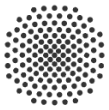

University of Stuttgart
Institute for Visualization and Interactive Systems

Signal Processing and Analysis of Human Brain Potentials (EEG)

Team : Capybara
Members : Samar Siddiqui [Matriculation : 3761951]
Prasoon Tiwari [Matriculation : 3768310]
Devesh Talsania [Matriculation : 3760334]

Task : Pattern Symmetry
Analysis : Event Related Potentials (ERP)
Paper : Symmetry perception and affective responses: A combined EEG/EMG
Authors : Alexis D.J. Makin, Moon M Wilton, Anna Pecchinenda, Marco Bertamini.

Supervisor : Prof. Dr. Benedikt Ehinger

Abstract

Event-Related Potential (ERP) analysis is a famous method in understanding the brain's response to specific stimuli. In this report, we explore the neural mechanisms that is linked with the visual perception and brain processing by analysing EEG data collected from subjects who were shown two types of visual patterns: Symmetrical and random (Asymmetrical).

The study focuses on identifying and comparing key ERP components P1 and N1, which are associated with early visual processing and Sustained Posterior Negativity (SPN). Preprocessing techniques, including Bandpass filtering, Artefacts removal, and Baseline correction, were applied to clean the data, which makes it better for Analysis.

This study contributes to understanding the brain's visual and cognitive processing. It highlights the usability of ERP analysis in understanding neural responses to stimuli. Future work could extend this analysis by incorporating advanced machine learning techniques and exploring individual variances in neural responses.

Table of Contents

S. No.	Title	Section	Page
1	Introduction	-	4
2	Theoretical Background	2.1 Electroencephalography (EEG)	5
		2.2 Event Related Potentials (ERPs)	5
		2.3 Visual processing	5-6
3	Experiment	3.1 Participants	6
		3.2 Apparatus	6
		3.3 Design	7
		3.4 Procedure	7
		3.5 Stimuli	7
4	Methodology	4.1 Data for analysis	7
		4.2 Software, Tools and Libraries used.	7
		4.3 Loading of Data	7-8
4.4	Preprocessing	4.4.1 Filtering	8-9
		4.4.2 Artefacts Removal using ICA	9-13
		4.4.3 Epoching or Segmentation	14-15
		4.4.4 Baseline Correction	15-16
5	Analysis	5.1 Calculating evoked response	16-17
		5.2 Finding peaks P1 and N1	17-18
		5.3 plotting graphs for evoked response for PO8 and PO7	18-19
		5.4 Useful Data extraction for all subjects for further analysis	19-21
		5.5 Comparison of PO8 and PO7 for ‘symmetrical’ and ‘asymmetrical’	21-23
6	Results	-	23
7	Additional analysis	-	23-24
8	Conclusion	-	25
8	References	-	25-26

1. Introduction

The human brain is a complicated system that processes a lot of information daily. Among these informations, vision plays main role in perceiving and interpreting the environment, just like any main character in movies (Just for reference). The electroencephalogram (EEG), a technique for recording brain activity, has become a basis for understanding neural processes associated with visual perception. Event-Related Potentials (ERPs), derived from EEG data, are time-locked responses to specific sensory, cognitive, or motor events. They provide valuable insights into the neural activity with respect to time, enabling researchers to examine how the brain processes different types of information when presented in front of the subject [10].

Symmetry is a fundamental feature in visual perception, as symmetrical patterns are often perceived as more aesthetically pleasing and meaningful than random patterns [1]. This raises several questions about the neural mechanisms related to the preference for symmetry and also the efforts required to process random patterns by human brain. Understanding these mechanisms is crucial for advancing our knowledge of human perception and cognitive functions.

This report focuses on analysing EEG data using ERP techniques to investigate the brain's response to symmetrical and random patterns. By examining key ERP components such as P1, N1, which are associated with early visual processing and cognitive evaluation, we aim to identify distinct neural signatures for these stimuli. The focus will be. Mostly on occipital electrodes positions, because earlier studies [3][4] suggests the positions have higher variable activities in pattern recognition cases. Our study uses advanced signal processing methods to preprocess EEG data, extract ERPs, and derive meaningful insights from the result. The objectives of this study are as follows :

- To analyse EEG data using ERP techniques to understand the brain's response to symmetrical and random patterns.
- To identify and compare key ERP components associated with the processing of these patterns.
- To interpret the results in the context of existing literature on visual perception and cognitive processing.

This report is structured as follows: Section 2 provides the theoretical background, including an overview of EEG, ERPs, and visual processing. Section 3 describes the Experiment conducted. Section 4 Describes methodology employed in the study, including data acquisition, preprocessing, and analysis techniques. Section 5 presents the Analysis, while section 6 present the result of the analysis. The report concludes with a summary of the study's outcomes and suggestions for future research.

2. Theoretical Background

2.1 Electroencephalography (EEG)

An Electroencephalography (EEG) is a non-invasive technique used to measure electrical activity in the brain. It records voltage fluctuations resulting from the activity of neurons, particularly those in the cerebral cortex, through electrodes placed on the scalp. These electrical signals are generated by postsynaptic potentials of neurons and provide a real time measure of brain function.

2.2 Event Related Potentials (ERPs)

Event-Related Potentials (ERPs) are time-locked brain responses that occur as a direct result of specific sensory, cognitive, or motor events. They are derived from the electroencephalogram (EEG) by averaging EEG segments that are aligned to the onset of a stimulus or event. This averaging process filters out unrelated neural activity, isolating the event-specific components. ERPs are composed of distinct components, such as P1, N1, etc. named based on their polarity (positive or negative) and timing (First, second). These components are associated with different stages of neural processing. ERPs are widely used in neuroscience and psychology to investigate brain functions related to perception, attention, memory, and language. They also have clinical applications in diagnosing neurological and psychiatric conditions, such as Alzheimer's disease and schizophrenia [12]. Our focus is on P1 and N1. These are early components of the ERP waveform observed in EEG data. These components are closely associated with sensory processing, particularly in response to visual and auditory stimuli [11][10]. They are among the first brain responses to an external event and provide insights into early attentional

P1 : P1 (also called P100) is a positive deflection in the ERP waveform, occurring approximately 80–130 ms after the onset of a stimulus. It Peaks around 100 ms and it’s Polarity is Positive (upward deflection on the graph). The Amplitude is Influenced by stimulus intensity and attention.

N1 : N1 (also called N100) is a negative deflection in the ERP waveform, occurring approximately 90–150 ms after stimulus onset. It Peaks around 100 ms but varies depending on the stimulus type (visual, auditory, or somatosensory). The polarity is negative (downward deflection on the graph). The amplitude is sensitive to stimulus intensity, attention, and task relevance.

Sustained Posterior Negativity (SPN) : The Sustained Posterior Negativity (SPN) is an ERP component associated with the processing of visual symmetry and regularity. It reflects sustained neural activity in the posterior occipito-parietal brain regions and is typically observed in tasks involving the perception of symmetrical patterns compared to random patterns. The SPN is a late ERP component that begins around 300 ms after stimulus onset and continue for the duration of the stimulus presentation or task engagement. It is most prominent over occipito-parietal electrodes (e.g., PO7 and PO8) and reflects activity in the extrastriate visual cortex. The SPN is a negative-going waveform, meaning that the electrical potential recorded at these posterior electrodes is more negative for symmetrical stimuli than for random stimuli.

2.3 Visual processing

Visual processing refers to the series of neural mechanisms that the brain uses to interpret and make sense of visual information captured by the eyes. It begins with the retina in the eye, where

light is converted into electrical signals, and continues through the path to the visual cortex and other brain regions [7][8][9].

Stages of Visual Processing :

- Retinal Processing : The retina contains photoreceptors (rods and cones) that detect light intensity and colour. The signals are transmitted to retinal ganglion cells and sent to the brain through the optic nerve.
- Optic Chiasma and Pathways : At the optic chiasma, signals from each eye are split, with information from the left visual field going to the right hemisphere and vice versa.
- Lateral Geniculate Nucleus (LGN) : Located in the thalamus, the LGN processes basic visual features like edges, contrast, and colour.
- Primary Visual Cortex (V1) : In the occipital lobe, V1 processes fundamental visual features such as orientation, spatial frequency, and motion.
- Higher-Order Processing : Signals are sent to higher-order areas like the ventral stream, which processes object recognition, and the dorsal stream, which handles spatial relationships and motion.

Role of Symmetry in Visual Processing :

Symmetry is a key feature in visual perception, as the brain is naturally aware of symmetrical patterns. Studies suggest that symmetrical stimuli are processed more efficiently, evoking stronger neural responses [2], as they require less cognitive effort to decode.

3. Experiment

In the Experiment, participants were presented with two types of stimuli : black and white abstract patterns with two-fold reflectional symmetry, or equivalent random patterns (Fig. 1). The patterns were on the screen for 3 s, after which participants pressed one button if they had seen a Symmetrical pattern and another for a random pattern. The EEG data was simultaneously recorded from the scalp and EMG from muscles on the face, most importantly from the left Zygomaticus Major (ZM) and Corrugator Supercilii (CS, frowning muscle).

3.1 Participants

Twenty-four participants (age 18 to 41, 8 male, 0 left-handed) took part. They were reimbursed for their time. All had normal or corrected-to normal vision and were naive respect to the hypotheses of the study. The study had approval from the local ethics committee.

3.2 Apparatus

EEG data sampled continuously at 512 Hz from 64 scalp electrodes that were embedded in an elasticised cap and arranged according to the international 10–20 system. Two additional electrodes, called common mode sense (CMS) and driven right leg (DRL) were used as reference and ground, respectively. Two facial electrodes were positioned over the left smiling muscle (Zygomaticus Major, ZM) and two were positioned over the left frowning muscle (Corrugator Supercilii, CS) muscle. The arrangement of facial electrodes followed standard guidelines [16]. Horizontal and vertical electrooculograms were recorded to detect eye movements and blinks in each condition. VEOG and HEOG electrodes were positioned above and below the right eye, and on the outer canthi of both eyes, respectively. Facial muscle and EOG activity was recorded using the external channels of the BioSemi system and also sampled at 512 Hz.

