# ELECTROENCEPHALOGRAM MOVEMENT IMAGERY

Udhayan Dherman (Christopher)
Dept of Engineering and Fabrication Technologies
Douglas College
New Westminister, Canada
udhayan.dherman02@gmail.com

***Abstract*—This paper describes basic methods of Electroencephalogram classification. This recorded data in return will be run through a processing algorithm and be sent wirelessly to a microcontroller to then be used to satisfy conditional statements in order to control electrical components. The EEG recording will be captured using electrodes over the scalp which then will be amplified on the brain-computer interface in order to analyze. The processing algorithm consists of multiple node functions, in which the raw EEG data is run through and optimized. The clean-filtered EEG signals are transferred via wifi, thus granting us the ability to control electrical applications wirelessly.**

***Keywords—Electroencephalography, EEG, Data processing, Machine Learning, Movement Imagery***

## I. INTRODUCTION

Systems that intertwine with one another have always been an interest of mine. Therefore, having the chance to be able to work on the Electroencphagram project for the ENGR-2999 Capstone was a great opportunity. Before any work began on the project, Previous research was done on what Electroencephalography was about and how the concept of it worked. Everything done beyond that was work done in ENGR-2999 lab time. Many of the concepts talked about in this paper granted me the chance to venture into a new domain. Most of the Mathematical concepts throughout the paper were taught in previous courses so understanding the mathematical aspects was not bad. The objective of the project is to be able to capture EEG (Electroencephalogram) signals to then be processed using a basic machine-learning algorithm that classifies the difference between moving your right hand and moving your left hand. Therefore, now having obtained the trained data, which then can be used to control 2 LED'S in order to simulate whether the subject is thinking left or right.

## II. HARDWARE AND SOFTWARE.

Thanks to the Engineering department at Douglas College, they were able to supply me with the physical equipment needed to measure EEG signals. This section deals with the type of equipment that will be used in the project and why the specific equipment was chosen.

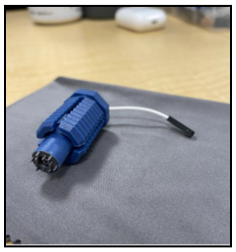


*Figure 1: Spiked Electrode*

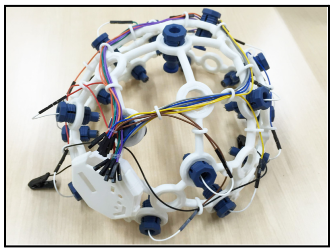


*Figure 2: Ultra-Cortex IV*

### *A. Hardware*

In Figure 1 we have what is called a spiked electrode, the ends of the electrode come into contact with the skin on your head. Be cautious when placing them on your head as applying too much pressure can cause a large impedance which leads to faulty data. The electrodes consist of a plastic housing, with the metal electrode tip attached to a spring so that the user has a firm and comfortable fit. At the end of the electrode, a wire is connected which will then connect to other longer wires that will run through the (Figure 2) Ultra-Cotrex IV helmet. The Ultra-Cortex is the structural placement of where each electrode will sit over the parts of the brain. The electrode will cover the: Frontal Left, Frontal Right, Central, Temporal Left, Temporal Right, Parietal, and Occipital. According to Y. Wang [1] the Frontal and Parietal part of the brain is the only aspect we really care about. Those sections deal with motor functions and sensory input. Later in the paper, we will discuss the reason for all 8 electrodes, instead of just 3 to cover the Frontal and Parietal sections.

Now, let's take a look at the final piece of hardware that we need to record EEG signals, the microcontroller. OpenBCI provides a variety of boards to use, but in our case, we will be using the Cyton Board. The specifications page [2] can be found at the end of the paper in the References section. The Cyton acts as a middleman, between the subject and the computer. Once an electrode interacts with an electric field, the Cyton transports this information to the computer through something called the OpenBCI dongle. The dongle communicates information through the serial port of your computer.

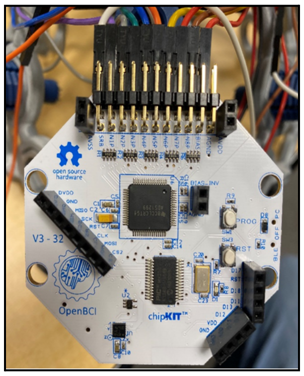


Figure 3: Cyton Board

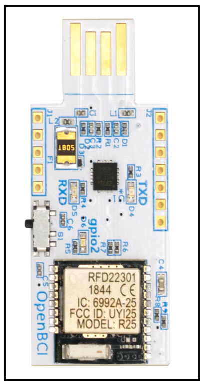


Figure 4: OpenBCI Dongle

There is no specific order in which you must connect the electrodes to the Cyton board, by this I mean relative to the N#P pin. But, in order to know which electrodes map to which section of the brain, one would strongly suggest mapping out a personal layout.

