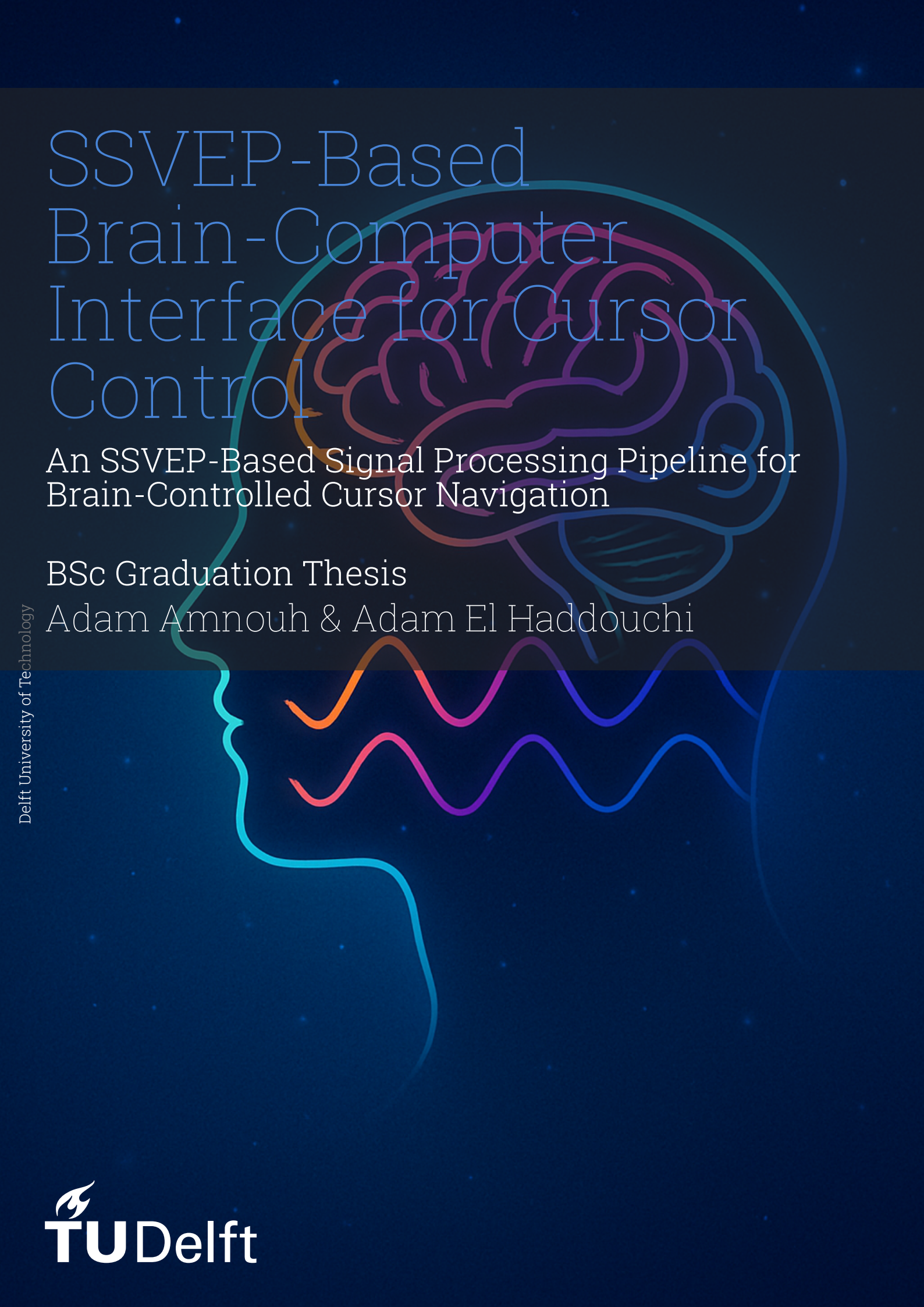
SSVEP-Based Brain-Computer Interface for Cursor Control



An SSVEP-Based Signal Processing Pipeline for Brain-Controlled Cursor Navigation

BSc Graduation Thesis

Adam Amnouh & Adam El Haddouchi

Delft University of Technology

TU Delft

SSVEP-Based Brain-Computer Interface for Cursor Control

An SSVEP-Based Signal Processing Pipeline for Brain-Controlled Cursor Navigation

by

Adam Amnouh & Adam El Haddouchi

Student Name	Student Number
Adam Amnouh	5822289
Adam El Haddouchi	5476526

Supervisors: Tiago Costa and Dante Muratore
Project Duration: April, 2025 - June, 2025
Faculty: Faculty Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft

Abstract

This thesis investigates a steady-state visually evoked potential (SSVEP)–based brain-computer interface (BCI) for controlling a computer cursor. The system employs two frequency-detection methods: filter-bank canonical correlation analysis (FBCCA) and ensemble task-related component analysis (eTRCA). Performance was evaluated using EEG data from a public dataset (MAMEM) and a self-recorded dataset. Classification accuracy, information transfer rate (ITR), and signal-to-noise ratio (SNR) were analyzed.

Results indicate that the MAMEM dataset, featuring a 256-channel high-density electrode setup, yielded superior classification accuracy—averaging 70.3% with FBCCA and reaching up to 93.3% peak accuracy in some subjects. In contrast, the self-recorded dataset, acquired with an 8-channel OpenBCI headset, showed lower performance, with FBCCA average accuracies ranging from 33.3% to 71.1%, and a maximum observed peak accuracy of 93.3% in isolated trials. FBCCA proved particularly effective in real-time conditions due to its calibration-free design, achieving real-time classification accuracies of 67.86% and 62.5% in two separate test sessions. In addition, system latency was assessed, and it was found that the complete signal processing pipeline executes within 0.232 seconds. eTRCA achieved higher offline accuracy on the MAMEM data when sufficient training trials were available but underperformed on the self-recorded dataset due to limited calibration data. Real-time cursor control was successfully demonstrated using two target frequencies (8.57 Hz and 12.00 Hz), confirming the practical viability of SSVEP-based BCIs. Future work should aim to improve signal quality, enhance spatial resolution through better electrode placement, and reduce the need for user-specific calibration to enable more reliable and accessible real-world applications.

Preface

Adam Amnouh & Adam El Haddouchi
Delft, June 2025

This paper marks the completion of our Bachelor's graduation project, conducted as part of a collaborative effort to develop a real-time Brain-Computer Interface (BCI) system. The project was carried out by three dedicated groups: GUI and Stimulus Design, Signal Acquisition, and Signal Processing and Classification. Our group was responsible for the latter, focusing on decoding SSVEP signals from EEG data to enable cursor control through brain activity.

Working on this project has been both challenging and rewarding. It provided us with the opportunity to apply and expand our knowledge in signal processing, while also learning how to collaborate effectively within a multidisciplinary team. The experience gave us valuable insights into the complexity and potential of BCI technologies.

We would like to sincerely thank our supervisors, Tiago Costa and Dante Muratore, for their ongoing support, guidance, and constructive feedback throughout the course of the project. Their expertise and encouragement played a crucial role in helping us stay motivated and on track.

We hope this work will contribute to further developments in the field of BCI, and we are excited to see how future students and researchers will build upon it.

Contents

1 Introduction

1.1. State-of-the-Art Analysis

Brain-Computer Interfaces (BCIs) are systems that enable direct communication between the brain and an external device by translating neural activity into actionable commands, bypassing peripheral nerves or muscular control [1]. Non-invasive BCI systems, particularly those based on electroencephalography (EEG), are very attractive due to their safety, portability, and relatively low cost [2]. BCIs have shown promise in restoring communication or control abilities to individuals with severe motor impairments. In recent years, however, BCI applications have extended into broader domains including entertainment, neurofeedback training, and assistive technologies for the general population [3], [4].

Among various BCI paradigms, the Steady-State Visual Evoked Potential (SSVEP) approach stands out for its robustness, minimal training requirements, and high information transfer rates. SSVEPs are brain responses elicited when a user focuses on a visual stimulus flickering at a constant frequency, making this paradigm particularly suitable for real-time applications [5]. Thanks to advances in signal processing and machine learning, modern SSVEP-based BCIs now achieve significantly improved accuracy and responsiveness, bringing real-time control applications closer to practical deployment [6]. However, achieving seamless integration between signal acquisition, real-time processing, and graphical interface design remains a key technical challenge [7].

1.2. Problem Definition

Despite the growing interest in non-invasive Brain-Computer Interfaces (BCIs), many real-time systems still face significant challenges in signal processing, particularly in decoding Steady-State Visually Evoked Potentials (SSVEPs) with low-latency and high accuracy using consumer-grade EEG hardware. These challenges are exacerbated by the practical limitations of low channel count, environmental noise, and inter-subject variability in SSVEP responses [8], [9].

Real-time BCI applications, such as cursor control, demand both rapid response and consistent performance. However, many existing pipelines require extensive calibration, long signal windows, or high-density electrode arrays, limiting their practicality for general use. The need for reliable classification from brief EEG epochs recorded via dry electrodes further complicates the signal processing task, often resulting in reduced signal-to-noise ratios (SNR) and degraded system responsiveness [10].

This thesis addresses these limitations by designing and implementing a signal processing and classification pipeline optimized for low-latency and calibration-minimal operation. The system targets two-class cursor control using an 8-channel OpenBCI headset and evaluates two prominent frequency recognition methods: Filter Bank Canonical Correlation Analysis (FBCCA) and Ensemble Task-Related Component Analysis (eTRCA). These algorithms were selected due to their established performance in SSVEP detection under real-time constraints [8], [9].

The scope of this work is shaped by practical constraints, including limited EEG spatial resolution, hardware compatibility with the OpenBCI platform, and integration with GUI components developed

by collaborating subgroups. These constraints inform design decisions throughout the pipeline and highlight trade-offs between algorithmic performance and system usability.

1.3. Overview of the BCI Pipeline and Implementation Scope

The developed Brain-Computer Interface (BCI) system consists of three principal components: the Graphical User Interface (GUI), the Data Acquisition subsystem, and the Signal Processing module. Together, these elements form an integrated real-time loop for interpreting Steady-State Visually Evoked Potential (SSVEP) responses and enabling cursor control through neural activity.

Figure 1.1 presents the high-level system architecture. The operational flow is initiated within the GUI, which manages user interaction and initiates communication with the EEG headset through the Lab Streaming Layer (LSL) protocol. LSL provides a standardized interface for real-time acquisition and synchronization of biosignals, enabling a seamless data stream between the GUI and the backend.

Upon activation, the EEG headset continuously transmits raw neural signals to both the GUI and the backend components. Within the Data Acquisition subsystem, these signals undergo preprocessing steps, including filtering and noise suppression, to enhance the signal-to-noise ratio. The filtered EEG data is then forwarded to the Signal Processing unit, where SSVEP-related frequency features are extracted and classified.

The classification module is responsible for determining the intended focus of the user based on the frequency-locked neural activity. A predicted frequency label and associated confidence score are generated as output and sent back to the GUI. The GUI interprets this classification result to update the interface, such as moving a cursor or providing other real-time feedback.

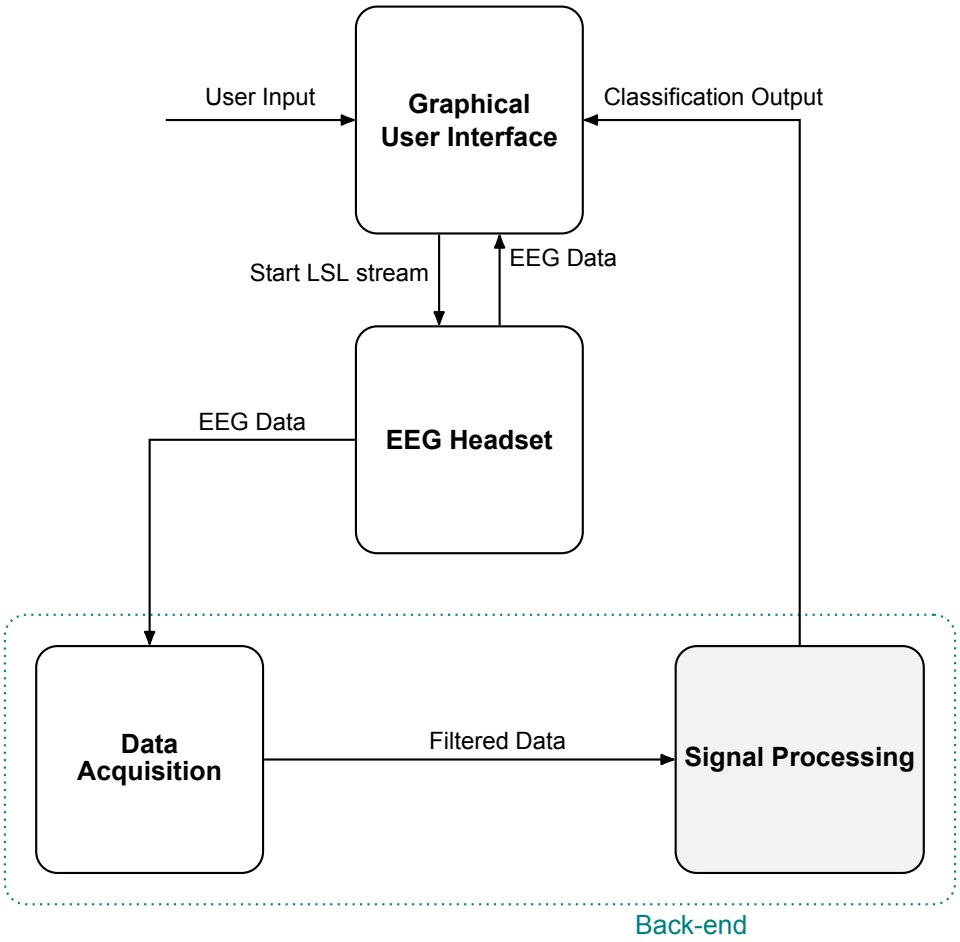


Figure 1.1: Overall System Architecture

This work forms part of a broader collaborative project in which multiple subgroups contributed to the design and implementation of the BCI system. The responsibilities described in this thesis were centered on the signal processing and classification track. Key tasks undertaken include:

- Implementation and optimization of the FBCCA and eTRCA classification algorithms.
- Validation of these algorithms using both publicly available datasets (e.g., MAMEM) and self-recorded EEG signals.
- Ensuring real-time system performance through the use of short signal windows and lightweight computational methods.
- Delivering classification outputs in a structured format suitable for direct integration with the GUI subsystem.

Through the careful development and evaluation of the signal processing pipeline, the system aims to demonstrate the feasibility of real-time BCI control using non-invasive EEG signals and consumer-grade hardware.

1.4. Structure of the Thesis

The thesis is continued in Chapter 2, where the programme of requirements is outlined, including the system's functional goals, performance targets, and integration needs. In Chapter 3, the chosen signal processing and classification methods—such as FBCCA and eTRCA—are presented, along with supporting techniques like filtering and performance metrics. Chapter 4 describes how the system was implemented, detailing how the datasets were processed and how the algorithms were integrated into a working pipeline. In Chapter 5, the empirical evaluation is provided, covering both the validation of dataset signal quality and the correctness of the classification methods. The system's performance is also reported based on various datasets and real-time tests. In Chapter 6, potential improvements and directions for future work are explored. Finally, in Chapter 7, the main findings are summarized and concluding reflections are offered. Additional material, such as source code and extended results, is included in the appendices, with all references compiled at the end.