3.3 Design

This Experiment involved a single within-subjects factor [shape, (symmetrical, random)] with 80 trials of each condition. Symmetrical and random patterns were presented in a randomised sequence for each participant.

3.4 Procedure

Each trial started with a fixation cross, which was on the screen for 1.5 s, and was followed by a black and white pattern, which was on the screen for 3 s. The trial structure is shown in Fig. 1. At the end of each trial, participants were prompted with a response screen to report whether they had seen a symmetrical or a random pattern. They had up to 3 s to enter an appropriate response. In the Experiment, participants pressed one button for reflectional symmetry, and another for random patterns.

3.5 Stimuli

Stimuli were black and white patterns. In each quadrant, there were nine square elements, four of which were black and five were white. The size of the elements varied between 0.25° and 1° of visual angle, and the orientation was either 45° or 90°. The position nearest the central fixation cross was always white, so the fixation cross was never occluded. The background was a black circle and a white diamond (Fig. 1). The circle had a diameter of 5.11°.

4. Methodology

4.1 Data for analysis

The data used for analysis is downloaded from NEMAR website [17]. Reference to which was provided on the course website. The data is in BIDS format. There is a separate data file of recordings for each participant. There are 64 EEG channels and 8 EOG channels.

4.2 Software, Tools and Libraries used.

The analysis was done using Python Programming language. The platform used for the purpose is Jupyter. The Libraries used are Pandas, Matplotlib, numpy and most importantly MNE-Python. MNE is an Open-source Python package for exploring, visualising, and analysing human neurophysiological data: MEG, EEG, sEEG, ECoG, NIRS, and more. We chose MNE because the data is in BIDS and it is easier to process BIDS data using MNE. Also, MNE has a smooth operation with preprocessing of data using pipelines [15].

4.3 Loading of Data

The data is in BIDS format, we are using MNE library of python to load the data for analysis. Which is done in the way depicted in the code (C1).

Here, subject_id refers to the participant data File name/ID. Passing sunID will load data for that subject/ participant. *set_montage* function is used because EEG data typically include signals from multiple electrodes, but the data itself does not include information about where these electrodes were placed on the scalp. A montage provides this placement information, mapping each channel to a specific location in 3D space. Many EEG analyses, such as source localisation, topographical plotting, or spatial filtering, which will be used in this report later on, require precise knowledge of electrode positions. Setting the montage ensures that these analyses are grounded in the correct configuration.

```
[3]: #loading the BIDS data
def loadData(sunId):
    bids_root = "/Users/sammy/Documents/SP EEG/pro_bids"
    #specify id in individual notebook
    subject_id = sunId
    bids_path = BIDSPath(subject=subject_id,task="jacobsen",datatype='eeg', suffix='eeg',root=bids_root)
    raw = read_raw_bids(bids_path)
    raw.load_data()
    raw.set_montage('standard_1020',match_case=False, on_missing='ignore')
    #dropping because these channels are causing troubles in ICA.
    raw.drop_channels(['EXG1', 'EXG2', 'EXG3', 'EXG4','EXG5', 'EXG6', 'EXG7', 'EXG8'])
    return raw
```

Fig 1 : Code for loading data

The *raw.drop_channels()* function is used to remove 8 channels that are related to EOG. Reason for doing this is because when we tried to run data cleaning techniques, it was showing that these channels are either faulty or does not have relevant data, the system couldn't map these EOG channels to electrode positions for EEG analysis. In short, these channels are EOG channels and are not used in analysis of EOG. These channels are 'EXG1', 'EXG2', 'EXG3', 'EXG4','EXG5', 'EXG6', 'EXG7' and 'EXG8'. The output after data loading is shown in Fig 2.

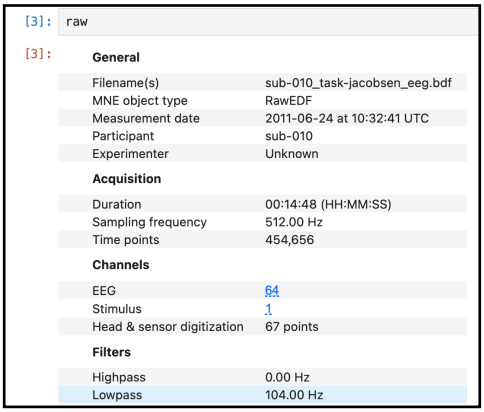

Fig 2 : Output after loading data

4.4 Preprocessing

Preprocessing of data refers to the set of techniques applied to raw data to clean, transform, and prepare it for analysis. In the context of EEG and ERP studies, preprocessing is crucial to ensure the reliability and accuracy of results, as raw EEG data often contains noise and artefacts that can obscure meaningful signals such as eye blinks, muscle movements etc [12].

4.4.1 Filtering

Filtering of EEG data is a preprocessing step used to remove unwanted noise and focus on specific frequency components relevant for the analysis. EEG signals often contain various types of noise, such as low-frequency drifts, high-frequency electrical interference, and artefacts such

```
[4]: #Filtering
def filterData(raw):
    #filtered at highpass 0.05Hz and Lowpass 50Hz
    raw.filter(l_freq = 0.05 , h_freq = 50)
    return raw
```

Fig 3 : Code for filtering data for 0.05 Hz - 50 Hz range

as eye blinks, muscle movements etc., which can be confusing and it may deflect us from finding meaningful neural activity. Filtering enhances the signal-to-noise ratio by removing these irrelevant frequency components. Another advantage of filtering is that we can extract the frequency range from the data, that we need to focus [13]. In short, we just pick what is of our interest and throw away rest of the Trash. That is why, it is an important step for pre-processing to consider. The filtering is done using Filtering function in MNE. See Fig 3.

The filtering of EEG data is done for highness 0.05Hz to lowpass 50Hz which is a common preprocessing step remove unwanted noise and retain frequency components of interest. The ERP components P1 and N1 typically occur at low frequencies (below 30 Hz). Filtering out frequencies outside this range focuses the analysis on the signals of interest while minimising noise. But if we shorten our range for filtering, then meaningful information may get lost [13]. So, It's like we have to choose optimal range.

Why did we chose 0.05 Hz for high-pass ? : we used 0.05 Hz as high-pass filter to remove slow drifts and baseline wander caused by movement, respiration, or electrode impedance changes. And also to ensure that the ERP components P1, N1 and SPN are not distorted by low-frequency artefacts.

Why did we chose 50 Hz for low-pass ? : we used 50 Hz as low-pass filter to remove high-frequency noise from the signal, such as muscle artefacts like jaw clenching, eye blinks etc or external electrical interference eg. power line noise at 55/60 Hz). This filter reduces many other high-frequency noise, enabling clearer visualisation and measurement of ERPs like SPN, which is important for assessing symmetry processing. The 0.05–50 Hz filter range is a standard choice in ERP research because it balances noise reduction with the preservation of meaningful neural signals. After applying filtering, the *raw* object reflects the changes as circled Fig 4.

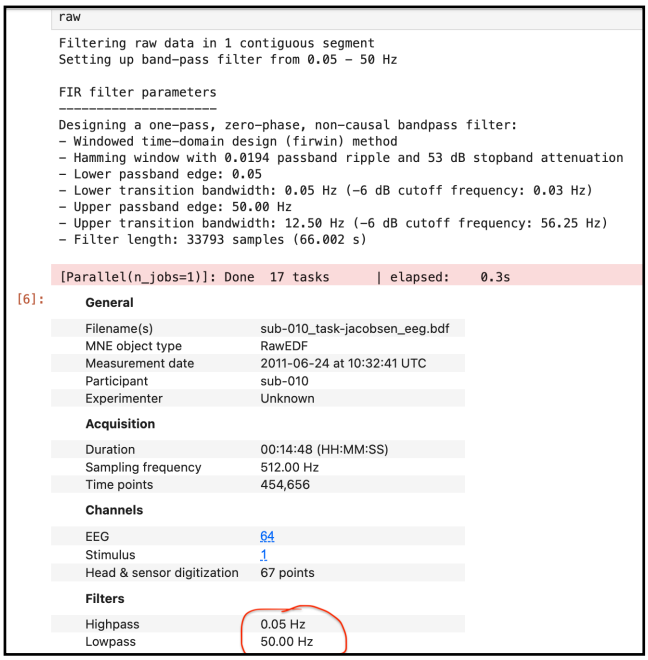

Fig 4 : raw object after filtering

4.4.2 Artefacts Removal using ICA

In this section we discuss the most important part of the preprocessing which is artefacts removal. We used Independent Component Analysis (ICA) for identifying and removing artefacts. ICA is a mathematical technique used in EEG signal processing to separate a set of mixed signals into their independent underlying components. It is like separating each type of grain from a bowl containing mixture of these grains all together. In the context of EEG, ICA helps to decompose the recorded brain signals into sources that may represent neural activity, artefacts, or other physiological processes.

Why do we need to perform ICA? : EEG signals often contain artefacts such as eye blinks, muscle movements, and heartbeats that can obscure the brain's electrical activity. ICA separates these artefacts into separate components, allowing their removal while preserving neural signals [18] [19]. Also, EEG electrodes record mixed signals from various neural and non-neural sources for eg. power lines. ICA separates these signals into statistically independent components. By removing artefacts and irrelevant sources, ICA enhances the quality of the data for further analysis.

