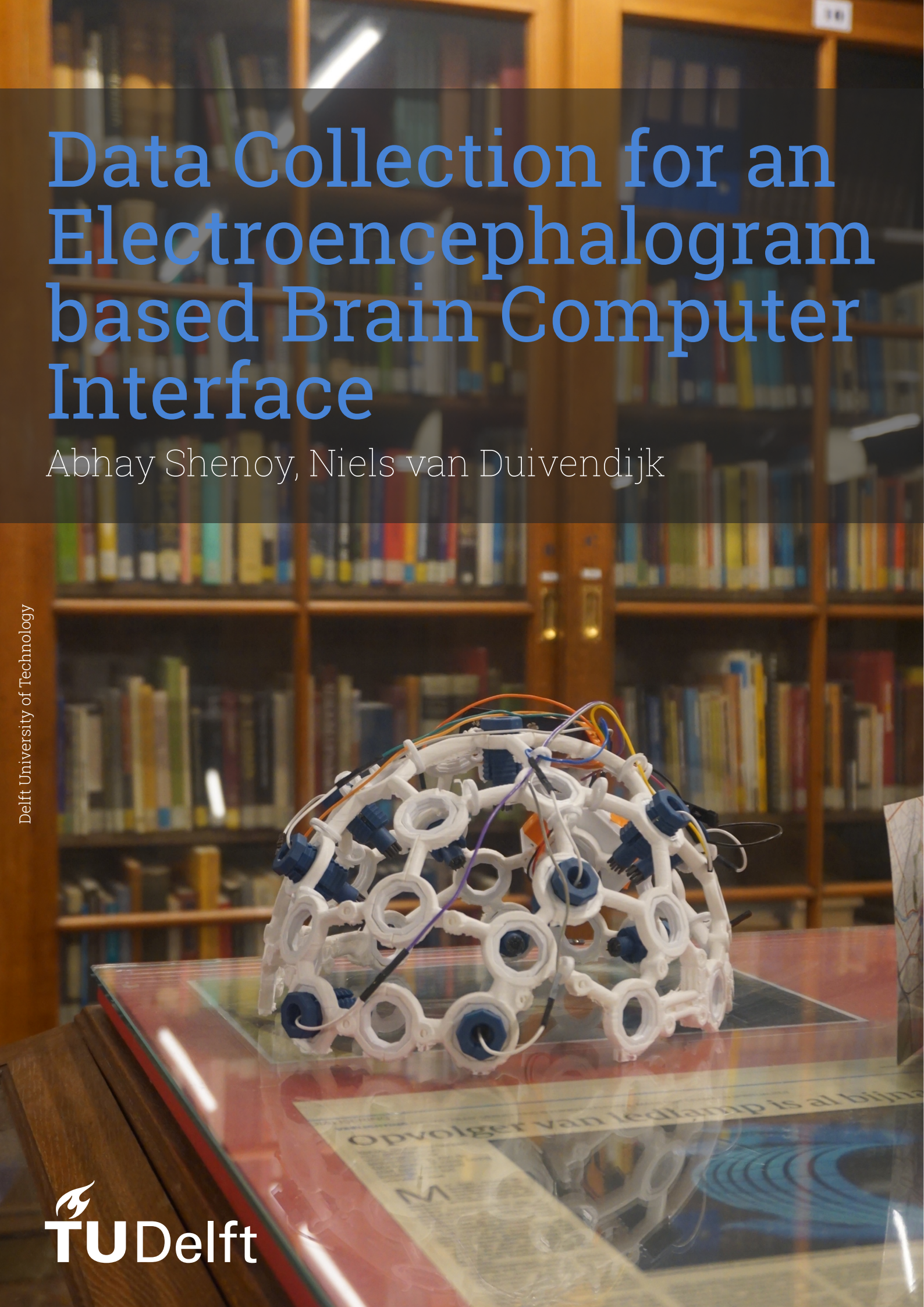
Data Collection for an Electroencephalogram based Brain Computer Interface

Abhay Shenoy, Niels van Duivendijk

Delft University of Technology



TU Delft

Data Collection for an Electroencephalogram based Brain Computer Interface

by

Abhay Shenoy, Niels van Duivendijk

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on Monday June 24th, 2024 at 15:30 PM.

Student number:	5510473, 4913825	
Project duration:	April 22, 2024 – June 28, 2024	
Thesis jury members:	dr. B. Hunyadi,	TU Delft, Supervisor
	Prof. dr. ir. O. Isabella,	TU Delft, Jury Chair
	M. Fieback,	TU Delft

This thesis is confidential and cannot be made public until June 28, 2024.

TU Delft

Preface

This thesis paper will be centered around the design and creation of a Brain Computer Interface (BCI) that is controlled through electroencephalogram (EEG) signals based on a Motor Execution paradigm. This project forms the Bachelor Graduation Project which is done in a group of 6 people. The group is further divided into 3 subgroups being data collection, machine learning and interface design. The goal of this thesis will primarily focus on the measurement and data collection process within the pipeline of the whole system. We would love to express our gratitude to Bori Hunyadi for supervision and guidance during our project. Thanks to our colleagues: Akram Chakrouni, Adnane Acudad, Ilyaas Shousha and Oussama Seddouki for their continued collaboration during the project and we sincerely hope that this project will be pursued further after this thesis.

Abhay Shenoy, Niels van Duivendijk
Delft, July 2024

Abstract

The main goal of this project is to utilize a commercially available OpenBCI Ultracortex IV for the measurement of Electroencephalogram(EEG) signals. A pipeline consisting of preprocessing, classification and extraction is employed to transform the motor execution EEG signal into a singular Left or Right output. This output is then further displayed on an Interface that offers the option to either calibrate or play a simple game.

The Ultracortex and relevant software were used to determine the sensor layout, with the placement of the sensors focused on areas which exhibited high cortical activity during motor execution. Experiments were strategically designed to optimize our chance of successful readings and OpenVIBE was used in conjecture with preprocessing filters to save the raw and filtered data which was further sent to the Machine Learning group.

The collected data was analyzed through Spectrograms, Power Spectral Density(PSD) and Event-Related Desynchronization/Synchronization(ERDS) plots. The analysis aimed to confirm whether the desired activity occurred and whether the observed patterns resemble those documented in other research papers.

The data from the headset is live-streamed to the interface via Lab Streaming Layer(LSL) where it undergoes further filtering before being sent to the Machine learning group. This process was done through python libraries which then allowed for efficient and effective communication between the other groups.

Contents

Nomenclature

This thesis incorporates a variety of abbreviations and specialized terminology. Table 1 shows a brief overview of the abbreviations in this project and Table A.1 shows descriptions of the jargon words that may be encountered in this thesis. Please refer to the these tables for clarification on any unfamiliar terms or abbreviations.

Abbreviations

Abbreviation	Definition
BCI	Brain Computer Interface
CAR	Common Average Reference
EEG	Electroencephalogram
ERD	Event Related Desynchronization
ERDS	Event Related Desynchronization/ Synchronization
ERS	Event Related Synchronization
ERSP	Event Related Spectral Perturbation
FFT	Fast Fourier Transform
FY	Fourier Transform
ICA	Independent Component Analysis
Mark IV	OpenBCI Ultracortex Mark IV
ME	Motor Execution
MEG	Magnetoencephalography
MI	Motor Imaging
PSD	Power Spectral Density
SNR	Signal-to-noise ratio
SSVEP	Steady-State Visually Evoked Potentials
STFT	Short-time Fourier transform

Table 1

Descriptions

Definition	Description
EEGBrowser	A graphical interface for viewing EDF files
EEGLAB	A MATLAB based toolbox for processing and viewing recorded EEG data
Epoch	A partition of a timesignal to be evaluated and/ or classified
OpenBCI	Graphical user interface for EEG data streaming
OpenVIBE	A software platform for designing and testing BCI's

Table 2

1

Introduction

The goal of the project is to develop an Electroencephalogram (EEG) based Brain-Computer Interface (BCI) that utilizes motor execution signals generated in the brain. Given the complexity of the project, it has been logically divided into three components: data collection, machine learning, and the interface. This report will detail the headset used, the choices made on pre-processing, and the process of streaming data to the interface. The machine learning model will extract common features and develop a model capable of accurately predicting left and right motor execution movements. The interface group will dynamically implement training and viewing screens to visualize the EEG signals and play a collection of simple games.

For a comprehensive understanding of this project, it is important to understand the concepts of EEG and BCI. EEG signals, or Electroencephalograms, measure electrical activity in the brain. EEG signals are hard to read as their magnitude is of a few microvolts and must pass through the scalp and hair. Brain-computer interfaces (BCI) are a current research topic where signals from EEG or Magnetoencephalography (MEG) are converted into computer-readable instructions. There exist several BCI-related paradigms such as Motor Imagery (MI), Steady-State Visually Evoked Potentials (SSVEP), or P300 [20]. The chosen paradigm for the report is a derivation of motor imagery called Motor Execution (ME) in which control of a limb evokes an EEG signal. This paper [12] about finding new features in motor-related brain activity explains that motor tasks blocking of ongoing activity in the μ-band (8Hz – 13 Hz) of an EEG record, i.e., Event-Related Desynchronization (ERD) takes place. To further clarify, motor execution is less affected by BCI illiteracy[4], a phenomenon in which test subjects struggle to produce distinguishable signals. Motor execution also produces much more robust and identifiable signals as compared to motor imaging hence the choice for using it[19]. Examples of integrating the EEG paradigms in a BCI would include left and right movements for motor execution. Different EEG Paradigms also have different frequency bands where patterns and movements are most observable. For motor execution, the most common frequency bands that will be considered are the alpha and beta bands, which are frequencies from 8Hz - 12Hz and 12Hz - 30Hz. The moment when a stimulus or task is presented to a user is called a stimulation with expected differences from non-event-related data from varying brain stimulation. For the Motor execution paradigm, expected behaviors include Event-Related Desynchronization (ERD) and Event-Related Synchronization (ERS). A decrease of power from the baseline (ERD) followed by an increase (ERS) should be observed when trials are averaged. EEG signals read by headsets also are especially susceptible to interference with some common sources of artifacts including movements, eye blinks, cardiac movements, muscle movements, and power-line noise. These interference sources reduce the overall signal-to-noise ratio of the signal and therefore need to be further addressed. Muscle movements and cardiac artifacts occur at a frequency range of above 35hz which is not of relevance in the alpha and beta band, the bandpass filters for 8Hz - 30Hz will therefore remove this interference.

The project is conducted using a pre-provided OpenBCI Ultracortex Mark IV headset, which is used in tandem with OpenVIBE to process and collect data to be sent to the Machine learning group. The collected methodologies and EEG signals will be analyzed to determine the quality of our data.

2

Requirements

To ensure the success of the project, a set of requirements has been established to serve as a benchmark. Each subgroup within the project has its own unique set of requirements. To maintain coherence across all subgroups, a set of global requirements has also been defined. The group requirements are defined based on the target audience of gaming users with possible future applications in healthcare. The requirements, which are a mix of qualitative and quantitative will aid in more precisely defining the goals for the project based on the target audience.

2.1. Group requirements

- The delay for the Machine learning model to decision should be less than 0.75s.
- The classification accuracy must be at least 75%.
- The headset to data receiving latency must be less than 0.10s.
- The machine learning and streaming must be done in Python to ensure easy compatibility.
- The final demo must allow for a user to wear the headset and make left/right decisions in a game.
- The final demo must be able to train and fit new user's data to a refitted ML model.
- The group must use an OpenBCI Ultracortex mark IV to stream the read EEG data.

The general requirements are kept simple to ensure that the subgroups have the most ability to implement features while still being able to cohesively integrate.

2.2. Subgroup requirements

- At least 3 testing experiments should be created to best extract EEG data from the test subjects for eye blinks and motor execution.
  - Create repeatable OpenVIBE scenarios that would allow for others to also gather data.
  - Create a testing document to show steps that must be taken for headset setup and data acquisition.
- There must be at minimum n=200 trials for 1 test subject to best work for a single person.
- Electrode layout should be chosen in a manner that spatial patterns for blinks show a 3x increase in signal amplitude over baseline.
- Electrode layout should be chosen in a manner that spatial patterns for motor execution show a 2x increase in signal amplitude over baseline.
- Filter the data from the EEG signal using a bandpass or high pass filter to extract relevant frequency components which should be sent to machine learning.
- The latency of the system from streaming the data from the headset to Python after filtering should be less than 170 ms.

  - The latency for the signal acquisition on Python should be less than 70 ms.
  - The latency for the signal filtering should be less than 100ms.
- The output data of the preprocessing subgroup should be saved in a .EDF file format with respective stimulation annotations.
- Filtering and streaming should be able to be performed on Python or using a Python library to allow for easy integration with the other subgroups.

The goals were internally set as something that was practically achievable by the end of the project, the decisions taken to implement these goals and whether they were achieved will be further elaborated throughout the document.

3

Hardware overview

The implementation of BCI necessitates the use of a device that can capture EEG signals from the brain. This chapter will elaborate on the specific hardware utilized during the project for the experiments, as well as various design design decisions regarding the hardware. A comprehensive overview of all hardware connections is visualized in figure 3.1.

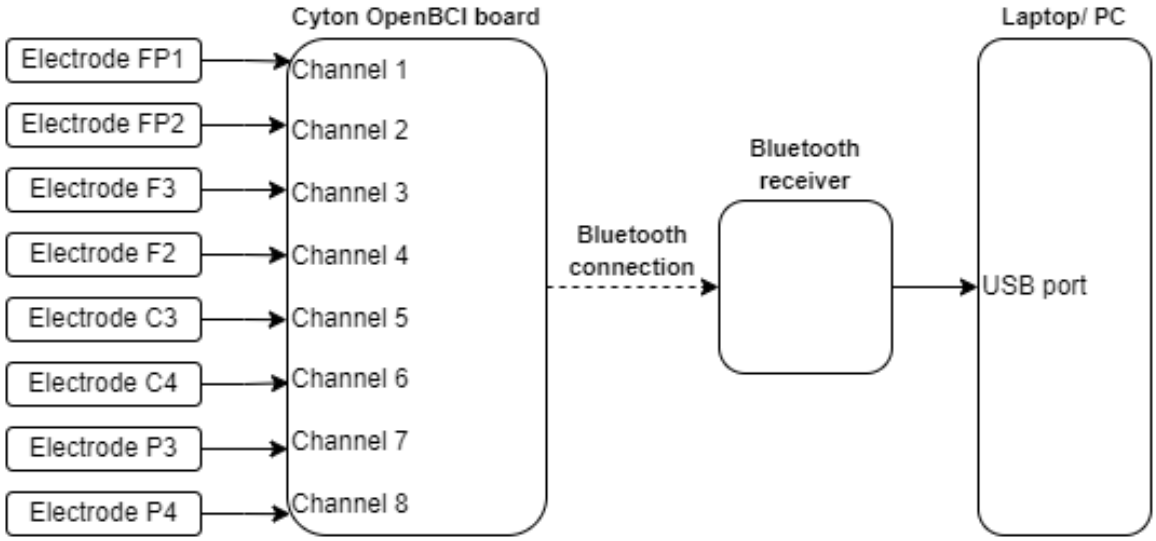


Figure 3.1: Diagram of the hardware connections for an 8-channel cyton Ultracortex Mark IV headset

3.1. Ultracortex

In order to capture EEG signals, our group was provided with an OpenBCI Ultracortex Mark IV headset. The Mark IV uses a Bluetooth connection for connecting wirelessly with a USB Bluetooth receiver. This receiver is plugged into a computer, enabling data reading through either dedicated software that accesses the USB port or custom-coded programs developed in an external coding environment, such as Python.

3.2. Reading EEG Signals

When a large group of neurons in the brain synchronously fire, they produce electric fields. Each electrode consists of 12 spikes, also called combs are made of a conductive material called silver chloride. When an electrode is placed firmly against the scalp, it will pick up the electric field created by

the group of neurons. The signal, albeit weak and typically within the range of a couple of micro-volts, is then sent to a channel of the Cyton OpenBCI Board which will sample the signal with a sampling rate of 256Hz. The cyton board will in turn send the digital signal to a computer for processing. The detailed workings of the board as presented on the OpenBCI website [3].

3.3. Electrode Layout

The Mark IV that the team used can connect a maximum of 8 channels to the electrodes, so the team was tasked to find an optimal electrode layout for an eight-electrode configuration. It is important to base this layout on the paradigm that will be experimented with, in our case motor execution. It must be known what area of the brain activates during a motor execution event. The paper [5] shows experiments with motor execution where a majority of cortical activity is present. It therefore becomes clear that C3 and C4 are the most important sensors for detecting motor execution. FP1 and FP2 are also important placements as these are used to detect blink artifacts [1]. Section 6.2 and section 7.2 elaborate more on blink artifacts and their detection. With these four sensor placements being certain, the other four sensors should be placed strategically for the optimal detection of motor execution. An overlay of the brain activity from paper [5] with the helmet shows approximate locations that could be good for detection. This results however in an asymmetrical layout, while other sources suggest symmetrical brain activity for motor execution. A 2d topographic map of the brain during motor execution is overlaid with a photo of the headset to gauge sensor regions that would likely experience the most cortical activity, given the variance in motor execution trials by subject a rough mapping of areas was used to gather sensor locations. Figure 3.2 confirms that a majority of the activity during motor execution occurs around the C3 and C4 electrodes. This figure is only shown to get a general sense of where the brain is active and should be taken with a grain of salt as this shows a likely 2D representation of the cortex combined with a 3D top-view of the Mark IV. Figure 3.3 then shows the final electrode layout that the team chose for the experiments. Their connections to the channels of the Cyton OpenBCI board are shown in table A.1.