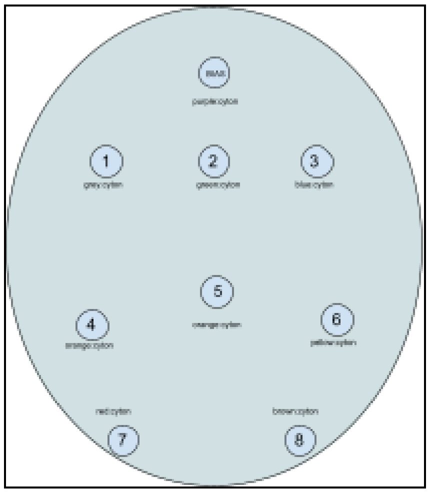


Figure 5: Electrode Mapping

The Cyton consists of 10 pins. Eight of them are N#P pins and the remaining two are the SRB (System Reference Bus) and BIAS pins. The SRB and BIAS pins are your only exception to which order you can connect the electrodes. The reference electrode (SRB) helps to establish a common reference point for the bioelectrical signals being measured by the active electrodes, this is the ear clip. The BIAS pin is a ground connection that ensures a stable electrical potential.

### B. *Software*

Next is the software application provided by OpenBCI, this allows us to view these detected voltages. The application is called OpenBCI GUI. The OpenBCI GUI is OpenBCI's powerful software tool for visualizing, recording, and streaming data from the OpenBCI Boards. More specifications and installation information will be provided in the References section [3].

## III. UNDERSTANDING WHAT WE ARE READING

While conducting the background research needed, any prior knowledge of this concept was hard to come about. Therefore, if there were materials found, it was highly advanced and needed much more additional time. For instance, what are we looking for? Is it just some voltage like in a circuit or perhaps some type of frequency occurrence? Therefore, the decision was made to set a timeline of about 7 days for strictly research purposes only. This was not only for my benefit but also because if any other future Douglas College Engineering student decides to continue working on the Electroencephalogram project, they will have a strong foundation to learn from.

### A. *How do the Electrodes work?*

The electrodes detect tiny electrical charges that result from the activity of your brain cells. This activity consists of electrical impulses [4]. Mayo Foundation for Medical Education and Research (MFMER) (2021) [5] explains that when an electrical impulse is generated by a neuron in the brain, it creates a small electrical field around the neuron. These are then detected by the electrode and then amplified on the computer. Therefore, the time series [6] will be our main widget for displaying biosensing data. It processes and shows the electrophysiological signal (caused by changes in the membrane potential of individual cells) in real time, with each graph representing the voltage detected at one point in time by an electrode. It measures the absolute amplitude of the signal in voltage, in units of μVrms (microvolts, root mean squared). “µVrms”, this value is the RMS (Root mean squared) voltage in microvolts (10^-6).

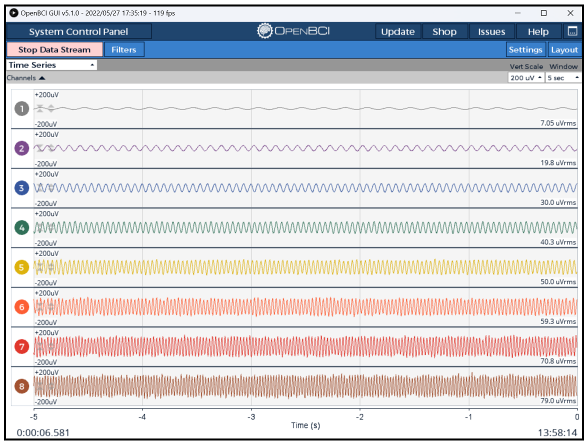


*Figure 6: OpenBCI GUI - 8 channel Time Series*

## *B. What are µVrms?*

Consider 2 circuits. One circuit has a 30V DC power supply with a resistor connected in series of 5Ω. The other circuit has some unknown AC voltage supply “x”, with the exact same resistor connection. In the DC circuit:

$$Power = \frac{V^2}{R} = \frac{(30)^2}{5} = 180W$$

Now to generate 180W of power in the AC circuit we need a voltage that is equivalent to 30V DC. Supplying an AC circuit with 30V will not generate 180W of power. So, to calculate the required voltage we need to find the peak voltage generated by the sine function, where:

$$V_{pk} = \sqrt{2} \cdot V_{rms} = \sqrt{2} \cdot 30V = 42.4V$$

Thus, we need 42.4V supplied in the AC circuit to generate the 180W of power.

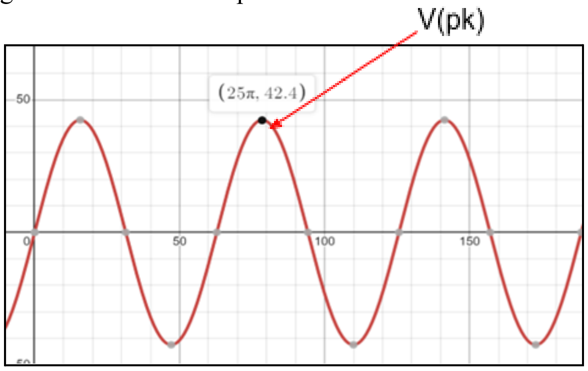


*Figure 7: generated sine graph representing peak Voltage of an AC voltage*

Therefore, we can represent Vrms as:

$$V_{rms} = \frac{V_{pk}}{\sqrt{2}}$$

This is the value that will be displayed on the time series but the voltages recorded are far smaller giving them a range of around (10^-6), hence the µ (micro).