We performed ICA using ICLabel in MNE python [15]. It is a machine learning-based tool designed to automatically classify independent components (ICs) derived from EEG data into distinct categories. It simplifies the process of identifying the sources of ICs (e.g., brain activity, artefacts) by using a pre-trained neural network on a dataset of manually labeled ICs. It simplifies the process of analysis for us by identifying whether a given IC represents neural activity or an artefact. Which ultimately reduces the manual effort required to inspect and classify ICs during EEG preprocessing. It removes the components that it classifies as other than Brain. Or we have to specify which classified components to keep and which one should be discarded.

ICLabel classifies ICs into seven categories :-

1. Brain : Neural activity originating from cortical areas. Typically localised and matches expected EEG frequency bands (e.g., alpha, theta).
2. Muscle : Artefacts caused by muscle contractions. Characterised by high-frequency activity (>20 Hz) and irregular patterns.
3. Eye : Artefacts from eye blinks. Dominates frontal electrodes and exhibits low-frequency activity.
4. Heart (ECG) : Artefacts caused by cardiac activity. Periodic patterns synchronised with the heartbeat.
5. Line Noise : Electrical interference, often at 50 Hz or 60 Hz. Appears as narrowband noise in the frequency spectrum.
6. Channel Noise : Irregular noise due to poor electrode contact or hardware issues. Localised to specific channels.
7. Other : Components that do not fit into the above categories, but can be one of belonging to one of the above category also, maybe because of the error of ICA.

Choosing the number of components to run ICA is another important task. Because if we choose a large number of components, then ICA will divide the data into that much components which can lead to a lot of components classified as noise/artefacts. And then these components would be removed. In short, if we choose a large number for number of components while running ICA, there are higher chances of loosing useful data. If we choose the components to be less. Then, ICA would even classify noises and other artefacts as useful, which is not good for our analysis. Because we need to clean the data as much as possible. So, it is important to choose optimal number for number of components [19].

Why did we choose number of components as 20 ? : as discussed above, it is important to choose the number of components in the optimal range. We first ran ICA with 10 components, the results we got after running ICA were something like only 6-7% of the components were discarded, averaging all the participants. Which is not good for analysis, as it may still contain a lot of noises. Then we ran the analysis with 30 components, from the results, approximately 23% of the components were discarded, averaging all the Participants. Which is nearly quarter of the data. So, it is not good as it may have discarded a lot of data, which is useful. So, we came up with using 20 components. In this case, only 14-15% of the components were discarded which is a good data cleaning result for us by far. Not too many components are discarded, neither too less.

ICA was implemented in the way depicted in fig 5. First we have to create an Object for ICA which is a class in *MNE.preprocessing* in python. We specify number of components, random state and preferred method in parameters while defining the ICA object. The FastICA (Fast Independent Component Analysis) method is widely used in EEG preprocessing because it is computationally efficient, robust, and well-suited for separating independent sources in datasets. It minimises the risk of losing meaningful brain activity while removing artefacts. FastICA assumes that independent sources are non-Gaussian, except for Gaussian component. This is consistent with the nature of EEG data. Neural signals and artefacts are typically non-Gaussian. FastICA uses this assumption to optimise the separation of components more effectively than simpler methods like Principal Component Analysis (PCA). It also allows for different contrast or objective functions to estimate independent components e.g, logcosh, exp, or cube. This feature makes it adaptable to different datasets and requirements. FastICA has been extensively validated in EEG preprocessing pipelines, showing good performance [20].

```
[10]: #ICA
def performICA(raw):
    #intialising ICA
    ica = ICA(n_components=20, random_state=42, method='fastica')
    ica.fit(raw)

    # Using mne_icalabel to classify components
    ic_labels = label_components(raw, ica, method='iclabel')

    # Printing the classifications
    print("ICLabel classifications:")
    print(ic_labels)

    # Identify Eye, muscle artifacts and channel noise
    Artifact_comp = [i for i, label in enumerate(ic_labels['labels']) if label == 'eye blink' or 'channel noise' or 'muscle artifact']
    print(f"Artifacts identified: {Artifact_comp}")

    # Excluding artifacts related components
    ica.exclude = Artifact_comp

    # Apply ICA cleaning to the raw data and plotting components
    raw_clean = ica.apply(raw)
    ica.plot_components()
    x = range(0,20)
    ica.plot_properties(raw,picks=x,psd_args={'fmax': 35.},reject=None);
    #labels of all components
    print(ic_labels["labels"])
```

Fig 5 : code of implementing ICA on the data

Then we fit our data in ICA object to separate components. Now, we have to define a new object of *iclabel.label_components* which will classify the data into different labels (mentioned on page 10). We can see which component is classified to which label by printing the object.

Now, comes the main task for data cleaning, which is removing the components that are classified by ICA that are not brain activity, which is, other than ‘brain’ and ’other’ labels in the classification. We have to iterate through all the components and see their labels, if it is irrelevant then we add it to a list. We have to store all the irrelevant components in this list like ‘muscle artefacts’, ‘eye blink’,’channel noise’. Then we run *ica.exclude* and provide the list that we stored, that will exclude the components that we do not want in our data. Finally, we apply these changes to the *raw* object. We depicted all the topographies of all the components by plotting each separately by *ica.plot_components()* function. The results can be seen in fig 6. We have also displayed properties of all the components separately, which includes spectrum and power spectral density (PSD) graph, which helps us to identify noises in the channel (Fig 7).

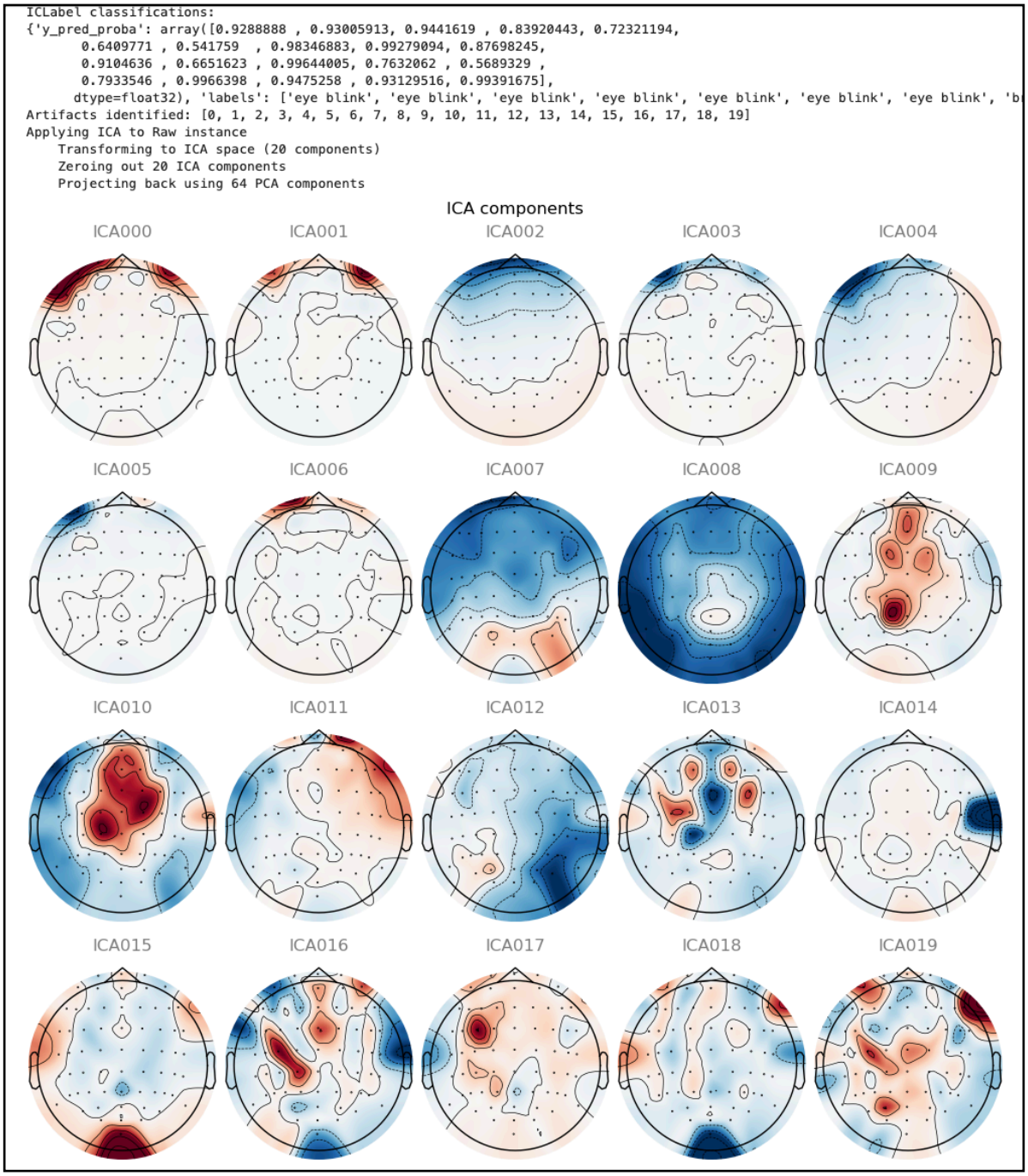

Fig 6. Shows ICA components topography for subject 10.

In the figure, we can see first component ICA000. The activity in the topography is clearly visible in the front of the head, specifically in the eye region. After seeing this, we can conclude that the first component that is ICA000 can be classified as ‘eye blink’. Our analysis also classify this to ‘eye blink’ which can be seen in the figure itself at the top where iclabel classification List is printed. The properties of the component is also shown in the code separately, snapshot can be seen in fig 7(a).

We can also see third component ICA002. The activity in the topography is clearly visible in the front of the head, specifically in the eye region. After seeing this, we can conclude that the first component that is ICA000 can be classified as ‘eye blink’. Our analysis also classify this to ‘eye blink’ which can be seen in the figure itself at the top where iclabel classification List is printed. The properties of the component is also shown in the code separately, snapshot can be seen in fig 7(b).