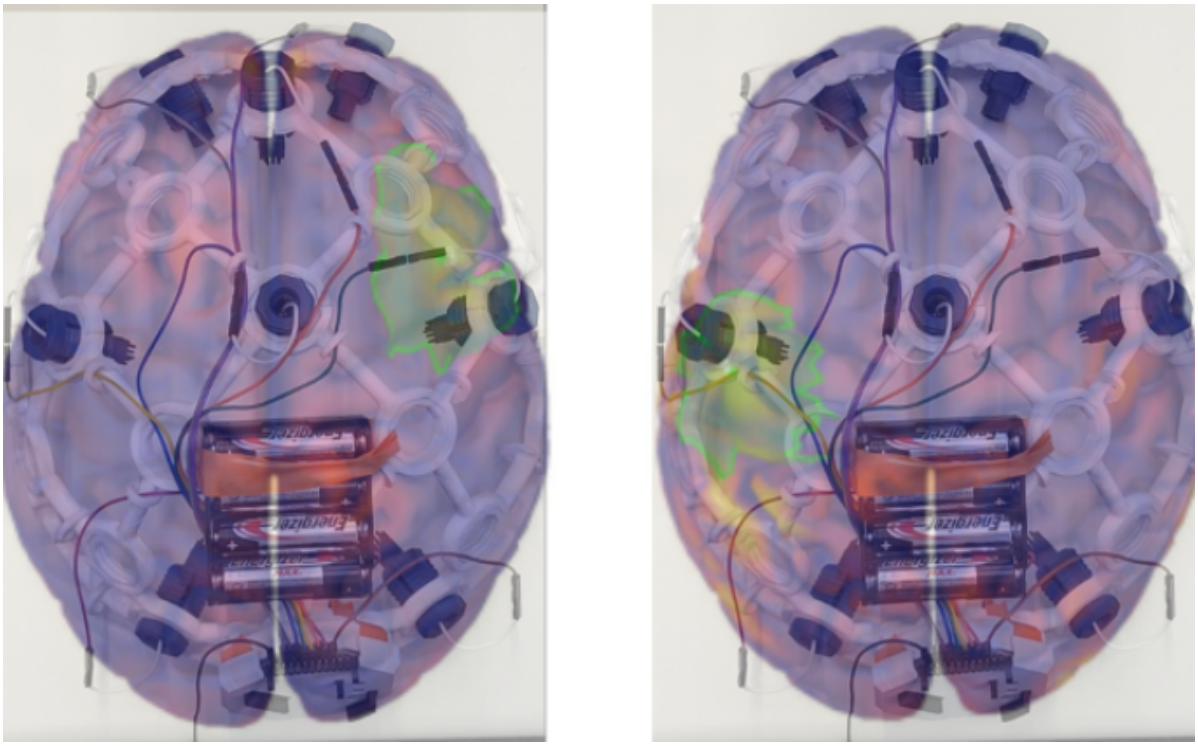


Figure 3.2: An overlay of the brain activity from paper [5] with the Mark IV

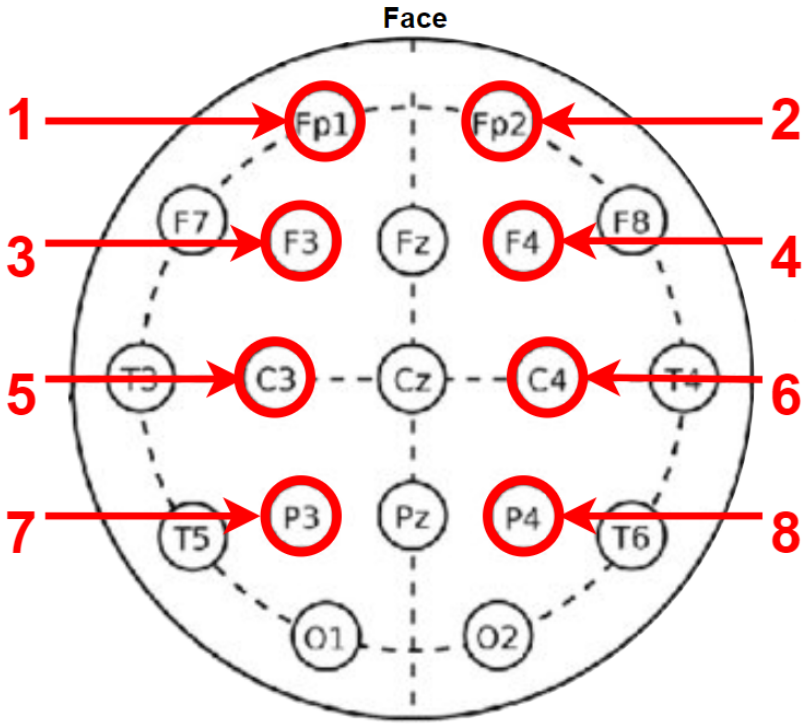


Figure 3.3: The final electrode layout used for the experiments

4

Software overview

Functionality and reliability were a priority in choosing the ideal software for data acquisition. The software should be easy to understand, compatible with the Mark IV headset, and able to export .EDF files to meet the subgroup goal requirements. The ability for the platform to natively export LSL streams would also be beneficial to troubleshoot data streaming-related issues. The choice of used software based on merits is presented below.

4.1. Acquisition

Data acquisition is the primary focus of the subgroup and therefore it was of great importance to use software that is reliable and could export in .edf files. A few standout choices for the data acquisition were BCI2000, OpenBCI GUI, OpenVIBE[7], Python, Matlab, and TMSI Apex. Each of these programs has a set of advantages and disadvantages as shown in table 4.1[17].

Software	Benefits	Drawbacks
BCI2000	• Designed for BCI • Mark 4 headset support • Real-time processing possible	• Limited support/small user base • Difficulty with LSL communication • Questionable reliability
MATLAB	• Live signal processing possible • Familiar MATLAB environment • Able to store to .EDF file	• Headset support is limited • No live signal displays
OpenBCI GUI	• Designed for Mark 4 headset • Live signal view • LSL streams possible	• Cannot write to .edf • Limited compatibility
OpenVIBE	• Works with many headsets • Able to write to .edf • Block-based modelling • LSL streaming possible	• Lackluster signal view • Reliability
Python	• LSL streaming possible • Works with many headsets • Able to write to .edf	• Missing GUI • More difficult setup required
TMSI Apex	• GUI available • Works with many headsets	• LSL streaming difficult • Cannot write to .EDF • More difficult setup required

Table 4.1: Showing benefit vs drawback analysis for all considered data acquisition software

As visible from the table every software had its own set of benefits and drawbacks which meant choosing which software best fit our needs. Software like TSMI Apex, and BCI2000 missed certain functionality and were the least documented software which led us to choose against them. MATLAB was not chosen due to no live signal display capability and OpenBCI GUI was chosen instead used for data viewing. The workflow we desired benefited from simpler modeling and more documented setups which were fit by OpenVIBE perfectly, this is the reason that OpenVIBE was chosen as the data acquisition software for the project. A detailed overview of OpenVIBE and a few relevant boxes has been provided below.

4.1.1. OpenVIBE

OpenVIBE as a software can be split into two main components the acquisition server and designer. The working test setup of acquisition software and designer is shown in the image 4.1 and 4.2 [14]. The acquisition server acts as a bridge to read the signals from the USB port and forward them to the designer. The Mark IV has options to use in both WiFi mode or using 2.4 GHz which both allow data to be streamed, given the chosen 2.4 GHz dongle the acquisition server interfaces with the relevant COM port and retrieves the transmitted signal. The acquisition server also contains many settings to correct for device drift which is an accumulating delay throughout using the headset and identifiers of the test subject which is useful for .EDF trial data sorting. The sample count per block essentially chooses the epochs of the buffer data to be sent. The captured and streamed data is then sent through a virtual IP port which can be further read by the OpenVIBE designer. This is the reason that OpenVIBE was chosen as the data acquisition software for the project.

The OpenVIBE designer on the other hand is a block-based modeling section of OpenVIBE which allows for the setup of hardware-independent measuring and modeling scenarios. Many forms of filtering, processing, and classification functionality are built into the designer. However, the most important aspect is the filtering and stimulation functionality of OpenVIBE. Using the same IP port as the acquisi-

tion server, a scenario was created that applied temporal bandpass filtering with Graz visualization to capture the relevant motor execution signal. The acquisition server box after receiving the data on the IP port outputs many things such as the raw signal, time scale, and units of the signals. The signal is then filtered through an 8-30Hz butter worth bandpass filter as further elaborated in the chapter 6. The filtered signal is displayed using a signal display block along with being sent to the.EDF writer's block. The Graz simulator and visualization are both blocks that run lua scripts to execute a standard Graz testing setup with baseline cross and arrow commands for left and right motor execution[9]. This stimulation is then further written along within the.EDF file as stimulation which is used for further epoching and processing.

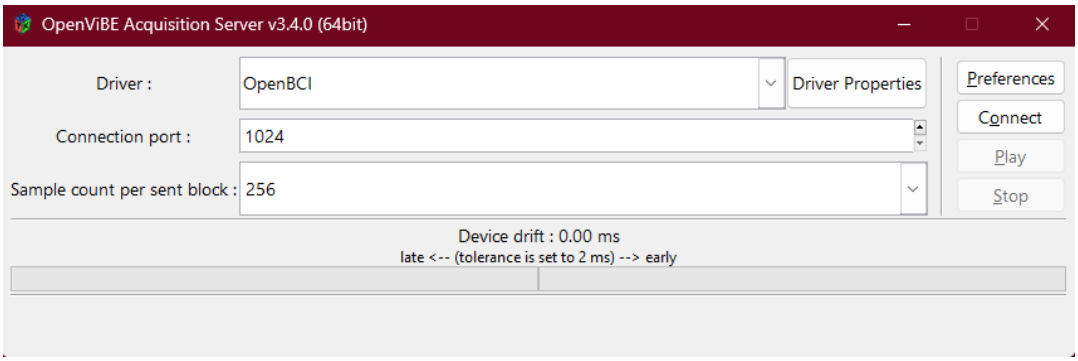


Figure 4.1: Workflow of OpenVIBE acquisition server used to set and initialize communication with the headset

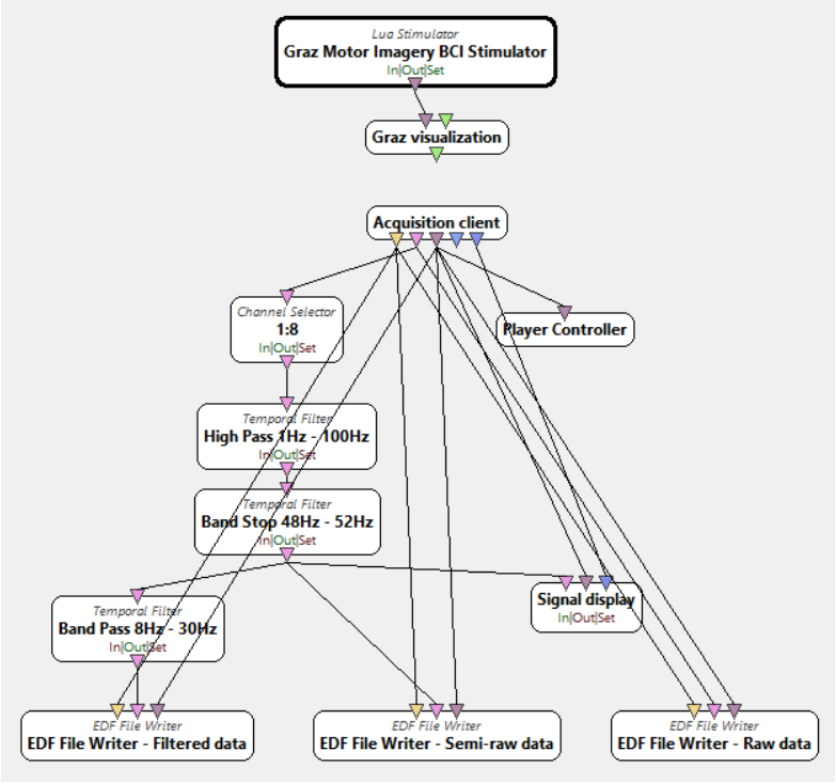


Figure 4.2: Workflow of OpenVIBE designer used to design and model tests to gather data

4.2. Data viewing

While data collection is important, being able to view and inspect the data post-measurements to examine for faults is just as crucial. OpenVIBE as mentioned before severely could not simply inspect

data post-recording with .EDF annotations. The choice for data viewing was therefore split between OpenBCI, EDFbrowser, and EEGLAB with each used for different purposes. The typical workflow of data viewing post-trial measurement is described below.

4.2.1. OpenBCI GUI

OpenBCI GUI is free-to-use compatible software that works natively with OpenBCI headsets with the ability to plot[10], filter, and store EEG signals. OpenBCI GUI has a very simplistic workflow in the fact that it was the most ”plug and play” for visualizing the EEG data streams. The dongle is plugged into the USB port of the computer after which OpenBCI GUI is opened. The start stream button was used to visualize the time domain signal along with an FFT and either a topographic head plot or a 10s interval live spectrogram. The use of OpenBCI is to check whether no periodic interference is observed and no channels are railed. A simple test of motor execution was then performed with special attention paid to the FFT plot where an increase should be noticed, given all the above was done the data collection with OpenVIBE could proceed.

4.2.2. EDF browser

EDF browser is free software that natively can plot EEG signals, apply filters, and simple data visualization. The data loaded in the EDF browser can be viewed split by channel or together with the further ability to change the amplitude and timescale of the signals. The annotation section also allows partial filtering of annotations showing signals after a certain stimulation and a real-time player that plays the signal back in real-time. Filters such as butter-worth bandpass and high pass filters are also applicable to the EEG signal with further simple amplitude or power spectrum displays. The software in the data collection workflow was used to quickly view the signals post-measurement to ensure that the activity did indeed occur after a stimulation. The whole signal was also inspected for anomalies or artifacts that would later need to be removed.

4.2.3. EEGLAB

EEGLAB is a MATLAB-based extension for processing and visualizing EEG data, the software has quite a large scope of abilities with the most relevant sections being blink inspection, topographic maps, and a spectral Event-related Synchronization/Desynchronization (ERDS) analysis. The data is initially loaded where channels are mapped to signals using the channel definition. The data is then further inspected for eye blinks and other prevalent artifacts that would need correction. The native Independent Component Analysis (ICA) decomposition which is a blind source separation technique is then used with a standard run-ICA algorithm to distinguish artifacts from wanted EEG signals. The data if needed is then rejected using the ICA classification section to further remove eye blink activity[1], the plots of the spectral ERSD are finally viewed to ensure whether the components are either not noticeable or of a certain amplitude. The use of the software in our data collection pipeline is to better visualize and remove eye-blink artifacts.

4.3. Data streaming

Finally, the data from the headset will be streamed from the headset to a python environment for further display and use. The LSL framework was chosen due to its ease of use and availability in the previously used software. To ensure further simplicity in implementing the chosen method of data streaming, python is chosen as the platform of implementation.

5

Data Acquisition

The goal of this project is to capture motor execution signals from an EEG headset which entails creating test setups best suited for maximizing usable results. The tests were designed in a method where other experimenters trying to follow this setup will be easily able to replicate it, the analysis and created methods are presented below.

5.1. Factors for Error

Due to the sensitive nature of the EEG systems, there were a multitude of different factors that need to carefully be considered. A further clarification required is that the errors mentioned below are all factors that may cause problems in the EEG signal however are all avoidable, artifacts discussed further in chapter 6.2 refer to processes in the environment or in the human body which are realistically not possible to control such as heartbeat and eye blinks. The factors for error will therefore be further evaluated and are located below.

5.1.1. Movement

Motor execution in theory entails movement of a limb followed by a certain action. The movement during this action could shift the headset which can create unwanted EEG signals. Figure 5.1 below shows the effect of a slight movement of the headset on the EEG signal.

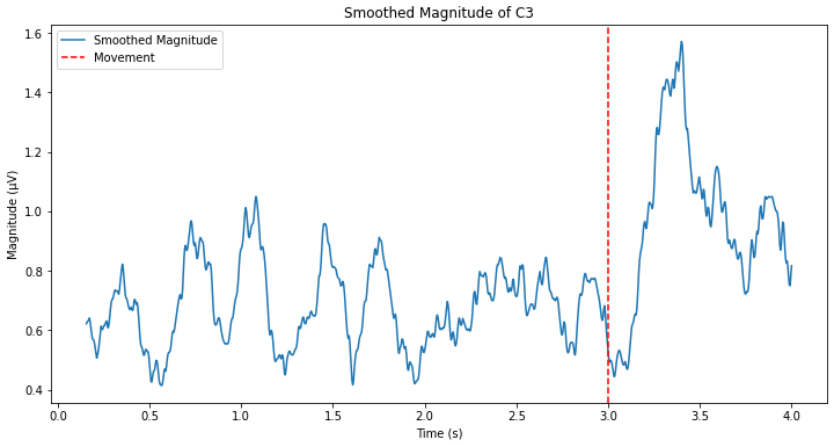


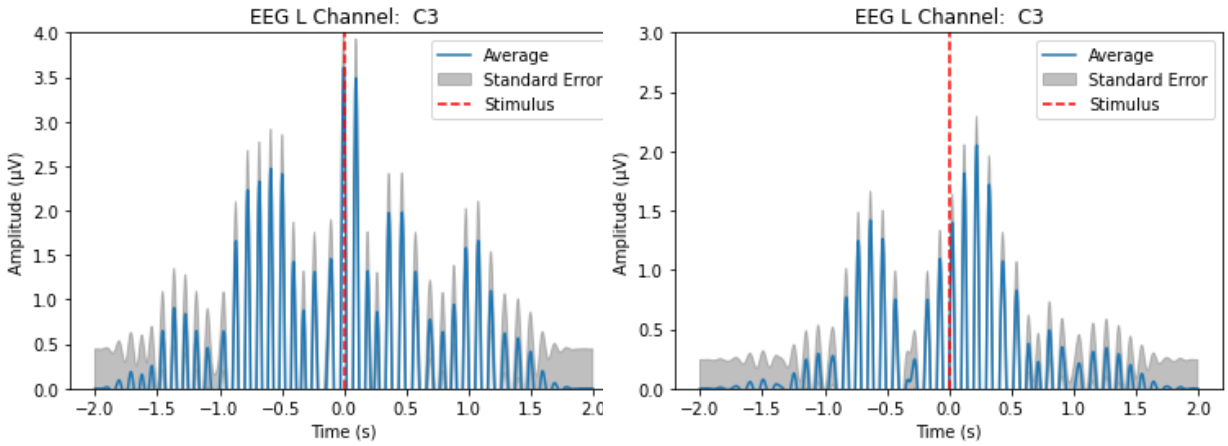
Figure 5.1: Effect of natural movement on an 8Hz - 30Hz filtered squared magnitude EEG signal, the signal is smoothed using a moving average. The code for the plots can be found in Appendix B

When a substantial enough movement was induced, the signal would spike and need at least 5 to 10s to come back to baseline. In an experimental scenario, this is non-ideal as the movement caused an anomaly that would ruin legibility for a certain event with a chance of also a subsequent event. The

anomaly also pertains to all channels meaning that certain channels during the event could not be ignored. The problem with the error is that being so heavily reliant on movement for motor execution signals, there would naturally be movement across the body which is not ideal. To prevent the error a multitude of steps were taken such as changing arm movements and introducing stability electrodes and mechanisms to keep the head still. Instead of moving the whole arm, which would lead to head movements, an active change was made to lock the elbows to a certain point where only the forearms were allowed to move, this ensured that shoulder and head movement would be limited. To further limit the sliding of the headset during movements more electrodes were added at O1, O2, F7, and F8 as non-readable channels where they were not connected to the main board but rather placed as stability electrodes which they were tightly pressed against the head to reduce movement and slip of the headset. The last optimization made to reduce the movement was by ensuring that the test subject pressed their back against a chair with their head and then remained in a neutral position. This also further reduced the risk of random movements.

5.1.2. Concentration

The concentration of the subject also played a major role in the data received from the test subject. While being tested the subject had a limited span of concentration which limited many factors while designing the experiment. The subject was placed in a quiet isolated room with only the Graz visualization present to eliminate any external forms of distraction. During the testing, no people were near the test subject. To ensure proper movement the subject was initially guided to perform the movement with the OpenBCI-GUI to ensure that signals were detected. To visualize the effect of fatigue on the recorded data, two tests were used. The first test was an initial recording of the ”Motor execution basic test collection setup” as shown in figure 5.4 and included a single movement that had to be performed, while the second test was the same test done after 9 previous consecutive recordings. The normalized squared magnitude of the signal is used to visualize the strength of the signal around the epoch. The recording setup initially consisted of filtering between 8Hz - 30Hz and then taking the squared magnitudes of the filtered signal for the epochs, the filtering and calculation of squared magnitude are further defined in chapter 6 and 7 respectively. Figure 5.2 shows the difference in signal squared magnitude as shown below.



(a) Average magnitude of bandpass filtered signal for concentrated subject

(b) Average magnitude of bandpass filtered signal for not concentrated subject

Figure 5.2: Comparison of a for a 5 trial averaged concentrated vs not concentrated subject performing a left trial, the plots are 4s around the stimulation(t=0s) and are filtered and normalized from 8Hz - 30Hz. The code for this plot is available in Appendix B.1

While the plots seem similar the normalized magnitude of the concentrated signal was much higher than the non-concentrated subject. The testing was therefore designed and chosen to have 3 tests followed by breaks of at least 5 to 10 minutes.