2

Programme of Requirements

This chapter presents the formal requirements for the development of a signal processing and classification pipeline within a steady-state visually evoked potential (SSVEP)-based Brain-Computer Interface (BCI) system. The intended goal of the system is to enable real-time two-class cursor control via non-invasive EEG signals acquired using an OpenBCI headset. To realize this objective, a structured set of requirements has been established, categorized into general system requirements, classifier-specific requirements, and implementation constraints.

2.1. General System Requirements

Functional Requirements

- G.1 The system shall be capable of identifying user visual attention towards flickering stimuli corresponding to distinct control commands.
- G.2 It shall output a classification result in real-time, based on a brief segment of EEG data following stimulus onset.
- G.3 The system shall allow two-target control, assigning one flicker frequency to each directional command.
- G.4 Output confidence scores shall be provided to quantify prediction certainty, enabling threshold-based command validation.

Performance Requirements

- G.5 The average classification accuracy during real-time operation shall exceed 70%, based on the user's intended selections.
- G.6 The latency between stimulus onset and output decision (signal processing pipeline delay) shall not exceed 400 ms.
- G.7 The system shall produce consistent classification results using a maximum of 2 seconds of EEG data per trial.

Implementation Requirements

- G.8 All signal processing and classification procedures shall be implemented in Python, using libraries suitable for real-time execution (e.g., NumPy, SciPy).
- G.9 The system shall operate on mid-range consumer hardware without requiring GPU acceleration.
- G.10 EEG data shall be acquired using the OpenBCI headset with dry electrodes, operating at a sampling rate of 250 Hz.

2.2. Classification Module Requirements

Functional Requirements

C.1 The classifier shall receive preprocessed EEG epochs of fixed length and return a prediction label and associated confidence value.

C.2 The method shall be capable of distinguishing between at least two classes corresponding to defined flickering frequencies.

C.3 The classifier shall be modular and capable of integration with the GUI component.

C.4 The output format shall consist of a label (Left or Right) and a normalized confidence value in the range [0,1].

Performance Requirements

C.5 The classifier shall achieve at least 67% accuracy on real-time trials using unseen data.

C.6 In controlled offline experiments using synthetic or benchmark datasets, accuracy shall exceed 80%.

C.7 The classifier shall operate within a computational latency of less than 200 ms per prediction.

Implementation Requirements

C.8 The classification method shall be based on Filter Bank Canonical Correlation Analysis (FBCCA) and ensemble task-related component analysis (eTRCA).

C.9 Sub-band weights and canonical correlation scores shall be used to compute a final class decision and confidence level.

3

Methodology

3.1. Literature review

The classification of Steady-State Visual Evoked Potentials (SSVEPs) in Brain-Computer Interfaces (BCIs) has prompted extensive research into signal processing and machine learning techniques. Numerous algorithms have been developed to enhance the accuracy and speed of SSVEP-based systems. Among these, Filter Bank Canonical Correlation Analysis (FBCCA) and Ensemble Task-Related Component Analysis (eTRCA) have emerged as prominent methods due to their balance between performance, computational efficiency, and applicability in real-time environments.

FBCCA, an enhancement of Canonical Correlation Analysis (CCA), applies a bank of band-pass filters to extract frequency-specific information, including fundamental and harmonic components of the SSVEP response. This approach, introduced by Chen et al. [11], has been shown to significantly improve classification accuracy over traditional CCA, particularly in noisy conditions and with short data windows. Earlier foundational work by Lin et al. [12] and Bin et al. [13] established the use of CCA in BCI applications, while FBCCA extended its utility by emphasizing harmonic contributions and improving robustness. Comparative studies such as those by Liu et al. [14] and Aghili et al. [15] affirm FBCCA's competitiveness, making it suitable for use even with low-cost EEG devices.

Ensemble Task-Related Component Analysis (eTRCA), building upon the earlier Task-Related Component Analysis (TRCA) method proposed by Tanaka et al. [16], offers a supervised alternative that utilizes subject-specific training data. It enhances signal components that are temporally stable across trials, learning spatial filters tailored to individual users. The ensemble version, proposed by Nakanishi et al. [17], combines multiple spatial filters to further improve classification reliability. Although it requires calibration, eTRCA has demonstrated high accuracy in within-subject settings and has outperformed other methods such as eCCA and standard TRCA in multiple studies [18].

Several other classification strategies were evaluated but not adopted due to various limitations. Extended CCA (eCCA), which combines artificial reference signals with subject-specific EEG templates [19], offers moderate performance gains over standard CCA but at the cost of increased computational complexity. Machine learning models like Support Vector Machines (SVMs) and Linear Discriminant Analysis (LDA) have also been explored [20], [21], often relying on hand-engineered features such as spectral power or spatial patterns. While potentially effective, these approaches require extensive preprocessing and may lack the temporal resolution needed for rapid SSVEP detection.

Deep learning-based approaches, particularly Convolutional Neural Networks (CNNs), have attracted attention for their ability to learn discriminative features directly from raw EEG data [22]. However, these models typically demand large training datasets, are computationally intensive, and often lack generalizability across subjects without retraining or fine-tuning [23]. Their limited interpretability and resource demands currently constrain their practicality in real-time or consumer-grade settings.

Other methods, including mutual information-based feature selection [24], multiset CCA [25], and adaptive filtering techniques, offer alternative perspectives but have seen limited adoption due to implemen-

tation complexity or inconsistent results across experimental setups.

In summary, FBCCA and eTRCA were selected based on their strong empirical performance, theoretical soundness, and practical usability. These methods represent two distinct but complementary approaches—unsupervised and calibration-free in the case of FBCCA, and supervised with subject adaptation in the case of eTRCA—making them well-suited for evaluation in both general-purpose and personalized BCI applications.

3.2. Preprocessing

Before the methods can be applied, the signal must first be preprocessed. EEG preprocessing aims to improve the signal-to-noise ratio by removing unwanted artifacts. A common first step in many EEG pipelines is notch filtering: a narrow band-stop filter, typically centered at 50 Hz in Europe or 60 Hz in North America, which is designed to suppress mains electricity interference. In a comprehensive study on EEG signal enhancement, Wang et al. [26] describes how a 50/60 Hz notch filter is commonly used to attenuate power line noise, but also emphasize that if the notch is too wide, it may distort nearby signal components. This warning draws from earlier signal-processing studies, such as Ai et al. (2018), which analyze the effects of filter width on adjacent EEG frequency bands.

Following the removal of line noise, band-pass filtering is used to isolate the frequency range relevant to steady-state visual evoked potentials (SSVEPs), typically between 5 and 50 Hz. A widely used option is the Butterworth filter, a type of infinite impulse response (IIR) filter known for its maximally flat frequency response in the passband. This ensures minimal amplitude distortion within the band of interest. In practice, a fourth-order zero-phase Butterworth filter is commonly employed by applying the filter forward and then backward through the data to eliminate phase distortion. This approach results in a linear-phase band-pass filter, ideal for preserving the temporal structure of SSVEP responses.

3.2.1. Notch Filter

A notch filter is a narrow-band bandstop filter designed to eliminate a specific frequency component while leaving other frequencies largely unaffected. In EEG, the notch is set to the power-line frequency (50 or 60 Hz). This removes the strong sinusoidal interference from electrical mains. However, notch filtering can introduce ringing or distort signals very near the notch. For this reason, the notch is made as narrow as possible. In practice, EEG toolboxes often implement an IIR notch (e.g. second-order) at 50 Hz, sometimes also filtering harmonics (e.g. 100 Hz). As noted in the literature, notch filtering "is usually used to attenuate power line noise", though one must balance noise removal against the risk of waveform distortion.

3.2.2. Butterworth Filter

The Butterworth filter is a standard IIR filter known for a "maximally flat" frequency response in its passband. In other words, it has no ripples in the passband and a smooth roll-off toward the stopband. Its magnitude response is given by

$$|H(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2n}} \tag{3.1}$$

for cutoff frequency ω_c and filter order n. In EEG-BCI preprocessing, Butterworth filters are often used for band-pass (and sometimes low-pass) filtering of EEG. For example, a fourth-order Butterworth band-pass from 5–50 Hz might be applied to isolate SSVEP components. To avoid phase distortion, the filter is typically applied forward and backward (zero-phase filtering). Wang et al. reported using "a zero-phase fourth-order Butterworth filter" in an EEG processing pipeline. In summary, the Butterworth filter provides a gentle, flat passband ideal for preserving signal amplitude within the SSVEP band, while attenuating out-of-band noise.

3.3. Methods

3.3.1. FBCCA

Following preprocessing, the data can be forwarded to one of the primary methods presented in this study, namely FBCCA. Filter Bank Canonical Correlation Analysis (FBCCA) is an enhanced method

used in SSVEP-based Brain-Computer Interfaces (BCIs) for frequency recognition. It is designed to overcome the limitations of traditional Canonical Correlation Analysis (CCA) by leveraging the spectral characteristics of SSVEPs across multiple frequency bands, including harmonics.

The process begins with a multichannel EEG signal $X \in \mathbb{R}^{C \times T}$, where C is the number of EEG channels and T the number of time samples. As shown in Figure 3.1, the EEG signal is passed through a filter bank that decomposes it into m subbands. Each subband is configured to isolate a specific frequency range, targeting either the fundamental stimulus frequency or one of its harmonics. This subband decomposition helps to capture the frequency-specific energy distribution of the SSVEP response more effectively than full-band analysis.

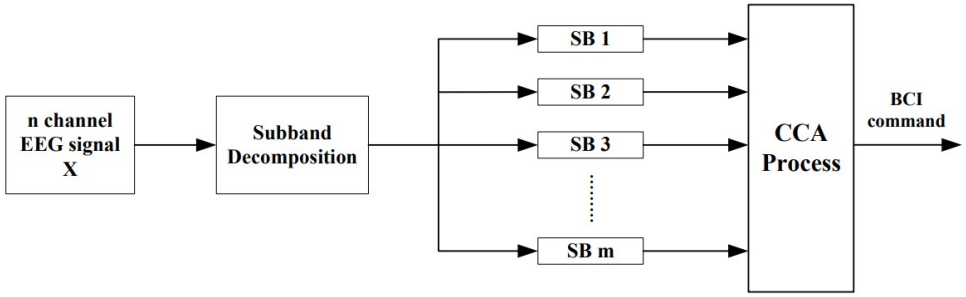


Figure 3.1: FBCCA framework: Subband decomposition of the EEG signal followed by CCA processing for each subband.

Once the signal is decomposed, each subband is processed independently. For a given target frequency f_i, a reference signal Y_{f_i} is constructed using sine and cosine components at f_i and its harmonics. The form of the reference signal is typically:

$$Y_{f_i}(t) = \begin{bmatrix} \sin(2\pi f_i t) \\ \cos(2\pi f_i t) \\ \sin(4\pi f_i t) \\ \cos(4\pi f_i t) \\ \vdots \\ \sin(2\pi N f_i t) \\ \cos(2\pi N f_i t) \end{bmatrix} \tag{3.2}$$

where N is the number of harmonics included. Canonical Correlation Analysis is then performed between each subband of the EEG signal and each reference signal. CCA identifies pairs of linear transformations (weight vectors) for the EEG and reference signals that maximize the correlation between them. Mathematically, the goal is to find vectors $\mathbf{w}_x$ and $\mathbf{w}_y$ that maximize:

$$\rho = \frac{\mathbb{E}\left[(\mathbf{w}_x^\top X)^\top (\mathbf{w}_y^\top Y)\right]}{\sqrt{\mathbb{E}\left[(\mathbf{w}_x^\top X)^2\right] \cdot \mathbb{E}\left[(\mathbf{w}_y^\top Y)^2\right]}} \tag{3.3}$$

This canonical correlation ρ_{ij} is computed for each combination of subband j and stimulus frequency f_i.

Figure 3.2 illustrates the core logic of this process. For each stimulus frequency f_i, a corresponding reference signal Y_i is compared with the EEG data using CCA. The canonical correlation coefficient ρ_i obtained from each comparison reflects how well the EEG data matches that stimulus frequency. Among all tested frequencies, the one yielding the highest ρ_i is selected as the recognized stimulus.

The innovation in FBCCA lies in combining the correlation results from each subband to produce a final decision. Instead of relying on the output of a single full-band CCA computation, FBCCA assigns a weight w_j to each subband j, then calculates a weighted sum of the squared canonical correlations:

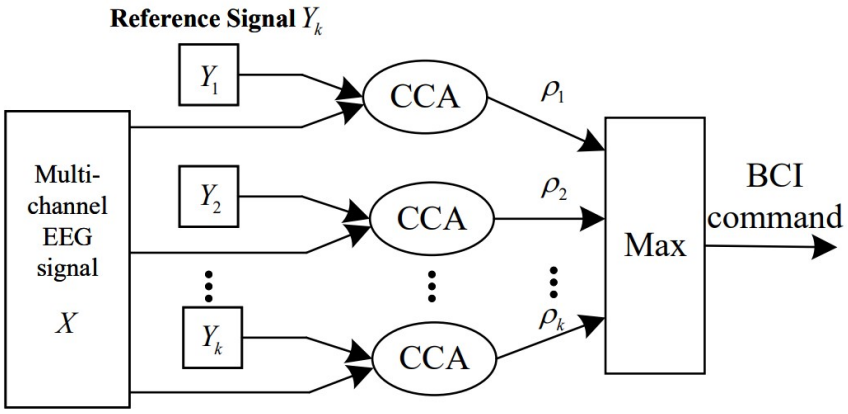


Figure 3.2: CCA computation between the EEG signal and multiple frequency-specific reference signals. The frequency with the highest ρ_k is selected as the BCI output.

$$\rho_i = \sum_{j=1}^{m} w_j \cdot \rho_{ij}^2 \tag{3.4}$$

This weighted aggregation exploits the fact that different frequency bands contain different SSVEP harmonics with varying strength. Low-frequency subbands usually contribute more strongly to the recognition accuracy, and thus receive higher weights.

The final step is to select the stimulus frequency f^* that corresponds to the highest aggregated correlation score:

$$f^* = \arg\max_{f_i} \rho_i \tag{3.5}$$

This approach allows FBCCA to more accurately extract frequency-specific information from EEG data compared to standard CCA, especially under low signal-to-noise ratio (SNR) conditions or when dealing with multiple simultaneous stimuli. By decomposing the signal and exploiting harmonics through filter banks, FBCCA offers a robust and high-resolution method for SSVEP-based BCI classification.

3.3.2. eTRCA

Another method employed in this paper is eTRCA. To enhance the extraction of steady-state visual evoked potentials (SSVEPs) in brain–computer interfaces (BCIs), the ensemble task-related component analysis (eTRCA) method has been proposed as an extension of the traditional task-related component analysis (TRCA) approach.Instead of treating each stimulus independently, eTRCA enhances signal detection by incorporating spatial information from all stimulus classes simultaneously, resulting in greater robustness to noise and variability.

As illustrated in Fig. 3.3(b), the method begins by computing a spatial filter for each stimulus condition based on repeated EEG trials. These filters are derived by maximizing the reproducibility of the brain response across trials of the same stimulus. Let $\mathbf{w}_n \in \mathbb{R}^C$ denote the spatial filter for stimulus n, where C is the number of EEG channels. These filters are computed using the TRCA criterion, which seeks the direction in EEG space that maximizes inter-trial covariance while minimizing intra-trial variability.