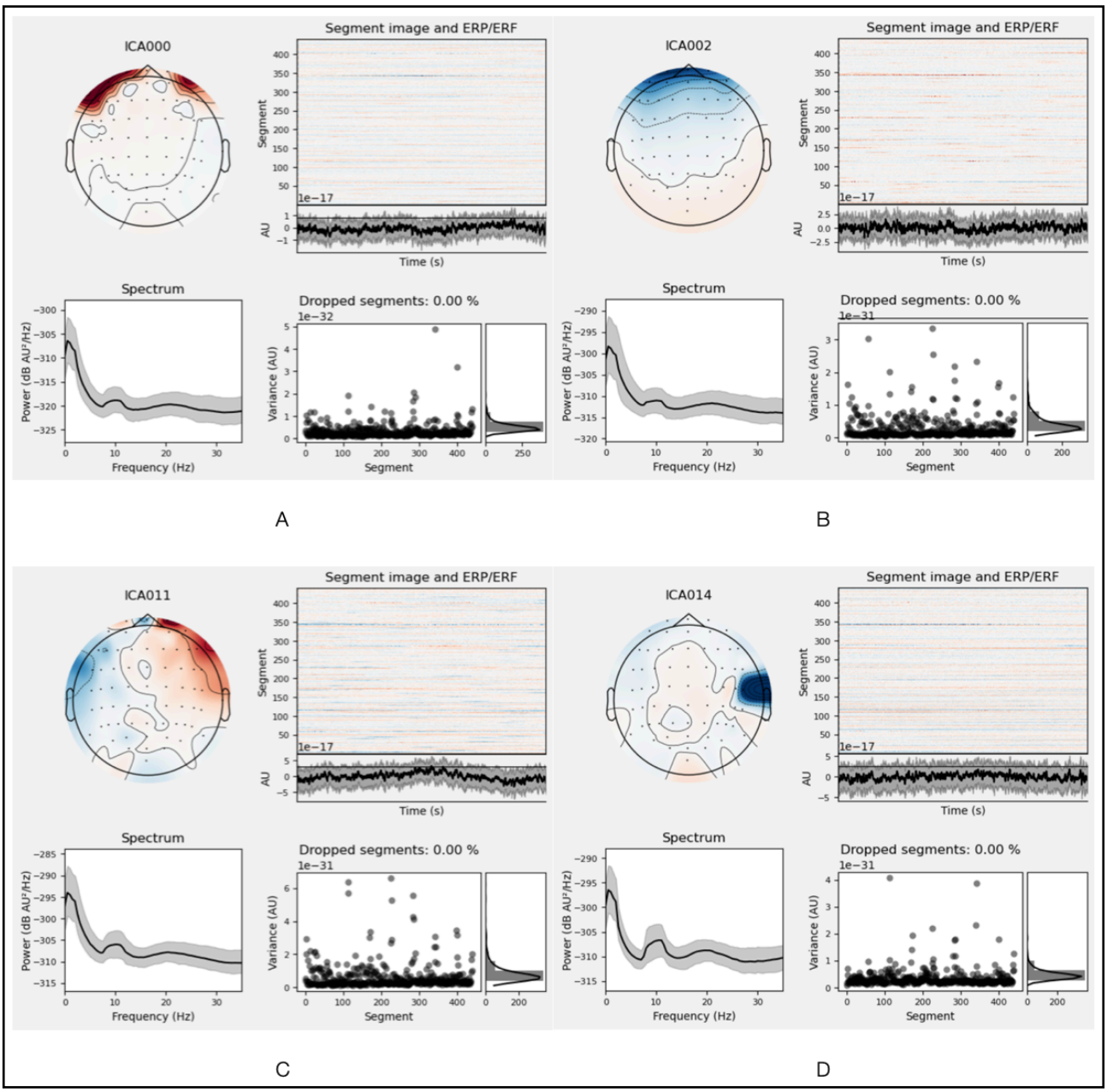

Fig 7 : Showing properties of components separately

If we notice in the seventh component that is ICA014, we can see there are some positions having no activity and also at some points it is very high. It may be due to movement of electrode or head or can be cause by some other factor eg, sweat. These are considered to be channel noises and this is not good for data analysis, so it can not be considered to be included. As we can see it to be channel noise, so does our code. It classifies this component as channel noise, can be seen in the list. The properties of the component is also shown in the code separately, snapshot can be seen in fig 7(d).

We can see twelfth component ICA011. The activity in the topography is clearly visible as in some places it is higher and lower in some regions, depicting bipolarity. After seeing this, we can conclude that the this component that is ICA011 can be classified as ‘brain’ activity. Our analysis also classify this to ‘brain’ which can be seen in the list printed. The properties of the component is also shown in the code separately, snapshot can be seen in fig 7(d).

4.4.3 Epoching or Segmentation

Epochs in EEG data are short, time-locked segments of continuous EEG recordings that are extracted relative to specific events or stimuli. These segments represent the brain's response during a defined time window relative to the event of interest and are the foundation for event-related potential (ERP) analysis. These events can be sensory, cognitive, or triggers, such as visual stimuli, auditory signals, or button presses. Each epoch has a defined time window relative to an event, typically including a pre-event (baseline) period and a post-event period. Each epoch represents the brain's response to a single instance of the event, capturing its variability. Epoching will allow us to focus on brain activity associated with these events only. It is like focusing on specific points of data [12][10].

The epoching is done as shown in Fig 8. The event names in the downloaded data is termed as '1' and '3'. All the event named '1' are the events where symmetrical stimuli was presented whereas '3' are all the events where Asymmetrical stimuli occurred. So, we mapped these values by using a dictionary, '1' mapped to 'Symmetrical' and '3' mapped to 'Asymmetrical' for making our analysis interpretations better. *eventName* function returns all the event instances defined in the data. *getEpochs* processes these events object and returns the epochs after segmenting. The ouput/ details after epoching the *raw* object can be seen in fig 9.

[6]:
```
#for getting events
def eventName(raw):
    # Specify stimulus channel
    events = find_events(raw, stim_channel='Status', initial_event = True)
    print(events)
```

[7]:
```
#epoching the data
def getEpochs(raw):
    events = find_events(raw, stim_channel='Status')  #stimuli channel
    event_id = {
        'Symmetrical': 1,
        'Asymmetrical': 3
    }
    #epoching from -0.2s to 1s.
    epochs = Epochs(raw, events, event_id, tmin=-0.2, tmax=1.0, preload=True)
    return epochs
```

Fig 8 : generating epochs / segments for events

[10]:
```
#epoching the data (raw)
epochs = getEpochs(raw)
```

```
Trigger channel Status has a non-zero initial value of {initial_value} (consider using initial_event=True to detect this event)
Removing orphaned offset at the beginning of the file.
160 events found on stim channel Status
Event IDs: [1 3]
Not setting metadata
160 matching events found
Setting baseline interval to [-0.19921875, 0.0] s
Applying baseline correction (mode: mean)
0 projection items activated
Using data from preloaded Raw for 160 events and 615 original time points ...
0 bad epochs dropped
```

Fig 9 : epochs generated

Why did we chose time frame -0.2s to 1s ? : The duration of epochs should be chosen based on the expected latency of ERP components. In our case. We are considering P1, N1 and SPN. P1 and N1 are post stimuli hikes usually occurring between 80-200 ms. But for SPN, we need a little longer frame to analyse the waveforms after appearance of stimuli and after N1 hike. Whether the wave is staying negative for a while or not. So, we choose -0.2s to get pre-event, for baseline correction as well as considering pre-event brain activities. And upper limit of the frame as 1s to get just enough data for SPN analysis.

4.4.4 Baseline Correction

Baseline correction is a preprocessing step used in EEG and ERP analysis to that removes slow shifts, drifts or baseline offsets in the signal, ensuring that the neural activity of interest is accurately represented by normalising the signal. This ensures that the data represents relative changes in voltage rather than absolute values. It improves the comparability of trials and conditions. This step adjusts each epoch to a common reference level by subtracting the average signal value from a baseline Period. The code for the baseline correction is shown in Fig 10. The epochs are passed as parameters and it performs baseline correction for all the segments in epochs and return modified epochs. The mode of correction is Mean, means it is averaging the offsets for normalisation [10].

```
[8]: #applying baseline correction
     def baselineCorrection(epochs):
         #from -0.2s to 0s
         epochs.apply_baseline(baseline=(-0.2,0))
         return epochs
```

Fig 10 : code for baseline correction

Why did we use baseline correction ? : It Removes low-frequency drifts and artefacts caused by electrode movements or may be some physiological factors. And it also aligns epochs to a common reference, ensuring that post-event activity is comparable across trials. So, the the data analysis step has a starting point, where to begin. Another reason for using this is that it improves the interpretability of ERP components by ensuring they reflect event-related activity rather than baseline variations.

Why did we use the period -0.2s to 0s ? : While performing baseline correction, it is necessary to choose a baseline period carefully to avoid mixing it with pre-stimulus neural activities. In short, we don't need any irrelevant data/activity get mixed in the data that we are trying to

[12]: epochs

[12]:

General	
MNE object type	Epochs
Measurement date	2011-06-24 at 10:32:41 UTC
Participant	sub-010
Experimenter	Unknown
Acquisition	
Total number of events	160
Events counts	Asymmetrical: 80 Symmetrical: 80
Time range	-0.199 – 1.000 s
Baseline	-0.200 – 0.000 s
Sampling frequency	512.00 Hz
Time points	615
Metadata	No metadata set
Channels	
EEG	64
Stimulus	1
Head & sensor digitization	67 points
Filters	
Highpass	0.05 Hz
Lowpass	50.00 Hz

Fig 11 : epochs object after baseline correction

extract for analysis. If the baseline range were too short or improperly chosen, it could add noise into the ERP data, leading to misinterpretation of results.