5.1.3. Electrode Contact

Differences in subjects such as hair length and hair density play a major role in the quality of the recorded signal. Hair means that the electrodes do not properly contact the scalp which reduces the SNR of the received signal. The spiky electrodes were used in conjecture with the ”railed” indicator on OpenBCI GUI to limit any chances of the headset not contacting the scalp, this did not guarantee errors from a poor connection, however, it was used to mitigate it.

5.2. Referencing

The choice of reference for electrodes was the last important step before designing the test setups and ideally a reference point would be chosen with zero potential to accurately determine the difference in magnitude of the brain signals. A reference point on the human body with zero interference however does not exist and therefore allows for multiple different methods for referencing[16], the OpenBCI headset allows for uni-polar or bi-polar referencing with the reference point being the earlobe. There are also other referencing techniques such as Mastoid referencing and common average referencing which present their unique benefits and drawbacks. Mastoid referencing is a very often used referencing technique where the reference point is the mastoid bone present right behind the ear. The mastoid is often chosen due to the distance from the cerebral cortex, which means that the reference is less likely to be any generated cortical signals, the drawback however for high electrode measure setups or measurements near mastoids result in noise in the reference. Common Average Reference (CAR) entails averaging the signals of all electrodes to get a reference signal which is then further used. The benefit of such a method is that biases on single points have less of an effect on CAR than other reference methods. The drawback however is that average referencing is very sensitive to bad channels and therefore the average reference can easily be shifted[6]. The used test setup was especially susceptible to noise and therefore meant that common average referencing could be used. Mastoid references were also not chosen due to the different hardware needed to attach to a mastoid. The choice of referencing was therefore chosen as bipolar referencing. Re-referencing is a process where a reference method can be changed to better evaluate data, re-referencing can cause increased noise and more distortion which is prevented by recording data with more electrodes An important note is that signals recorded could always be re-referenced later using software such as EEGLAB allowing for better data extraction.

5.3. Test Methodologies

Given the previously made optimizations, test methodologies were chosen to accommodate these factors and to ensure the best chance of getting a good motor execution signal while having the highest possible SNR. The testing methodology for both blink detection and motor execution is shown and further explained below.

5.3.1. Blink testing

The blink testing was done as an initial setup for testing whether EEG signals were truly measurable by the headset. Blinks being a very easily detectable artifact in EEG systems meant that it formed a logical start for testing to ensure that EEG signals with the relevant pattern were observable. The main power in a blink signal lies in the delta band being from 0Hz to 4Hz. The filter that was therefore chosen was a combination of a 5th order IIR high pass 1Hz filter along with a power line suppressing 48Hz - 52Hz notch filter. This allowed for the removal of DC components while maintaining the main power in the blink signal. The filters were also chosen at 1Hz as this was the lowest possible frequency where a majority of the DC component was still removed. Further in testing the trial was kept to a maximum of two minutes as we found that it was the maximum time of continued focus for a test subject. The experiment consisted of a 10s start window where all the EEG headset channels would calibrate and settle to a baseline. This was then followed by a 5s window of the subject preparing for the experiment. There is then a stimulation every 5s through an auditory system of a beep playing on a speaker till the time of 120s when the test ends. Figure 5.3 shows the visualized test setup.

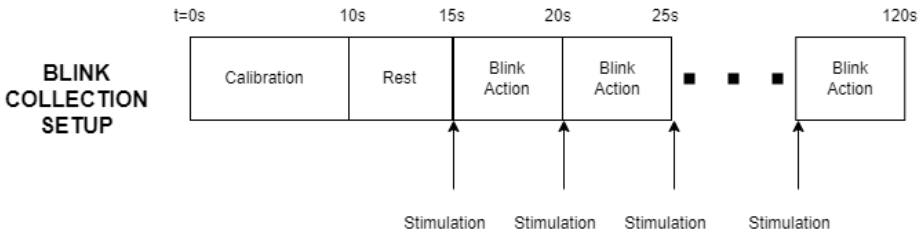


Figure 5.3: Test setup designed to capture blinks with a stimulation every 5s after 15s of initial calibration

5.3.2. ME testing

The motor execution testing was split into 2 relevant phases with the first being a test to see whether motor execution was detectable through the headset and the second to gather data. Motor execution as described prior is much easier to detect than its alternative motor imaging, however, the initial unfamiliarity with the EEG headsets and the relatively small magnitudes of EEG signals caused doubt on whether it was possible to detect and separate visually between left and right trials. The first test was designed in conjecture with a bandpass filter from 8Hz to 30Hz which encompassed the alpha and beta bands, this frequency band also adequately removed the large DC component and power line noise. A Graz test was used which is a standardized testing system for motor imagery and motor execution, the test consists of a cross on the screen followed by arrows pointing left and right. The Graz test for all motor execution trials was slightly adjusted to increase the time for acting and the rest time in between with the diagram of the relevant Graz setup presented in Appendix A.3. The initial test started with the standard 10s of calibration followed by 5s of rest. One single stimulation at 15s was presented and the test ended at 20s. The test was kept simple as the goal was to see frequency domain activity in the relevant alpha frequency band. Figure 5.4 shows the first test setup named ”Motor execution basic test collection setup”. The second test was designed to keep close to the 2 minutes and a mix of 5 left and right trials would be conducted. To prepare for the experiment 30s was designated for calibration and a further 3.5s for rest. A stimulation would then be shown every 10 seconds until 133.5s. The testing was chosen in such a manner to ensure that the participant could remain focused while gathering the highest amount of trial data per test possible. Figure 5.4 also shows the second test setup named ”Motor execution collection setup”.

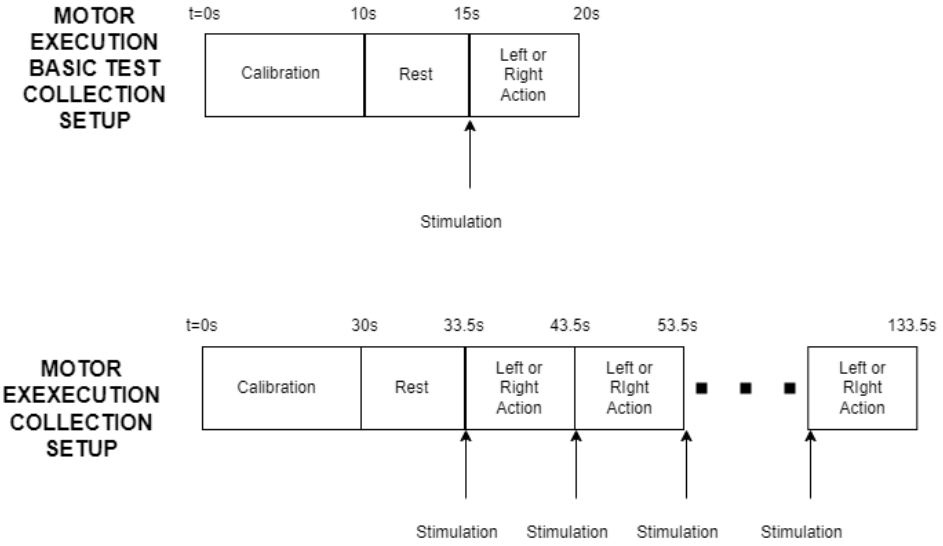


Figure 5.4: Test setups designed to capture motor execution signals with a 1 stimulation for the basic test and a stimulation every 10s after 33.5s of calibration

6

Preprocessing

The machine learning module requires both raw and filtered data for the classification of its model. Raw data itself is not useful as it contains DC components and noise from the power line and thus requires several preprocessing procedures to make analysis possible and classification easier. This will be referred to as the semi-raw data signal. The filtered data requires a few additional preprocessing steps. This chapter describes all the necessary steps of manipulating the data to achieve useful semi-raw and filtered data. Figure 6.1 shows a block diagram of the preprocessing pipeline.

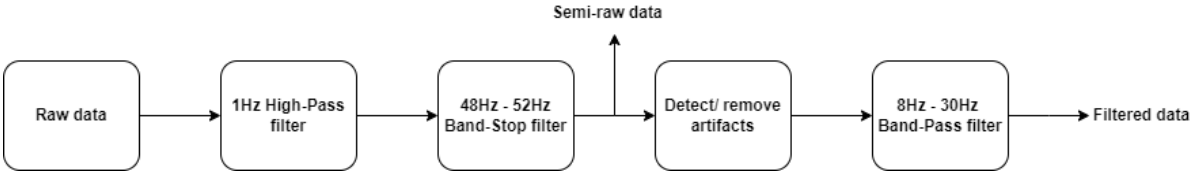


Figure 6.1: Block diagram of the EEG signal preprocessing pipeline to produce semi-raw and filtered signals

6.1. Filtering

Filtering refers in this case to the process of isolating frequencies using a high-pass, low-pass, band-pass, or band-stop filter. The team filters the signal mainly using the OpenVIBE software, which offers the option to use Butterworth or Chebyshev filter types.

6.1.1. Filterting mathematics

Mathematically a random filtering process is easiest explained as windowing in the frequency domain. One may use a Fast Fourier Transform (FFT) function to convert the filtered signal to a function that shows the amplitude of frequency components in the original signal. The equation for the Fourier transform is shown in equation 6.1. Keep in mind that this equation shows the Fourier transformation of the continuous time-domain signal, yet we will transform the sampled version of this signal which is discrete. Also, keep in mind that the FFT algorithm differs from the actual mathematical function, yet the process results in the roughly same outcome.

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi f t}\, dt \tag{6.1}$$

Similarly to equation 6.1, the signal can be converted back to a time domain signal using the inverse Fourier transform shown in equation 6.2.

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{i2\pi f t}\, df \tag{6.2}$$

In case certain frequencies must be isolated, the frequency-dependent signal could simply be multiplied by a transfer function that magnifies the desired frequencies and suppresses the other frequencies. This process is shown in figure 6.2. The multiplication in the frequency domain translates to a convolution in the time domain, which is mathematically shown in equations 6.3.

$$\begin{aligned} Y(f) &= X(f)G(f) \\ y(t) &= x(t) * g(t) = \int_{-\infty}^{\infty} x(\tau)g(t-\tau)\,dt \end{aligned} \tag{6.3}$$

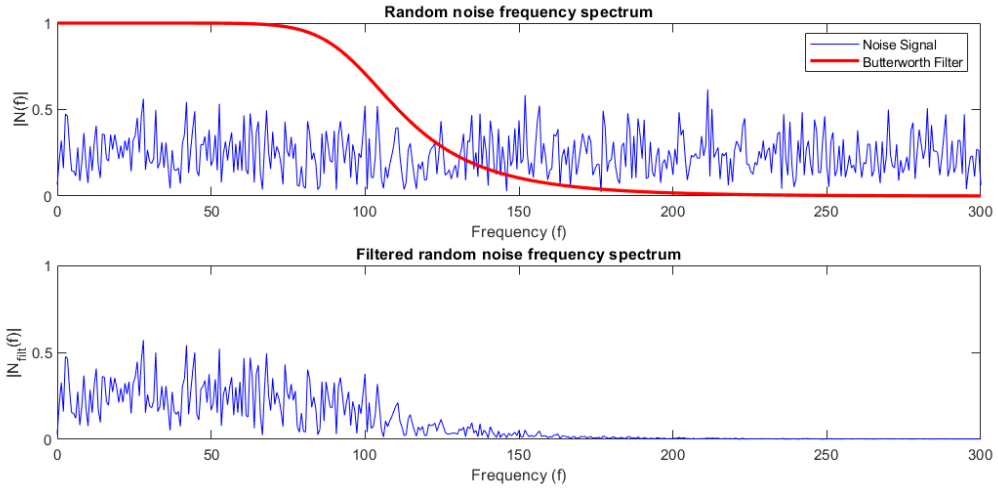


Figure 6.2: Example of a 5th-order Butterworth filter which is filtering random noise

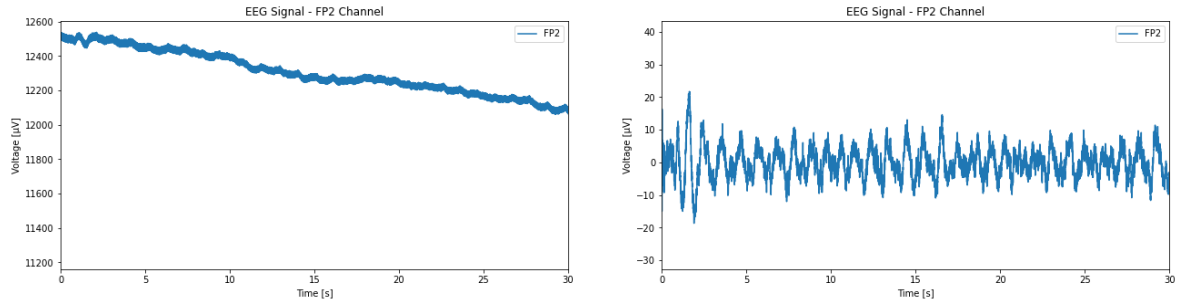
Depending on the desired output, different transfer functions can be used as filters. As mentioned before, OpenVIBE offers transfer functions for Chebyshev and Butterworth. These are mathematically represented with the transfer function in equations 6.4. Note that both equations allow the variables n and ω_0 to be chosen. The variable n represents the order of the filter, and ω_0 represents the cutoff frequency. The cutoff frequency should be chosen depending on the desired structure of the filter, as it defines where the signal starts or stops being suppressed. This varies per application of each filter. The order of the filter was chosen to be 5th order, as this was the maximum order that the OpenVIBE software allowed.

$$G_n(\omega) = \frac{1}{\sqrt{1+(\omega/\omega_0)^{2n}}} \quad \text{(Butterworth)} \qquad G_n(\omega) = \frac{1}{\sqrt{1+\epsilon^2 T_n^2(\omega/\omega_0)}} \quad \text{(Chebyshev)} \tag{6.4}$$

All filtering was digitally performed in the OpenVIBE environment. More information about the characteristics of the filters that this software offers can be found on the website [7].

Semi-raw signal

The raw data is not useful without having performed mainly two preprocessing steps. First of all the DC components of the signal must be removed as the signal will otherwise shoot up or down without oscillating around a center point. This phenomenon is shown in figure 6.3. The team used a high-pass filter with a cutoff frequency of 1Hz to remove these DC components. The other important preprocessing step regarding the raw data is to remove any interference coming from the electric fields created by nearby power lines. As the team's experiments were done in the Netherlands, which uses a frequency of 50Hz on the power lines, a notch (Band-stop) filter was added with 48Hz - 52Hz as the cutoff frequencies to remove any interference coming from the electric fields created by the power lines.



(a) With DC components

(b) Without DC components

Figure 6.3: Raw signal of channel FP2 with and without DC components

Filtered signal

The alterations made to the raw signal for it to become a useful filtered signal are that the artifacts should be removed from the data and that the irrelevant frequencies should be removed from the signal. The team is mostly interested in the alpha and beta waves of the EEG signal as the ME paradigm is significantly more active in these active frequencies. The alpha waves are the waves with frequencies between 8Hz - 12Hz and the beta waves correspond to the frequencies between 12Hz - 30 Hz. The team decided to use a Band-Pass filter of 8Hz - 30Hz.

Some artifacts have a relatively low frequency of <5Hz. Measurements showed that blink artifacts for instance are most prominently present in these lower frequencies. This is shown in the Event-related Spectral Perturbations (ERSP) of one of the blink experiments shown in figure 7.1. It would be inconvenient to try and detect these artifacts for removal after filtering away most low frequencies with the previously mentioned Band-pass filter of 8Hz - 30Hz. For this reason, it is wise to remove the artifacts before band-filtering irrelevant frequencies. The process of the removal of the artifacts is elaborated on in section 6.2.

6.2. Artifact removal

Several artifacts are present in most EEG data streams that should be removed for a more optimal signal, likely resulting in a higher accuracy of the machine learning model. Some examples of these artifacts are blink artifacts, cardiac artifacts, and muscular artifacts.

Blink artifacts

One very common type of artifact that will almost certainly show up in every EEG dataset is an artifact created by the subject blinking. A blink artifact is generated by the upward deviation of the eyes and the closing of the lid. This phenomenon is called the 'Bells phenomenon'. A more in-depth study [8] shows these artifacts appearing in EEG readings for different eye movements and closings.

Blink artifacts have a distinct look in the time and frequency domain. The time domain signal of several blinks is shown in figure 6.4. Due to the filtering already removing most of the frequency components of a blink artifact, the team has decided not to implement a full removal of these artifacts due to time constraints. The team has however devised several ways to try and remove this artifact fully.

The blink is most notably present in the FP1 and FP2 sensors, located closest to the eyes. The location of these sensors is shown in figure 3.3. The blink artifacts will be prominently present in the data of these sensors and it is therefore easy to detect an event. One could simply ignore the epoch in which this artifact is present. This removes the ability to detect an ME event during the ignored epoch, but it also rids the data of every single false positive that might be triggered due to a blink event. This trade-off will be worth it in the team's opinion. This method is most likely the easiest way to remove a blink artifact.

Another method to remove a blink artifact would be to suppress the amplitude of the signal solely during a blink event. This way there is no need to ignore an entire epoch. One could guess the length of a blink event by having a standard duration of the signal being suppressed in case a blink is detected.

Since not every blink event will have a duration, an even better solution will be to use machine learning to detect the specific duration of a blink, isolating and suppressing solely the blink event of the signal.

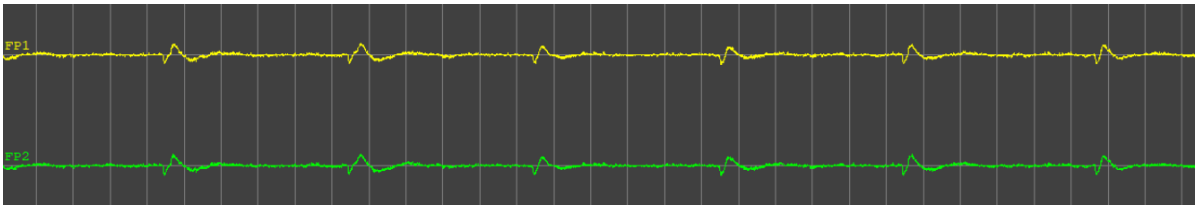


Figure 6.4: Measured raw datasignal from FP1 and FP2 during several blink events

Muscular artifacts

An artifact one may encounter in their EEG data is a muscular artifact. Muscular artifacts arise from the movement of muscles under the skin that may be close to an EEG sensor. It is known to be difficult to remove. The paper [2] describes how a muscle movement can cause interference in the EEG readings, and describes how primarily the clenching of the jaw will be visible on the EEG data. The team has not noticed any significant muscular artifacts present in their data and it is therefore decided not to spend time removing these artifacts from the data.

Cardiac artifacts

Finally, an artifact the team looked at to potentially remove from the data is a cardiac artifact. These artifacts arise from the pulsation of the heart being picked up in the data due to a sensor placed on the scalp being close to a vein under the skin. The team encountered many oscillatory interference in the data, some of which were thought to be cardiac artifacts. These oscillatory interferences will be elaborated on in chapter 9.

The team's solution to getting rid of cardiac interference would be to move the Mark IV slightly on the scalp to get the sensor with interference away from the vein that is picking up this artifact. Another method the team discussed to remove this artifact is to filter out the frequency components that are expected to come with a cardiac artifact. A logical thought would be that a cardiac signal is a periodic signal averaged between 60bpm - 100bpm, or 1Hz - 1.67Hz for a resting heart. These frequencies could be easily removed by moving up the cutoff from the high pass filter, mentioned in section 6.1, from 1Hz to 2Hz. It is worth mentioning that the filtered signal would not need this adjustment to remove these frequencies as this signal has been Band-filtered from 8Hz-30Hz. The paper [13] however states "*cardiac or electrocardiogram (ECG) artifact is one of the most devastating physiological artifacts, which arises due to heart movements and is characterized by the presence of highly energetic quasi-periodic QRS complex in the frequency range of 5–22.5 Hz*". These frequencies are impossible to remove without a very complex solution. The team decided it was not worth the time and effort to try and remove the artifacts in these frequency ranges.