## IV. DATA PROCESSING

This is the most crucial part of the project, -processing collected data. Without this step, applying raw data will leave a large error percentage in the application stage. If we take a look at (Figure 6), we see a diagram known as a pipeline. In this pipeline, there are multiple sections, called nodes. Each node has a function that alters the raw data running through it. The Software that allows us to do this is NeuroPype. NeuroPype is a powerful platform for real-time brain-computer interfacing (BCI), neuroimaging, and neural signal processing, which supports a range of biosignal modalities (NeuroPype, 2015) [7]. This section of the paper explains the function of each node and why this specific alteration was needed.

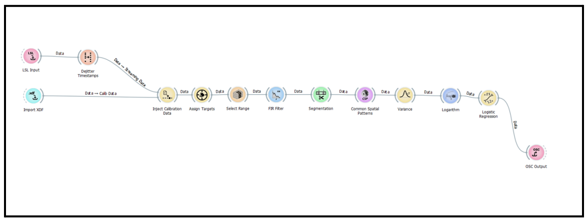


Figure 8: Neuropype PipeLine data processing - Larger scale at pg. 10.

**LSL (Lab Streaming Layer) Input**: this is a feature that allows the software to receive real-time data from a wide range of EEG and other neurophysiological recording devices that support the LSL protocol. Therefore, this node is needed in order to receive live EEG data.

**Dejitter Timestamps**: this node is a data processing node that helps to correct the timing of events or data points that are recorded in a variable time resolution. It works by taking time series input of data points that are recorded at variable time intervals and then using statistical methods to estimate the true time of each data point. The rest of the pipeline requires these timestamps, in order to select a manageable amount of data.

**Import XDF**: Imports the xdf data from local files that have been recorded already and populated with EEG data. This is calibrated which will be crucial to the operating state of the node further on.

**Inject Calibration Data**: This node is used to inject calibration data into the data processing pipeline. The Inject Calibration Data node will first let through the calibration data, which will then trickle down your subsequent processing pipeline, giving every node a chance to calibrate, and providing a good ground truth. After that, the Inject Calibration Data node will let through the regular streaming data from your actual data source, so that the pipeline can do its regular processing without unwanted signals that could interfere with the data.

**Assign Targets**: This node allows you to define which of the markers in the data should be used for event-related analysis. In our case, the node will accept the marker strings (left or right) and convert them into an integer (1 or 0), and take these numbers as the target values.

**Select Range**: This node will take any nonempty chunk that has the desired axis, and then extract the desired selection range. This node can only work thanks to the Dijetter Timestamps node, so the pipeline can now work with a manageable data amount.

**FIR Filter (Finite impulse response)**: This is a signal-filtering node. We are using passbands [6, 7] Hz and [30, 32] Hz, meaning that the filter will allow signals in these frequency ranges to pass through relatively untouched. Overall, setting the FIR filter frequencies to [6, 7, 30, 32] would be appropriate as we want to isolate a specific frequency component in the input signals, such as alpha (8-12 Hz) and gamma (30-100 Hz) waves.

**Segmentation:** This node is used to cut fixed-length segments out of a continuous time series. This only works with a time series and marker streams. The markers are flagged by the Assign Targets stream and then work on in Segmentation. In our case, we are cutting from 0.5s to 3.5s of the segment for each target marker.

**Common Spatial Patterns:** The CSP node identifies spatial filters that optimize the separation of two or more classes of EEG signals based on their covariance matrices. Firstly, we need to segment the incoming stream in fixed durations, we can call these "clips". This is done in the previous node - the Segmentation node. Then, computation of the covariance matrices of each clip is needed to then average the matrices. A covariance matrix is a square matrix that summarizes the covariances between multiple variables in a dataset [8]. In other words, it measures the degree to which two variables vary together. So, in our situation we are using 8 electrodes, thus 8 channels resulting in 8 different voltage values.

Recall from pg.1- section A, it was mentioned that the project only needed to use 3 electrodes to examine the differences in order to classify movement imagery. Therefore, the reason the project needs all 8, is due to the CSP model has issues dealing with a zero entity in its arguments.

| | Voltage(μVrms) | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ |
|---|---|---|---|---|---|---|---|---|---|
| Time (s) | | | | | | | | | |
| 0.5 | | | | | | | | | |
| 1 | | | | | | | | | |
| 1.5 | | | | | | | | | |
| 2 | | | | | | | | | |
| 2.5 | | | | | | | | | |
| 3 | | | | | | | | | |
| 3.5 | | | | | | | | | |
| | | | | | | | | | |