In ERP studies, a 200 ms pre-stimulus (-0.2s) baseline is a common choice because it provides sufficient time to establish a stable baseline without overlapping with the response caused by the stimulus. The time window -0.2s to 0s is enough before the stimulus is presented to avoid addition of other neural activity or artefacts (e.g., muscle activity, eye movements) into the final data. Fig 11 shows the output after baseline correction is applied, it shows details of the epochs object. The time range and baseline details of the epochs object shows the changes have been applied.

5. Analysis

After preprocessing the data, we have *epochs* Object. It contains all the recordings of EEG data within the range -0.2s to 1s of the event, with baseline corrected and for all channels of all the similar events, for example, 'symmetrical' event. The data is now clean, as far as we are concerned. We now have to perform some calculations and visualisations for analysis.

5.1 Calculating evoked response

With the help of *epochs* object, we can now calculate *evoked* response. What is evoked response?. We have all the channels for all the events under same label, what we have to do is we have to average the recorded data for all the events, which are 80 events for 'symmetrical' and 80 for 'asymmetrical'. It will contain average values for each channel of all the instances where a specific type of event occurred. Fig 12 shows the implementation.

```
[14]: #computing evoked resonse for event type and then averaging it
      def calcEvoked(epochs):
          # for symmetrical
          evoked_symm = epochs['Symmetrical'].average()
          print("PLOT FOR SYMMETRICAL STIMULI")
          evoked_symm.plot()  # Basic waveform plot
          evoked_symm.plot_joint();  # Enhanced plot with topography
          # for asymmetrical/random stimuli
          evoked_asymm = epochs['Asymmetrical'].average()
          print("PLOT FOR ASYMMETRICAL / RANDOM STIMULI")
          evoked_asymm.plot()  # Basic waveform plot
          evoked_asymm.plot_joint();  # Enhanced plot with topography
          return (evoked_symm, evoked_asymm)
```

Fig 12 : evoked response

We are creating two variables for both the events, one for each. We can plot all the waveform generated in the evoked response in a graph that shows all the channels by *evoked_symm.plot().* This will provide us with some better insights, fig 13 shows the graph obtained after running *evoked_symm.plot().* In the graph, each line represent one channel / electrode position, the channels are colour coded according to the topography at top left. We can also plot a graph containing topography of head, which shows which electrode position had higher activity at what instances. Fig 14 shows the enhanced graph with topography. We can have clear insights about

at what position of the brain there were higher or lower potential changes, also helps in finding peaks.

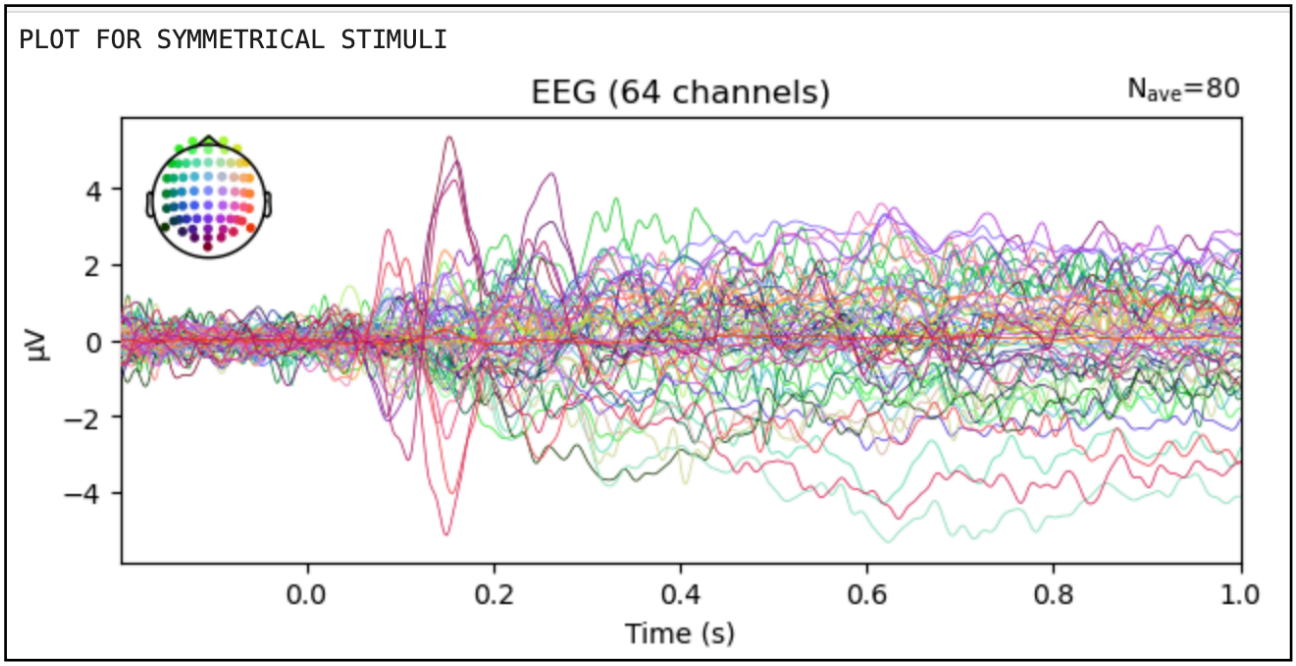

Fig 13 : evoked response ; average of all the epochs instances for symmetrical stimuli.

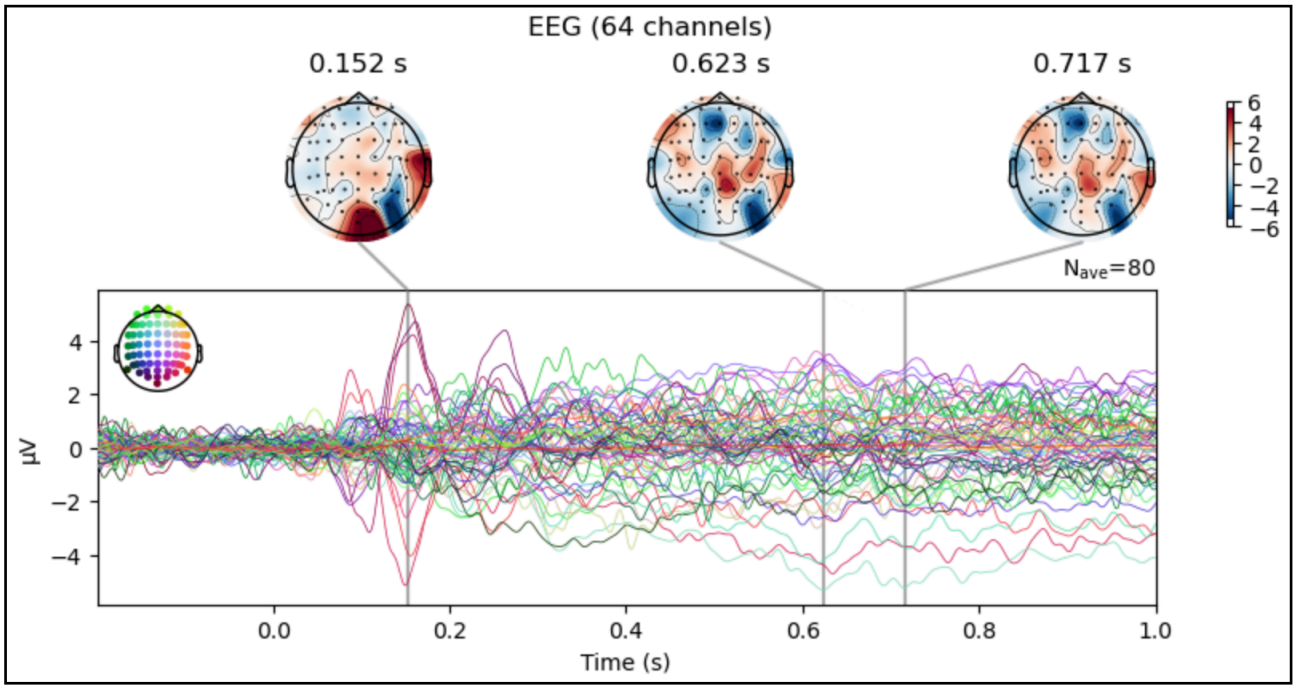

Fig 14 : graph with topography at peaks

5.2 Finding peaks P1 and N1

The next task is to find peaks, that is P1 and N1. We have to find 3 details, the time instance of the peak, the amplitude of the peak and the electrode position at which it occurs. We have to find for both the events separately. We will focus on positions like PO8 and PO7 more as these

positions are more related to image processing in the brain as suggested by previous studies and as discussed earlier in this report [3][4].

Fig 15 shows the code for finding the peaks and details. We used *get_peak()* function for identifying peak. We know that P1 occurs within 80-130ms after the event occurs, this defines the parameters tmin=0.08 and tmax=0.13. for getting P1, mode='pos' that is positive, while for N1 the mode='neg' that is negative. Return_amplitude is set to true because we also want the maximum aptitude. The function returns the electrode position of the peak, the latency (instance) of the peak and the amplitude of the peak in respective order.