7

Data Analysis

7.1. Background knowledge

The data analysis performed on the measurements requires some background knowledge. For standard reference, the table 7.1 below shows the mapping between channel number and name.

Channel Number	Channel Name
1	FP1
2	FP2
3	F3
4	F2
5	C3
6	C4
7	P3
8	P4

Table 7.1: Showing a table mapping channel number to channel names

Mathematical Descriptions

To gain a grasp of the visualization a little mathematical background is needed in time and frequency domain analysis techniques, the mathematical descriptions although mentioned briefly are used to visualize and therefore analyze the recorded signal. The mathematical description of the Short-time Fourier transform (STFT), power normalization, signal envelopes, and statistical analysis have been shown below.

The Fourier transform (FT) is a mathematical tool that is used to decompose a signal into all the frequency components that exist inside it, Fourier transforms are in essence time to frequency domain transforms that allow for more signal processing and further frequency domain visualization. The mathematical description of the Fourier transform is given below.

$$FT\{x(t)\} = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt$$

The equation above shows the FT with x(t) representing the signal in the time domain. While the FT is a useful analysis tool to discern frequency components, it also results in a loss of time information of the signal. Given the nature of EEG signals acting as non-stationary signals, a different related transform is used. The short-time Fourier transform (STFT) is a method to determine the sinusoidal frequency and phase of small sections of a signal as it changes over time. The Short-Time Fourier Transform

(STFT) is defined as shown below.

$$STFT\{x(t)\} = \int_{-\infty}^{\infty} x(\tau)w(t-\tau)e^{-j2\pi f\tau}d\tau \tag{7.1}$$

The equation above represents the STFT with x(t) representing the signal and w(t-τ) as the window function which is either a non-zero Gaussian or sine window for a small interval of time. Another technique used to analyze and visualize EEG signals is the Power Spectral Density PSD and normalization. The PSD of a signal is the amount of power present as a function of frequency. The PSD is usually calculated through the FT or one of its variants and is used to identify the dominant frequencies in a signal. While not in explicit scope for this paper, the Welches method is used to compute the PSD of finite-length signals. Normalization is then used to compare changes in frequency in a signal concerning a baseline. The equation for normalizing is presented below.

$$P_{norm}(f,t) = P(f,t) - \frac{1}{T_{Baseline}}\int_{t_{start}}^{t_{end}} P(f,t)dt \tag{7.2}$$

Some statistical tests are also used in Event-Related Spectral Perturbation (ERSP) such as paired t-test and Wilcoxon signed-rank test. The statistical tests are used to determine whether events observed in the signal are statistically significant. The tests compare two related samples from the same group where based on the distribution one of the two tests is used. A paired t-test is used for symmetric distributions around a mean and the Wilcoxon signed-rank test is used the rest of the time. The focus of the essay lies less on understanding the complete workings of statistical analysis, however, for the sake of completeness both formulas are provided below.

$$t = \frac{\bar{d}}{s_d/\sqrt{n}} \tag{7.3}$$

The Wilcoxon signed-rank test statistic W can be further calculated as shown below,

$$W = \sum_{i=1}^{n} sgn(x_i - y_i) \cdot rank(|x_i - y_i|) \tag{7.4}$$

Finally, the Hilbert transform is used to generate an analytical signal consisting of the signal as its real part and the Hilbert transform as its imaginary part. For a real signal $x(t)$, the corresponding analytical signal $x_a(t)$ is given below.

$$x_a(t) = x(t) + j \cdot \mathcal{H}[x(t)]$$

where $\mathcal{H}[x(t)]$ is the Hilbert transform of $x(t)$, and j is the imaginary unit. The Hilbert transform also shifts the phase of the original signal by 90 degrees. The Hilbert transform is then used to calculate the envelope of the signal which is then given by the magnitude of the analytical signal. The envelope is used to identify oscillatory activities while neatly visualizing changing patterns in a signal. The equation below shows the envelope of a real-time domain signal.

$$E(t) = |x_a(t)| = \sqrt{x(t)^2 + \mathcal{H}[x(t)]^2}$$

7.2. Results

As mentioned in chapter 5 the first experiments were done in order of increasing complexity. The simple blink testing was done initially to check whether the Mark IV headset could read signals. Blinks as mentioned before are also very easy to see in data. Motor execution was then tested with the main goal of seeing expected time-frequency spatial patterns.

7.2.1. Latency

The latency of the filters and system was determined using a simple Python time function where time was stored before running the code and was stopped after relevant Python filtering. The code for the time is present in B.1 with a screenshot of the delay captured and plotted in A.3.1. The delay from headset to filtered signal as presented by Python is 173.675 ms which is within 2% of the subgroup-defined goal

7.2.2. Blink Experiments

The experiment for blinking entailed blinking every 5s after an initial 15s for 120s total. Blinking artifacts occur due to the bells phenomenon as shown in [8] where the strongest visible peaks should have been on channel 1 (FP1) or 2 (FP2) with electrodes further from the eyes presenting weaker to nonexistent signals. The amplitude time plots are presented in 6.4.

In the channels, it is visible that the strongest activity occurs on channels 1 (FP1) and 2 (FP2) with almost no activity on channels 5 (C3), 6 (C4), 7 (P3), and 8 (P4). The positional weakening of the electrode signal from the forehead intuitively makes sense and shows that the different electrodes on the headset can read different signals in the brain spatially. As stated before blinks occur at quite a low frequency with most of the signal power component between 0Hz and 5 Hz. An EEG processing technique known as ERSP is used where the STFT of the normalized signal around epochs is plotted along with a PSD, the ERSP also uses an Independent Component Analysis (ICA) algorithm to plot increases and decreases over baseline power. The visualized ERSP of the signal is presented in the center, PSD on the left, and power concerning baseline shown at the bottom, are presented in figure 7.1 below.

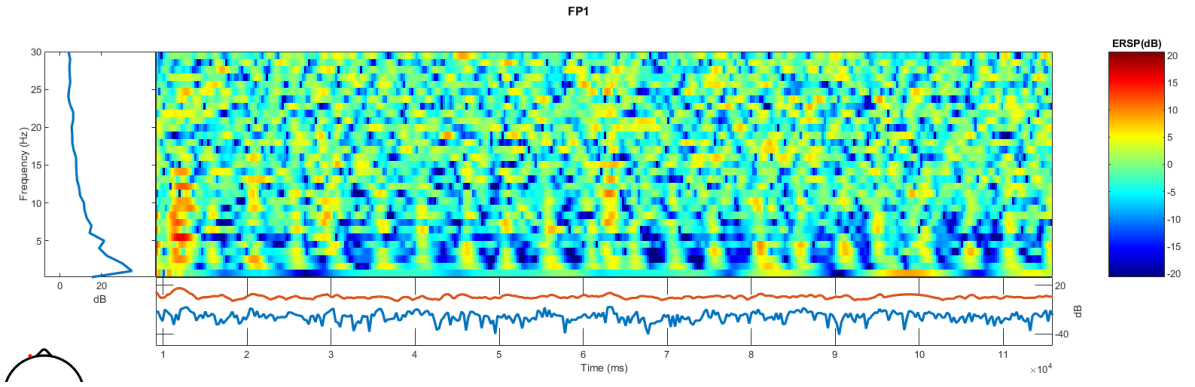


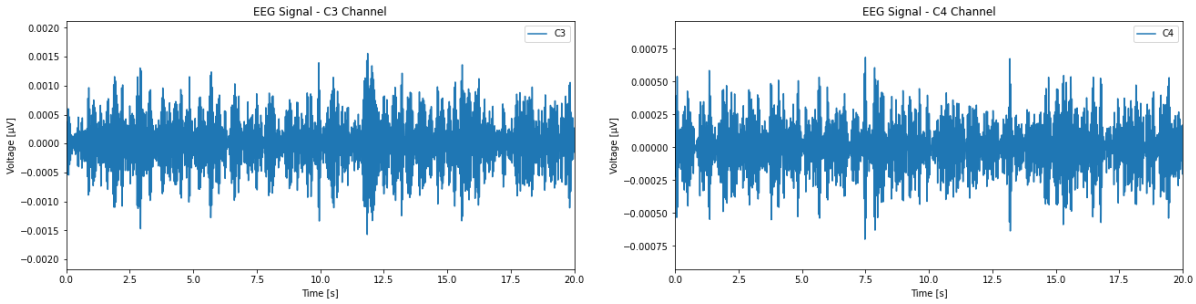
Figure 7.1: ERSP analysis plot of blinks performed by EEGLAB shown from 0Hz to 30Hz with a count of 400 samples

It can again quite easily be seen that the majority power of the signal lies in the delta frequency band (0.1Hz - 3.5Hz) and that the plot shows a periodic spectral activity due to the experiment of blinking every 5s. Theoretically, a filter of sufficiently high enough pass band and slope can result in all the effects being mitigated. The blinks also don't much affect channels 5 (C3) and 6 (C4) where cortical activity would be the highest for motor execution, this leads to the conclusion that with the preprocessing mentioned in chapter 6, blink artifacts would not have a major effect on motor execution.

7.2.3. Motor execution experiments

The subsequent experiment after blinking was detecting signal patterns of ERD/ERS commonly found during motor execution. The general testing consisted of distinguishing left and right movements in the alpha and beta bands (8Hz - 30Hz) of the signal. The goal of the section is to visualize whether motor execution signals are visible in time or frequency domains and whether there was a sufficient distinction between left and right trials. A great example of the results that would be expected is shown in the paper [15]. This paper shows a clear distinction between left and right movement with the ERS and ERD nicely visualized. The activities around trials were used to determine epochs and find the relevant aspects of a signal. An epoching setup of 2s before the epoch and 2s after the epochs was considered as it provided the most clarity for plots in most scenarios. The plot 7.2 shows the amplitude

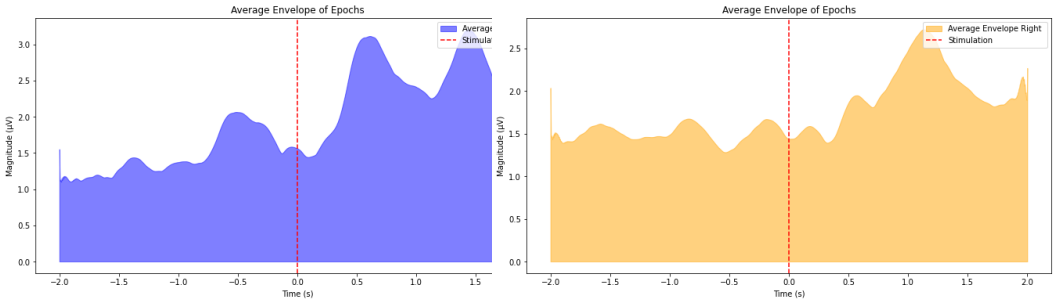
of a signal in channel C3 and its overall PSD has been placed below.



(a) Plotted channel C3 **(b)** Plotted channel C4

Figure 7.2: Comparison of a plotted time domain signal for a motor execution trial between channels C3 and C4 with an applied bandpass filter of 8Hz - 30Hz and between time t=0s and t=20s. The code for this plot is available in appendix B.1

The filtered signal in itself does not show any clear indication that motor execution signals are present. The normalized squared magnitude of the signal is therefore used to distinguish any sort of peaks or troughs that occur in the time domain. An average of all signals for each channel during the period of the epoch is used to calculate the normalized squared magnitude signal. By using the time before the stimulation as a baseline the plot should show clear ERS or ERD patterns. The plot for squared magnitude is presented in the appendix A.4. To better visualize trends in the squared magnitude, an envelope is used. The relevant procedure is determining the squared magnitude plot for all the relevant epochs where they would be time domain averaged and then the envelope would be determined. The plot 7.3 below shows the envelope of all the averaged left and right epochs respectively for 150 trials.

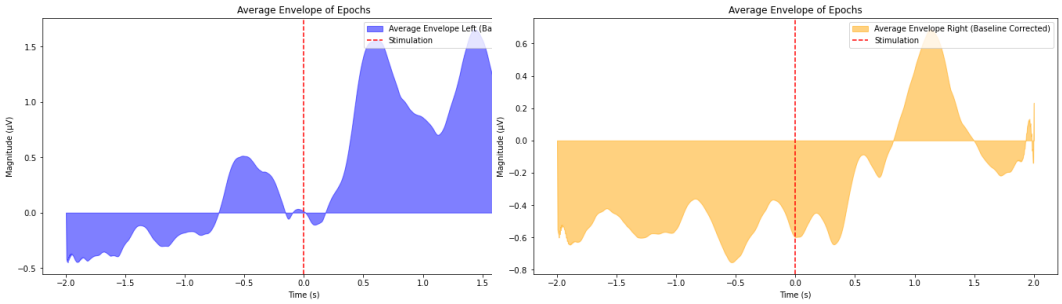


(a) Average Envelope of left trials channel 5 (C3) **(b)** Average Envelope of right trials channel 5 (C3)

Figure 7.3: Comparison of average envelopes of squared magnitudes for left and right for n=250 trials for channel 5 (C3). The code for this plot is available in appendix B.1

While a peak can be observed in the envelopes it is not very distinct and it does not show a good increase from the baseline period, this is corrected by using a baseline corrected squared magnitude where increases are respective to the value of a baseline value or in this case an average of a small time interval of 0.1s at the start of the epochs. The plotted average baseline corrected envelope of n=150 trials is presented in figure 7.4 below.

The average baseline corrected envelope shows a very distinct ERS behavior with a peak of 0.52 and 1.25 seconds after stimulation for left and right respectively. This is not an expected behavior that should be observed with n=150 trials and could indicate a systematic error or unknown and unexpected behaviors. There also hardly exists any ERD in the signal which would indicate that the ERS is much stronger than the ERD for both trials in channel C3. channel C4 also exhibited similar behavior across the n=150 trials. Another interesting point was that the plot for the right trials always lagged the left by about 0.73s which was highly unlikely, a delay of about 0.3s for both trials is expected due to inherit reaction times of the subject where they approximately occur at the same time after the stimulation. While some characteristics of ERD and ERS are visible in the averaged and normalized magnitude the only feature that shows distinctions is the time of peak occurrence after the stimulation, provided more than n=150 trials for left and right the result is likely statistically significant. The averaged PSD is also

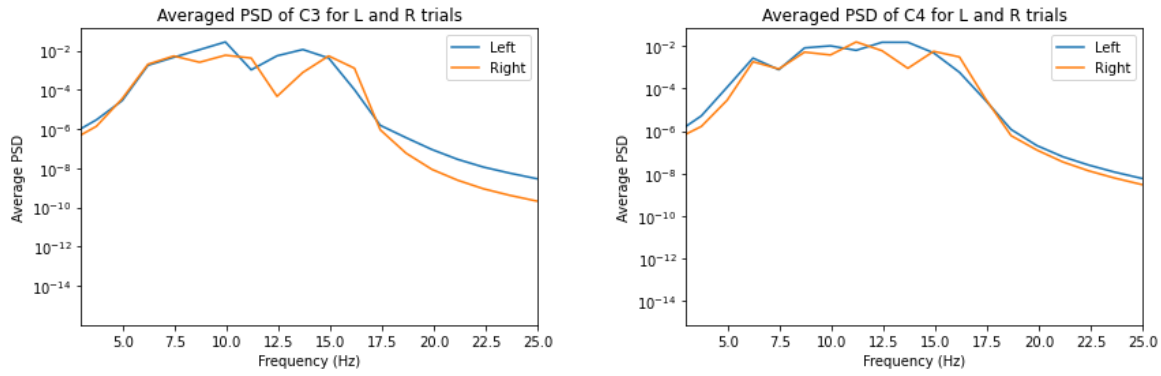


(a) Average baseline corrected envelope of left trials on channel 5 (C3)

(b) Average baseline corrected envelope of right trials on channel 5 (C3)

Figure 7.4: Comparison of filtered(8Hz - 30Hz) and averaged baseline corrected envelopes on channel 5 (C3) for n=250 left and right trials. The code for this plot is available in appendix B.1

used to check and see whether any ERS and ERD patterns are visible in the frequency domain. The PSD of the filtered signal is calculated and it is then averaged to show power frequency relationships across all epochs. The graph was averaged across 150 trials and was normalized with the baseline being used from the signal before the stimulation. The PSD of channels 5 (C3) and 6 (C4) are plotted for averaged left and right trials as shown in figure 7.5.



(a) Average PSD of channel 5 (C3)

(b) Average PSD of channel 6 (C4)

Figure 7.5: Comparison of filtered(8Hz - 15Hz) and averaged PSD between channel 5 (C3) and 6 (C4) for n=150 left and right trials. The code for this plot is available in appendix B.1

As seen from the plots above, the averaged PSD across the epoch does not seem to exhibit any major differences for left and right trials across any of the recorded channels. The peak of the PSD seems to lie around 10Hz for channel 5 (C3) and 11Hz for channel 6 (C4). The PSD therefore does not show significant separation between left and right trials which is quite unfortunate. A further method of visualization to distinguish left and right movement is ERDS plots. The ERDS plot is in essence a form a spectral estimation of the STFT and is used to visualize the change of frequencies over time with a focus on seeing regions of activity (ERS) and regions with negative activity (ERD). The paper [11] has a great explanation that goes more in-depth on the basic principles of ERD and ERS. The settings used for the ERDS plot included using a 0.5s hamming window and 80% overlap as these produced the most clear visualizations. These specific settings were used as they provided the best time resolution while sufficiently separating frequencies in the concerned frequency band of 8Hz -30Hz. The ERDS plot of both averaged left and right trials are shown in figures 7.6 and 7.7 below.