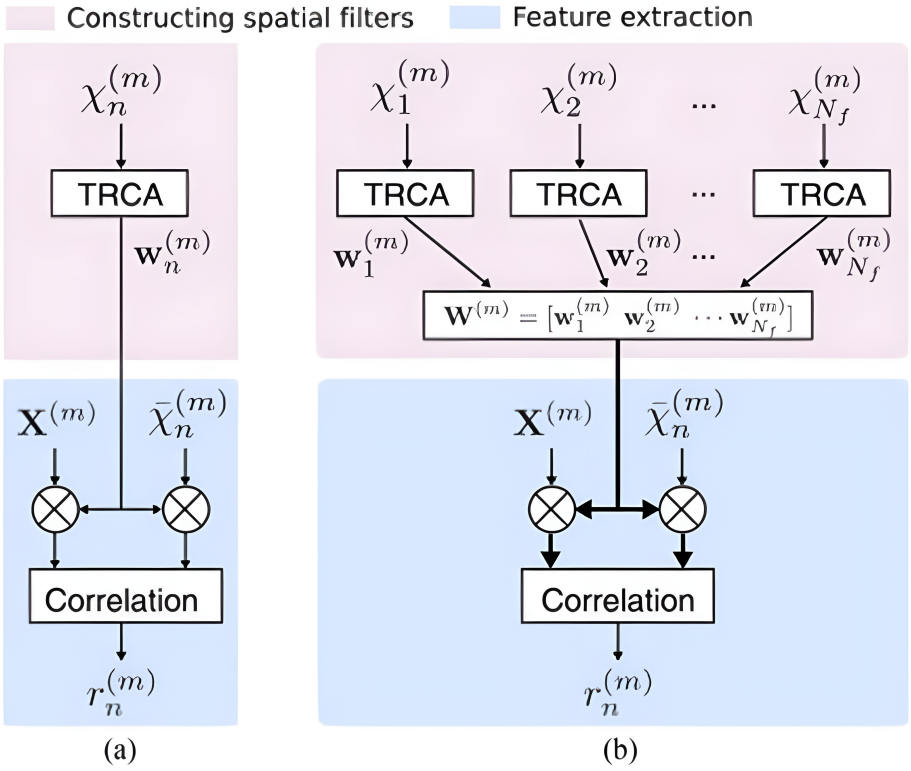


Figure 3.3: Overview of the proposed methods. (a) Standard TRCA method using individual class filters; (b) Ensemble TRCA (eTRCA), which integrates filters from all classes and computes a two-dimensional correlation. Adapted from [17].

Instead of using each $\mathbf{w}_n$ separately, eTRCA constructs an ensemble spatial filter matrix $\mathbf{W}$ by collecting all filters into a single structure:

$$\mathbf{W} = [\mathbf{w}_1,\ \mathbf{w}_2,\ \ldots,\ \mathbf{w}_N]\,, \tag{3.6}$$

where N is the number of stimulus classes. This ensemble filter exploits the fact that the spatial patterns of SSVEP responses across nearby frequencies tend to be similar, and therefore combining them can improve the overall signal-to-noise ratio (SNR).

Given a test EEG trial $\mathbf{X} \in \mathbb{R}^{C\times T}$, where T is the number of time samples, and a corresponding class-specific template $\bar{\mathbf{X}}_n \in \mathbb{R}^{C\times T}$ formed by averaging previous trials, both are projected using the ensemble filter:

$$\mathbf{Y} = \mathbf{W}^\top \mathbf{X}, \qquad \mathbf{Y}_n = \mathbf{W}^\top \bar{\mathbf{X}}_n. \tag{3.7}$$

The similarity between the projected trial and the projected template is then quantified using a two-dimensional correlation:

$$\rho_n = \mathrm{Corr2D}(\mathbf{Y}, \mathbf{Y}_n), \tag{3.8}$$

where ρ_n serves as the classification feature for stimulus n. This correlation measures both spatial and temporal similarity between the test signal and the class template after filtering.

To further enhance detection performance, the analysis is repeated across multiple frequency sub-bands. For each sub-band m, a correlation value $\rho_n^{(m)}$ is computed. These values are then aggregated using a weighted sum of squares:

$$F(n) = \sum_{m=1}^{M} a_m \cdot \left(\rho_n^{(m)}\right)^2, \tag{3.9}$$

where $a_m = m^{-1.25} + 0.25$ is the weight for sub-band m, and M is the total number of sub-bands. The final decision is made by selecting the stimulus n with the maximum score:

$$\hat{n} = \arg\max_n F(n). \tag{3.10}$$

The entire process, shown in Fig. 3.3(b), highlights how eTRCA incorporates spatial filters from all stimuli, uses them to project both test and template signals, and applies two-dimensional correlation to produce a robust measure of similarity. This ensemble strategy allows the system to exploit inter-class spatial structure and improves detection accuracy, particularly in scenarios with many stimuli and short time windows.

3.4. Evaluation metrics

3.4.1. FFT

To analyze the datasets used in this paper, the Fast Fourier Transform (FFT) will be applied. FFT allows for the identification of frequency peaks even before implementing classification algorithms. It is widely used to extract frequency-domain features from EEG signals. By taking the discrete Fourier transform of each EEG epoch, one obtains the amplitude (or power) spectrum. In SSVEP analysis, the stimulus frequency appears as a sharp peak in this spectrum. Specifically, if $x[n]$ is the EEG signal of length N, the DFT is

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i2\pi nk/N}, \tag{3.11}$$

where k indexes frequency bins. By choosing N so that the stimulus frequency f coincides with one of the FFT bins (e.g. by using an integer number of cycles in the window), the response at f can be estimated precisely. Typically, one computes the magnitude $|X[k]|$ or power $|X[k]|^2$ and looks for the largest peak at or near the known stimulation frequencies. As Kartsch et al. note, “SSVEP responses are traditionally computed from the power spectrum at the tag frequency”. In practice, the FFT is computed over the selected electrodes (often occipital channels) and averaged across them or across epochs. The peak amplitude (or signal-to-noise ratio) at each stimulus frequency can then be used as a feature for classification.

3.4.2. Signal-to-Noise Ratio in SSVEP

To quantitatively assess the strength and clarity of the recorded SSVEP responses, the signal-to-noise ratio (SNR) is computed. This metric provides a measure of how prominently the stimulus-locked signal stands out from background neural and environmental noise.

Welch power--spectral density

Before any SNR is calculated, EEG power is estimated with the Welch PSD. Welch’s method splits each epoch into overlapping segments, applies windowing, computes a periodogram for every segment, and then averages those periodograms. Averaging lowers the variance of single-segment FFTs, while windowing limits spectral leakage. For the present data, a 2s window with 50% overlap was used, giving a frequency resolution of $\Delta f = 0.2$ Hz. This resolves the 8.57 Hz and 12 Hz targets yet leaves enough segments for a stable average. The resulting spectrum $P(f)$ is therefore smoother and more reliable than a raw FFT and is used in both SNR definitions below.

Time-domain SNR

Time-domain SNR in SSVEP experiments is defined by comparing the signal power during stimulation against a separate baseline (no-stimulus) period. One computes the power spectral density (PSD) of

the EEG during stimulation and during baseline, then takes their ratio:

$$\text{SNR}_{\text{time}} = \frac{P_{\text{stim}}(f_0)}{P_{\text{base}}(f_0)}. \tag{3.12}$$

For instance, if the EEG shows a spectral peak at the stimulus frequency during flicker, this peak's power is divided by the power at the same frequency in the baseline recording [27].

Frequency-domain SNR

Frequency-domain SNR leverages the narrowband nature of SSVEP responses by estimating noise from neighboring frequency bins within the same trial. If $P(f_0)$ is the power at the target stimulation frequency f_0, and P_{noise} the average power of adjacent bins (excluding f_0), then

$$\text{SNR}_{\text{freq}} = \frac{P(f_0)}{P_{\text{noise}}}. \tag{3.13}$$

often expressed in decibels as $10 \log_{10}[P(f_0)/P_{\text{noise}}]$. An SNR > 1 indicates a clear spectral peak above the local noise floor.

3.4.3. Accuracy

The most important evaluation metric used in this study is accuracy. Classification accuracy, a fundamental performance metric in BCI research, is defined as the fraction of trials correctly classified 3.14. Mathematically, if N_{correct} is the number of correct identifications out of N_{total} trials, then

$$\text{Accuracy} = \frac{N_{\text{correct}}}{N_{\text{total}}}. \tag{3.14}$$

This definition is commonly used in the literature. In multiclass SSVEP experiments with balanced classes, accuracy by itself usually suffices. Sometimes one also reports the error rate $1 - P$ or related measures such as balanced accuracy if classes are imbalanced. Other standard metrics (precision, recall, F1-score) can be computed from the confusion matrix if needed, but in typical SSVEP-BCI reporting they are less common.

4

Implementation

The implementation of the proposed SSVEP-based BCI system was carried out in a structured manner across several stages involving both offline and real-time data. In order to design and validate the signal processing pipeline, initial development was conducted using a publicly available EEG dataset [28], which provided controlled experimental conditions suitable for algorithm refinement. This phase was followed by the application of the same methods to EEG data collected with a consumer-grade OpenBCI headset, allowing for an evaluation under more realistic and user-defined conditions. Ultimately, the pipeline was adapted for real-time use to enable direct control of a computer interface through brain activity. The sections that follow present the technical details and considerations of this implementation process in chronological order.

4.1. The MAMEM dataset

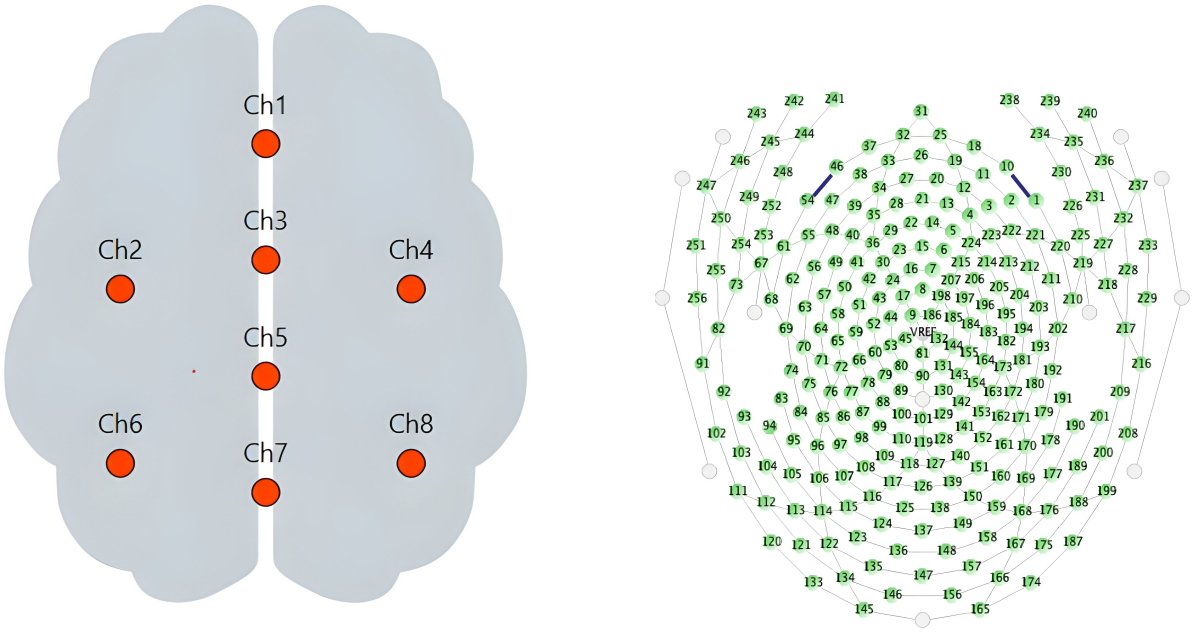
4.1.1. Dataset Description

The MAMEM SSVEP dataset comprises EEG recordings from 11 healthy volunteers (8 males, 3 females, ages 24–39) obtained during a visual flicker task. EEG was captured with the EGI Geodesic EEG System 300 (GES 300) using a 256-channel HydroCel Geodesic Sensor Net at a sampling rate of 250 Hz. Five visual stimulus frequencies (6.66, 7.50, 8.57, 10.00, 12.00 Hz) were presented as flickering magenta boxes at the center of a 22” LCD screen (refresh rate 60 Hz). Each stimulus flickered for 5 seconds (one trial), followed by 5 seconds of rest (black screen).

Each subject performed up to five recording sessions. Each session began with 100 seconds of rest (fixation on a blank screen), followed by an adaptation phase consisting of eight 5-second flicker trials, with each frequency selected randomly and separated by 5-second rest periods. After a further 30-second rest, the main experiment continued with each of the five stimulation frequencies presented in ascending order; for each frequency, three 5-second flicker trials were delivered, each separated by a 5-second rest. In total, 15 main trials and 8 adaptation trials were collected per session. Some sessions with recording issues (e.g., program crashes) were excluded, resulting in a final dataset comprising 1104 valid 5-second trials.

4.1.2. Electrode Mapping and Channel Selection

Figure 4.1 shows the two EEG headsets used: (a) the OpenBCI 8 channel EEG headset and (b) the 256 channel HydroCel Geodesic Sensor Net used in the MAMEM recordings. Only channels common to both systems were retained for analysis. Specifically, electrodes numbered 21, 59, 183, 81, 126, 101, 107, and 160 in the 256 channel layout were chosen, because these positions match the user’s 8-channel montage; these eight electrodes were relabeled as channels 1 to 8 in the reduced data set (see Fig. 4.1). This selection ensures that the analyses use the intersection of both sets of electrodes, avoiding mismatches in spatial sampling between systems.



(a) The OpenBCI 8-channel EEG headset.

(b) The 256-channel Geodesic Sensor Net (HydroCel).

Figure 4.1: Electrode configurations for the two EEG systems. Only electrodes common to both systems were used: channels 1–8 correspond to electrodes 21, 59, 183, 81, 126, 101, 107, and 160 of the 256-channel net.

4.1.3. Data Segmentation Methodology

The continuous EEG recordings were segmented into stimulus trials based on the digital event markers (DIN_1) recorded by the Stim Tracker device. In the raw data, the DIN_1 channel indicates stimulus onset and offset: it is high (1) during each 5-second flicker and low otherwise. The indices n_k at which DIN_1 rises from 0 to 1 were identified as the start of each trial. If successive rising-edge events were separated by more than 1250 samples (5 seconds at 250 Hz), this gap signified a rest interval rather than a continuation of the same trial. In other words, a new trial was marked whenever $n_{k+1} - n_k > 1250$.

After identifying the onset index n_k of each trial, a 5-second epoch was extracted. Let $f_s = 250$ Hz be the sampling rate. For trial k, the raw EEG segment E_k was defined as

$$E_k[m] = x[n_k + m], \quad m = 0, 1, \ldots, 5f_s - 1, \tag{4.1}$$

i.e. $5f_s = 1250$ samples starting at the trigger index n_k. Each epoch E_k is a multichannel time series (with the selected 8 channels) of length 1250 samples.