Fig 16 shows the output of the code in fig 15. The position, amplitude and latency of peaks P1 and N1 are shown with the topography of the instance. This data is saved for further analysis.

```
[15]: #finding peak and latency for P1 and N1
def findPeak(evoked):
    # Find P1 (positive peak) within 80-130 ms
    p1, p1_latency, p1_amplitude = evoked.get_peak(tmin=0.08, tmax=0.13, mode='pos', return_amplitude=True)
    # Find N1 (negative peak) within 140-200 ms
    n1, n1_latency, n1_amplitude = evoked.get_peak(tmin=0.14, tmax=0.2, mode='neg', return_amplitude=True)
    print(f"P1: Latency={p1_latency*1000:.1f} ms, Amplitude={p1_amplitude*1000000:.3f} µV")
    print(f"N1: Latency={n1_latency*1000:.1f} ms, Amplitude={n1_amplitude*1000000:.3f} µV")
    print("P1 at : ", p1, " ; N1 at : ", n1)
    evoked.plot_topomap(times=[p1_latency, n1_latency])
    P1 = [p1, p1_latency, p1_amplitude]
    N1 = [n1, n1_latency, n1_amplitude]
    return [P1,N1]
```

Fig 15 : code for finding peak P1 and N1

```
[17]: #now we will find the N1 and P1 peaks ; their latnecy and amplitude also the channel it came from
#topography shows the brain involement
x = findPeak(evoked[0]) # this is for symmetrical stimuli
```

```
P1: Latency=87.9 ms, Amplitude=2.912 µV
N1: Latency=150.4 ms, Amplitude=-5.136 µV
P1 at :  PO8  ; N1 at :  PO8
```

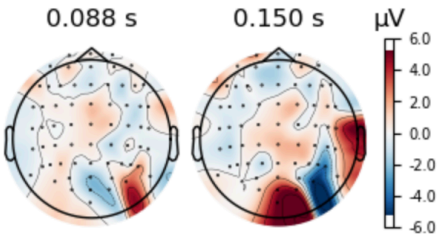

Fig 16 : P1 and N1 peaks for symmetrical stimuli event

5.3 plotting graphs for evoked response for PO8 and PO7

Our next step is to shift our focus to only those channels that matter for now, that are PO8 and PO7. So, we have to analyse evoked responses for these positions. By plotting graphs for 'symmetrical' and 'asymmetrical' events together we can see the difference in amplitudes for both peaks and also the sustained posterior negativity. We used the code in fig 17 to draw these graphs using *mne.viz.* Fig 18 shows the two graphs for PO8 and PO7 plotted by code in fig 17.

```
[16]: #plotting PO8 electrode events – Depicting P1 and N1
      def drawEvokedEvents(evoked_symm, evoked_asymm, pick):
          mne.viz.plot_compare_evokeds([evoked_symm, evoked_asymm], picks=pick)
```

```
[20]: #now we plot symmetrical and asymmetrical stimuli events averaged, side by side on a graph for channels picked above
      for i in range(0, len(pick)):
          drawEvokedEvents(evoked[0], evoked[1], pick[i])
```

Fig 17 : plotting graphs for evoked response for PO8 and PO7

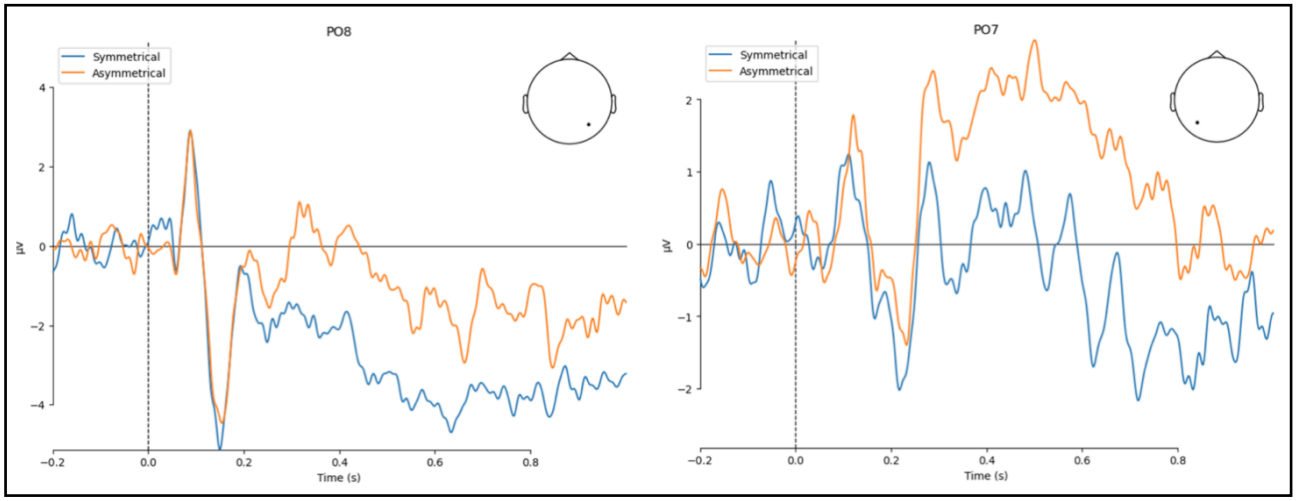

Fig 18 : graphs for symmetrical and asymmetrical stimuli events combined, of evoked response for PO8 and PO7.

5.4 Useful Data extraction for all subjects for further analysis

All the steps we performed above are applied to the data of only one participant (subject 10, sample). We have to perform these steps for all the participants EEG recordings and extract data from it. For that we load all the participants data, and perform above steps and store them in an array. Fig 19 shows the implementation. *loadEvoked()* function performs all the steps after loading data and retrieves *evoked* response data for all participants and append the object to compList. The *Evoked* object has the electrode position, the latency (instance) and the amplitude of the peak P1 and N1. And compList now has *evoked* object for all the participants.

```
[2]: def loadEvoked(subId):
         raw = loadData(subId)
         raw = filterData(raw)
         performICA(raw)
         epochs = getEpochs(raw)
         epochs = baselineCorrection(epochs)
         evoked = calcEvoked(epochs)
         return evoked
```

```
[14]: #all subject ids
      subIds = ['001','002','003','004','005','006','007','008','009','010','011','012','013','014','015','016','017','018','019','020','021','022',
```

```
[16]: # making list of all evoked variables from the data, subjects IDs.
      compList = []
      for i in range(0,24):
          temp = loadEvoked(subIds[i])
          compList.append(temp)
```

Fig 19 : data extraction

Our next step is separating evoked response data for ‘symmetrical’ event and ‘asymmetrical’ event into different list. Which is done by the code in fig 20. We need the data to be converted to pandas data frame because it is easier to run statistical analysis techniques on dataframes. The list *symEvoked* contains data of all the participants for ‘symmetrical’ events and *asymmEvoked* List has all ‘asymmetrical’ events. We picked only PO8 and PO7 channels for all the data that we are extracting because that is where our main focus is.

```
[2]: #for getting Data in Dataframe form
def getData(evoked):
    #making copy so that original object stays intact
    x = evoked.copy()
    x.pick(['PO8','PO7']) #picking these 2 channels
    # Get the data and time points
    data = x.data  # Shape is (n_channels, n_times)
    times = x.times  # Time points in seconds

    # Convert data to a pandas DataFrame
    df = pd.DataFrame(data.T, columns = ['PO8','PO7'])  # Transpose to make time points as rows
    df['time'] = times  # Add time column
    return df
```

```
[19]: #extract pandas dataframe for symmetrical stimuli for all evoked variables
symmEvoked = []
asymmEvoked = []
for i in range(0,24):
    temp_symm = getData(compList[i][0])
    symmEvoked.append(temp_symm)
    temp_asymm = getData(compList[i][1])
    asymmEvoked.append(temp_asymm)
```

Fig 20 : separating data based on event

Now, our next task is to combine recorded and cleaned data that we derived above into a single data frame. So that one data frame contains data for all participants for a single channel for a single event. In short, we will have 4 data frames ; for PO8 channel and ‘symmetrical’ event, for PO8 channel and ‘asymmetrical’ event, for PO7 channel and ‘symmetrical’ event and for PO7 channel and ‘asymmetrical’ event. Which is done by code in fig 21. Also, we are adding mean of the data as a new column. Save this data to a new csv file so that we do not have to run our code all over again.

```
[39]: #combining all the dataframes into one dataframe
#for PO8 symmetrical
combined_PO8_sym = pd.concat(sym_PO8, axis=1)
combined_PO8_sym.columns = ['PO8_1','PO8_2','PO8_3','PO8_4','PO8_5','PO8_6','PO8_7','PO8_8','PO8_9','PO8_10','PO8_11','PO8_12','PO8_13',
                            'PO8_14','PO8_15','PO8_16','PO8_17','PO8_18','PO8_19','PO8_20','PO8_21','PO8_22','PO8_23','PO8_24']
combined_PO8_sym['Avg_PO8'] = combined_PO8_sym.mean(axis=1)

combined_PO7_sym = pd.concat(sym_PO7, axis=1)
combined_PO7_sym.columns = ['PO7_1','PO7_2','PO7_3','PO7_4','PO7_5','PO7_6','PO7_7','PO7_8','PO7_9','PO7_10','PO7_11','PO7_12','PO7_13',
                            'PO7_14','PO7_15','PO7_16','PO7_17','PO7_18','PO7_19','PO7_20','PO7_21','PO7_22','PO7_23','PO7_24']
combined_PO7_sym['Avg_PO7'] = combined_PO7_sym.mean(axis=1)

combined_PO8_asym = pd.concat(asym_PO8, axis=1)
combined_PO8_asym.columns = ['PO8_1','PO8_2','PO8_3','PO8_4','PO8_5','PO8_6','PO8_7','PO8_8','PO8_9','PO8_10','PO8_11','PO8_12','PO8_13',
                             'PO8_14','PO8_15','PO8_16','PO8_17','PO8_18','PO8_19','PO8_20','PO8_21','PO8_22','PO8_23','PO8_24']
combined_PO8_asym['Avg_PO8'] = combined_PO8_asym.mean(axis=1)

combined_PO7_asym = pd.concat(asym_PO7, axis=1)
combined_PO7_asym.columns = ['PO7_1','PO7_2','PO7_3','PO7_4','PO7_5','PO7_6','PO7_7','PO7_8','PO7_9','PO7_10','PO7_11','PO7_12','PO7_13',
                            'PO7_14','PO7_15','PO7_16','PO7_17','PO7_18','PO7_19','PO7_20','PO7_21','PO7_22','PO7_23','PO7_24']
combined_PO7_asym['Avg_PO7'] = combined_PO7_asym.mean(axis=1)
```

```
[42]: combined_PO8_sym.to_csv('combined_PO8_sym.csv', index=True, header=True)

[43]: combined_PO7_sym.to_csv('combined_PO7_sym.csv', index=True, header=True)

[44]: combined_PO8_asym.to_csv('combined_PO8_asym.csv', index=True, header=True)

[45]: combined_PO7_asym.to_csv('combined_PO7_asym.csv', index=True, header=True)
```