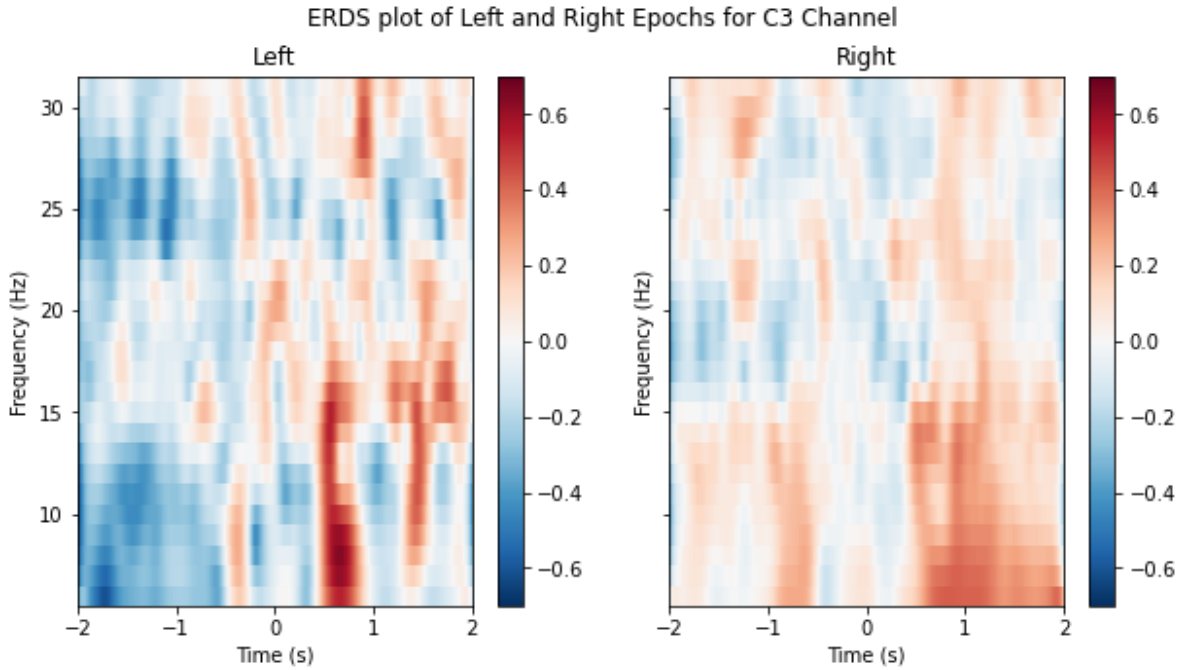


Figure 7.6: ERDS plot of filtered(8Hz - 30hz) and averaged epoch signals across n=150 L and R trials on channel 5 (C3). The code for this plot is available in appendix B.1

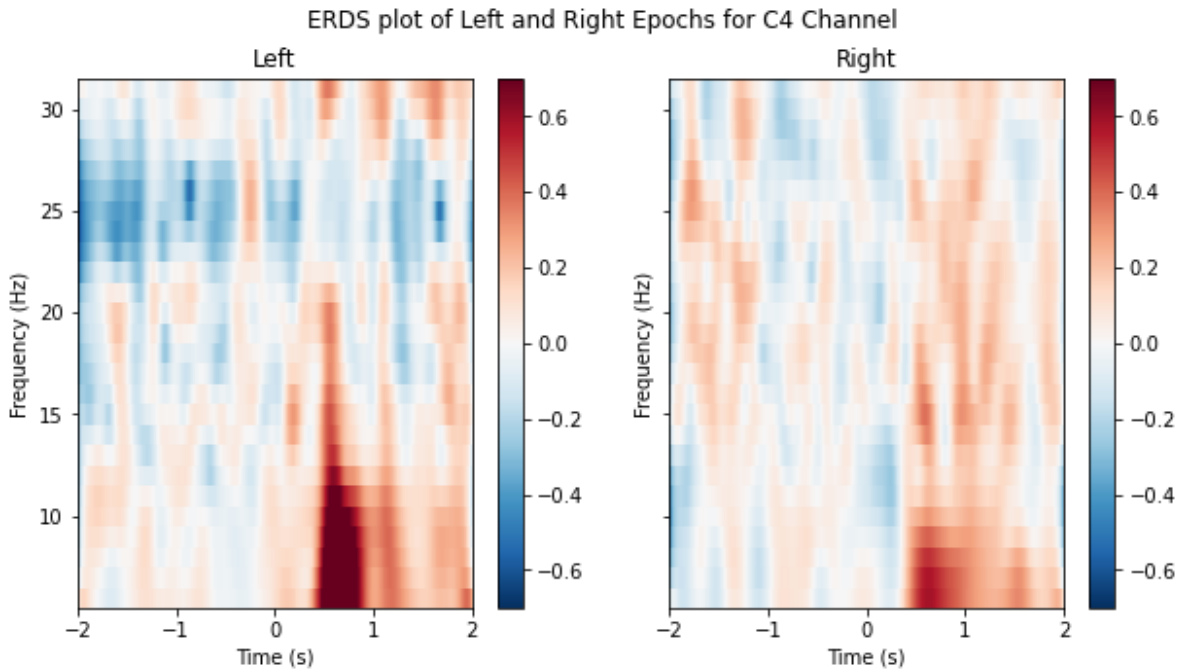
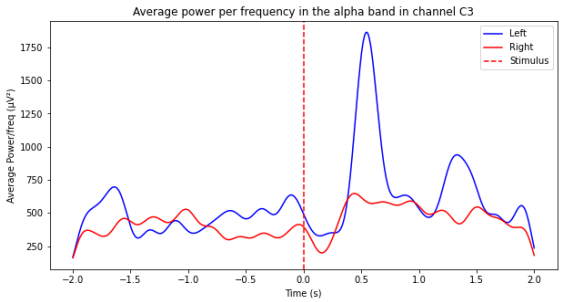
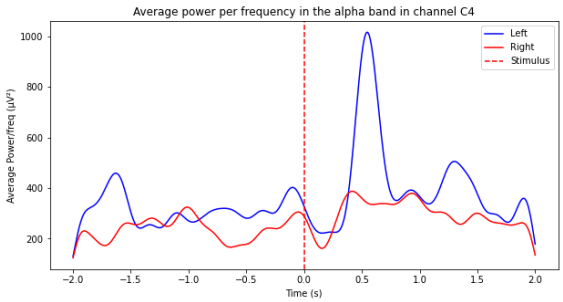


Figure 7.7: ERDS plot of filtered(8Hz - 30Hz) and averaged epoch signals across n=150 L and R trials on channel 6 (C4). The code for this plot is available in appendix B.1

The plots show ERD/ERS activity but to very different extents, with the color bar referring to a change in power from baseline in dB hence a dark red area with a value of 0.7 represents a linear 1.16 increase in power from baseline. In the plot, it is quite visible that there is spectral ERS activity roughly 550ms after the stimulus for the left scenario in channel 5 (C4) where the right trials exhibited a weaker more spread-out signal for channel 6 (C4). ERD is also much more visible in the first plot with time before

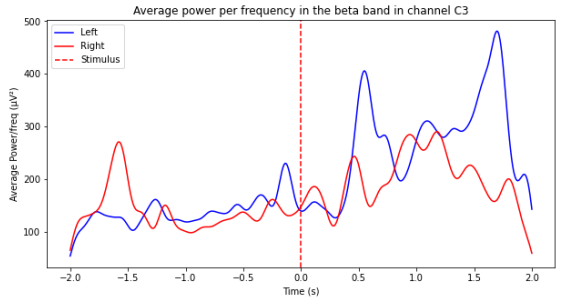


(a) Average alpha band power C3

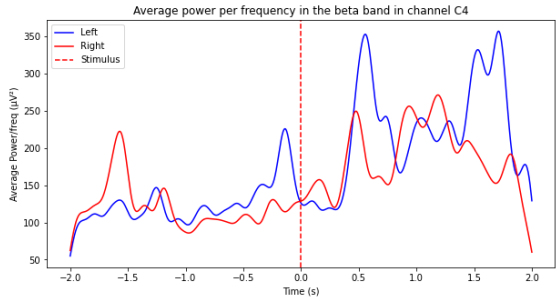


(b) Average alpha band power C4

Figure 7.8: Average alpha band(8-12hz) power for the signal . The code for this plot is available in appendix B.1



(a) Average beta band power C3



(b) Average beta band power C4

Figure 7.9: Average beta band(12-20hz) power for the signal. The code for this plot is available in appendix B.1

the stimulation at t=0s showing ERD activity. The first plot of channel 5 (C3) also shows this however to a much lower extent and with the power of right trials being slightly more concentrated in time. The plots were also averaged over n=150 epochs for each trial which indicated the statistical certainty of the data. This entails that while the analysis prior shows that while it is quite easy to see that motor execution is taking place by using the above tools it is hard to see the distinction between left and right trials. Features such as PSD and amplitude plots do not show any differences whereas ERDS plots and magnitude plots can visualize ERDS much better. The figures below show a

The spectrograms in figures 7.6 and 7.7 shows the signal power of each frequency overtime during an event, but another interesting visualization of the signal power is to average the signal power over all frequencies in a specific frequency band. This results in a line chart of the alpha band (8Hz - 12Hz) and beta band (12Hz - 30Hz) power during an event. Figures 7.8 and 7.9 show the average power per frequency of channels C3 and C4 for both the alpha and beta band. As the left part of the brain controls the right arm, and the right part of the brain controls the left arm, we expect the power in C3 for the right to be dominant over the left and we expect the power in C4 for the left to be dominant over the right. Interestingly, our experiments show for both channels C3 and C4 that the left signal is much larger. Several reasons may explain this result. The experimental setup may have been faulty regarding the sturdiness of the headset. The subject could have been moving the headset while performing the trials for the 'left' motor execution, while the headset remained still for the 'right' motor execution trials. This would have had to happen over many trials to make a significant impact on the average trial and is therefore unlikely but still possible. Another explanation could be that the electrode C3 was not placed entirely against the scalp, thereby picking up a lesser signal.

Although it is difficult to make a solid distinction between left and right through this data, it is obvious that the motor execution is happening. ERS and ERD are visible however not to the extent of clearly distinguishing left and right trials. While a concrete distinction could not have been made, the machine learning model with a collection of features may prove to be far more effective at distinguishing the trials.

8

Conclusion

To conclude this thesis we achieved a majority of the subgroup goals set in the beginning with sufficient to great levels of competence. Research was done into both hardware and software regarding the best methods to acquire and process data. Electrode layouts were chosen to be placed on areas with the most cortical activity during motor execution and software was chosen to be able to store recorded data in.EDF files with relevant annotations that was sent to the machine learning subgroup. Testing was designed around problems that were found such as movement, concentration, and electrode contact with robust solutions created to minimize these negative effects in the recorded data. Three separate testing experiments were created in a repeatable and logical manner to ensure that signals being received from the headset were as clear as possible a testing document as provided in supplement with this document was also successfully created and used. The achieved increase in amplitude for blinks was approximately 3x compared to the baseline. The achieved increase in power for the motor execution movements from baseline was 1.6667x on channel 5 (C3) and 2.0667x on channel 6 (C4) which was more than satisfactory to visualize the ERS patterns, ERD signals remain hard to visualize. There were also 150 trials measured for one test subject which while falling short of the goal of 200 was sufficient for current data visualization and analysis. Further recordings are also planned post June 14th, 2024 in which the aim is to measure more than 100 more datasets to better improve the machine learning models. The preprocessing step was designed to filter the blinks with a 5th-order butter worth 1Hz high pass filter and the motor execution signals were filtered with an 8hz-30hz 5th-order butter worth bandpass filter, allowing for extraction and visualization of patterns both temporally and spatially. The latency of signal acquisition came to approximately 172.183ms which is a little bit above the goal of 170ms, the goal of achieving a delay of 170ms was also considered to be achieved sufficiently. To analyze the blink signals ERSP was used which showed clear spectral blink patterns with varying strength across the electrodes on the headset with a peak in signal power around 1Hz. The plots to visualize motor execution allowed for the verification that motor execution signals were detectable and whether left and right trials could be visually distinguished. The plots of the PSD of the signal did not yield much information whereas ERDS plots and envelope plots showed much clearer motor execution patterns. The Envelope plots also showed lagging data across left and right trials which was not expected behavior. For the group requirements, the goal succeeded in creating usable motor execution signals which were sent to the Machine learning subgroup. Integration is also underway with all other subgroups to create a working system, meaning that there is no concrete way to verify whether the group requirements were achieved The data collection subgroup has successfully achieved most of the subgroup goals set to a sufficiently high standard with further progress remaining on more data collection, latency reduction, and systems integration.

9 Discussion

This project endured many hindrances that are of relevance to mention in this thesis. This chapter will elaborate on potential objections to the validity of the measurements and results. This chapter will also discuss improvements that could be made to the experiments.

9.1. Interference

This project lost much time due to some strange interference that the team encountered. A very large percentage of the recorded measurements had to be thrown out of the datasets due to this strange interference. Figure 9.1 shows a good example of the sudden interference on one of the channels. More examples of this interference in the raw signal are shown in section A.5. The team has not been able to pinpoint the cause of this interference but has figured out the following parameters. The interference is periodic. The interference is independent of one's location since it was observed on various levels of the faculty as well as right outside of the faculty. The interference seems to be around 8Hz, with peaks about 3 times the size of EEG data peaks. It is likely that the interference is caused by either a strong antenna inside the faculty that emits frequencies of around 8Hz, or the interference is caused by a hardware issue where some periodic signal from one of the components is leaking into the data. Although this may be a software issue, it is very unlikely as this was observed in several data acquisition programs.

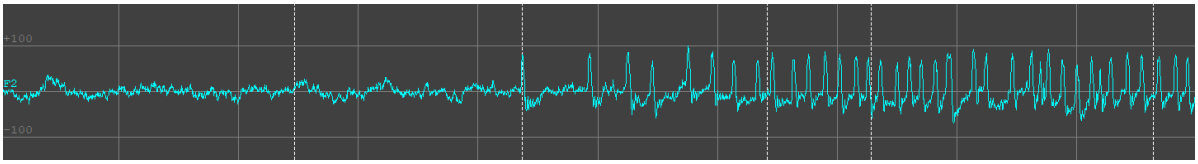


Figure 9.1: Example of the signal suddenly picking up interference

9.2. Conclusiveness of the results

9.2.1. Hardware

Although the team tried to make sure that the EEG helmet was always harnessed incorrectly, this was a rather difficult task. The electrodes must be placed directly against the scalp to pick up decent data, which is difficult for a person with hair. We have seen in a previous thesis [18] that the standard deviation of an EEG signal is dependent on the amount of hair a person has. This combined with the sub-optimal sturdiness of the helmet resulted in many times one or more channels becoming rather noisy. Some of the datasets likely had one or more channels that had more noise than they were supposed to have. This could have influenced the results. The team recommends any new research group experiment on the person with the least amount of hair, as well as watching the EEG signals live while mounting the Mark IV. This way one could easily identify a noisy channel before starting measurements.

9.2.2. Software

Some strange filtering behavior by the 'Modifiable temporal filter' function of the OpenVIBE software was observed. This function supposedly allows the user to more specifically customize a filter. The first datasets that were sent to the Machine Learning group used this filter. This filtering method however was much less optimal than the 'Temporal filter' function. Figure 9.2 shows the same raw signal, where one uses the 'Temporal filter' function and the other uses the 'Modifiable Temporal Filter' function. Section 6.1 explains how the filters were applied to obtain this 'raw' signal. The team still does not know where the vast difference comes from, but recommends any new research team use the 'Temporal filter' over the 'Modifiable temporal filter'.

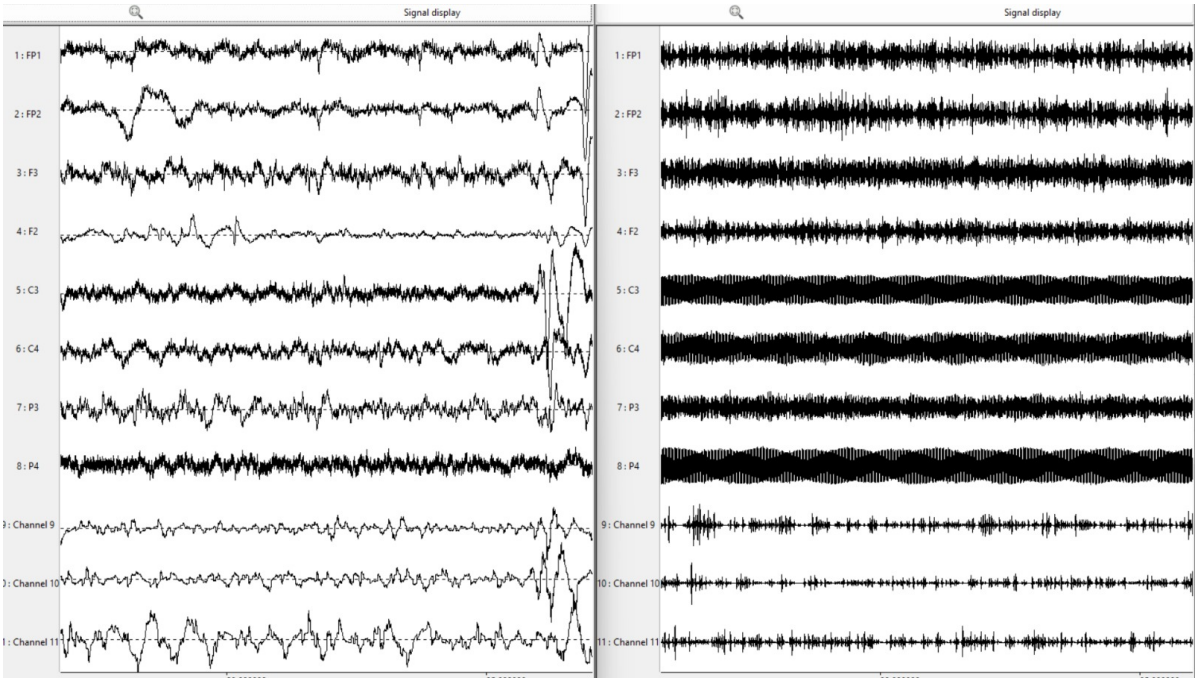


Figure 9.2: Left: Temporal filter, Right: Modifiable temporal filter

9.2.3. Continued Development

While our group achieved sufficient progress there is still a lot to be pursued, especially in terms of collecting more data or extending to motor imagery. Due to time constraints, the amount of data recorded was sufficient however it could be extended by capturing trials for more than one test subject, for a much broader high-accuracy application we would suggest at least 300 trials and 10 unique people. The use of other gel-based headsets could also be considered for better comfort and results along with more feature analysis to visualize the difference between left and right trials. Overall the field of motor execution-based BCIs seems very promising and we hope that this research will be picked up further.

References

[1] Xiao Jiang; Gui-Bin Bian; and Zean Tian. “Removal of Artifacts from EEG Signals: A Review”. In: *NIH* (Feb. 2019). DOI: 10.3390/s19050987. URL: https://www-ncbi-nlm-nih-gov.tudelft.idm.oclc.org/pmc/articles/PMC6427454/.

[2] Xun Chen et al. “ReMAE: User-Friendly Toolbox for Removing Muscle Artifacts from EEG”. In: *IEEE Transactions on Instrumentation and Measurement* 69 (5 2020). ISSN: 15579662. DOI: 10.1109/TIM.2019.2920186.

[3] *Cyton Data Format*. URL: https://docs.openbci.com/Cyton/CytonDataFormat/. (accessed: 24.04.2024).

[4] Suzanna Becker; Kiret Dhindsa; Leila Mousapour; Yar Al Dabagh. “BCI Illiteracy: It’s Us, Not Them. Optimizing BCIs for Individual Brains”. In: *IEEE* (Feb. 2022). DOI: 10.1109/BCI53720.2022.9735007. URL: https://ieeexplore.ieee.org/document/9735007.

[5] Amita Giri, Lalan Kumar, and Tapan Gandhi. “Cortical Source Domain Based Motor Imagery and Motor Execution Framework for Enhanced Brain Computer Interface Applications”. In: *IEEE Sensors Letters* 5 (12 2021). ISSN: 24751472. DOI: 10.1109/LSENS.2021.3122453.

[6] Quanying Liu; Joshua H Balsters; Marc Baechinger; Onno van der Groen; Nicole Wenderoth and Dante Mantini. “Estimating a neutral reference for electroencephalographic recordings: the importance of using a high-density montage and a realistic head model”. In: *Pubmed* (Aug. 2014). DOI: 10.1088/1741-2560/12/5/056012. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4719184/.

[7] OpenVibe Inria. *Title of the Website*. 2023. URL: https://openvibe.inria.fr/features/.

[8] Masaki Iwasaki et al. “Effects of eyelid closure, blinks, and eye movements on the electroencephalogram”. In: *Clinical Neurophysiology* 116 (4 2005). ISSN: 13882457. DOI: 10.1016/j.clinph.2004.11.001.

[9] Mitul Kumar Ahirwal; Narendra D. Londhe. “Offline Study of Brain Computer Interfacing for Hand Movement Using OpenVIBE”. In: *IEEE* (Aug. 2011). DOI: 10.1109/access.2023.3293421. URL: https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/5978930.

[10] OpenBCI. *The OpenBCI GUI*. 2023. URL: https://docs.openbci.com/Software/OpenBCISoftware/GUIDocs/.

[11] G. Pfurtscheller and F. H. Lopes Da Silva. *Event-related EEG/MEG synchronization and desynchronization: Basic principles*. 1999. DOI: 10.1016/S1388-2457(99)00141-8.