*Figure 9: Table representing voltage values entering the CSP node*

The following table (Figure 7) represents a small set of data for each channel from a time initial start time of 0.5(s) to final end time of 3.5(s). This was a parameter set in the segmentation node. ($X_n$) is the measured voltage across the scalp in (μVrms). So, the covariance matrix will be an 8x8 matrix. Each entry of the covariance matrix is calculated using the following formula [9]:

$$Cov(X_n X_m) = E(X_n X_m) - E(X_n)E(X_m)$$

To find the Expected Value (E) of one channel, use the following formula:

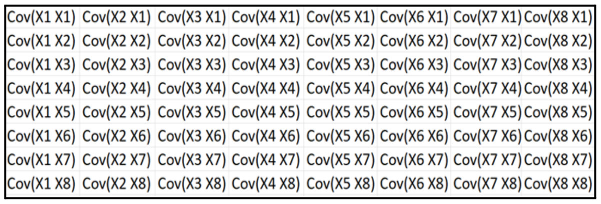
$$E(X_n) = \sum_{(n=1)}^{k} n(X_n)$$

To find the Expected Value (E) of the product of two channels, use the following formula:

$$E(X_n X_m) = [\sum_{(i=1)}^{k} i(X_n)] \cdot [\sum_{(i=1)}^{k} i(X_m)]$$

Where "k", is the end of the clip and "n" is the beginning of the new set of the data with a new time axis done by the Segmentation node.

The following matrix is produced, (Figure 10). A property of the covariance matrix is that is symmetric along the diagonal, so the lower-left triangle is the same as the upper-right triangle. So, it would be reasonable to only work out either side.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Cov(X1 X1) | Cov(X2 X1) | Cov(X3 X1) | Cov(X4 X1) | Cov(X5 X1) | Cov(X6 X1) | Cov(X7 X1) | Cov(X8 X1) |
| Cov(X1 X2) | Cov(X2 X2) | Cov(X3 X2) | Cov(X4 X2) | Cov(X5 X2) | Cov(X6 X2) | Cov(X7 X2) | Cov(X8 X2) |
| Cov(X1 X3) | Cov(X2 X3) | Cov(X3 X3) | Cov(X4 X3) | Cov(X5 X3) | Cov(X6 X3) | Cov(X7 X3) | Cov(X8 X3) |
| Cov(X1 X4) | Cov(X2 X4) | Cov(X3 X4) | Cov(X4 X4) | Cov(X5 X4) | Cov(X6 X4) | Cov(X7 X4) | Cov(X8 X4) |
| Cov(X1 X5) | Cov(X2 X5) | Cov(X3 X5) | Cov(X4 X5) | Cov(X5 X5) | Cov(X6 X5) | Cov(X7 X5) | Cov(X8 X5) |
| Cov(X1 X6) | Cov(X2 X6) | Cov(X3 X6) | Cov(X4 X6) | Cov(X5 X6) | Cov(X6 X6) | Cov(X7 X6) | Cov(X8 X6) |
| Cov(X1 X7) | Cov(X2 X7) | Cov(X3 X7) | Cov(X4 X7) | Cov(X5 X7) | Cov(X6 X7) | Cov(X7 X7) | Cov(X8 X7) |
| Cov(X1 X8) | Cov(X2 X8) | Cov(X3 X8) | Cov(X4 X8) | Cov(X5 X8) | Cov(X6 X8) | Cov(X7 X8) | Cov(X8 X8) |

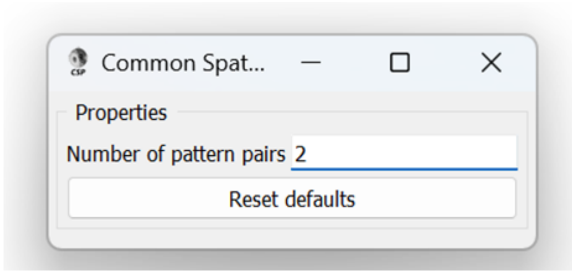
*Figure 10: Covariance matrix of all 8 channels*

Now, you might be thinking how would the pipeline compute the matrix for class (Right) where the data has not been run through if the class (Left) has been triggered? In fact, the data for all classes are collected during a different session and the segmentation and covariance matrix calculations are performed offline with pre-recorded data while the pipeline is booting. You should observe the message, "CSP model training…" in the boot log. Once the covariance matrices have been calculated for each class, they need to be averaged and then can be used to compute the spatial filters, which can then be applied to new data to extract the features for classification or other analyses. To calculate the average of the classes, add the corresponding entries of the RIGHT and LEFT matrices resulting in the matrix (A). Then divide each entry in (A) by 2 to produce the average matrix ($A^*$).

Having obtained the average matrix the next step is to calculate the spatial filter matrix that maximizes the difference between the covariance matrices of the two classes [9][10]. We do this by applying Eigendecomposition [11][12][13]. We find the eigenvalues by taking:

$$det(A^* - \lambda I) = 0$$

where "I" is the identity matrix. (A*-λI) leaves us with a matrix where all the values down the diagonal are subtracted by lambda(λ). After taking the determinate, solve for λ from the characteristic equation. This will produce a few eigenvalues, which then can be used to find their corresponding eigenvectors. The 2 eigenvectors with the highest eigenvalues are chosen as they represent the max difference between the two classes. These are then normalized and subtracted from each other to produce a final spatial filter. This is why we set the parameter in the CSP node as 2 as we want to eigenvectors.