Fig 21 : code for combining the data frames and storing as csv files

5.5 Comparison of PO8 and PO7 for 'symmetrical' and 'asymmetrical'

We have csv files with us containing the records that we need for comparison. We will load these files using pandas *read_csv()* function and run comparisons. Also, setting index for loaded pandas data frame for csv file is necessary, otherwise we would not be able to plot data. There will be 4 data frames as there are 4 csv files. We will plot all the columns, that represents electrode values of all the participants at specific instances averaged over all the epochs for one event. Fig 22 shows this implementation. We have plotted all the column with alpha 0.5 for reducing opacity except the average column.

```
[5]: plt.figure(figsize=(16, 12))
     for col in df.columns[:-1]:  # Exclude the 'Average' column
         plt.plot(df.index, df[col], alpha=0.5, label=col)  # Lower opacity

     # Highlight the average column
     plt.plot(df.index, df['Avg_PO8'], color='black', linewidth=2, label='Average', alpha=1)
     plt.xlabel('Time')
     plt.ylabel('Aplitude')
     plt.title('PO8 for Symmetry Pattern')
     plt.axhline(y=0, color='red', linestyle='-', linewidth=2)
     plt.legend(loc='best');
```

Fig 22 : code for plotting the data frame

Similarly, we will perform the same steps for all the data frames that we have to get 4 different graphs. Fig 23(a) shows the graph for PO8 electrode for 'symmetrical' stimuli averaged over all epochs for all the participants and bold line shows the average of all. Fig 23(b) shows the graph for PO8 electrode for 'asymmetrical' stimuli averaged over all epochs for all the participants and bold line shows the average of all. Fig 23(c) shows the graph for PO7 electrode for 'symmetrical' stimuli averaged over all epochs for all the participants and bold line shows the average of all. Fig 23(d) shows the graph for PO7 electrode for 'asymmetrical' stimuli averaged over all epochs for all the participants and bold line shows the average of all.

Analysing these graphs, we have a clear picture that in both the electrodes there are positive hikes for both events at around 80-100ms post event, and a negative hike at around 160-180ms post event. Also, for 'symmetrical' event for both the electrodes, the graph tends to be more negative after the Negative hike occurred as compared to the case of 'asymmetrical' event.

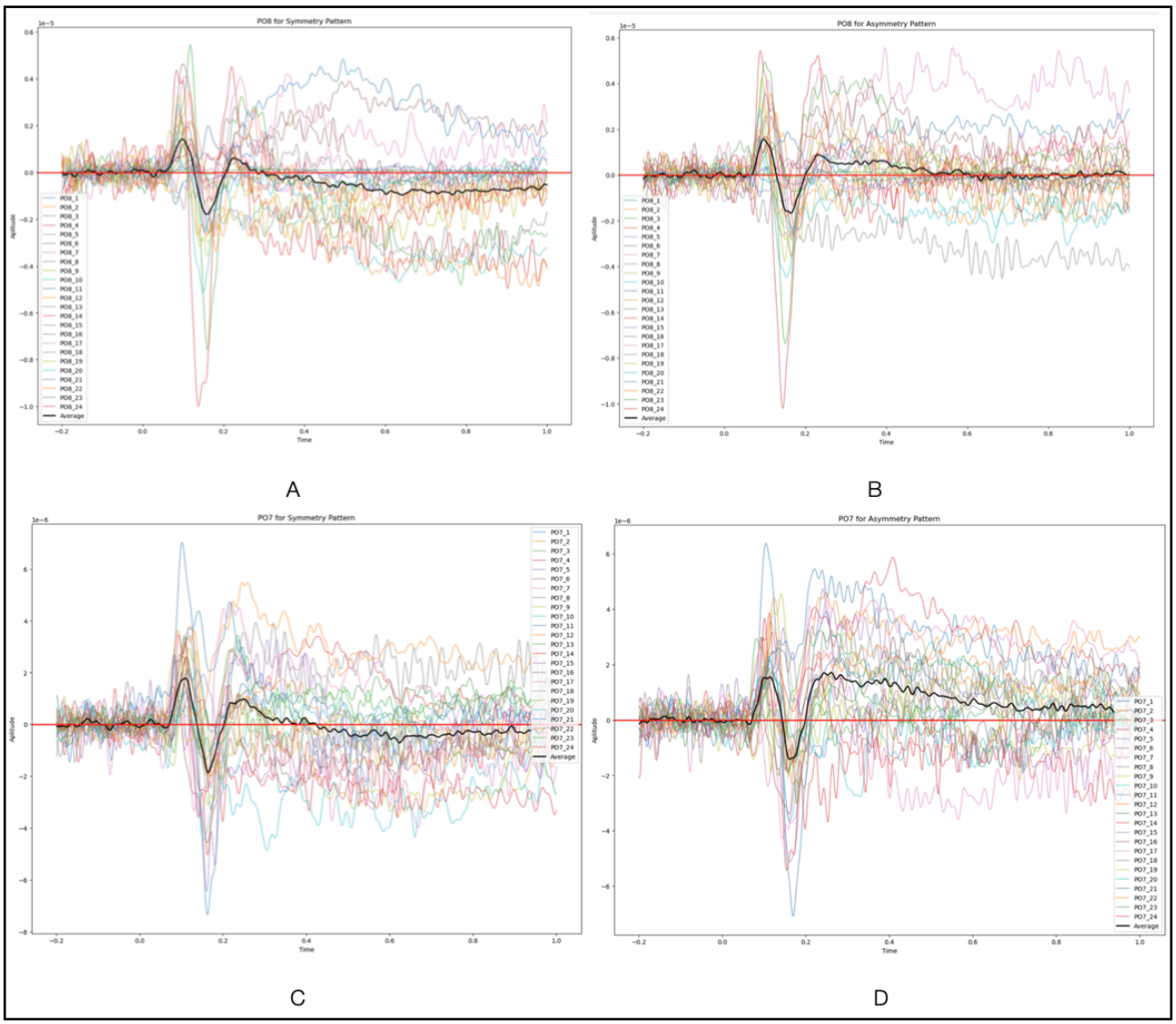

Fig 23 : graphs obtained for electrode positions and events

We will now extract this average columns and plot them side by side for both the events together to compare the waveform and differences. Fig 24 shows the code, while fig 25 shows the graphs obtained. The black line represents ‘symmetrical’ stimuli while the red one represents ‘asymmetrical’ stimuli, both are the averages column from the data frames we obtained above. The green line divides the region for positive and negative amplitude.

```
plt.figure(figsize=(12, 8))
# Highlight the average column
plt.plot(df.index, df['Avg_PO8'], color='black', linewidth=2, label='Symmetry Average PO8', alpha=1)
plt.plot(df2.index, df2['Avg_PO8'], color='red', linewidth=2, label='Asymmetry Average PO8', alpha=1)
plt.xlabel('Time')
plt.ylabel('Aplitude')
plt.title('PO8 for Asymmetry and Symmetry Pattern')
plt.axhline(y=0, color='green', linestyle='-', linewidth=2)
plt.legend(loc='best');
```

Fig 24 : code for plotting average columns side by side for both events for one electrode position

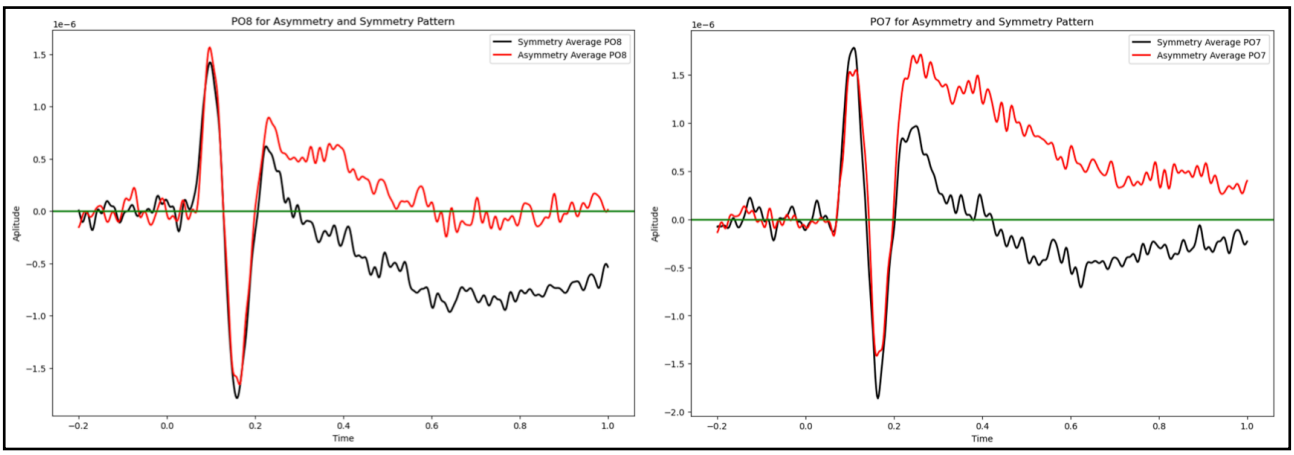

Fig 25 : graphs from the code above

6. Results

The PO7 and PO8 electrodes, which are over occipito-parietal areas, were analysed because these regions are known to process visual symmetry. The P1 Component, 100–130 ms post stimulus onset, reflects early sensory processing in the visual cortex. The amplitude of the P1 component did not significantly differ between symmetrical and asymmetrical patterns, visualised in fig 18 (left) for subject 10. While in fig 25 there is a slight difference between the two, maybe because it is average of all the participants or it may be possible due to some noise. But, in case of most of the participants, there was 0, or close to 0 difference between P1 hikes for asymmetrical and symmetrical events. The N1 Component, 170–200 ms post-stimulus onset, is associated with intermediate visual processing, particularly attentional mechanisms and pattern recognition. We found that N1 amplitude was significantly greater for symmetrical patterns compared to asymmetrical patterns. Can be seen in fig 18 and well as fig 25. The Sustained Posterior Negativity (SPN) reflects later stage processing and is often linked to attentional engagement with regular patterns. After analysis, we noticed that the SPN was larger for symmetrical patterns than asymmetrical ones. It is clearly visible in fig 25, where we see a huge gap between red and black lines after N1.