Each extracted epoch was then bandpass and notch filtered to remove drifts and line noise. A digital notch filter at 50 Hz (mains frequency) was applied to attenuate power-line interference. Additionally, a zero-phase bandpass filter in the range 5–45 Hz was applied to retain the SSVEP frequency components and suppress slow drifts and high-frequency noise. In the time domain, the combined filtering can be represented as a convolution of the epoch with the filter impulse responses:

$$Y_k[n] = (h_{\text{notch}} * h_{\text{bp}} * E_k)[n], \tag{4.2}$$

where $h_{\text{notch}}[n]$ and $h_{\text{bp}}[n]$ are the impulse responses of the 50 Hz notch and the 5–45 Hz bandpass filters, respectively. The result $Y_k[n]$ is the final preprocessed epoch for trial k.

4.1.4. Usage in FBCCA and eTRCA

Each preprocessed epoch was organized as a matrix of size $C \times N$ (channels $\times$ time samples), with $C = 8$ selected channels and $N = 1250$ samples per trial. These epoch matrices served as the inputs to the frequency recognition algorithms. Specifically, for each trial k, the filtered EEG data Y_k (an 8×1250 array) was provided as input to both FBCCA and eTRCA methods. The details of feature extraction and classification by FBCCA and eTRCA are given in chapter 3.

4.2. The Self-Measured Dataset

4.2.1. Dataset Description

An independent EEG dataset was recorded specifically for this study, following a controlled experimental protocol designed to replicate the visual stimulation paradigm used in the MAMEM dataset. EEG signals were recorded from five healthy participants using an 8-channel OpenBCI-compatible headset. All measurements were conducted in a quiet, darkened room to minimize ambient light and acoustic interference. The measurement setup is shown in Figure 4.2.

Figure 4.2: Measurement setup used for EEG data acquisition.

Each subject participated in between two and five recording sessions. The stimulus protocol included five visual stimulation frequencies: 6.66 Hz, 7.50 Hz, 8.57 Hz, 10.00 Hz, and 12.00 Hz. Each session began with a 30-second resting (calibration) period during which the subject fixated on a black screen. This was followed by a sequence of trials, each consisting of repeated stimulation with a single target frequency.

Each target frequency was presented in three repetitions, with each repetition consisting of 5 seconds of stimulation followed by 5 seconds of baseline (black screen). After the third repetition, an additional 10-second baseline was included before the next frequency began. This resulted in the following temporal structure for a single frequency:

- 5 s stimulation
- 5 s rest
- 5 s stimulation
- 5 s rest
- 5 s stimulation
- 15 s rest (final 5 s rest + 10 s baseline)

The demographic details of the participants are summarized in Table 4.1.

Subject ID	Gender	Hair Type
S001	Male	Thick
S002	Male	Regular
S003	Male	Regular
S004	Male	Short
S005	Male	Thick

Table 4.1: Overview of subjects in the self-measured dataset.

4.2.2. Data Segmentation Methodology

The EEG recordings were stored in CSV format with samples collected at 250 Hz. The raw data consisted of eight EEG channels. Specific sample windows corresponding to flicker intervals were predefined for segmentation. For each trial i, the segment indicates the start and end indices of the flicker period. In total, 15 such intervals were extracted:

$$(s_i, e_i) = (s_0 + 2500i, s_0 + 2500i + 1250), \quad i = 0, 1, \ldots, 14 \tag{4.3}$$

Each window was 5 seconds long (1250 samples). For example, trial 1 was extracted from samples 7500 to 8750, trial 2 from 10000 to 11250, and so on. These segments represented periods of active visual stimulation.

Before analysis, each extracted trial was subjected to the following filtering steps:

1. A notch filter was applied at 50 Hz to remove power-line interference:

$$H_{\text{notch}}(f) = \frac{s^2 + 1}{s^2 + \frac{\omega_0}{Q} s + 1}, \quad \omega_0 = \frac{2\pi f_0}{f_s}, ; f_0 = 50 \text{ Hz} \tag{4.4}$$

2. A 4th-order Butterworth bandpass filter was used to retain components between 5 and 45 Hz:

$$H_{\text{band}}(f) = \frac{1}{\sqrt{1 + \left(\frac{f}{f_c}\right)^{2n}}}, \quad f_c \in [5, 45], ; n = 4 \tag{4.5}$$

Each trial T_k was then filtered by convolution:

$$T_k^{\text{filtered}} = h_{\text{band}} * (h_{\text{notch}} * T_k) \tag{4.6}$$

where $*$ denotes convolution, and h_{notch} and h_{band} represent the respective impulse responses of the filters. The final output was a 3D tensor of shape $(15, 8, 1250)$, representing 15 trials, 8 channels, and 1250 samples per trial.

4.2.3. Usage in FBCCA

The preprocessed epochs from the self-measured dataset were used as direct input to the FBCCA classification pipeline. Each trial, structured as an 8×1250 matrix, was passed to the FBCCA method for frequency detection by computing canonical correlations with reference sine/cosine signals at the five target frequencies.

4.3. Real-Time BCI Evaluation Using FBCCA

4.3.1. Real-Time Measurement Description

To evaluate the system's ability to operate in real-world scenarios, a real-time EEG-based interface was implemented. This interface allowed subjects to control a two-class cursor by focusing on one flickering visual targets, each associated with a distinct stimulation frequency. In contrast to the offline experiments—where five different frequencies were tested to assess their detectability—only two frequencies were used in this setup: 8.57 Hz for the left direction and 12.00 Hz for the right direction.

These specific frequencies were chosen because they had shown the highest classification accuracy in offline analyses (as discussed in Section 5).

The user interface, shown in Figure 4.3, presented two large yellow flickering boxes, one on each side of the screen. Each box flickered at one of the two selected frequencies. A circular cursor was positioned at the center, and its movement reflected the system's classification of the user's intent. If the user looked at the left target (8.57 Hz), the system aimed to detect this frequency and move the cursor left. Similarly, a gaze at the right target (12.00 Hz) was intended to result in rightward cursor movement. The interface was developed by the GUI subgroup, as described in their report [29].

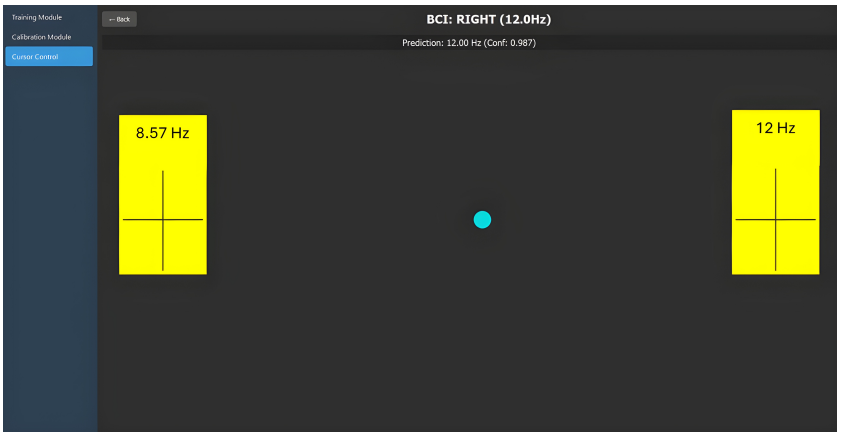


Figure 4.3: Real-time BCI. The left box flickers at 8.57 Hz and the right at 12.00 Hz. The cursor indicates the system's classification output, while the confidence level quantifies the correlation strength between the EEG signal and the target frequency.

Displayed above the interface was a real-time confidence score. This score quantified how strongly the system believed that the EEG signal matched one of the two target frequencies. The confidence value was calculated as a normalized correlation score between the user's EEG signal and the theoretical reference signals generated for each frequency. Thus, it indicated not only the chosen direction but also how reliable the classification decision was.

In addition to functional control, baseline measurements were conducted in which the participant did not look at either of the flickering boxes. These neutral trials were used to observe the natural confidence fluctuations when no intentional command was given. This information helped determine a threshold confidence level below which the system would interpret the signal as neutral or resting, avoiding unintended cursor movement.

4.3.2. Mathematical Framework for Real-Time FBCCA

Since the full FBCCA method, including sub-band decomposition and frequency scoring, has already been introduced in Section 3, this subsection focuses solely on the confidence value used in real-time classification.

After the classification frequency f^* is selected by maximizing the aggregated correlation score as described in Equation 3.5, the associated canonical correlation ρ_i is used to quantify how closely the EEG signal matches the reference. This score is then normalized over the sum of the sub-band weights:

$$\text{Confidence} = \frac{\rho_i}{\sum_{j=1}^{m} w_j} \tag{4.7}$$

This confidence value ranges from 0 to 1 and expresses the system's certainty in its prediction. A higher confidence suggests a stronger and more reliable match to one of the stimulation frequencies, while lower values may indicate either ambiguous brain activity or a resting state. In real-time usage, this confidence measure is critical for suppressing unintended control actions and determining baseline thresholds.

5

Empirical Evaluation

In order to assess the effectiveness and practical applicability of the implemented methods, a systematic evaluation was carried out. This chapter presents the validation procedures and empirical results derived from both offline and real-time experiments. The performance of the system is analyzed across several dimensions, including signal quality, frequency detection accuracy, and classification confidence.

5.1. Validation

5.1.1. FFT-Based Inspection of Target Trials (MAMEM Dataset)

To confirm whether the visual stimulation successfully evoked measurable SSVEP responses, a frequency-domain analysis was conducted using the FFT, as described in chapter 3. Each EEG trial was converted to the frequency domain, and spectral amplitudes were averaged across all channels. The resulting spectra were examined to detect peaks corresponding to the stimulation frequencies.

Figure 5.1 displays the FFT power spectra of fifteen trials, grouped by target frequencies: 6.66 Hz, 7.50 Hz, 8.57 Hz, 10.00 Hz, and 12.00 Hz. In each plot, the red dashed line indicates the frequency the subject was instructed to focus on.

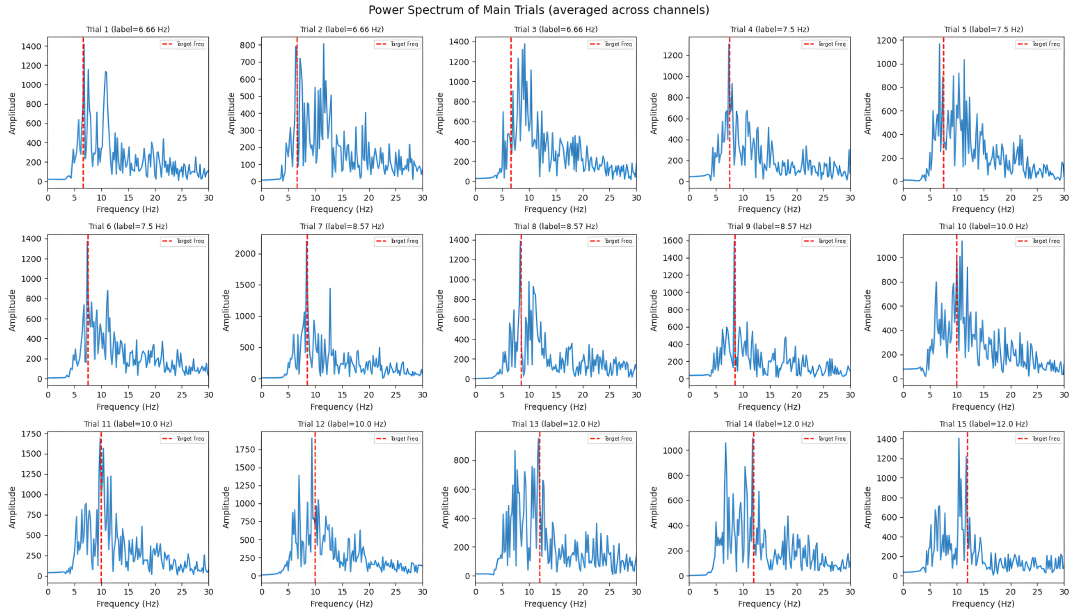


Figure 5.1: FFT power spectra of 15 EEG trials grouped by stimulus frequency - MAMEM Dataset.

Clear peaks are observable near the target frequencies in most cases, confirming the presence of frequency-locked SSVEP activity. For instance, trials labeled 6.66 Hz and 10.00 Hz exhibit sharp, consistent peaks exactly at those frequencies. Slight variability in amplitude or minor spectral noise can be noted across some trials, which is expected due to differences in individual responses or environmental factors. Nonetheless, the presence of clear spectral energy at the target frequencies validates the recorded responses.

5.1.2. FFT-Based Inspection of Target Trials (Self-Measured Dataset)

To evaluate the presence of SSVEP responses in the self-recorded dataset, the FFT was applied to each trial after averaging across all EEG channels, similar to the procedure used for the MAMEM dataset. Figure 5.2 shows the resulting power spectra for 15 trials, grouped by the corresponding stimulus frequency.

Compared to the MAMEM data, the spectra from the self-measured trials display increased variability in peak clarity and alignment. In several instances, particularly at lower frequencies such as 6.66 Hz and 7.5 Hz, the spectral peaks are either broad or slightly shifted away from the target.

Despite these variations, multiple trials still exhibit discernible energy concentrations near the intended frequencies, notably at 10 Hz and 12 Hz. The presence of these peaks suggests that the SSVEP responses were successfully elicited, albeit with less consistency than in the controlled MAMEM dataset.

Overall, the frequency-domain analysis supports the viability of the self-measured data for further investigation, though the results indicate a somewhat noisier signal profile.

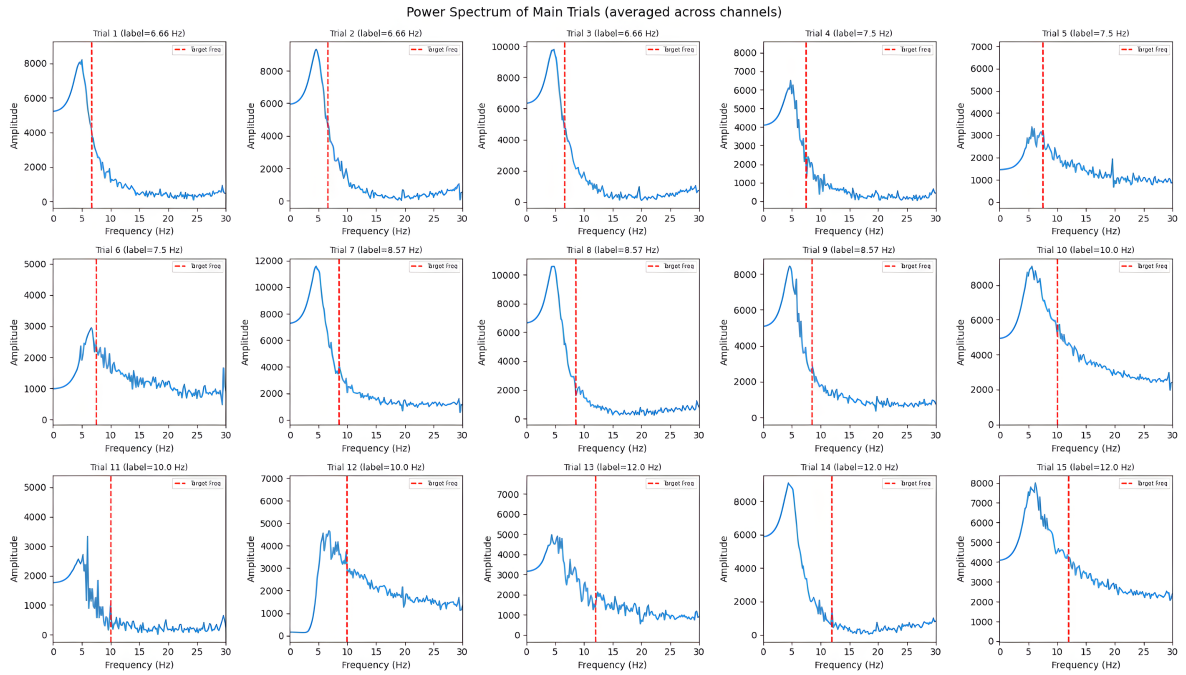


Figure 5.2: FFT power spectra of 15 EEG trials grouped by stimulus frequency - Self-Measured Dataset.