Number of pattern pairs 2

*Figure 11: CSP Parameter menu*

In summary, the eigenvectors represent the spatial filters, that maximize the max difference in variance between the 2 classes [14][15]. And the eigenvalues represent the amount of variance captured by each vector. Therefore, by obtaining this information we have a pattern-like relationship that can be used to classify the two classes.

Next, is the Application of the Spatial Filter. Once the spatial filters have been calculated, they are applied to the EEG data to extract features. This is done by applying an element-wise multiplication of the spatial filter and the live EEG data sample resulting in a spatially filtered EEG signal.

Let **v** = the spatial filter eigenvector, and **u** = the live EEG vector.

$$v = < v_1, v_2, \dots, v_8 >$$

$$u = < u_1, u_2, \dots, u_8 >$$

Therefore, applying element-wise multiplication we get:

$$w = < v_1u_1, v_2u_2, \dots, v_8u_8 >$$

Where **w** = is the spatially filtered vector of the live EEG data.

**Variance**: is a measure of dispersion, meaning it is a measure of how far a set of numbers is spread out from their average value. Thus, the Variance Node takes the spatially filter EEG signal as an input and then calculates the variance over a time interval set as a parameter in the pipeline. Using the following formula:

$$Var(w) = E(w^2) - E(w)^2$$

Where $w$ = the spatially filtered vector. E = Expected -value.

Thus, the output of the Variance node is a time series of variance values calculated from multiple spatially filtered signals.

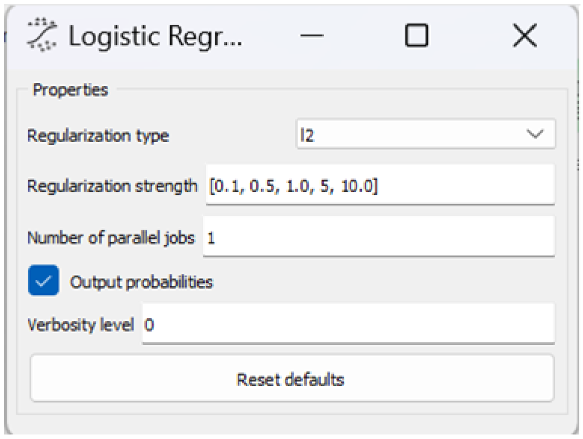
**Logarithm**: This node takes the logarithm of each element moving through it. Taking the logarithm of the variance can help to stabilize the variance over time [16]. As EEG signals are volatile. Therefore, by applying the logarithm of base "e", (which is just the natural log) we find the variability of the signal over time using the following formula:

$$ln(\alpha) = \beta$$

Where α = variance value and β = the variability scalar value.

**Logistic Regression**: "Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables." -IBM [17]. So, the type of logistic regression we will be using is Binary logistic regression. As we only want to classify between Left and Right. This approach has only two possible outcomes, (0,1)[19].

Node Parameters:


Logistic Regr...

Properties

Regularization type: l2

Regularization strength: [0.1, 0.5, 1.0, 5, 10.0]

Number of parallel jobs: 1

✓ Output probabilities

Verbosity level: 0

Reset defaults

*Figure 12: Logistic Regression Parameter menu*

Regularization Type: regularization is a technique used to prevent overfitting. The model tries to learn from the details along with the noise in the data and tries to fit each data point on the curve and improve the generalization performance of the model [18]. [L2], regularization involves adding a penalty proportional to the squared value of the coefficients [18]. Which controls the complexity of the model by shrinking the coefficients towards zero but does not set any coefficients exactly to zero[17].

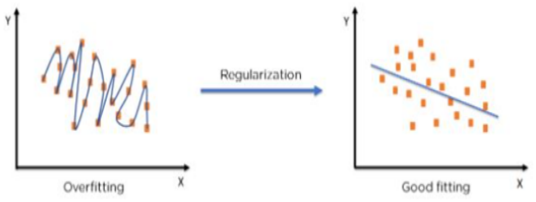


*Figure 13: Difference between Overfitting and Good Fitting*

Regularization Strength: The regularization strength is specified as [0.1, 0.5, 1.0, 5, 10.0], which means that the logistic regression algorithm will be run using these values of the regularization strength parameter (C) in a grid search. The grid search will typically train and evaluate multiple models using different values of C and choose the value of C that yields the best performance on a validation.

Now to find the binary outcome of the Logistic Regression, use the following formula:

$$\rho = \frac{1}{1+e^{-z}}$$

This will produce values between (0,1) allowing one to use this for classification purposes.

**OSC Output:** This node will allow us to export our data to a microcontroller. The parameters of the OSC output are the IP address, Receiver port, and message address. The IP address is in reference to the micro-controller IP address, I will explain how to find that later on. Next, the receiver port is what you set for the microcontroller and make sure you set the same value in the OSC Output. Finally, the message address is how we will name our messages. In our case we will use the address, "/data" and when we specify later in the micro-controller code we only want to receive data from that address.

