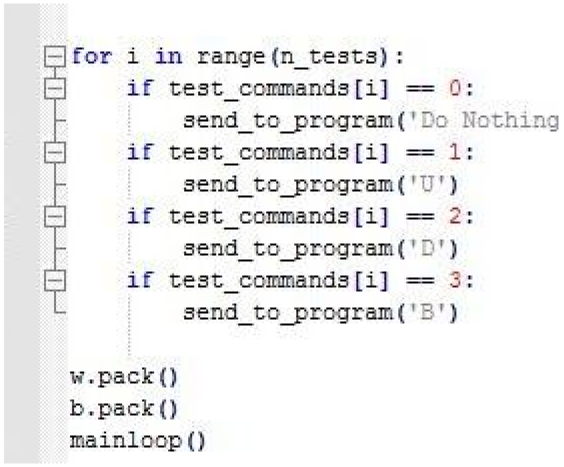
Fig. slideKeyboardOutput: output of the slider and button code that used a keyboard input

The code was later altered to use randomly selected numbers to pick which command will be used. The function "send_to_program" (**Fig. randomSelect1**) shows that once a character is inputted, it will be used to control the slider and button. It functions the same as the code above; the actions are based on an input of a "b", "d", or "u". The letter that is passed into the function is determined by the for loop (**Fig. randomSelect2**). This loop selects a number from the n_test array that was specified in the code. The output can be seen below in Fig. randomSelectOutput.



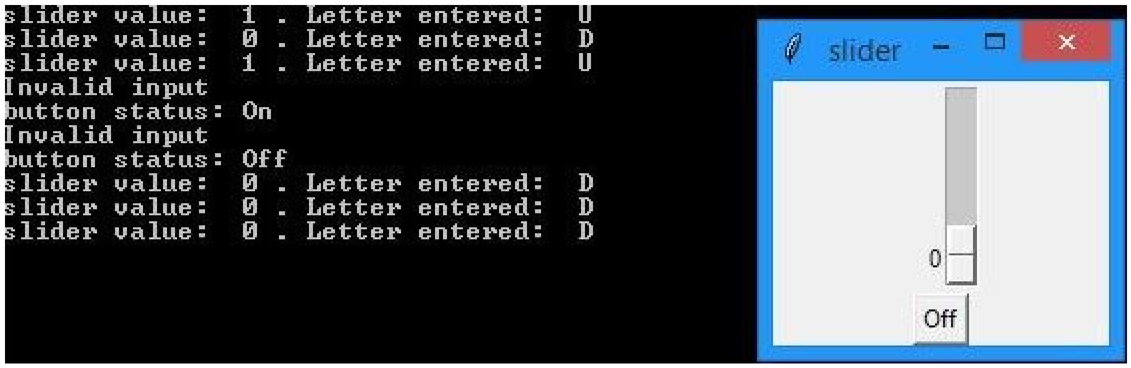
```
from tkinter import *
import numpy as np

n_tests = 10  #The number of tests you can run.
zero_prob = 0.5  #The probability of getting a zero in the list
other_prob = (1-zero_prob) / 3  #The other probabilities for the non-zeros
test_commands = np.random.choice([0,1,2,3], n_tests,
p = [zero_prob, other_prob, other_prob, other_prob])

def send_to_program(x):
    slide = w.get()
    if(x == 'U'):
        if(slide < 100):
            slide = slide + 1
            w.set(slide)
        print('slider value: ', slide,'. Letter entered: ', x)
    elif(x == 'D'):
        if(slide > 0):
            slide = slide - 1
            w.set(slide)
        print('slider value: ', slide,'. Letter entered: ', x)
    elif(x == 'B'):
        if(b['text'] == 'Off'):
            b['text'] = 'On'
            print('button status: On')
        else:
            b['text'] = 'Off'
            print('button status: Off')
    else:
        print('Invalid input')

master = Tk()
w = Scale(master, from_=100, to=0)
b = Button(master, text = 'Off')
master.title("slider")
```

Fig. randomSelect1: First part of the code for selecting random commands



```
for i in range(n_tests):
    if test_commands[i] == 0:
        send_to_program('Do Nothing
    if test_commands[i] == 1:
        send_to_program('U')
    if test_commands[i] == 2:
        send_to_program('D')
    if test_commands[i] == 3:
        send_to_program('B')

w.pack()
b.pack()
mainloop()
```

Fig. randomSelect2: The second part of the code for selecting random commands



```
slider value:  1 . Letter entered:  U
slider value:  0 . Letter entered:  D
slider value:  1 . Letter entered:  U
Invalid input
button status: On
Invalid input
button status: Off
slider value:  0 . Letter entered:  D
slider value:  0 . Letter entered:  D
slider value:  0 . Letter entered:  D
```

Fig. randomSelectOutput: The output of the randomSelect code

Ideal Project Finish

Concerning the neural network, under ideal conditions the original Bollinger-interface would be used on live data, and a network would be chosen to convert the score matches of a signal to a command that the program could use. We believe that given more time and our original microcontroller, we could better process the data coming from the MindController to turn it into something usable and visually identifiable as EEG data. Under ideal circumstances, the 10-20 EEG electrode placement would have been implemented. This method would have allowed us to collect live data from different areas of the brain to analyze the various wave frequencies. Subsequently, we would have determined which waves were appropriate to our specific application. At the moment, the code will completely run through and print out the final slider and button. An ideal finish for this portion of the code would be if the slider and button would display as soon as the code starts running so we could watch as the slider and button change.

Wiki-links

1. https://sites.google.com/udel.edu/mindcontroller/home (Main page)
2. https://sites.google.com/udel.edu/mindcontroller/Oliver (Oliver's Page)
3. https://sites.google.com/udel.edu/mindcontroller/AJ (AJ's Page)
4. https://sites.google.com/udel.edu/mindcontroller/Jayson (Jayson's page)
5. https://sites.google.com/udel.edu/mindcontroller/Stephen (Stephen's page)

APPENDIX

Equipment Needed:

• 2 electrode probes per channel
• 1 Instrumentation amplifier IC per channel
• Up to 2 quad op amps per channel
• Resistors and capacitors of varying sizes to support the filtering and amplification ICs
• A microcontroller with 8+ ADC channels
• A project box or other container to house the amplifier and help shield from noise
• Software for visualizing incoming EEG signals
• Machine learning / neural network software
• A computer to process incoming EEG signals and run the demo program

Budget and costs:

• $50 - pack of electrodes
• $40 - microcontroller
• $12 / instrumentation amplifier IC
• $1-2 / quad op amp IC
• $10 metal project box

Costs may increase in the event that we decide to add more channels or have a PCB produced.