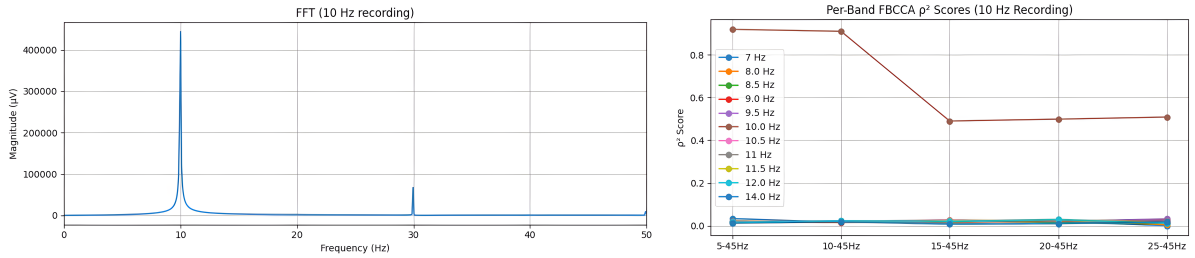
5.1.3. Synthetic Signal Validation of FBCCA

To assess the functionality of the Filter Bank Canonical Correlation Analysis (FBCCA) method, a controlled experiment was conducted using a 10 Hz sine wave produced by a function generator. The signal was applied to one of the electrodes of the EEG headset, with the ground connected to the reference electrode. A sinusoidal waveform with a peak-to-peak voltage of 1 V was introduced into the system, emulating a continuous SSVEP signal at 10 Hz. This setup was designed to verify whether the FBCCA method correctly identifies the stimulation frequency by returning a maximum correlation score at 10 Hz.

The EEG signal was recorded and saved in a CSV file. A FFT was then applied to the data to visualize the frequency content of the signal. As shown in Figure 5.3A, a clear peak appears at 10 Hz, which confirms that the 10 Hz signal was successfully captured by the system.

Following the FFT analysis, the FBCCA method was applied to the signal. As described in Chapter 3, the FBCCA algorithm splits the signal into five overlapping subbands. For each target frequency, a correlation value (ρ^2 score) is computed per subband, and the sum of these five scores is used to determine the final correlation score, as given by Equation 3.4. The results are shown in Figure 5.3B. As expected, the correlation score for 10 Hz is significantly higher than for all other candidate frequencies, clearly demonstrating the correct functioning of the FBCCA method.

Together, the FFT and FBCCA results confirm that the system accurately detects and correlates with the true stimulation frequency. The frequency domain analysis confirms the presence of the 10 Hz signal, while the FBCCA correlation scores validate that the method effectively identifies the correct frequency from among multiple candidates.



(a) FFT of the 10 Hz signal.

(b) Per-band FBCCA ρ^2 scores.

Figure 5.3: Functional assessment of FPCCA with a 10 Hz sine wave: (A) shows a spectral peak at 10 Hz and its harmonic at 20 Hz, (B) confirms the highest correlation at 10 Hz across all subbands.

5.2. Results

MAMEM Dataset

Subject ID	Gender	Hair Type	FBCCA Avg. (%)	FBCCA Peak (%)	ETRCA Avg. (%)
S001	Male	Regular	77.78	80.00	40.00
S002	Male	Regular	78.67	93.33	33.33
S003	Male	Thick	37.78	40.00	20.00
S004	Male	Short	80.00	86.67	26.67
S005	Female	Thick	33.33	53.33	26.67
S006	Female	Thick	46.67	66.67	40.00
S007	Male	Regular	64.00	73.33	53.33
S008	Female	Thick	31.11	40.00	40.00
S009	Male	Short	81.33	86.67	53.33
S010	Male	Regular	74.67	80.00	26.67
S011	Male	Regular	69.33	80.00	40.00

Table 5.1: Overview of subject-specific performance metrics using FBCCA and ETRCA methods (5 targets).

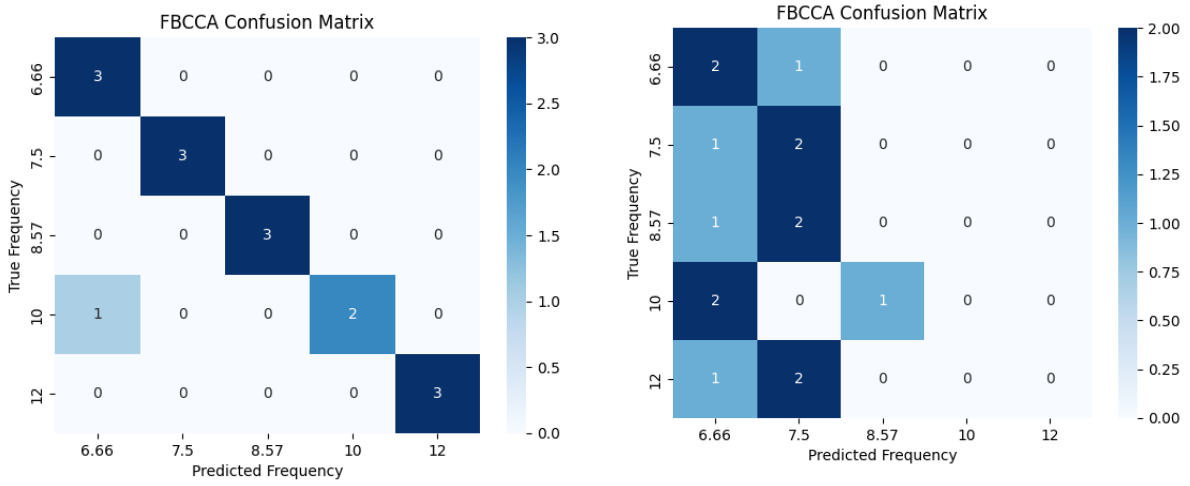
As shown in Table 5.1, the classification accuracies for each subject are presented based on the application of FBCCA and ETRCA methods to the MAMEM dataset. These results are derived from the experimental pipeline detailed in chapter 4. The FBCCA average accuracy is computed over five trials per subject, as described in the same chapter. The table displays both the average and peak performance using FBCCA, as well as the average accuracy achieved through ETRCA, giving a comparative overview of the results across subjects.

Self-Measured Dataset

Subject ID	Gender	Hair Type	Trials	FBCCA Avg. (%)	FBCCA Peak (%)
S001	Male	Thick	2	36.67	40.00
S002	Male	Regular	3	60.00	73.33
S003	Male	Regular	3	60.00	80.00
S004	Male	Short	5	71.11	93.33
S005	Male	Thick	2	33.33	40.00

Table 5.2: FBCCA performance results on the self-measured dataset (5 targets).

Table 5.2 shows the classification results obtained using the FBCCA method on the self-measured dataset. The number of trials conducted for each subject is also reported. As described in chapter 4, these accuracies are based on the experimental setup and evaluation procedure outlined therein. The average FBCCA accuracy per subject is computed over the number of trials specified, while the peak value indicates the highest individual performance observed.



(A) Confusion matrix of the best FBCCA result.

(B) Confusion matrix of the lowest-scoring FBCCA trial (Subject 5).

Figure 5.4: Comparison between the best and worst FBCCA classification results on the self-measured dataset.

Figure 5.4 shows a comparison between the highest and lowest FBCCA classification performances obtained on the self-measured dataset. The confusion matrix in (A) displays a strong diagonal structure, indicating correct classification across most frequency targets. In contrast, the matrix in (B) shows scattered misclassifications, with less alignment along the diagonal.

Self-Measured Dataset

To evaluate the real-time performance of the developed Brain-Computer Interface (BCI) system, two experimental sessions were conducted, visualized as time-series plots of confidence scores and predicted stimulus frequencies in Figure 5.5. The BCI leverages Steady-State Visually Evoked Potentials (SSVEPs) elicited by visual stimuli flickering at distinct frequencies: 8.6 Hz for the left target and 12.0 Hz for the right target, as depicted in the interface design (see Figure 4.3).

Based on a qualitative assessment of multiple real-time performance traces, a confidence threshold of 0.2 was established. Confidence scores exceeding this threshold were interpreted as indicative of intentional target selection, whereas scores below this boundary were considered reflective of baseline or non-selective neural states. In addition, system latency was assessed, and it was found that the complete signal processing pipeline executes within 0.232 seconds.

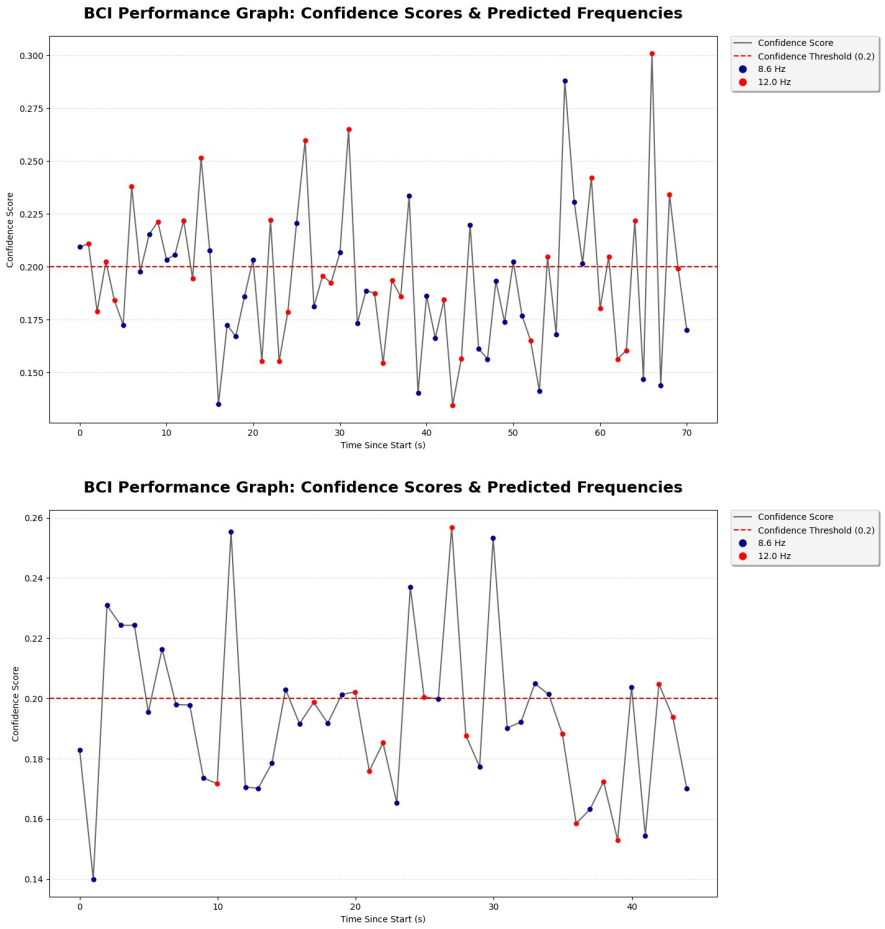


Figure 5.5: Real-time BCI predictions with confidence scores. Red: 12.0 Hz (right), blue: 8.6 Hz (left), dashed line: 0.2 confidence threshold.

Session A (Left Panel of Figure 5.5): This session lasted approximately 70 seconds. The participant initially focused on the right stimulus (12.0 Hz) from the start of the trial until second 35, shifted attention to the left stimulus (8.6 Hz) until second 57, and finally returned to the right stimulus for the remainder of the session. A total of 28 classification events surpassed the confidence threshold, of which 19 corresponded to the correct target as per the participant's intended selection. This yields an overall real-time classification accuracy of **67.86%**.

Session B (Right Panel of Figure 5.5): This session lasted approximately 50 seconds. The participant alternated attention between the target stimuli, following a predefined pattern. During this recording, 24 classification points exceeded the confidence threshold, with 15 correctly matching the attended target frequency. The resulting accuracy for this session was **62.5%**.

5.2.1. SNR results

Time Domain SNR Results

Time-domain SNR was computed by comparing the power at each stimulation frequency during flicker against the corresponding frequency's power in baseline recordings. Mean SNR values across subjects and conditions are presented in table 5.3.

Frequency-Domain SNR Results

Frequency-domain SNR was calculated by dividing the power at each target frequency by the average power of its neighboring bins. The distribution of frequency-domain SNR values across participants is shown in table 5.3. This measure emphasizes the presence of the SSVEP peak above the local spectral background.

	Time-SNR		Time-SNR		Freq-SNR		Freq-SNR	
Trial	**lin mean**	**lin std**	**dB mean**	**dB std**	**lin mean**	**lin std**	**dB mean**	**dB std**
1	3.203	2.175	5.056	3.374	1.063	0.083	0.265	−10.794
2	1.882	1.029	2.745	0.123	1.772	1.659	2.485	2.199
3	124.714	124.205	20.959	20.941	1.703	1.406	2.313	1.479
4	1.204	1.110	0.806	0.454	1.050	1.022	0.211	0.096
5	15.864	12.129	12.004	10.838	4.985	2.238	6.976	3.498
6	10.702	9.280	10.294	9.679	4.105	1.265	6.133	1.020
7	3.709	0.846	5.692	−0.730	3.242	1.686	5.108	2.267
8	8.210	3.564	9.144	5.522	8.872	6.726	9.480	8.278
9	4.740	2.195	6.758	3.415	1.896	0.927	2.779	−0.331
10	0.872	0.754	−0.594	−0.473	0.929	0.897	−0.320	−0.473
11	2.642	2.465	4.219	3.918	3.797	3.695	5.794	5.676
12	0.397	0.284	−4.015	−5.462	0.188	0.044	−7.266	−13.560

Table 5.3: Time- and frequency-domain SNR from an average accuracy recording with 6 trials per target. The two target frequencies are 8.57 Hz and 12 Hz.

In Table 5.3, the frequency-domain SNR values line up with the single-trial spectra. Ratios above about 2, as in Trials 5 and 8, come with a clear peak at the stimulus frequency. Ratios near or below 1, as in Trials 2, 4, 10, and 12, match spectra that follow a $1/f$ curve and show no peak. Trial 3 exposes a weakness of the time-domain metric; its very high baseline-referenced SNR is driven by a short broadband burst, so the frequency-domain measure still reports only a modest response. Overall, the frequency-domain SNR gives the clearest view of stimulus-locked activity, while the time-domain SNR mainly flags large power shifts that may be artifacts. Because most trials sit close to the noise floor, we find no significant peaks at the intended stimulation frequencies, and the data lack SSVEP responses strong enough for high-quality classification.