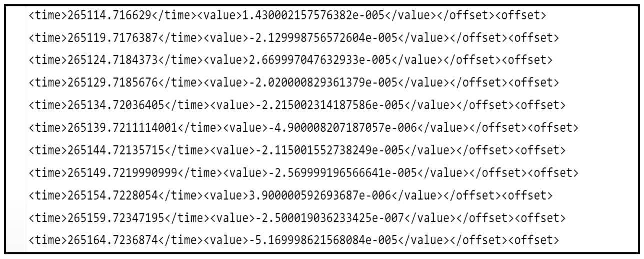
## V. RECORDING CALIBRATION DATA

This section deals with the recording of the calibration data that is needed to be run through the pipeline in order to provide a good reference. There are 3 parts to the setup. The first part deal with writing a script, which is needed to attach markers to the voltages in order for the pipeline algorithm to work. Python was a brainer choice, as there was a specific library that was needed which provided, PyLSL. PyLSL is an interface to Lab Streaming Layer. LSL is an overlay network for real-time exchange of time series between applications [20]. The main purpose of the script is to collect the value read at an electrode and store it with the marker label, left or right. Thus, when the program prompts the subject to think of moving their left or right hand, the subject should then only imagine moving the requested hand. The script will then record those values with the string marker attached to it and send it over to Lab Recorder, this is the second part of the setup.

Before we get to the second part of the setup, there are techniques that need to be kept in mind when understanding the rest of this section. As this relates to setting up the subject, in which we will record EEG data. The most important part when collecting data is that the user does not move or tense any muscles. They should only imagine the movement that is intended to be recorded. Another important part is to consider the type of hair your subject has. During my first few attempts at data collection, the electrode channels were not in operating condition. My hair is thick and long, thus not giving the electrode the contact it needs. Fortunately, this issue was solved: by not using any conditioner the day before; parting my hair a certain way; not consuming any caffeine for 6 hours, and using spiked-tipped electrodes instead of flat-tipped electrodes. Another important technique is that when collecting data, the trials have to be in randomized order and be in the range of 80-100 events. This induces a more realistic situation, which in return allows the application aspect to work better.

LabRecorder is the default recording program that comes with LSL. It allows the recording of all streams on the lab network into a single file, with time synchronization between streams (LabRecorder, 2016)[21]. Lab Recorder acts as a middleman between the Python script and the OpenBCI GUI. The Lab Recorder allows communication between the OpenBCI GUI where voltage values are collected which then are sent to the Python script where it is attached with markers and then sent back to the Lab Recorder where it will be stored in an xdf file. If you would like to what the data looks like, any online xdf viewer would suffice.

The data I recorded looks like the following:



```
<time>265114.716629</time><value>1.430002157576382e-005</value></offset><offset>
<time>265119.7176387</time><value>-2.129998756572604e-005</value></offset><offset>
<time>265124.7184373</time><value>2.669997047632933e-005</value></offset><offset>
<time>265129.7185676</time><value>-2.020000829361379e-005</value></offset><offset>
<time>265134.72036405</time><value>-2.215002314187586e-005</value></offset><offset>
<time>265139.7211114001</time><value>-4.900008207187057e-006</value></offset><offset>
<time>265144.72135715</time><value>-2.115001552738249e-005</value></offset><offset>
<time>265149.7219990999</time><value>-2.569999196566641e-005</value></offset><offset>
<time>265154.7228054</time><value>3.900000592693687e-006</value></offset><offset>
<time>265159.72347195</time><value>-2.500019036233425e-007</value></offset><offset>
<time>265164.7236874</time><value>-5.169998621568084e-005</value></offset><offset>
```

*Figure 14: Snippet of Calibration data*

The final part of recording calibration data is setting up OpenBCI's GUI widgets. We will be using two widgets: Time series and Networking widgets. For the Time Series Widget settings, make sure the PGA gain is all set to [x24], the input type is set to [ normal], the Bias include for all channels should be [yes], SRB2 should all be yes and SRB1 should all be [no]. For the Networking widget, we will need to change to the LSL protocol. Set the Data Type to [Time series], set the type to [EEG], Filter set to [OFF], and make sure that the name is set to something unique. Once you have the EEG stream running and the Python program running, update Lab Recorder and check mark both streams and start. We will have now obtained data that looks likes (Figure 14).

## VI. COMMUNICATION WITH THE OUTSIDE WOLRD

This section of the paper deals with how we can now communicate with physical components using our processed EEG data. The idea is to have live EEG data running through our pipeline to then be transferred via wifi to the microcontroller that will communicate with its pins. The microcontroller chosen is a NodeMCU (Node MicroController Unit). The reason for using this controller is for its System-on-Chip (SoC) The ESP8266 has a CPU, RAM, and networking (Wifi) [22]. The networking module is what we are aiming for. This will allow us to communicate with physical projects by sending data over wifi. The IDE we will be using to code the microcontroller will be Arduino IDE, as Arduino has the option to import the ESP8266 package.