[12] Elena Pitsik et al. “Motor execution reduces EEG signals complexity: Recurrence quantification analysis study”. In: *Chaos* 30 (2 2020). ISSN: 10541500. DOI: 10.1063/1.5136246.

[13] Rakesh Ranjan, Bikash Chandra Sahana, and Ashish Kumar Bhandari. “Cardiac Artifact Noise Removal From Sleep EEG Signals Using Hybrid Denoising Model”. In: *IEEE Transactions on Instrumentation and Measurement* (2022). ISSN: 15579662. DOI: 10.1109/TIM.2022.3198441.

[14] Kavita V. Singala and Kiran R. Trivedi. “Connection setup of openvibe tool with EEG headset, parsing and processing of EEG signals”. In: 2016. DOI: 10.1109/ICCSP.2016.7754278.

[15] Zhichuan Tang et al. “A brain-machine interface based on ERD/ERS for an upper-limb exoskeleton control”. In: *Sensors (Switzerland)* 16 (12 2016). ISSN: 14248220. DOI: 10.3390/s16122050.

[16] Dezhong Yao; Yun Qin; Shiang Hu; Li Dong; Maria L. Bringas Vega and Pedro A. Valdés Sosa. “Which Reference Should We Use for EEG and ERP practice?” In: *Pubmed* (Apr. 2018). DOI: 10.1007/s10548-019-00707-x. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6592976/.

[17] C. Brunner; G. Andreoni; L. Bianchi; B. Blankertz; C. Breitwieser; S. Kanoh; C. A. Kothe; A. Lecuyer; S. Makeig; J. Mellinger; P. Perego; Y. Renard; G. Schalk; ´ I. P. Susila; B. Venthur; and G. R. Muller-Putz. “BCI Software Platforms”. In: *IEEE* (Apr. 2022). URL: https://sccn.ucsd.edu/~scott/pdf/Brunner_BCI11.pdf.

[18] Thibault Los Wesmond Lee. “EEG based BCI: measurement and quality control”. In: (2023).

[19] Hai-Jiang Meng; Yan-Ling Pi; Ke Liu; Na Cao; Yan-Qiu Wang; Yin Wu; and Jian Zhang. “Differences between motor execution and motor imagery of grasping movements in the motor cortical excitatory circuit”. In: *NIH* (Aug. 2018). DOI: 10.7717/peerj.5588. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6118197/.

[20] Reza Abiri ; Soheil Borhani; Eric W Sellers; Yang Jiang; Xiaopeng Zhao. “A comprehensive review of EEG-based brain-computer interface paradigms”. In: *PubMed* (Nov. 2015). DOI: 10.1088/1741-2552/aaf12e. URL: https://pubmed.ncbi.nlm.nih.gov/30523919/.

A

Figures and Tables

A.1. Hardware overview

Channel	Electrode
1	FP1
2	FP2
3	F3
4	F2
5	C3
6	C4
7	P3
8	P4

Table A.1: Electrode connections to the channels of the Cyton OpenBCI board

A.2. Preprocessing

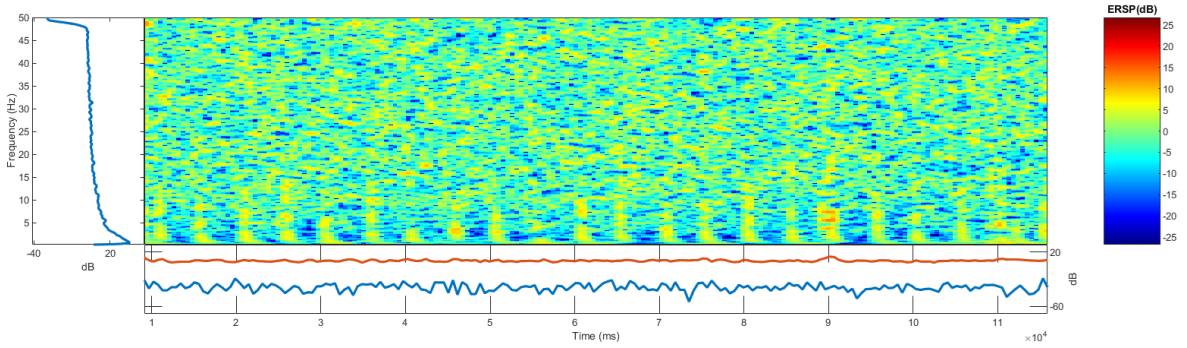


Figure A.1: The spectrogram of an experiment where the subject periodically blinked

A.3. Data Acquisition

A.3.1. Latency

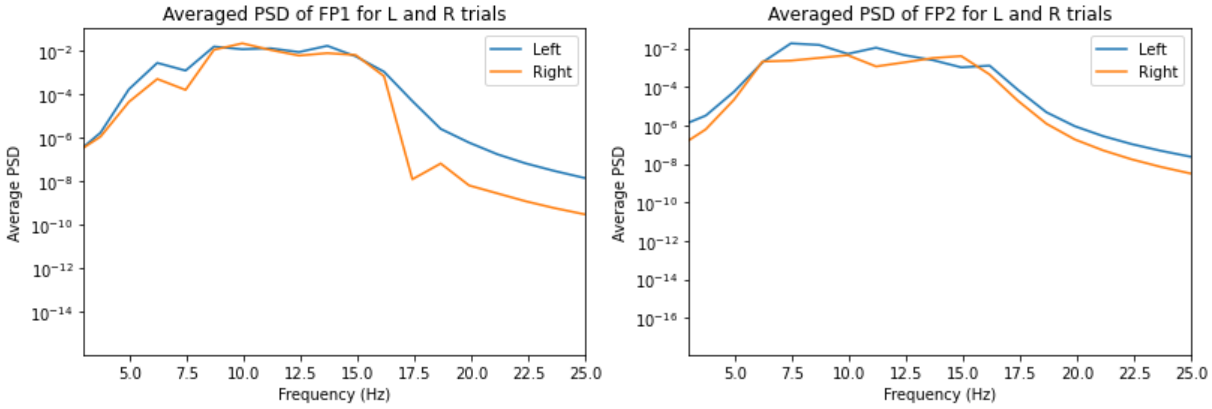
```
IIR filter parameters
---------------------
Butterworth bandpass zero-phase (two-pass forward and reverse) non-causal filter:
- Filter order 20 (effective, after forward-backward)
- Cutoffs at 8.00, 12.00 Hz: -6.02, -6.02 dB

The filter operation took 0.1736752986907959 seconds.
Used Annotations descriptions: ['OVTK_GDF_Cross_On_Screen', 'OVTK_GDF_End_Of_Session',
'OVTK_GDF_End_Of_Trial', 'OVTK_GDF_Feedback_Continuous', 'OVTK_GDF_Left', 'OVTK_GDF_Right',
'OVTK_GDF_Start_Of_Trial', 'OVTK_StimulationId_BaselineStart', 'OVTK_StimulationId_BaselineStop',
'OVTK_StimulationId_Beep', 'OVTK_StimulationId_ExperimentStart', 'OVTK_StimulationId_ExperimentStop',
'OVTK_StimulationId_Train']
Not setting metadata
150 matching events found
Setting baseline interval to [-2.0, 0.0] s
Applying baseline correction (mode: mean)
0 projection items activated
```

Figure A.2: Latency measured of the headset to filtered data using python measured using a time function

A.4. Plots

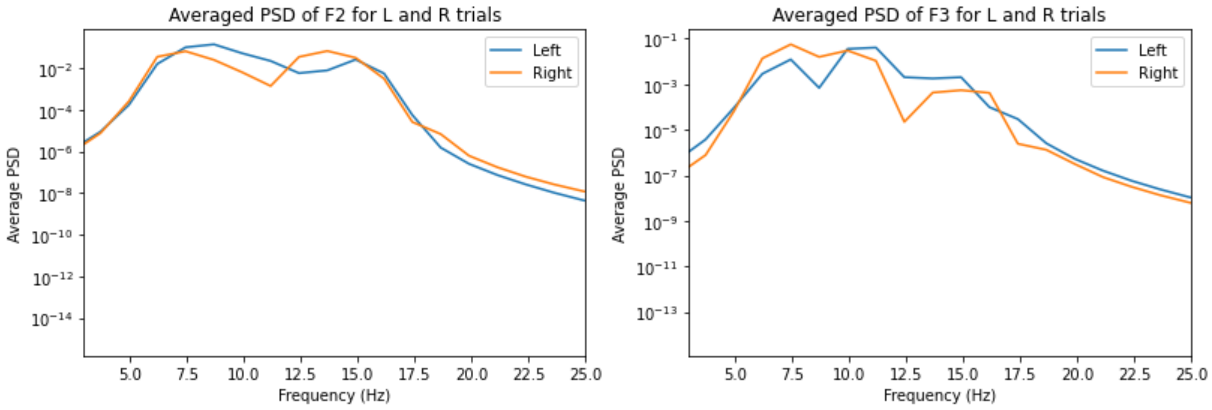
A.4.1. PSD plots



(a) Average PSD of channel 1(FP1)

(b) Average PSD of channel 2(FP2)

Figure A.3: Comparison of filtered(8-30hz) and averaged PSD between channel 1 and 2 for n=150 left and right trials



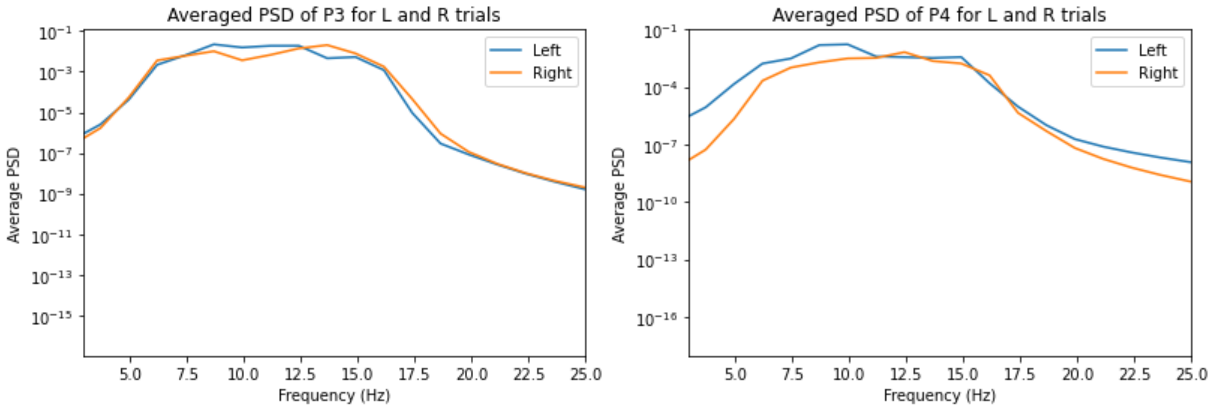
(a) Average PSD of channel 3(F3)

(b) Average PSD of channel 4(F2)

Figure A.4: Comparison of filtered(8-30hz) and averaged PSD between channel 3 and 4 for n=150 left and right trials

A.4.2. Squared Magnitude plots

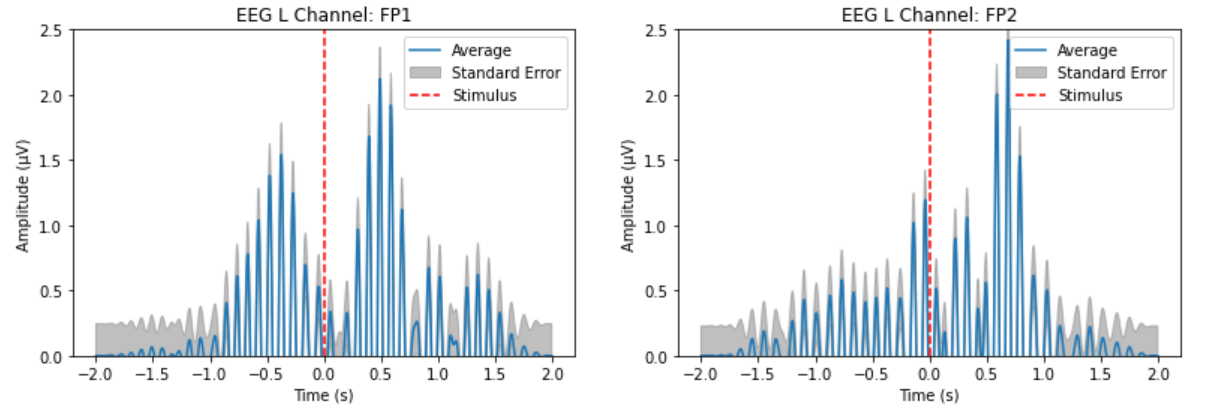
A.5. Discussion



(a) Average PSD of channel 7(P3)

(b) Average PSD of channel 8(P4)

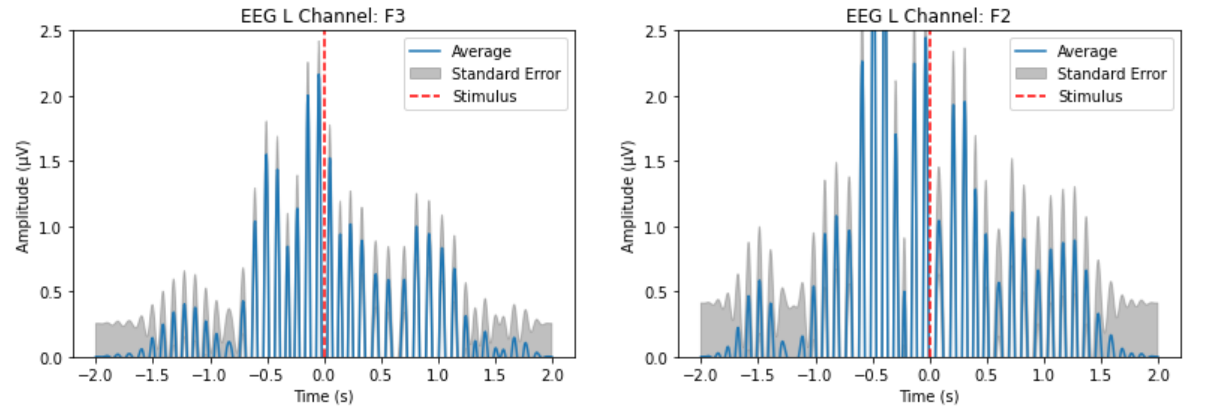
Figure A.5: Comparison of filtered(8-30hz) and averaged PSD between channel 7 and 8 for n=150 left and right trials



(a) Average PSD of channel 1(FP1)

(b) Average PSD of channel 2(FP2)

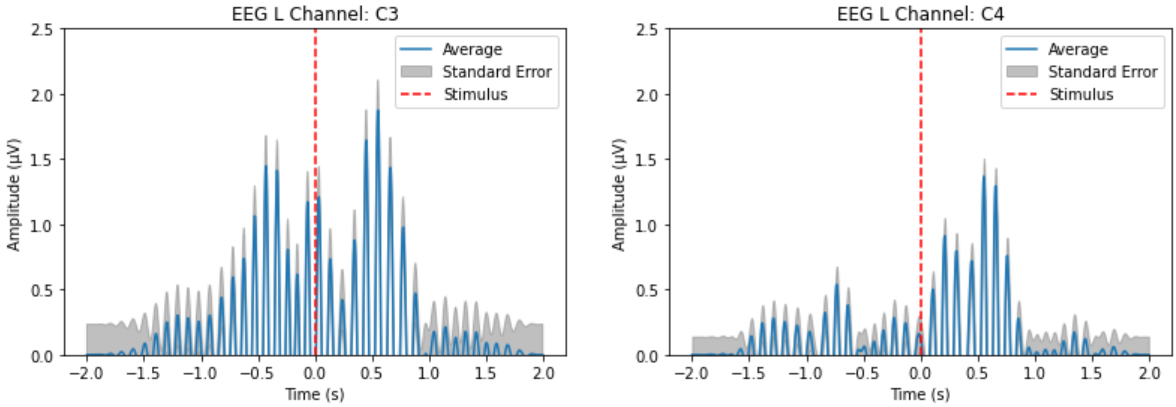
Figure A.6: Comparison of filtered(8-30hz) and averaged squared magnitude plot between channel 1 and 2 for n=150 left and right trials



(a) Average PSD of channel 3(F3)

(b) Average PSD of channel 4(F2)

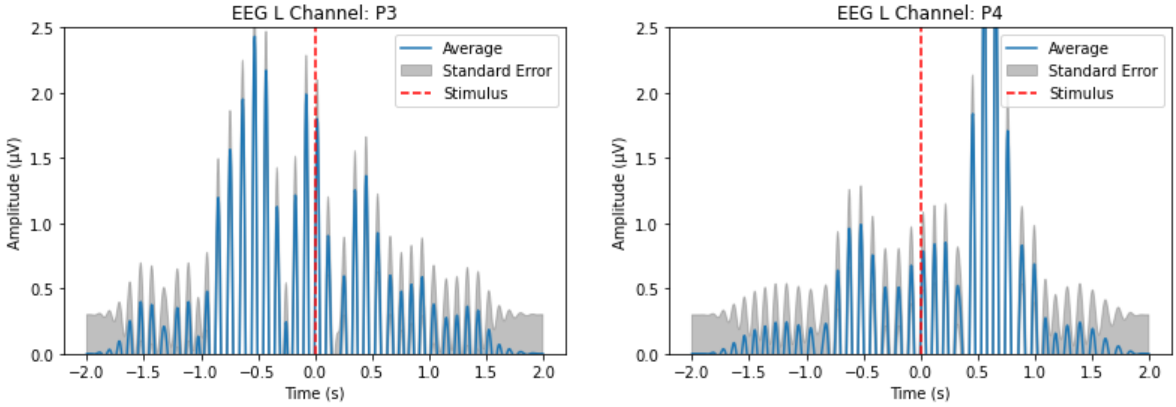
Figure A.7: Comparison of filtered(8-30hz) and averaged squared magnitude plot between channel 3 and 4 for n=150 left and right trials



(a) Average PSD of channel 5(C3)

(b) Average PSD of channel 6(C4)

Figure A.8: Comparison of filtered(8-30hz) and averaged squared magnitude plot between channel 5 and 6 for n=150 left and right trials



(a) Average PSD of channel 7(P3)

(b) Average PSD of channel 8(P4)

Figure A.9: Comparison of filtered(8-30hz) and averaged squared magnitude plot between channel 7 and 8 for n=150 left and right trials