7. Additional Analysis - Decoding

For additional analysis, we chose decoding to move further. We chose decoding because it works well for classification tasks (like this experiment of classifying patterns based on symmetry) and it is highly effective for detecting asymmetries in spatial patterns. It is also good for decoding hemispheric differences rather than explain brain function in depth. We used Common spatial patterns (CSP), it is a supervised feature extraction method that finds spatial filters to maximise variance differences between two classes (e.g., left vs. right hemisphere) and helps in identifying asymmetric activation patterns across brain regions. CSP is used for extracting features, then with the results we train Linear Discriminant analysis (LDA) Classifier to analyse left-right hemisphere asymmetry. LDA finds a linear decision boundary between symmetric EEG patterns that maximise the separation between two classes. It works well with CSP to enhance spatial differences. LDA assumes that EEG features are linearly separable, which may not always be true.

```python
# Convert list to NumPy arrays
X = np.concatenate(all_epochs, axis=0)  # (n_total_epochs, n_channels, n_times)
y = np.concatenate(all_labels, axis=0)  # Labels

# Feature Extraction using CSP
csp = CSP(n_components=4, reg=None, log=True, norm_trace=False)

# Classification using LDA
lda = LinearDiscriminantAnalysis()

# Create a pipeline
pipeline = Pipeline([('CSP', csp), ('LDA', lda)])

# Split into training/testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Train the model
pipeline.fit(X_train, y_train)

# Predict on test set
y_pred = pipeline.predict(X_test)
```

Fig 26 : Code for LDA Trainer

We Choose n components as 2 because LDA works well with a small number of highly informative features. More components increase dimensionality, leading to overfitting. Too many components capture irrelevant noise instead of meaningful patterns. We compare left v/s right hemisphere signals, and train LDA for classification. LDA performance was neither bad, nor excellent. It acquired an accuracy of 50% (Fig 27)

```python
# ------------------------ 7. Plot Confusion Matrix ------------------------
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=["Left", "Right"], yticklabels=["Left", "Right"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("LDA Confusion Matrix")
plt.show()
```

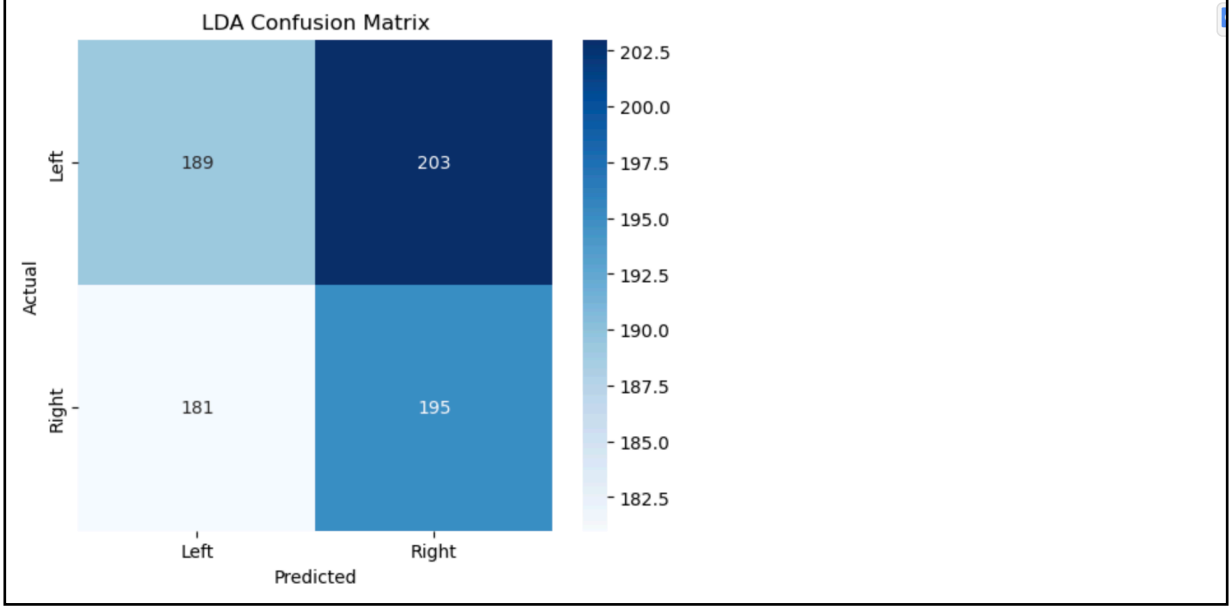

Fig 27 : LDA Performance

Conclusion

Our analysis shows distinct neural activity for symmetrical and asymmetrical patterns. Symmetrical patterns evoked stronger and faster ERP responses, indicating high attention and efficient visual processing. On the other hand, the asymmetrical patterns resulted in comparatively weaker ERP components, which suggests high effort in interpreting random stimuli. These findings align with existing theories on the brain's preference for symmetry in visual stimuli. As we presented the results, we conclude that there are no effect of symmetry at the Early stage (P1), indicating that basic visual encoding is unaffected by symmetry. At Intermediate stage (N1), Symmetry influences attention mechanisms and pattern recognition in brain. And the Later stage (SPN), Sustained engagement with symmetry patterns reflects higher-order processing inside brain. This aligns with previous findings, particularly the work by Jacobsen and Höfel (2003) [6]. Understanding visual processing is essential for fields such as cognitive neuroscience, artificial intelligence (e.g., computer vision), and clinical research on conditions like visual impairments or neurological disorders. This research can be extended by combining EMG to get in depth correlation between the brain activity and different patterns.

References

1. Cárdenas, R. A., & Harris, L. J. (2006). Symmetrical decorations enhance the attractiveness of faces and abstract designs. Evolution and Human Behaviour.
2. Eisenman, R. (1967). Preference for symmetry and the rejection of complexity, Psychonomic Science.
3. Höfel, L., & Jacobsen, T. (2007a). Electrophysiological indices of processing aesthetics: spontaneous or intentional processes? International Journal of Psychophysiology.
4. Höfel, L., & Jacobsen, T. (2007b). Electrophysiological indices of processing symmetry and aesthetics: a result of judgment categorisation or judgment report. Journal of Psychophysiology.
5. Jacobsen, T., & Höfel, L. (2002). Aesthetic judgments of novel graphic patterns : analyses of individual judgments. Perceptual and Motor Skills.
6. Jacobsen, T., & Höfel, L. (2003). Descriptive and evaluative judgment processes: behavioural and electrophysiological indices of processing symmetry and aesthetics. Cognitive, Affective, & Behavioural Neuroscience.
7. Treder, M. S. (2010). Behind the looking glass: a review on human symmetry perception. Symmetry.
8. Tyler, C. W. (1995). Empirical aspects of symmetry perception. Spatial Vision.
9. Wagemans, J. (1995). Detection of visual symmetries. Spatial Vision.
10. Luck, S. J. (2014). An Introduction to the Event-Related Potential Technique. MIT Press.
11. Coles, M. G. H., & Rugg, M. D. (1995). "Event-related brain potentials: An introduction." Electrophysiology of Mind: Event-Related Brain Potentials and Cognition. Oxford University Press.
12. Kappenman, E. S., & Luck, S. J. (2016). "Best practices for using event-related potentials in clinical research: Guidelines and recommendations." Psychophysiology.
13. Widmann, A., Schröger, E., & Maess, B. (2015). "Digital filter design for electrophysiological data – A practical approach." *Journal of Neuroscience Methods.*
14. Bruce, V., Green, P. R., & Georgeson, M. A. (2014). Visual Perception: Physiology, Psychology, and Ecology (4th ed.). Psychology Press.

15. Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L., & Hämäläinen, M. S. (2013). MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience.*
16. *Fridlund, A. J., & Cacioppo, J. T. (1986). Guidelines for human electromyographic research. Psychophysiology.*
17. *Data :* https://nemar.org/dataexplorer/detail?dataset_id=ds004347
18. Vigário, R. N. (1997). "Extraction of ocular artefacts from EEG using independent component analysis." Electroencephalography and Clinical Neurophysiology.
19. Jung, T. P., Makeig, S., Humphries, C., Lee, T. W., McKeown, M. J., Iragui, V., & Sejnowski, T. J. (2000). "Removing electroencephalographic artifacts by blind source separation." *Psychophysiology.*
20. Hyvärinen, A., & Oja, E. (2000). "Independent Component Analysis: Algorithms and Applications." Neural Networks