This allows us to compile code for the ESP8266 microcontroller using its relative environment.

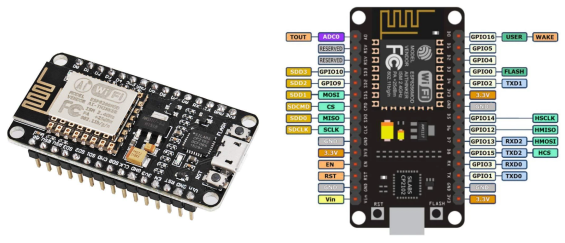


*Figure 15: NodeMCU Microcontroller*

Therefore, this section has all to do with application events. The idea is to transfer data over to the NodeMCU via wifi, where it will be then used to run functions in the NodeMCU. Here is the process step by step from the beginning. The subject will put on the helmet a prepare while the booting phase is running. They will have a contraption consisting of a NodeMCU.When the subject is thinking of either moving their left or right hand, this data is sent wirelessly to the NodeMCU. Where it will then check conditions and run functions in correspondence with the data sent. Therefore, the whole idea of this project is to have mind control powers.

There are 2 parts to the section, the circuit build for the NodeMCU and the code needed to communicate with the microcontroller. The circuit does not have to be complicated, as the objective is to determine if we are able to actually use brain signals to communicate with electrical components. Therefore, the circuit consists of 2 LEDs, one red and one green. The schematic of the circuit can be found in the files provided with the paper. The second part is the code needed. For this part, we can divide its objectives into 3 separate sub-sections. Firstly, connecting to a network. Before any sort of communication is done a connection to a network will need to be established. This can be done using 2 important libraries: [ESP8266WiFi.h], and [WiFiUdp.h]. The ESP8266WiFi library provides a wide collection of C++ methods (functions) and properties to configure and operate an ESP8266 module in station and/or soft access point mode (ReadTheDocs). The second one WiFiUdp.h is needed specifically for the programming of UDP routines (ReadTheDocs). The next part of the code deals with handling the OSC messages. The idea is to first see if the OSC message is a float type, and if the message is a float we can store that value into a variable and be able to use that for application purposes such as switching on an LED. The last part of the code deals with handling exceptions of the OSC messages. OSCErrorCode is a protocol that will handle all thrown exceptions, instead of coding your own triggers. The code for the project can be found in the files provided with the paper.

## VII. BUDGET

| Item: | Quantity: | Price: |
|---|---|---|
| Electrode | 8 | $60.80 |
| Cyton Board | 1 | $1350.50 |
| Ultra-Cortex Mark IV | 1 | $76.44 |
| OpenBCI Dongle | 1 | $270.27 |
| Wire | 8 | $6.98 |
| NodeMCU | 1 | $5.46 |
| Total | | $1770.45 |

**The Ultra-Cortex Mark IV, was 3D printed. Therefore the above cost is relative to the amount of filament needed to print the part.*

## VIII. REFERENCES

[1] Y.Wang, "Design of electrode layout for motor imagery-based brain-computer interface", May. 10, 2007. [OnlineDocument].Available:https://sccn.ucsd.edu/~yijun/pdfs/EL07.pdf

[2] OpenBCI, "OpenBCI Documentation: Cyton Specs", 2016. [Online]Available:https://docs.openbci.com/Cyton/CytonSpecs/

[3] OpenBCI, "OpenBCI Documentation: The OpenBCI GUI", 2016.[Online].Available:https://docs.openbci.com/Software/OpenBCISoftware/GUIDocs/#:~:text=The%20OpenBCI%20GUI%20is%20OpenBCI's,party%20software%20such%20as%20MATLAB

[4] John Hopkins Medicine, "Health: Electroencephalogram". [Online].Available:https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electroencephalogram-eeg#:~:text=The%20electrodes%20detect%20tiny%20electrical,provider%20then%20interprets%20the%20reading

[5] Mayo Clinic, “Tests and Procedures: EEG”, 2022. [Online]. Available:https://www.mayoclinic.org/tests-procedures/eeg/about/pac-20393875

[6] OpenBCI, “OpenBCI Documentation: GUI Widget Guide”, 2016.[Online].Available:https://docs.openbci.com/Software/OpenBCISoftware/GUIWidgets/#:~:text=%E2%80%8B,all%20of%20the%20available%20widgets.