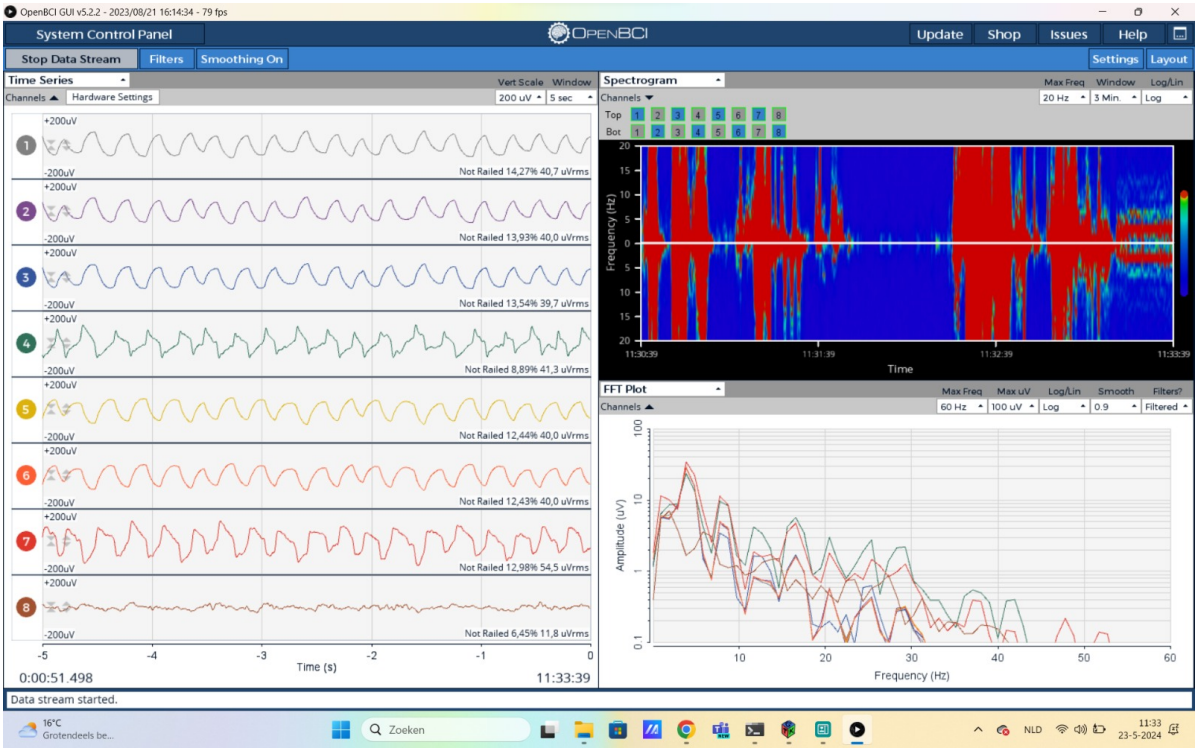


Figure A.10: Note the right edge of the spectogram: very clearly a specific band interference

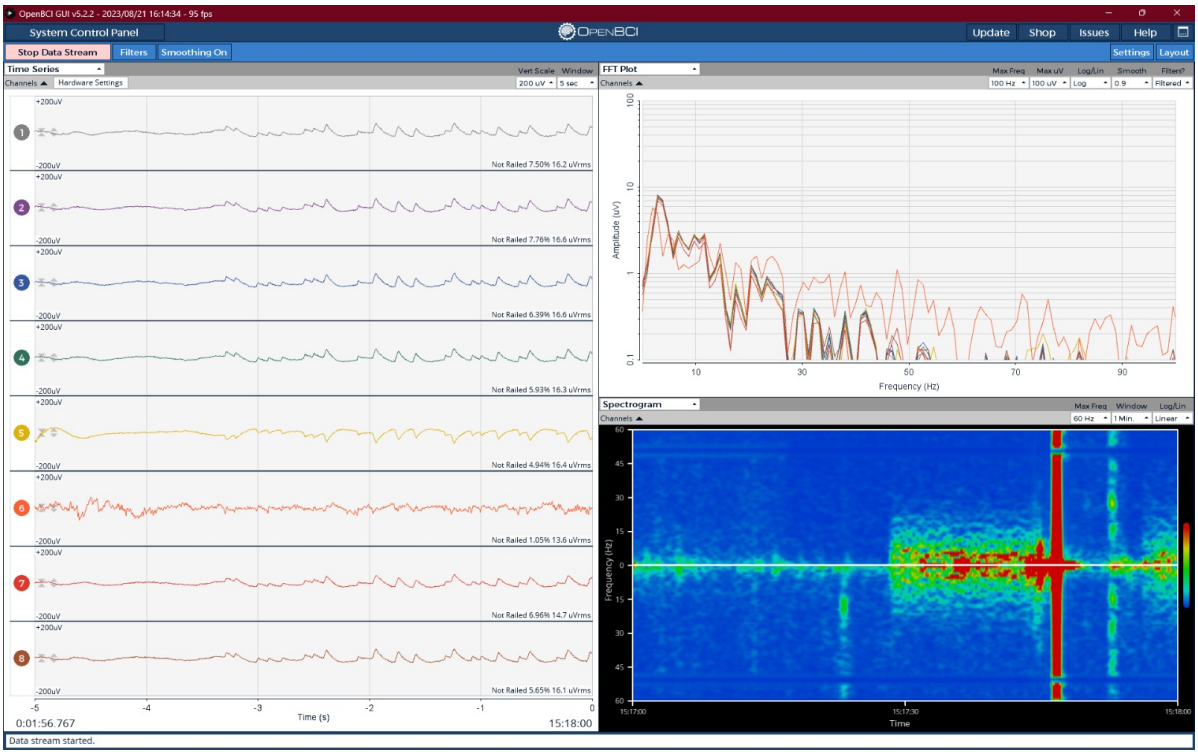


Figure A.11

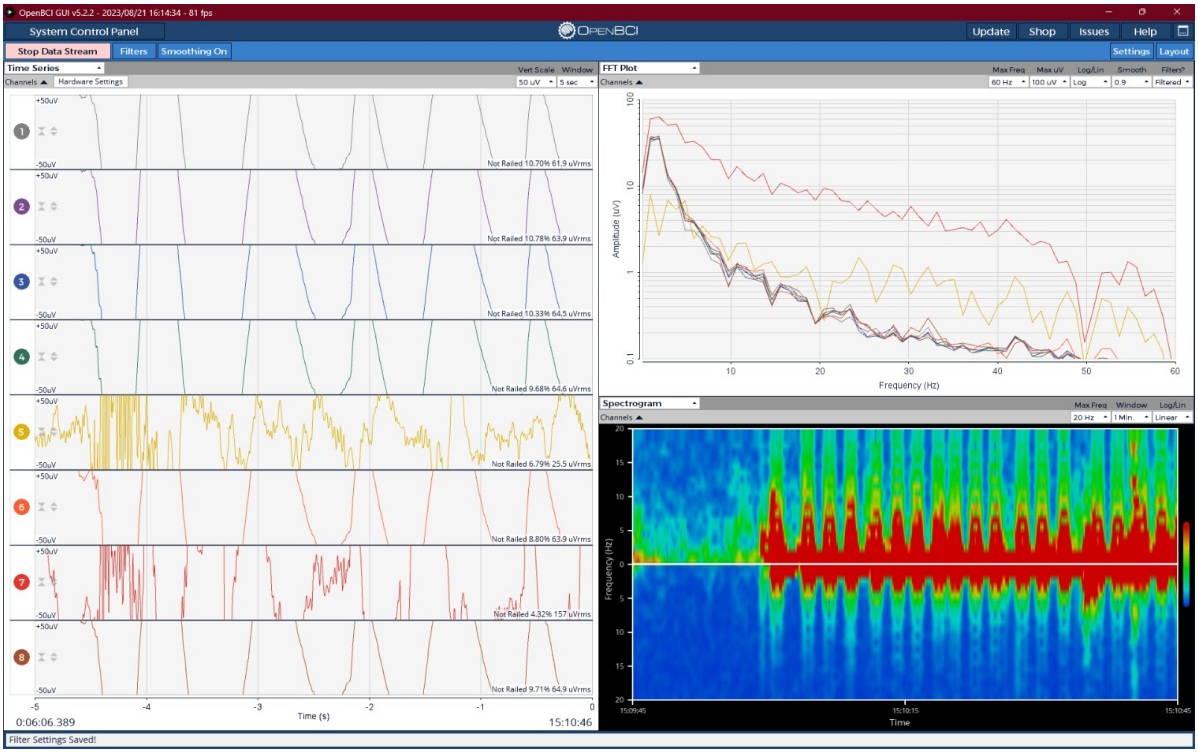


Figure A.12

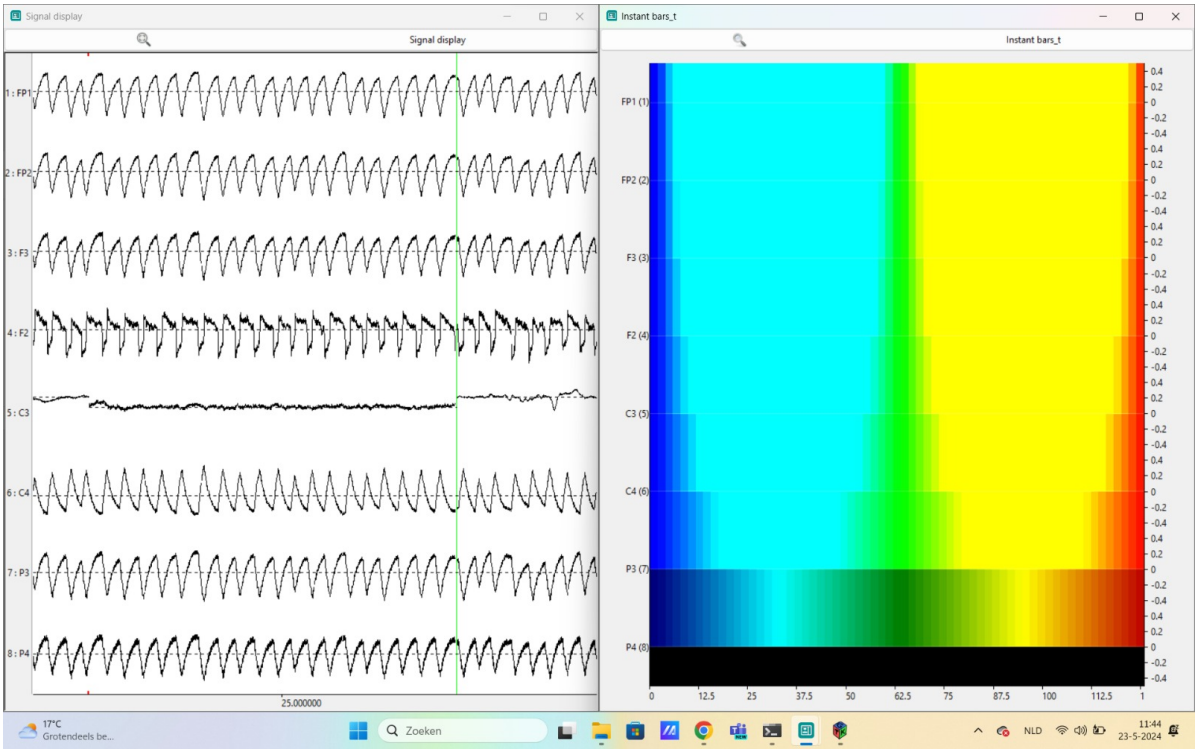


Figure A.13

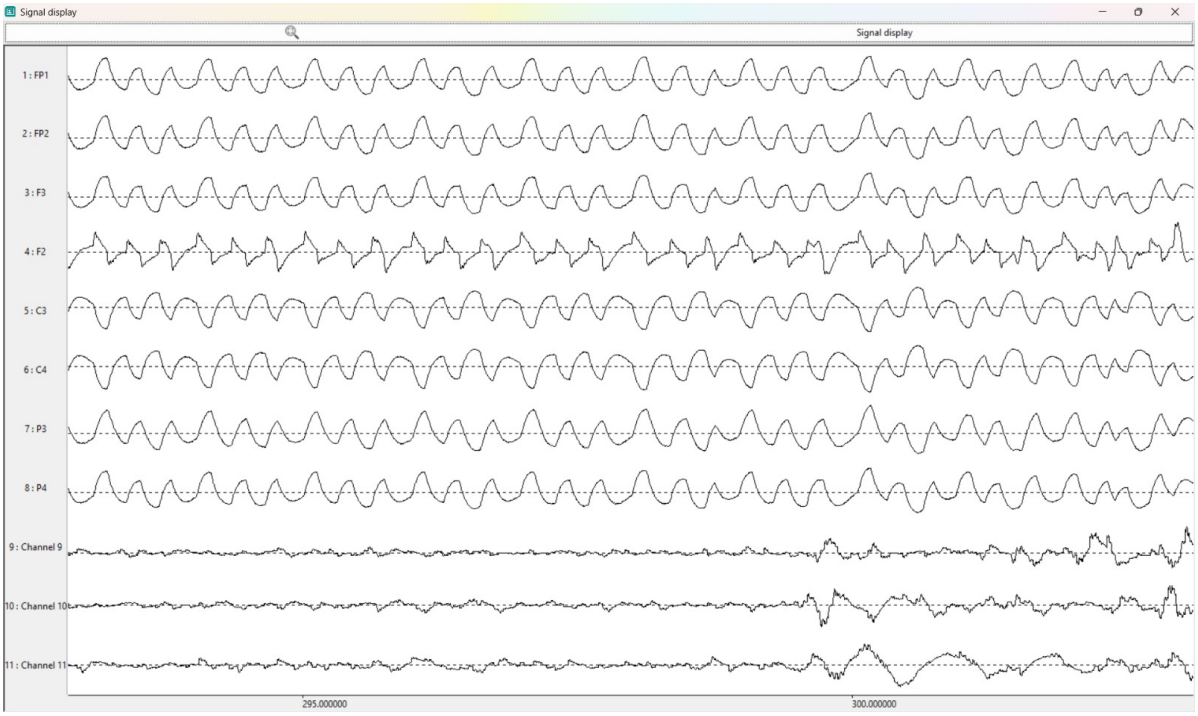


Figure A.14

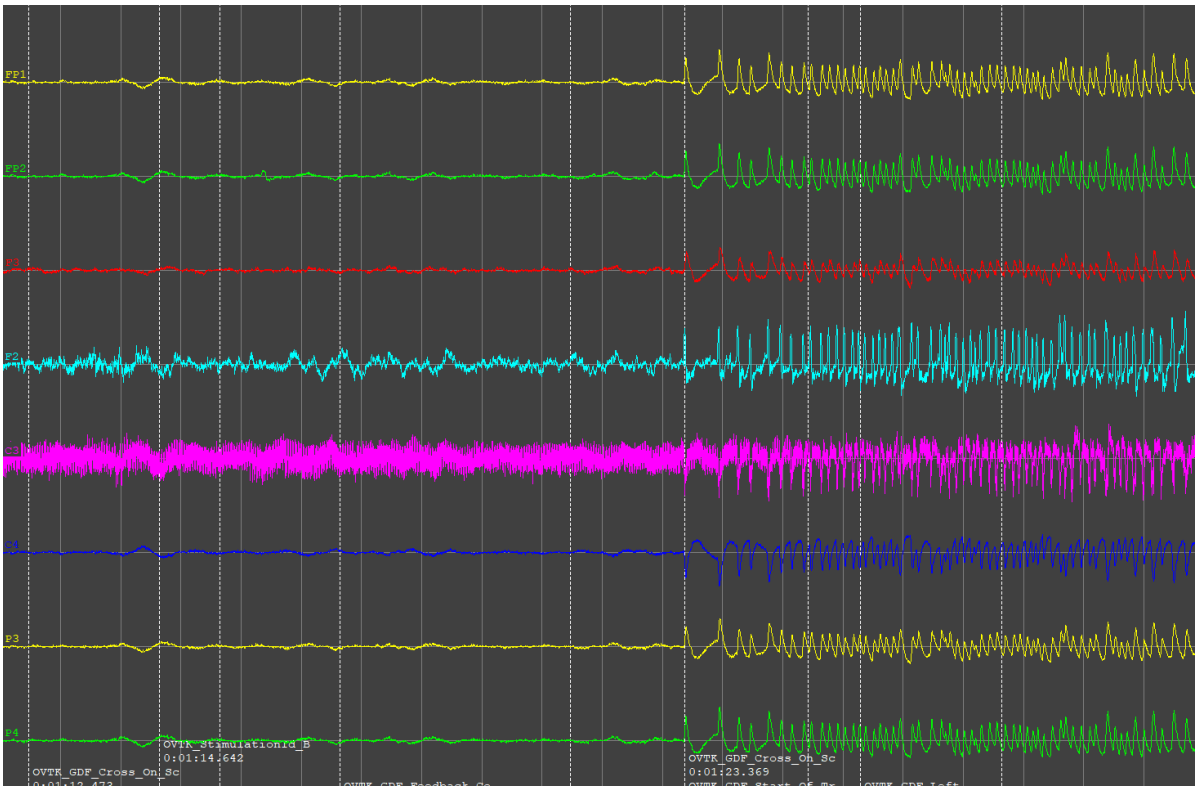


Figure A.15

B

Code

B.1. Python

Average Power Alpha/Beta band

```python
"""
@author: Niels van Duivendijk
"""

import mne
import numpy as np
import os
import matplotlib.pyplot as plt

# Directory containing the EDF files
edf_directory = r'C:\Users\nvand\OneDrive\Documenten\EEG DATA\New filter data\RAW'

# Get a list of all EDF files in the directory
edf_files = [os.path.join(edf_directory, f) for f in os.listdir(edf_directory) if f.endswith('.edf')]

# Load the EDF files
raws = [mne.io.read_raw_edf(f, preload=True) for f in edf_files]

# Concatenate the raw data objects
raw = mne.concatenate_raws(raws)

#Define frequency band of interest
l_freq=13.
h_freq=30.

raw_filtered = raw.copy().filter(l_freq, h_freq, method='iir', iir_params={'order': 5, 'ftype': 'cheby1', 'rp': 0.1})

# Extract events from the raw data
events, event_id = mne.events_from_annotations(raw_filtered)

# Keep only the events of interest
event_id = { 'OVTK_GDF_Left': event_id['OVTK_GDF_Left'], 'OVTK_GDF_Right': event_id['OVTK_GDF_Right'] }

# Define the duration of the epochs
tmin, tmax = -2, 2  # start and end of an epoch relative to the event

# Use 'events' to create epochs
epochs = mne.Epochs(raw_filtered, events, event_id, tmin, tmax)

# Split the epochs based on the event type
epochs_left = epochs['OVTK_GDF_Left']
epochs_right = epochs['OVTK_GDF_Right']

```

```
# Compute TFR for left and right epochs
freqs = np.arange(l_freq, h_freq+1, 1.)
n_cycles = freqs / 2                        # different number of cycle per frequency

# Compute the TFR for the left epochs
power_left = mne.time_frequency.tfr_morlet(epochs_left, freqs=freqs, n_cycles=n_cycles, return_itc=False)

# Compute the TFR for the right epochs
power_right = mne.time_frequency.tfr_morlet(epochs_right, freqs=freqs, n_cycles=n_cycles, return_itc=False)

# Get the index of the channel of interest
index = raw.ch_names.index('C4')

# Sum the power across frequencies for the channel for each time point
power_left_avg = power_left.data[:, index, :].mean(axis=0)  # Sum across the frequency dimension and avereage
power_right_avg = power_right.data[:, index, :].mean(axis=0)

# Create time vector based on the epochs' time information
times = epochs.times

# Plot the summed power for the channel for left and right epochs
plt.figure(figsize=(10, 5))
plt.plot(times, power_left_avg, label='Left', color='blue')
plt.plot(times, power_right_avg, label='Right', color='red')
plt.axvline(0, color='r', linestyle='--', label='Stimulus')
plt.xlabel('Time (s)')
plt.ylabel('Average Power/freq (V²)')
plt.title('Average power per frequency in the beta band in channel C4')
plt.legend()
plt.show()
```