6

Analysis and Discussion

Algorithm Selection and Calibration

Due to time constraints and the limited quality of EEG recordings, the supervised eTRCA method could not be effectively utilized in this study. This method typically requires several consistent training trials to derive reliable spatial filters, which were not available in the self-recorded dataset. Consequently, classification accuracies obtained using eTRCA were consistently low. In contrast, the unsupervised FBCCA method demonstrated substantially better performance under the same conditions. Since FBCCA does not rely on prior calibration and is known to maintain robustness with short signal segments and lower signal quality, it was selected as the primary classification algorithm for both offline and real-time implementation.

Electrode Placement and Spatial Limitations

SSVEP signals are known to originate predominantly from the visual cortex, located in the occipital lobe. In the self-measured dataset, only three electrodes (channels 6, 7, and 8) were positioned over this region, as illustrated in Figure 4.1a. This limited spatial coverage introduced high sensitivity to electrode misplacement. If any of the occipital electrodes were improperly placed, the resulting signals lacked the necessary strength or clarity for reliable classification. By contrast, the MAMEM dataset included more than 50 electrodes across the occipital area, providing high spatial resolution and consistently robust SSVEP detection. This discrepancy in electrode density and placement precision likely contributed significantly to the reduced classification performance observed in the self-measured data.

Additionally, physical characteristics such as hair type were observed to significantly influence signal acquisition quality. As shown in Table 5.1 and Table 5.2, participants with thick hair generally exhibited lower classification accuracy compared to those with short or regular hair. For example, in the self-measured dataset, subjects with thick hair (S001 and S005) achieved FBCCA average accuracies of only 36.67% and 33.33%, respectively, whereas the subject with short hair (S004) reached 71.11%. A similar trend is evident in the MAMEM dataset, where thick-haired subjects (e.g., S003 and S005) had notably lower FBCCA average accuracies (37.78% and 33.33%) compared to those with short hair (e.g., S004 and S009, with accuracies of 80.00% and 81.33%).

This inverse relationship between hair thickness and classification accuracy can be attributed to the insulating properties of dense hair, which impedes effective electrode-skin contact and increases impedance at the sensor interface. As reported by Kappenman and Luck (2010), high impedance can significantly attenuate EEG signals and increase noise levels, thereby degrading signal quality and reducing classification performance [30].

Peak Detection Sensitivity and Classification Accuracy

Initial classification attempts using the MAMEM dataset yielded modest accuracy levels of approximately 40%, despite the presence of clear spectral peaks at the stimulation frequencies, as seen in Figure 5.1. It was observed that the frequency detection algorithm applied a strict matching criterion, requiring peaks to align exactly with the target frequencies. Given physiological variability and spectral

resolution limitations, actual peak positions often deviated by small margins (e.g., ±0.1 Hz). By relaxing the sensitivity and expanding the detection tolerance to ±0.2 Hz, classification accuracy improved markedly, reaching approximately 70%. These findings emphasize the importance of algorithmic flexibility in peak detection to accommodate realistic variations in spectral features.

Real-Time System Evaluation

To decouple algorithm performance from biological variability, synthetic SSVEP signals were injected into the system using a function generator. Sine waves at 12.0 Hz and 8.57 Hz were introduced to the amplifier through a voltage divider (see figure B.3), reducing the output to realistic EEG amplitudes between 10 and 100 μV. Even at these low signal strengths, the FBCCA classifier achieved 100% accuracy with confidence levels averaging around 0.6. Results from both the high-amplitude (1 V) and brain-strength (100 μV) tests are presented in the Appendix B (see figure B.2 and B.1). These findings confirm that the implemented classification algorithm is functional and effective under ideal signal conditions.

Real-time measurements using human EEG input yielded overall low accuracy, with the exception of two notable sessions that demonstrated substantially improved performance (see Figure 5.5). These isolated instances are likely attributable to better electrode contact or stronger individual responses. The generally poor performance across the remaining sessions confirms that inadequate signal quality, rather than algorithmic error, was the dominant limiting factor.

Future Work

Future work should consider the implementation of advanced data aggregation techniques to expand the available dataset for training and evaluation. By combining data across sessions and subjects, or using augmentation methods such as overlapping windows and synthetic trial generation, the variability of SSVEP responses can be better captured [31], [32]. This would be particularly beneficial for improving the generalizability of the eTRCA method, which demonstrated limited performance due to insufficient high-quality calibration data in the current study [17]. A larger, more diverse dataset could make eTRCA more robust and effective across different recording conditions.

With increased data availability through aggregation, future research may also explore the use of machine learning approaches for feature extraction and classification. These methods, including convolutional neural networks (CNNs) and deep belief networks, have shown promise in BCI applications but typically require substantial training data to avoid overfitting [33], [34]. In the present study, such methods were not pursued due to the known limitations of the self-recorded dataset, which lacked sufficient volume and consistency. However, with improved data collection and aggregation, the adoption of machine learning frameworks could significantly enhance classification accuracy and system adaptability.

Additionally, strategies to improve the signal-to-noise ratio (SNR) of EEG recordings should be explored. These may include hardware-based solutions such as electromagnetic shielding, proper cable grounding, and high-quality amplifiers [35], as well as experimental enhancements like increased electrode density in occipital areas and the use of standardized EEG caps for consistent placement [36]. Improving SNR directly impacts the clarity of SSVEP responses, thereby increasing the reliability of both traditional and machine-learning-based classifiers in future implementations.

7

Conclusion

This thesis has explored the design, implementation, and evaluation of a steady-state visually evoked potential (SSVEP)–based brain-computer interface (BCI) system for two-class cursor control. The system was developed using two classification methods—Filter Bank Canonical Correlation Analysis (FBCCA) and Ensemble Task-Related Component Analysis (eTRCA)—and tested on both a benchmark dataset (MAMEM) and a custom self-recorded dataset acquired with an 8-channel OpenBCI headset.

The empirical results demonstrated that FBCCA outperformed eTRCA under the given experimental conditions, particularly in the context of real-time classification and low-channel, low-SNR recordings. FBCCA enabled basic real-time cursor control with accuracies of 67.86% and 62.5% in two separate sessions. Conversely, eTRCA exhibited poor generalization and achieved low classification accuracy, frequently below 40%, likely due to its reliance on high-quality, subject-specific calibration data—something not available in the self-recorded setup. In addition to classification accuracy, system latency was assessed, and it was found that the complete signal processing pipeline executes within 0.232 seconds.

Performance discrepancies between the MAMEM and self-recorded datasets highlighted the critical role of signal quality, electrode density, and hardware stability. The high-density MAMEM dataset yielded notably higher classification accuracy and more consistent spectral responses due to superior spatial resolution and controlled experimental conditions. In contrast, the self-recorded data suffered from variability introduced by sparse electrode coverage, inconsistent contact, and environmental noise.

While the study confirms the feasibility of SSVEP-based cursor control using non-invasive EEG, it also underscores several limitations. Real-time performance remains modest and sensitive to noise and inter-subject variability. Future work should focus on improving signal quality through optimized electrode placement, enhanced filtering strategies, and the use of adaptive or hybrid classification methods. Furthermore, hardware improvements—such as standardized caps, shielding, and increased channel count—could substantially improve reliability and usability. Addressing these limitations will be essential for transitioning BCI technology from research prototypes to robust, user-friendly systems suitable for daily use.

References

[1] X.-Y. Liu, W.-L. Wang, M. Liu, *et al.*, "Recent applications of eeg-based brain-computer-interface in the medical field," *Military Medical Research*, vol. 12, Mar. 2025. DOI: 10.1186/s40779-025-00598-z.

[2] L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain computer interfaces, a review," *Sensors*, vol. 12, no. 2, pp. 1211–1279, 2012, ISSN: 1424-8220. DOI: 10.3390/s120201211. [Online]. Available: https://www.mdpi.com/1424-8220/12/2/1211.

[3] M. Orban, M. Elsamanty, K. Guo, S. Zhang, and H. Yang, "A review of brain activity and eeg-based brain–computer interfaces for rehabilitation application," *Bioengineering*, vol. 9, no. 12, 2022, ISSN: 2306-5354. DOI: 10.3390/bioengineering9120768. [Online]. Available: https://www.mdpi.com/2306-5354/9/12/768.

[4] G. A. M. Vasiljevic and L. C. de Miranda and, "Brain–computer interface games based on consumer-grade eeg devices: A systematic literature review," *International Journal of Human–Computer Interaction*, vol. 36, no. 2, pp. 105–142, 2020. DOI: 10.1080/10447318.2019.1612213.

[5] F.-B. Vialatte, M. Maurice, J. Dauwels, and A. Cichocki, "Steady-state visually evoked potentials: Focus on essential paradigms and future perspectives," *Progress in Neurobiology*, vol. 90, no. 4, pp. 418–438, 2010, ISSN: 0301-0082. DOI: https://doi.org/10.1016/j.pneurobio.2009.11.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301008209001853.

[6] E. Netzer, A. Frid, and D. Feldman, "Real-time eeg classification via coresets for bci applications," *Engineering Applications of Artificial Intelligence*, vol. 89, p. 103 455, Mar. 2020. DOI: 10.1016/j.engappai.2019.103455.

[7] S. Saha, K. A. Mamun, K. Ahmed, *et al.*, "Progress in brain computer interface: Challenges and opportunities," *Frontiers in Systems Neuroscience*, vol. Volume 15 - 2021, 2021, ISSN: 1662-5137. DOI: 10.3389/fnsys.2021.578875. [Online]. Available: https://www.frontiersin.org/journals/systems-neuroscience/articles/10.3389/fnsys.2021.578875.

[8] M. Nakanishi, Y. Wang, Y. Wang, Y. Mitsukura, and T.-P. Jung, "Enhancing detection of ssveps for a high-speed brain speller using task-related component analysis," in *2015 7th International IEEE/EMBS Conference on Neural Engineering (NER)*, IEEE, 2015, pp. 106–109.

[9] X. Chen, Y. Wang, M. Nakanishi, X. Gao, T.-P. Jung, and S. Gao, "Filter bank canonical correlation analysis for implementing a high-speed ssvep-based brain–computer interface," *Journal of neural engineering*, vol. 12, no. 4, p. 046 008, 2015.

[10] A. Rodriguez, A. Duenas, R. Ceres, L. Calderon, and J. A. Honrubia, "Low-cost ssvep-bci implementation using openbci hardware," *Biomedical Signal Processing and Control*, vol. 68, p. 102 678, 2021.

[11] X. Chen, Y. Wang, X. Gao, T.-P. Jung, and S. Gao, "Filter bank canonical correlation analysis for implementing a high-speed ssvep-based brain–computer interface," *Journal of Neural Engineering*, vol. 12, no. 4, p. 046 008, 2015.

[12] Z.-B. Lin, C.-L. Zhang, W. Wu, X. Gao, and S. Gao, "Frequency recognition based on canonical correlation analysis for ssvep-based bcis," *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 6, pp. 1172–1176, 2007.

[13] G. Bin, X. Gao, Z. Yan, B. Hong, and S. Gao, "An online multi-channel ssvep-based brain–computer interface using a canonical correlation analysis method," *Journal of neural engineering*, vol. 6, no. 4, p. 046 002, 2009.

[14] H. Liu, Z. Wang, R. Li, *et al.*, "A comparative study of stereo-dependent ssvep targets and their impact on vr-bci performance," *Frontiers in Neuroscience*, vol. 18, p. 1 367 932, 2024.

[15] S. N. Aghili, S. Kilani, E. Rouhani, and A. Akhavan, “A novel fast ica-fbcca algorithm and convolutional neural network for single-flicker ssvep-based bcis,” *IEEE Access*, vol. 12, pp. 630–642, 2023.

[16] T. Tanaka, O. Ozdenizci, and Z. Erkin, “Enhancement of steady-state visual evoked potential responses by the task-related component analysis method,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 21, no. 4, pp. 576–583, 2013. DOI: 10.1109/TNSRE.2012.2229296.

[17] M. Nakanishi, Y. Wang, Y. Wang, Y. Mitsukura, and X. Gao, “Enhancing detection of ssveps for a high-speed brain speller using task-related component analysis,” *IEEE Transactions on Biomedical Engineering*, vol. 65, no. 1, pp. 104–112, 2018.

[18] H. K. Lee and Y.-S. Choi, “Enhancing ssvep-based brain-computer interface with two-step task-related component analysis,” *Sensors*, vol. 21, no. 4, p. 1315, 2021.

[19] Y. Zhang, G. Zhou, J. Jin, X. Wang, X. Wang, and A. Cichocki, “L1-regularized multiway canonical correlation analysis for ssvep-based bci,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 2, pp. 212–222, 2014. DOI: 10.1109/TNSRE.2013.2282891.

[20] S. Bhattacharyya, P. Ganguly, and A. Chakrabarti, “Comparison of ssvep signal classification techniques using svm and ann for bci,” *International Journal of Advanced Computer Intelligence*, vol. 4, no. 2, pp. 263–267, 2014.

[21] A. de Oliveira, G. de Souza, R. Attux, K. Sameshima, and A. de Araujo, “Comparison of supervised learning classifiers for ssvep-based bci,” *Biomedical Signal Processing and Control*, vol. 40, pp. 359–374, 2018. DOI: 10.1016/j.bspc.2017.10.016.

[22] M. Kołodziej, K. J. Blinowska, and T. Durka, “Cnn-based classification of ssvep with limited training data,” *Applied Sciences*, vol. 13, no. 24, p. 13 350, 2023.

[23] F. Wu, R. Zhang, Y. Wang, Q. Ma, P. Xu, and D. Yao, “An analysis of deep learning models for ssvep classification in bci applications,” *IEEE Transactions on Neural Networks and Learning Systems*, 2023, Early Access.

[24] R. Zhang, Y. Liu, P. Zhou, and P. Li, “Frequency recognition in ssvep-based bci using multiset canonical correlation analysis,” *Biomedical Signal Processing and Control*, vol. 26, pp. 1–6, 2016. DOI: 10.1016/j.bspc.2015.11.011.

[25] X. Song, P. Xu, T. Liu, and R. Zhang, “Eeg classification based on multiway canonical correlation analysis,” *Biomedical Signal Processing and Control*, vol. 52, pp. 312–320, 2019. DOI: 10.1016/j.bspc.2019.04.012.

[26] F. Wang, L. Li, H. Zhang, and X. Gao, “A comprehensive review on motion trajectory reconstruction for eeg-based brain–computer interface,” *Frontiers in Neuroscience*, vol. 17, p. 1 078 208, 2023. DOI: 10.3389/fnins.2023.1078208.

[27] S. R. Schultz, “Signal-to-noise ratio in neuroscience,” *Scholarpedia*, vol. 2, no. 6, p. 2046, 2007, revision #137197. DOI: 10.4249/scholarpedia.2046.

[28] G. Stavropoulos, P. C. Petrantonakis, S. Hadjidimitriou, and I. Kompatsiaris, *Mamem ssvep dataset i: 256 channels, 11 subjects, 5 frequencies*, https://figshare.com/articles/dataset/MAMEM_EEG_SSVEP_Dataset_I_256_channels_11_subjects_5_frequencies_/2068677?file=3793738, Accessed June 2025, 2016.

[29] G. U. I. (Subgroup, “Ssvep-based brain-computer interface for cursor control,” https://drive.google.com/file/d/1V2QAy1FKJQ9DBIhUrB2OHrIA9DfWrb/view?usp=sharing, Bachelor’s thesis, Delft University of Technology, 2025.

[30] E. S. Kappenman and S. J. Luck, “The effects of electrode impedance on data quality and statistical significance in erp recordings,” *Psychophysiology*, vol. 47, no. 5, pp. 888–904, 2010.