[7] Neuropype, “Neuropype Documentation: Node Reference”, 2022.[Online].Available:https://www.neuropype.io/docs/nodes/neural.html

[8] N.Janakiev, “Understanding the Covariance Matrix”, *datascienceplus.com,* Aug. 3, 2018. [Online]. Available: https://datascienceplus.com/understanding-the-covariance-matrix/

[9] K.Siegrist, “Expected Value and Covariance Matrices”, *libretexts.org,*Apr.23,2022.[Online].Available:https://stats.libretexts.org/Bookshelves/Probability_Theory/Probability_Mathematical_Statistics_and_Stochastic_Processes_(Siegrist)/04%3A_Expected_Value/4.08%3A_Expected_Value_and_Covariance_Matrices

[10] D.Stansbury,“The Statistical Whitening Transform”, *theclevermachine.wordpress.com*, Mar. 30, 2015. [Online]. Available:https://theclevermachine.wordpress.com/2013/03/30/the-statistical-whitening-transform/

[11] A.Seb,“Eigenvalue Decomposition Explained”, *programmathically.com*, Dec. 2, 2020. [Online]. Available: https://programmathically.com/eigenvalue-decomposition/

[12] J.Fisher, “Eigenvalue Decomposition”, *guzintamath.com*, Feb.2,2019.[Online].Available:https://guzintamath.com/textsavvy/2019/02/02/eigenvalue-decomposition/

[13] J. Mourino, “Spatial Filtering the training process of a Brain Computer Interface”, La Sapienza University, June 2012 [OnlineDocument].Available:https://csilviavr.github.io/assets/publications/silvia01spatial.pdf

[14] L.Wasserman, “All of Statistics: a concise course in statistical inference” Springer texts in statistics, p. 51. ISBN 9781441923226,(2005).Available:https://en.wikipedia.org/wiki/Variance

[15] D.Wu, “Spatial Filtering for EEG-Based Regression Problems in Brain-Computer Interface”, University of California San Diego, Feb. 2, 2017.[Online Document]. Available: https://arxiv.org/pdf/1702.02914.pdf

[16] M.Pavlovic, “Log-normal Distribution – A simple explanation”,*towardsdatascience.com*,Feb.16,2022.[Online]. Available:https://towardsdatascience.com/log-normal-distribution-a-simple-explanation-7605864fb67c#:~:text=The%20variance%20of%20the%20log,the%20mean%20(see%20here)

[17] M.Banoula,“Regularization in Machine Learning”, *simplilearn.com*. Feb. 21, 2021. [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/regularization-in-machine-learning#:~:text=Regularization%20refers%20to%20techniques%20that,and%20prevent%20overfitting%20or%20underfitting

[18] IBM, “Logistic Regression: What is Logistic Regression”, 2019.[Online].Available:https://www.ibm.com/topics/logistic-regression

[19] Nuts and Bolts, “Introduction to Logistic Regression”, *appstate.edu*.[Online].Available:https://www.appstate.edu/~whiteheadjc/service/logit/intro.htm

[20] Liblsl, “Docs: Stream Outlets”, 2019. [Online]. Available:https://labstreaminglayer.readthedocs.io/projects/liblsl/ref/outlet.html

[21] Google Code, “Wikis: LabRecorder”. [Online]. Available: https://code.google.com/archive/p/labstreaminglayer/wikis/LabRecorder.wiki#:~:text=Overview,with%20time%20synchronization%20between%20streams

[22] Make-It, “NodeMCU ESP8266 – Detailed Review”, *make-it.ca.*[Online].Available:https://www.make-it.ca/nodemcu-details-specifications/

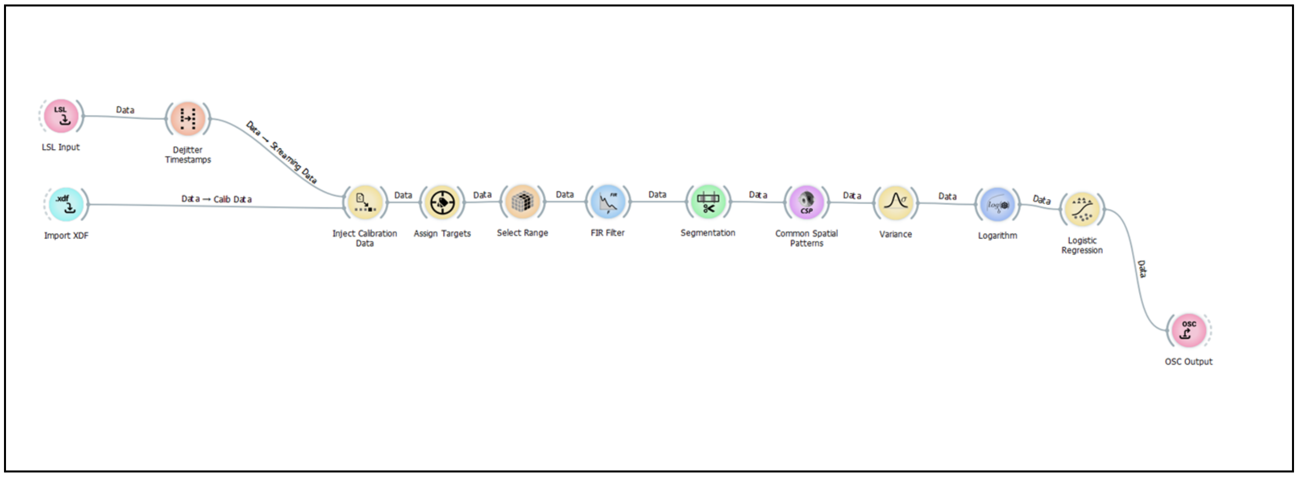


*Figure 8: Neuropype Pipeline for data processing*