Averaged PSD

```
"""
Created on Wed May 29 12:27:56 2024

@author: Abhay Shenoy
"""

import mne
import numpy as np
from scipy.signal import welch
import matplotlib.pyplot as plt
import os

# Directory containing the edf files
edf_directory = r'C:\Users\Abhay Shenoy\Documents\EEG DATA\NEW FILTERED EDF DATA'

# Get a list of all edf files in the directory
edf_files = [os.path.join(edf_directory, f) for f in os.listdir(edf_directory) if f.endswith('.edf')]

# Load the edf files
raws = [mne.io.read_raw_edf(f, preload=True) for f in edf_files]

# Concatenate the raw data
raw = mne.concatenate_raws(raws)

raw_filtered = raw.copy().filter(l_freq=7., h_freq=15., method='iir', iir_params={'order': 6, 'ftype': 'cheby1', 'rp': 0.1})

# Extract events from the raw data
events, event_id = mne.events_from_annotations(raw_filtered)

# Keep  the events of interest
event_id = { 'OVTK_GDF_Left': event_id['OVTK_GDF_Left'], 'OVTK_GDF_Right': event_id['OVTK_GDF_Right'] }
```

```

# Define the duration of your epochs
tmin, tmax = -2, 2

# epochs creation
epochs = mne.Epochs(raw_filtered, events, event_id, tmin, tmax)

# Split the epochs
epochs_left = epochs['OVTK_GDF_Left']
epochs_right = epochs['OVTK_GDF_Right']

# Average the epochs
epochs_average_left = epochs_left.average()
epochs_average_right = epochs_right.average()

# Define the mapping of channel indices to names
channel_mapping = {1: 'FP1', 2: 'FP2', 3: 'F3', 4: 'F2', 5: 'C3', 6: 'C4', 7: 'P3', 8: 'P4'}

# Compute PSD of the averaged epochs for every channell
for i in range(epochs_average_left.data.shape[0]):
    frequencies_left, power_left = welch(epochs_average_left.data[i], fs=raw.info['sfreq'])
    frequencies_right, power_right = welch(epochs_average_right.data[i], fs=raw.info['sfreq'
        ])

    # Normalize the PSD
    power_normalized_left = power_left #/ np.sum(power_left)
    power_normalized_right = power_right #/ np.sum(power_right)

    ch_name = channel_mapping.get(i+1, f'Channel␣{i+1}')

    # Plot normalized PSD
    plt.figure()
    plt.semilogy(frequencies_left, power_normalized_left, label='Left')
    plt.semilogy(frequencies_right, power_normalized_right, label='Right')
    plt.xlabel('Frequency␣(Hz)')
    plt.ylabel('Average␣PSD')
    plt.title(f'Averaged␣PSD␣of␣{ch_name}␣for␣L␣and␣R␣trials')
    plt.xlim([3, 25])
    plt.legend()
    plt.show()
```

Normalized Squared Magnitude

```

"""
Created on Wed May 29 12:27:56 2024

@author: Abhay Shenoy
"""

import mne
import time
import numpy as np
import os
import matplotlib.pyplot as plt
from scipy.stats import sem
from scipy.signal import hann, savgol_filter

# Directory containing the edf files
edf_directory = r'C:\\Users\\Abhay␣Shenoy\\Documents\\EEG␣DATA\\NEW␣FILTERED␣EDF␣DATA'

# Get all edf files in the directory
edf_files = [os.path.join(edf_directory, f) for f in os.listdir(edf_directory) if f.endswith(
    '.edf')]

# Load the edf files
raws = [mne.io.read_raw_edf(f, preload=True) for f in edf_files]

# Concatenate the raw data objects
raw = mne.concatenate_raws(raws)
```

```
27
28 start_time = time.time()
29
30 # Apply a bandpass filter between 8 and 12 Hz
31 raw_filtered = raw.filter(l_freq=8, h_freq=12, method='iir', iir_params={'order': 5, 'ftype':
       'butter'})
32
33 # Stop the timer
34 end_time = time.time()
35
36 # Calculate the elapsed time
37 elapsed_time = end_time - start_time
38
39 print(f"The␣filter␣operation␣was␣{elapsed_time}␣s.")
40
41 # Extract events
42 events, event_id = mne.events_from_annotations(raw_filtered)
43
44 # Keep the events of interest
45 event_id = { 'OVTK_GDF_Left': event_id['OVTK_GDF_Left'], 'OVTK_GDF_Right': event_id['
       OVTK_GDF_Right'] }
46
47 # duration of your epochs
48 tmin, tmax = -2, 2  # start and end of an epoch relative to the event
49
50 #create epochs
51 epochs = mne.Epochs(raw, events, event_id, tmin, tmax)
52
53 # Get epochs for the 'OVTK_GDF_Right' event
54 epochs_right = epochs['OVTK_GDF_Left']
55
56 # Average epochs
57 epochs_average_right = epochs_right.average()
58
59 # Convert data to microvolts
60 raw_data_microvolts = epochs_average_right.data * 1e6
61
62 # Normalize data
63 raw_data_microvolts = (raw_data_microvolts - np.mean(raw_data_microvolts)) / np.std(
       raw_data_microvolts)
64
65 # Get time vector
66 times = epochs_average_right.times
67
68 # Get channel names
69 ch_names = epochs_average_right.ch_names
70
71 fs = raw.info['sfreq']
72
73 # Plot each channel
74 for i, ch_name in enumerate(ch_names):
75     # Get the signal
76     signal = raw_data_microvolts[i]
77
78     n_samples = len(signal)  # No. of samples in 4s
79
80     # Hann window
81     window = hann(n_samples)
82
83     # Apply the Hann window
84     signal_windowed = signal * window
85
86     # Calculate the average and standard error
87     avg = np.mean(signal_windowed)
88     err = sem(signal_windowed)
89
90     # standard error array
91     err_array = np.full_like(signal_windowed, err)
92
93     # Plot
94     plt.figure()
```

```
    plt.plot(times, signal_windowed, label='Average')
    plt.fill_between(times, signal_windowed -  err_array, signal_windowed +  14 * err_array,
        color='gray', alpha=0.5, label='Standard Error')
    plt.axvline(0, color='r', linestyle='--', label='Stimulus')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude (µV)')
    plt.title('EEG L Channel: ' + ch_name)
    plt.ylim([0, 2.5])
    plt.legend()
    plt.show()
```

Envelope and Baseline corrected Envelope

```
"""
Created on Sun May 26 22:04:08 2024
SINGLE MAGNITUDE(NORMAL OR SMOOTHED)
@author: Abhay Shenoy
"""
import mne
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import hann
from scipy.signal import hilbert
import os

# Path to edf file
# Directory containing the EDF files
edf_directory = r'C:\\Users\\Abhay Shenoy\\Documents\\EEG DATA\\NEW FILTERED EDF DATA'

# Get a list of edf files in directory
edf_files = [os.path.join(edf_directory, f) for f in os.listdir(edf_directory) if f.endswith(
    '.edf')]

# Load edf
raws = [mne.io.read_raw_edf(f, preload=True) for f in edf_files]

# Concatenate the raw data
raw = mne.concatenate_raws(raws)
raw_c3 = raw.copy().pick_channels(['C4'])

# Crop the data (31 to 35 seconds)
raw_c3.crop(tmin=31, tmax=35)

# Get the data and times
data, times = raw_c3[:]

# magnitude
magnitude = np.abs(data[0])   # Convert from V to µV
fs = raw.info['sfreq']

n_samples = len(smoothed_magnitude)

# Create Hann window
window = hann(n_samples)

# Apply the window
signal_windowed = smoothed_magnitude * window

# Apply a bandpass filter between 8 and 12 Hz
raw_filtered = raw.filter(l_freq=8, h_freq=12, method='iir', iir_params={'order': 5, 'ftype':
     'butter'})

# Extract events
events, event_id = mne.events_from_annotations(raw_filtered)

# Keep the events of interest
event_id = { 'OVTK_GDF_Left': event_id['OVTK_GDF_Left'], 'OVTK_GDF_Right': event_id['
    OVTK_GDF_Right'] }

# Define duration of  epochs
```

```
tmin, tmax = -2, 2

# create epochs
epochs = mne.Epochs(raw, events, event_id, tmin, tmax)

# Get epochs for the 'OVTK_GDF_Left'
epochs_left = epochs['OVTK_GDF_Left']

# Average epoch
epochs_average_left = epochs_left.average()

# Get data and times from averaged epochs
data_avg, times_avg = epochs_average_left.data, epochs_average_left.times

# Compute magnitude of signal in microvolts
magnitude_avg = np.abs(data_avg[0])   # Convert from V to µV

# Compute smoothed signal
smoothed_magnitude_avg = moving_average(magnitude_avg, window_size)

# Apply Hann window to the signal
signal_windowed_avg = smoothed_magnitude_avg * window

# Get the epochs for the 'OVTK_GDF_Right' event
epochs_right = epochs['OVTK_GDF_Right']

# Average epochs
epochs_average_right = epochs_right.average()

# Get data and times from averaged epochs
data_avg_right, times_avg_right = epochs_average_right.data, epochs_average_right.times

# Compute magnitude of signal in microvolts
magnitude_avg_right = np.abs(data_avg_right[0])   # Convert from V to µV

# Compute smoothd signal
smoothed_magnitude_avg_right = moving_average(magnitude_avg_right, window_size)

# Apply Hann window to the signal
signal_windowed_avg_right = smoothed_magnitude_avg_right * window

# Compute envelope of the signal for each epoch
envelopes_right = [np.abs(hilbert(epoch)) for epoch in epochs_right]
envelopes_left = [np.abs(hilbert(epoch)) for epoch in epochs_left]

# Compute average envelope
average_envelope_right = np.mean(envelopes_right, axis=0)
average_envelope_left = np.mean(envelopes_left, axis=0)

# Baseline correction
baseline_corrected_envelope_right = average_envelope_right - average_envelope_right[0, 0]
baseline_corrected_envelope_left = average_envelope_left - average_envelope_left[0, 0]

#  shaded average envelope
plt.figure(figsize=(12, 6))

# Plot
#plt.fill_between(times_avg_right, average_envelope_right[0], color='orange',alpha= 0.5,
    label='Average Envelope Right ')
#plt.fill_between(times_avg_right, average_envelope_left[0], color='blue',alpha= 0.5, label='
    Average Envelope Left ')
#plt.fill_between(times_avg_right, 0, baseline_corrected_envelope_right[0], color='orange',
    alpha=0.5, label='Average Envelope Right (Baseline Corrected)')
plt.fill_between(times_avg_right, 0, baseline_corrected_envelope_left[0], color='blue', alpha
    =0.5, label='Average Envelope Left (Baseline Corrected)')


plt.axvline(0, color='r', linestyle='--', label='Stimulation')
plt.xlabel('Time (s)')
plt.ylabel('Magnitude (µV)')
plt.title('Average Envelope of Epochs')
```

```
plt.legend()
plt.show()
```

ERDS plot

```
"""
Created on Wed May 29 12:27:56 2024

@author: Abhay Shenoy
"""

import mne
import numpy as np
import os
from scipy.signal import welch
import matplotlib.pyplot as plt

# Directory containing edf files
edf_directory = r'C:\Users\Abhay␣Shenoy\Documents\EEG␣DATA\NEW␣FILTERED␣EDF␣DATA'

# Get list of all edf files in directory
edf_files = [os.path.join(edf_directory, f) for f in os.listdir(edf_directory) if f.endswith('.edf')]

# Load edf files
raws = [mne.io.read_raw_edf(f, preload=True) for f in edf_files]

# Concatenate the raw data
raw = mne.concatenate_raws(raws)

raw_filtered = raw.copy().filter(l_freq=7., h_freq=30., method='iir', iir_params={'order': 6, 'ftype': 'cheby1', 'rp': 0.1})

# Extract events
events, event_id = mne.events_from_annotations(raw_filtered)

# Keep events of interest
event_id = { 'OVTK_GDF_Left': event_id['OVTK_GDF_Left'], 'OVTK_GDF_Right': event_id['OVTK_GDF_Right'] }

#duration of your epochs
tmin, tmax = -2, 2

#create epochs
epochs = mne.Epochs(raw_filtered, events, event_id, tmin, tmax)

# Split the epochs
epochs_left = epochs['OVTK_GDF_Left']
epochs_right = epochs['OVTK_GDF_Right']

# Compute TFR
freqs = np.arange(6, 32., 1.)  # frequencies of interest
n_cycles = freqs /2  # different number of cycle per frequency

power_left = mne.time_frequency.tfr_morlet(epochs_left, freqs=freqs, n_cycles=n_cycles, return_itc=False)
power_right = mne.time_frequency.tfr_morlet(epochs_right, freqs=freqs, n_cycles=n_cycles, return_itc=False)

# Plot TFR
fig, ax = plt.subplots(1, 2, figsize=(10, 5), sharey=True)
power_left.plot([raw.ch_names.index('P3')], baseline=(-0.5, 0), mode='logratio', axes=ax[0], show=False, colorbar=True, vmin=-0.7, vmax=0.7)
power_right.plot([raw.ch_names.index('P3')], baseline=(-0.5, 0), mode='logratio', axes=ax[1], show=False, colorbar=True, vmin=-0.7, vmax=0.7)
ax[0].set_title('Left')
ax[1].set_title('Right')
plt.suptitle('ERDS␣plot␣of␣Left␣and␣Right␣Epochs␣for␣P3␣Channel')
plt.show()
```

Signal Plot

```
"""
Created on Mon Jun 10 10:23:23 2024

@author: Abhay Shenoy
"""

import mne
import matplotlib.pyplot as plt

# Path edf file
edf_file = r'C:\Users\Abhay␣Shenoy\Documents\EEG␣DATA\FILTERED␣EDF␣DATA\record
    -[2024.05.28-15.30.09]_FILTERED.edf'

# Load edf file
raw = mne.io.read_raw_edf(edf_file, preload=True)

raw_filtered = raw.copy().filter(l_freq=7, h_freq=30, method='iir', iir_params={'order': 6, '
    ftype': 'cheby1', 'rp': 0.1})

raw_eeg = raw_filtered.copy().pick_channels(['P3'])

# Get data and times from'C3' channel
eeg_data, eeg_times = raw_eeg[:]

# Plot the 'C3' channel data
plt.figure(figsize=(10, 5))
plt.plot(eeg_times, eeg_data[0, :], label='C3')
plt.title('EEG␣Signal␣-␣C3␣Channel')
plt.xlabel('Time␣[s]')
plt.xlim(0,20)
plt.ylabel('Voltage␣[µV]')
plt.legend()
plt.show()
```

Power bar

```
"""
Created on Wed May 29 12:27:56 2024

@author: Abhay Shenoy
"""

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib.colors import TwoSlopeNorm
import os

import mne
from mne.datasets import eegbci
from mne.io import concatenate_raws, read_raw_edf
from mne.stats import permutation_cluster_1samp_test as pcluster_test

# Directory containing edf files
edf_directory = r'C:\Users\Abhay␣Shenoy\Documents\EEG␣DATA\NEW␣FILTERED␣EDF␣DATA'

# Get a list of all edf files in directory
edf_files = [os.path.join(edf_directory, f) for f in os.listdir(edf_directory) if f.endswith(
    '.edf')]

# Load edf files
raws = [mne.io.read_raw_edf(f, preload=True) for f in edf_files]

# Concatenate the raw data
raw = mne.concatenate_raws(raws)

raw_filtered = raw.copy().filter(l_freq=7., h_freq=30., method='iir', iir_params={'order': 6,
     'ftype': 'cheby1', 'rp': 0.1})
```

```
32
33 # Extract events
34 events, event_id = mne.events_from_annotations(raw_filtered)
35
36 # Keepthe events of interest
37 event_id = { 'OVTK_GDF_Left': event_id['OVTK_GDF_Left'], 'OVTK_GDF_Right': event_id['
       OVTK_GDF_Right'] }
38
39 #duration of your epochs
40 tmin, tmax = -2, 2
41 event_ids = dict(OVTK_GDF_Left=2, OVTK_GDF_Right=3)
42
43
44
45 epochs = mne.Epochs(raw,event_id=['OVTK_GDF_Left', 'OVTK_GDF_Right'],tmin=tmin - 0.5,tmax=
       tmax + 0.5,picks=("C3","P3", "C4"), baseline=None, preload=True,)
46
47 freqs = np.arange(2, 36)
48 vmin, vmax = -1, 1.5
49 baseline = (-1, 0)
50 cnorm = TwoSlopeNorm(vmin=vmin, vcenter=0, vmax=vmax)
51
52 kwargs = dict(n_permutations=100, step_down_p=0.05, seed=1, buffer_size=None, out_type="mask"
       )
53
54 tfr = epochs.compute_tfr(
55     method="multitaper",
56     freqs=freqs,
57     n_cycles=freqs,
58     use_fft=True,
59     return_itc=False,
60     average=False,
61     decim=2,
62 )
63 tfr.crop(tmin, tmax).apply_baseline(baseline, mode="percent")
64
65
66 df = tfr.to_data_frame(time_format=None, long_format=True)
67
68 #Map frequency bands:
69 freq_bounds = {"_": 0, "delta": 3, "theta": 7, "alpha": 13, "beta": 35, "gamma": 140}
70 df["band"] = pd.cut(
71     df["freq"], list(freq_bounds.values()), labels=list(freq_bounds)[1:]
72 )
73
74 df_mean = (
75     df.query("time␣>␣1")
76     .groupby(["condition", "epoch", "band", "channel"], observed=False)[["value"]]
77     .mean()
78     .reset_index()
79 )
80
81 # Filter to relevant frequency bands:
82 freq_bands_of_interest = ["delta", "theta", "alpha", "beta"]
83 df = df[df.band.isin(freq_bands_of_interest)]
84 df["band"] = df["band"].cat.remove_unused_categories()
85
86 g = sns.FacetGrid(
87     df_mean, col="condition", col_order=['OVTK_GDF_Left', 'OVTK_GDF_Right'], margin_titles=
         True
88 )
89 g = g.map(
90     sns.violinplot,
91     "channel",
92     "value",
93     "band",
94     cut=0,
95     palette="deep",
96     order=["C3", "P2", "C4"],
97     hue_order=freq_bands_of_interest,
98     linewidth=0.5,
```

```
).add_legend(ncol=4, loc="lower center")

g.map(plt.axhline)#, **axline_kw)
g.set_axis_labels("", "ERDS")
g.set_titles(col_template="{col_name}", row_template="{row_name}")
g.fig.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.3)
```

B.2. Matlab

Noise filtering demonstration

```
%Author: Niels van Duivendijk

% Parameters
Fs = 1000; % Sampling frequency
T = 1/Fs; % Sampling period
L = 1500; % Length of signal
t = (0:L-1)*T; % Time vector

% Generate random noise signal
noiseSignal = randn(size(t));

% Compute the Fourier transform of the signal
Y = fft(noiseSignal);

% Butterworth filter design
n = 5; % Order of the filter
Wn = 0.2; % Normalized cutoff frequency (0 < Wn < 1)
[b, a] = butter(n, Wn); % Filter coefficients

% Apply the Butterworth filter to the noise signal in the time domain
filteredSignal = filter(b, a, noiseSignal);

% Compute the Fourier transform of the filtered signal
Y_filtered = fft(filteredSignal);

% Compute the single-sided spectrum of the filtered signal
P1_filtered = abs(Y_filtered/L);
P1_filtered = P1_filtered(1:L/2+1);
P1_filtered(2:end-1) = 2*P1_filtered(2:end-1);

% Define the frequency domain f for the filtered signal
f = Fs*(0:(L/2))/L;
f_filtered = Fs*(0:(L/2))/L;

% Plot the original and filtered frequency domain signal
figure;
subplot(2,1,1);
plot(f, abs(Y(1:L/2+1)/L)*10, 'blue');
title('Random noise frequency spectrum');
xlabel('Frequency (f)');
ylabel('|N(f)|');
ylim([0,1]);
xlim([0,300]);

% Frequency response of the filter
[h, w] = freqz(b, a, L/2+1, Fs);

% Plot the magnitude response (normalized frequency)
hold on; % Hold on to the current plot
plot(w, abs(h), 'red','LineWidth',2); % Plot the frequency response in red
legend('Noise Signal', 'Butterworth Filter');
hold off; % Release the plot hold

subplot(2,1,2);
plot(f_filtered, P1_filtered*5, 'blue');
title('Filtered random noise frequency spectrum');
xlabel('Frequency (f)');
ylabel('|N_{filt}(f)|');
ylim([0,1]);
```

```
60 xlim([0,300]);
```