[31] H. Cecotti and A. Graser, “Single-trial classification of event-related potentials in rapid serial visual presentation tasks using artificial neural networks,” *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 3, pp. 745–753, 2011.

[32] Y. Roy, H. Banville, I. Albuquerque, and A. Gramfort, “Eeg-gan: Generative adversarial networks for eeg data augmentation to improve ssvep classification,” *Journal of Neural Engineering*, vol. 19, no. 1, p. 016 028, 2022.

[33] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, “Eegnet: A compact convolutional neural network for eeg-based brain–computer interfaces,” *Journal of Neural Engineering*, vol. 15, no. 5, p. 056 013, 2018.

[34] P. Wang, X. Liu, B. Dong, C.-Y. Lu, and X. Li, “A review on deep learning techniques for eeg-based brain-computer interface applications,” *Sensors*, vol. 19, no. 1, p. 147, 2019.

[35] X. Li and F. Liu, “Improved ssvep detection using spatial-temporal filtering and eeg signal quality enhancement,” *Biomedical Signal Processing and Control*, vol. 60, p. 101 960, 2020.

[36] X. Song, P. Xu, T. Liu, and R. Zhang, “Eeg classification based on multiway canonical correlation analysis,” *Biomedical Signal Processing and Control*, vol. 52, pp. 312–320, 2019.

A

A Code

A.1. MAMEM Dataset Code

A.1.1. FBCCA

```
import os
from glob import glob
import scipy.io as sio
import numpy as np
from scipy.signal import iirnotch, butter, filtfilt
from sklearn.cross_decomposition import CCA
from sklearn.metrics import accuracy_score
from collections import defaultdict
# === Setup ===
data_dir = r"C:\Users\Adam\OneDrive\Documenten\Bep\BCI\Signal␣Processing\Code\Datasets"
mat_files = sorted(glob(os.path.join(data_dir, "S*.mat")))

# Constants
selected_channels = [
    80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
    139, 140, 141, 142, 143, 144, 145, 146, 147, 148
]
target_freqs = [6.66, 7.5, 8.57, 10.0, 12.0]
freqs = sorted(set([round(f + delta, 2) for f in target_freqs for delta in [-0.2, 0, 0.2]]))
main_labels = [0]*3 + [1]*3 + [2]*3 + [3]*3 + [4]*3  # 15 trials

# Results storage
results = []
# === Processing loop ===
for filepath in mat_files:
    try:
        filename = os.path.basename(filepath)
        subject = filename[:4]
        recording = filename[4]

        mat = sio.loadmat(filepath)
        eeg = mat['eeg']
        din1 = mat['DIN_1']
        sr = float(mat['samplingRate'])

        event_samples = [int(x[0][0]) for x in din1[3]]
        diffs = np.diff(event_samples)
        gap_idx = np.where(diffs > 1250)[0]
        block_starts = np.concatenate(([0], gap_idx+1))
        block_ends = np.concatenate((gap_idx, [len(event_samples)]))
        blocks = [(s,e) for s,e in zip(block_starts, block_ends)]
        main_blocks = blocks[8:23]
        main_starts = [event_samples[b[0]] for b in main_blocks]

        b_notch, a_notch = iirnotch(50/(sr/2), Q=30)
        b_band,  a_band  = butter(4, [5/(sr/2), 45/(sr/2)], btype='band')
```

```
47
48          def preprocess_epochs(eeg_data, starts, sr, selected_channels):
49              epochs = []
50              for start in starts:
51                  epoch = eeg_data[selected_channels, start:start+int(5*sr)]
52                  epoch = filtfilt(b_notch, a_notch, epoch, axis=1)
53                  epoch = filtfilt(b_band, a_band, epoch, axis=1)
54                  epochs.append(epoch)
55              return np.stack(epochs)
56
57          main_epochs = preprocess_epochs(eeg, main_starts, sr, selected_channels)
58
59          subbands = [(5,45), (10,45), (15,45), (20,45), (25,45)]
60          band_filters = [(butter(4, [lo/(sr/2), hi/(sr/2)], btype='band')) for lo, hi in
                subbands]
61          weights = [1.0/j for j in range(1, len(band_filters)+1)]
62
63          def generate_reference(f, N, fs, nh=3):
64              t = np.arange(N) / fs
65              return np.vstack([np.sin(2*np.pi*k*f*t) for k in range(1, nh+1)] +
66                               [np.cos(2*np.pi*k*f*t) for k in range(1, nh+1)]).T
67
68          def fbcca_classify(epochs, freqs, band_filters, weights):
69              n_trials = epochs.shape[0]
70              scores = np.zeros((n_trials, len(freqs)))
71              N = epochs.shape[2]
72              refs = {f: generate_reference(f, N, sr) for f in freqs}
73              for i in range(n_trials):
74                  for j, (b, a) in enumerate(band_filters):
75                      X_filt = filtfilt(b, a, epochs[i], axis=1).T
76                      for k, f in enumerate(freqs):
77                          cca = CCA(n_components=1)
78                          cca.fit(X_filt, refs[f])
79                          X_c, Y_c = cca.transform(X_filt, refs[f])
80                          rho = np.corrcoef(X_c[:, 0], Y_c[:, 0])[0, 1]
81                          scores[i, k] += weights[j] * (rho ** 2)
82              preds = np.argmax(scores, axis=1)
83              return preds
84
85          pred_dense = fbcca_classify(main_epochs, freqs, band_filters, weights)
86          pred_freqs = [freqs[i] for i in pred_dense]
87          mapped_preds = [min(range(len(target_freqs)), key=lambda j: abs(pred - target_freqs[j
                ]))
88                          for pred in pred_freqs]
89
90          acc = accuracy_score(main_labels, mapped_preds)
91          results.append((subject, recording, acc))
92          print(f"{subject} - Rec {recording} → Accuracy: {acc:.2%}")
93
94      except Exception as e:
95          print(f"Error processing {filename}: {e}")
96
97  # === Summary Output ===
98  print("\n=== Accuracy per Recording ===")
99  for subj, rec, acc in results:
100     print(f"{subj}{rec}: {acc:.2%}")
101
102 print("\n=== Average Accuracy per Subject ===")
103 per_subject = defaultdict(list)
104 for subj, rec, acc in results:
105     per_subject[subj].append(acc)
106
107 for subj in sorted(per_subject.keys()):
108     avg_acc = np.mean(per_subject[subj])
109     print(f"{subj}: {avg_acc:.2%}")
```

A.1.2. ETRCA

```python
import os
from glob import glob
import numpy as np
import scipy.io as sio
from scipy.signal import iirnotch, butter, filtfilt
from sklearn.metrics import accuracy_score, confusion_matrix
from collections import defaultdict
# === Setup ===
data_dir = r"C:\Users\Adam\OneDrive\Documenten\Bep\BCI\Signal Processing\Code\Datasets"
mat_files = sorted(glob(os.path.join(data_dir, "S*.mat")))

selected_channels = [
    80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
    139, 140, 141, 142, 143, 144, 145, 146, 147, 148
]
target_freqs = [6.66, 7.5, 8.57, 10.0, 12.0]
main_labels = [0]*3 + [1]*3 + [2]*3 + [3]*3 + [4]*3  # 15 trials
adapt_labels = [1, 0, 3, 4, 0, 2, 4, 1]              # fixed for all subjects

results = []
subject_best_acc = {}

# === Process all files ===
for filepath in mat_files:
    try:
        filename = os.path.basename(filepath)
        subject = filename[:4]
        recording = filename[4]

        mat = sio.loadmat(filepath)
        eeg = mat['eeg']
        din1 = mat['DIN_1']
        sr = float(mat['samplingRate'])

        event_samples = [int(x[0][0]) for x in din1[3]]
        diffs = np.diff(event_samples)
        gap_idx = np.where(diffs > 1250)[0]
        block_starts = np.concatenate(([0], gap_idx + 1))
        block_ends = np.concatenate((gap_idx, [len(event_samples)]))
        blocks = [(s, e) for s, e in zip(block_starts, block_ends)]

        adapt_blocks = blocks[:8]
        main_blocks = blocks[8:23]

        adapt_starts = [event_samples[b[0]] for b in adapt_blocks]
        main_starts = [event_samples[b[0]] for b in main_blocks]

        # Filters
        b_notch, a_notch = iirnotch(50/(sr/2), Q=30)
        b_band, a_band = butter(4, [5/(sr/2), 45/(sr/2)], btype='band')
        subbands = [(5, 45), (10, 45), (15, 45), (20, 45), (25, 45)]
        band_filters = [(butter(4, [lo/(sr/2), hi/(sr/2)], btype='band')) for lo, hi in
            subbands]

        def preprocess(eeg_data, starts):
            epochs = []
            for start in starts:
                ep = eeg_data[selected_channels, start:start+int(5*sr)]
                ep = filtfilt(b_notch, a_notch, ep, axis=1)
                ep = filtfilt(b_band, a_band, ep, axis=1)
                epochs.append(ep)
            return np.stack(epochs)

        adapt_epochs = preprocess(eeg, adapt_starts)
        main_epochs = preprocess(eeg, main_starts)

        # Templates
        n_classes = 5
        adapt_templates = {cls: [] for cls in range(n_classes)}
        for (b, a) in band_filters:
```

```
            band_data = filtfilt(b, a, adapt_epochs, axis=2)
            for cls in range(n_classes):
                idx = [i for i, lbl in enumerate(adapt_labels) if lbl == cls]
                if idx:
                    adapt_templates[cls].append(np.mean(band_data[idx], axis=0))
                else:
                    adapt_templates[cls].append(np.zeros((band_data.shape[1], band_data.shape
                        [2])))

        # TRCA weights
        W = {cls: [] for cls in range(n_classes)}
        for (b, a) in band_filters:
            band_data = filtfilt(b, a, adapt_epochs, axis=2)
            for cls in range(n_classes):
                idx = [i for i, lbl in enumerate(adapt_labels) if lbl == cls]
                if len(idx) < 2:
                    W[cls].append(np.zeros(band_data.shape[1]))
                    continue
                X_cls = band_data[idx]
                X_sum = np.sum(X_cls, axis=0)
                Q = np.zeros((X_cls.shape[1], X_cls.shape[1]))
                for trial in X_cls:
                    Q += trial @ trial.T
                S = X_sum @ X_sum.T - Q
                eigvals, eigvecs = np.linalg.eig(np.linalg.pinv(Q + 1e-6 * np.eye(Q.shape[0])
                    ) @ S)
                w = eigvecs[:, np.argmax(np.real(eigvals))]
                W[cls].append(np.real(w))

        # Classify
        pred_etrca = []
        for i in range(main_epochs.shape[0]):
            scores_cls = np.zeros(n_classes)
            for b_idx, (b, a) in enumerate(band_filters):
                trial = filtfilt(b, a, main_epochs[i], axis=1)
                for cls in range(n_classes):
                    w = W[cls][b_idx]
                    if np.allclose(w, 0):
                        corr_val = 0
                    else:
                        y_trial = w.T @ trial
                        y_temp = w.T @ adapt_templates[cls][b_idx]
                        if np.std(y_trial) == 0 or np.std(y_temp) == 0:
                            corr_val = 0
                        else:
                            corr_val = np.corrcoef(y_trial, y_temp)[0, 1]
                    weight = (b_idx+1)**(-1.25) + 0.25
                    scores_cls[cls] += weight * corr_val
            pred_etrca.append(np.argmax(scores_cls))

        acc = accuracy_score(main_labels, pred_etrca)
        results.append((subject, recording, acc))
        print(f"{subject} - Rec {recording} → eTRCA Accuracy: {acc:.2%}")

        # Update best accuracy per subject
        if subject not in subject_best_acc or acc > subject_best_acc[subject][1]:
            subject_best_acc[subject] = (recording, acc)

    except Exception as e:
        print(f" Error in {filename}: {e}")

# === Final Output ===
print("\n=== eTRCA Accuracy per Recording ===")
for subj, rec, acc in results:
    print(f"{subj}{rec}: {acc:.2%}")

print("\n=== Best Accuracy per Subject ===")
for subj in sorted(subject_best_acc.keys()):
    rec, best_acc = subject_best_acc[subj]
    print(f"{subj} → Best: {rec} with {best_acc:.2%}")
```

A.2. Self-Measured Dataset Code

```
import numpy as np
import pandas as pd
from scipy.signal import butter, filtfilt, iirnotch
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.cross_decomposition import CCA
def extract_trials_from_csv(filepath, sr=250):
    """
    Load EEG CSV, remove trigger channel, extract and filter 6 flicker trials.
    Each trial is 10 seconds (2500 samples) long.
    Returns: numpy array of shape (6, 8, 2500)
    """
    # Load data (skip header row), keep only first 8 EEG channels
    df = pd.read_csv(filepath, skiprows=1, header=None)
    raw = df.iloc[:, :-11].astype(float).values.T  # shape: (8, samples)
    print("Raw EEG shape:", raw.shape)

    # Flicker normaal 2 targets
    # flicker_samples = [
    #     (7500, 8750),     # tf1
    #     (10000, 11250),   # tf1
    #     (12500, 13750),   # tf1
    #     (17500, 18750),   # tf2
    #     (20000, 21250),   # tf2
    #     (22500, 23750),   # tf2
    # ]

    # # Flicker demba 2 targets 26 mei
    # flicker_samples = [
    #     (7500, 8750),     # tf1
    #     (10000, 11250),   # tf1
    #     (12500, 13750),   # tf1
    #     (17500, 18750),   # tf2
    #     (20000, 21250),   # tf2
    #     (22500, 23750),   # tf2
    # ]

    #Flicker 4 targets
    # flicker_samples = [
    #     (7500, 8750),     # tf1
    #     (10000, 11250),   # tf1
    #     (12500, 13750),   # tf1
    #     (17500, 18750),   # tf2
    #     (20000, 21250),   # tf2
    #     (22500, 23750),   # tf2
    #     (27500, 28750),   # tf3
    #     (30000, 31250),   # tf3
    #     (32500, 33750),   # tf3
    #     (37500, 38750),   # tf4
    #     (40000, 41250),   # tf4
    #     (42500, 43750)    # tf4
    # ]
    #Flicker 5 targets
    flicker_samples = [
        (7500, 8750),     # tf1
        (10000, 11250),   # tf1
        (12500, 13750),   # tf1
        (17500, 18750),   # tf2
        (20000, 21250),   # tf2
        (22500, 23750),   # tf2
        (27500, 28750),   # tf3
        (30000, 31250),   # tf3
        (32500, 33750),   # tf3
        (37500, 38750),   # tf4
        (40000, 41250),   # tf4
        (42500, 43750),   # tf4
        (47500, 48750),   # tf5
        (50000, 51250),   # tf5
```

```
        (52500, 53750)    # tf5
    ]

    # Define filters
    b_notch, a_notch = iirnotch(50 / (sr / 2), Q=30)
    b_band, a_band = butter(5, [5 / (sr / 2), 45 / (sr / 2)], btype='band')

    # Extract, filter, and store each trial
    trials = []
    for start, end in flicker_samples:
        eeg_segment = raw[:, start:end]  # (8, 2500)
        #eeg_segment = apply_car(eeg_segment)
        eeg_filtered = filtfilt(b_notch, a_notch, eeg_segment, axis=1)
        eeg_filtered = filtfilt(b_band, a_band, eeg_filtered, axis=1)
        trials.append(eeg_filtered)

    return np.stack(trials)  # shape: (6, 8, 2500)

sr=250
def setup_band_filters(sr):
    """Create bandpass filters for 5 sub-bands"""
    subbands = [(5, 45), (10, 45), (15, 45), (20, 45), (25, 45)]
    filters = []
    for low, high in subbands:
        b, a = butter(4, [low / (sr / 2), high / (sr / 2)], btype='band')
        filters.append((b, a))
    return filters
# def setup_band_filters(sr):
#     bands = [(8, 90), (8, 70), (8, 60), (8, 50), (8, 40), (8, 30), (8, 25), (8, 20)]
#     filters = []
#     for low, high in bands:
#         b, a = butter(4, [low / (sr / 2), high / (sr / 2)], btype='band')
#         filters.append((b, a))
#     return filters



def generate_reference(f, N, fs, nh=3):
    """Generate sine/cosine reference signal"""
    t = np.arange(N) / fs
    ref = np.vstack(
        [np.sin(2 * np.pi * k * f * t) for k in range(1, nh + 1)] +
        [np.cos(2 * np.pi * k * f * t) for k in range(1, nh + 1)]
    ).T
    return ref  # shape: (N × 2*nh)

def fbcca_classify(epochs, freqs, band_filters, sr=250, weights=None, default_threshold=0.02)
    :
    """
    Perform FBCCA classification on EEG epochs.

    Parameters:
        epochs: np.ndarray
            EEG data of shape (n_trials, n_channels, n_samples)
        freqs: list of float
            List of target frequencies
        band_filters: list of (b, a)
            List of IIR filter coefficients for sub-bands
        sr: int
            Sampling rate
        weights: list of float or None
            Weights for each sub-band
        default_threshold: float
            Threshold on max score to assign to "default" class (-1)

    Returns:
        preds: np.ndarray
            Predicted class indices (or -1 for default class)
        scores: np.ndarray
            Score matrix of shape (n_trials, n_freqs)
    """
```

```
    n_trials, n_channels, n_samples = epochs.shape
    n_freqs = len(freqs)
    scores = np.zeros((n_trials, n_freqs))
    refs = {f: generate_reference(f, n_samples, sr) for f in freqs}

    if weights is None:
        weights = [1 / (j + 1) for j in range(len(band_filters))]

    for i in range(n_trials):
        print(f"\nTrial {i+1}:")
        for j, (b, a) in enumerate(band_filters):
            X_filt = filtfilt(b, a, epochs[i], axis=1).T  # (samples × channels)
            weight = weights[j]
            for k, f in enumerate(freqs):
                Y = refs[f]  # (samples × harmonics)
                cca = CCA(n_components=1)
                cca.fit(X_filt, Y)
                X_c, Y_c = cca.transform(X_filt, Y)
                rho = np.corrcoef(X_c[:, 0], Y_c[:, 0])[0, 1]
                scores[i, k] += weight * (rho ** 2)
                print(f"  Sub-band {j+1}, Freq {f:.2f} Hz: corr = {rho:.4f}, weighted score
                    += {weight*rho**2:.4f}")

    # Determine predicted classes
    max_scores = np.max(scores, axis=1)
    preds = np.argmax(scores, axis=1)

    # Handle default class assignment
    preds[max_scores < default_threshold] = -1
    for i, score in enumerate(max_scores):
        if preds[i] == -1:
            print(f"Trial {i+1} classified as DEFAULT (no freq match, max score = {score:.4
                f})")
        else:
            print(f"Trial {i+1} predicted class = {preds[i]}, max score = {score:.4f}")

    return preds, scores

filepath = r"C:\Users\Adam\OneDrive\Documenten\Bep\BCI\Signal Acquisition\recordings\
    Recordings 27-05\Demba 6.66 7.5 8.57 10 and 12.csv"
# === Set parameters ===
sr = 250
freqs_to_test = [6.66, 7.50,8.57,10, 12]  # You control this
#true_labels = [0, 0, 0, 1, 1, 1]  # Adjust based on your ground truth
true_labels = [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]

# === Run full pipeline ===
main_epochs = extract_trials_from_csv(filepath, sr=sr)
band_filters = setup_band_filters(sr=sr)
preds, scores = fbcca_classify(main_epochs, freqs_to_test, band_filters, sr=sr)

# === Evaluation ===
print("Frequencies tested:", freqs_to_test)
print("Predicted indices: ", preds)
print("Predicted freqs:    ", [freqs_to_test[i] for i in preds])
print("True freqs:        ", [freqs_to_test[i] for i in true_labels])

correct = sum(p == t for p, t in zip(preds, true_labels))
print(f"\nAccuracy: {correct}/{len(true_labels)} = {correct / len(true_labels):.2%}")

# === Optional: Confusion Matrix ===
cm = confusion_matrix(true_labels, preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=freqs_to_test, yticklabels=freqs_to_test)
plt.xlabel("Predicted Frequency")
plt.ylabel("True Frequency")
plt.title("FBCCA Confusion Matrix")
plt.show()
```

A.3. Real-Time Code

```
def setup_band_filters(sr=250): # Added default sr
    subbands = [(5, 45), (10, 45), (15, 45), (20, 45), (25, 45)] # Standard sub-bands for FBCCA
    filters = []
    for low, high in subbands:
        # Ensure Nyquist frequency is not violated
        if high >= sr / 2:
            high = sr / 2 - 1 # Adjust high if it's too close or above Nyquist
            print(f"Warning: Adjusted high frequency for bandpass filter to {high} Hz due to sampling rate {sr} Hz.")
        if low >= high: # Ensure low is less than high
            print(f"Warning: Skipping sub-band ({low},{high}) as low >= high or high is too close to Nyquist for sr={sr}.")
            continue
        if low <= 0: # Ensure low is positive
            print(f"Warning: Skipping sub-band ({low},{high}) as low is not positive.")
            continue

        b, a = butter(4, [low / (sr / 2), high / (sr / 2)], btype='band')
        filters.append((b, a))
    if not filters: # Fallback if all sub-bands were invalid
        print(f"Warning: No valid sub-bands for sr={sr}. Using a default wide band [5, 40] Hz.")
        b, a = butter(4, [5 / (sr/2), min(40, sr/2 -1) / (sr/2)], btype='band')
        filters.append((b,a))
    return filters

def generate_reference(f, N, fs, nh=3):
    t = np.arange(N) / fs
    ref_signals = []
    for k in range(1, nh + 1):
        ref_signals.append(np.sin(2 * np.pi * k * f * t))
        ref_signals.append(np.cos(2 * np.pi * k * f * t))
    return np.vstack(ref_signals).T # shape: (N × 2*nh)

# --- FBCCA Classification Core (modified to be more general) ---
def fbcca_classify_core(eeg_segment, freqs_to_test, band_filters, sampling_rate, weights=None):
    """
    Core FBCCA classification logic for a single EEG segment.
    eeg_segment: shape (channels, samples) - pre-filtered (bandpass, notch)
    freqs_to_test: list of target frequencies
    band_filters: list of (b, a) tuples for sub-band filtering
    sampling_rate: The sampling rate of eeg_segment
    weights: weights for each sub-band (optional)

    Returns: predicted_freq_index, predicted_freq_value, max_cca_score
    """
    n_freqs = len(freqs_to_test)
    if n_freqs == 0:
        return -1, 0.0, 0.0 # No frequencies to test

    n_samples = eeg_segment.shape[1]
    if n_samples == 0:
        print("Warning: Empty EEG segment passed to fbcca_classify_core.")
        return -1, 0.0, 0.0

    # Generate reference signals for the given segment length
    refs = {f: generate_reference(f, n_samples, sampling_rate, nh=3) for f in freqs_to_test}

    scores_for_segment = np.zeros(n_freqs)

    for i_band, (b_sub, a_sub) in enumerate(band_filters):
        # Apply sub-band filter
        # Ensure eeg_segment has enough samples for filtfilt (at least 3 * max(len(a), len(b)))
        # For butter order 4, len(a) and len(b) are 5. So need at least 15 samples.
        if eeg_segment.shape[1] < 15: # A common requirement for filtfilt with order 4
            print(f"Warning: Segment too short ({eeg_segment.shape[1]} samples) for sub-band
```

```
                filtering. Skipping sub-band {i_band}.")
64              continue
65
66          X_sub_band_filtered = filtfilt(b_sub, a_sub, eeg_segment, axis=1).T # Transpose to (
                samples, channels)
67
68          for k_freq, f_target in enumerate(freqs_to_test):
69              Y_ref = refs[f_target] # Shape (samples, 2*nh)
70
71              # Ensure X and Y have the same number of samples (should be guaranteed by
                    N_samples)
72              if X_sub_band_filtered.shape[0] != Y_ref.shape[0]:
73                  print(f"Warning: Mismatch in samples for CCA: X_filt {X_sub_band_filtered.
                        shape[0]}, Y_ref {Y_ref.shape[0]}. Skipping.")
74                  continue
75
76              cca = CCA(n_components=1)
77              try:
78                  cca.fit(X_sub_band_filtered, Y_ref)
79                  X_c, Y_c = cca.transform(X_sub_band_filtered, Y_ref)
80                  # Handle potential all-zero transformed components (can happen with noisy/
                        short data)
81                  if np.all(X_c == 0) or np.all(Y_c == 0):
82                      rho = 0.0
83                  else:
84                      rho = np.corrcoef(X_c[:, 0], Y_c[:, 0])[0, 1]
85
86                  current_weight = weights[i_band] if weights is not None and i_band < len(
                        weights) else 1.0
87                  scores_for_segment[k_freq] += current_weight * (rho ** 2)
88              except ValueError as e:
89                  print(f"CCA ValueError for freq {f_target} Hz, sub-band {i_band}: {e}.
                        Skipping this CCA.")
90                  # This can happen if input contains NaNs or Infs, or if X/Y are rank-
                        deficient for CCA.
91                  continue # Skip this CCA calculation
92
93      if n_freqs == 0 or (np.sum(scores_for_segment) == 0 and n_freqs > 0) : # Handle no
            frequencies or all scores are zero
94          predicted_idx = -1
95          predicted_freq_val = 0.0
96          normalized_confidence_score = 0.0
97      else: # n_freqs > 0 and at least one score is non-zero
98          predicted_idx = np.argmax(scores_for_segment)
99          predicted_freq_val = freqs_to_test[predicted_idx]
100         raw_max_score = scores_for_segment[predicted_idx]
101
102         # Normalize the score:
103         # The denominator is the sum of weights of all sub-bands that *could* contribute.
104         # This assumes weights are positive. rho^2 is max 1.
105         # `weights` is the parameter passed to fbcca_classify_core.
106         # In process_live_eeg_segment, it's `fbcca_weights = [1.0] * len(sub_band_filters)`.
107
108         actual_sum_of_weights = 0.0
109         if weights is not None and len(weights) == len(band_filters):
110             # Sum positive weights. If all are 1.0, this is len(band_filters).
111             actual_sum_of_weights = sum(w for w in weights if w > 0)
112         elif weights is None:
113             # If weights were None, accumulation loop used 1.0 per band.
114             actual_sum_of_weights = float(len(band_filters))
115         else:
116             # Mismatched weights length. This is an issue. Fallback for robustness.
117             print(f"Warning: Mismatch in weights length for normalization. len(weights)={len(
                    weights)}, len(band_filters)={len(band_filters)}. Assuming default weights of
                    1.0 for normalization denominator.")
118             actual_sum_of_weights = float(len(band_filters)) # Fallback
119
120         if actual_sum_of_weights > 0:
121             normalized_confidence_score = raw_max_score / actual_sum_of_weights
122             # Clip to ensure the score is strictly within [0, 1]
123             # due to potential floating point inaccuracies or unexpected rho values.
```

```
            normalized_confidence_score = np.clip(normalized_confidence_score, 0.0, 1.0)
        else:
            # This case (actual_sum_of_weights <= 0) implies no bands or all weights non-positive.
            # If raw_max_score is also 0 (likely), then 0.0 is correct.
            # If raw_max_score is > 0 but sum_weights is 0, it's an anomaly; 0.0 is a safe fallback.
            normalized_confidence_score = 0.0

    return predicted_idx, predicted_freq_val, normalized_confidence_score

# --- 2. Basic Preprocessing (Filters) ---
    # Bandpass filter (e.g., 5-45 Hz)
    b_band, a_band = butter(4, [5 / (sampling_rate / 2), min(45, sampling_rate/2 - 1) / (sampling_rate / 2)], btype='band')
    # Notch filter (e.g., 50 Hz or 60 Hz depending on region)
    # Assuming 50Hz for now, make this configurable if needed
    notch_freq = 50
    if notch_freq >= sampling_rate / 2:
        print(f"Warning: Notch frequency {notch_freq}Hz is too high for sampling rate {sampling_rate}Hz. Skipping notch filter.")
        eeg_filtered_band = filtfilt(b_band, a_band, eeg_segment_selected_channels, axis=1)
    else:
        b_notch, a_notch = iirnotch(notch_freq / (sampling_rate / 2), Q=30)
        eeg_filtered_band = filtfilt(b_band, a_band, eeg_segment_selected_channels, axis=1)
        eeg_filtered_final = filtfilt(b_notch, a_notch, eeg_filtered_band, axis=1)

    # --- 3. FBCCA Classification ---
    # Get sub-band filters (these are independent of the main bandpass/notch applied above)
    sub_band_filters = setup_band_filters(sr=sampling_rate)

    if not sub_band_filters:
        print("Error: Could not set up sub-band filters for FBCCA.")
        return -1, 0.0, 0.0

    # Weights for sub-bands (can be optimized, e.g., based on SNR of harmonics)
    # For now, using equal weights as a starting point.
    # Example: weights = [k**(-1.25) + 0.25 for k in range(1, len(sub_band_filters) + 1)]
    fbcca_weights = [1.0] * len(sub_band_filters) # Equal weights

    predicted_idx, predicted_val, confidence = fbcca_classify_core(
        eeg_filtered_final,
        target_frequencies,
        sub_band_filters,
        sampling_rate,
        weights=fbcca_weights
    )

    return predicted_idx, predicted_val, confidence


# --- Existing Offline Plotting and Main Execution (for testing/reference) ---
def plot_trial_fft_offline(trials, target_freqs_values, sr=250, max_freq=30):
    n_trials = trials.shape[0]
    cols = 3
    rows = int(np.ceil(n_trials / cols))
    fig, axes = plt.subplots(rows, cols, figsize=(15, 4 * rows))
    axes = axes.flatten()
    for i in range(n_trials):
        trial = trials[i]
        signal = trial.mean(axis=0)
        fft_vals = np.abs(np.fft.rfft(signal))
        fft_freq = np.fft.rfftfreq(len(signal), d=1/sr)
        axes[i].plot(fft_freq, fft_vals, label='FFT')
        axes[i].axvline(target_freqs_values[i], color='red', linestyle='--', label='Target Freq')
        axes[i].axvline(2 * target_freqs_values[i], color='green', linestyle='--', label='2nd Harmonic')
        axes[i].set_xlim(0, max_freq)
        axes[i].set_title(f"Trial {i+1} - Target: {target_freqs_values[i]} Hz")
        axes[i].set_xlabel("Frequency (Hz)")
```

```
        axes[i].set_ylabel("Amplitude")
        axes[i].legend()
    for j in range(n_trials, len(axes)): fig.delaxes(axes[j])
    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    print("--- Testing Offline Pipeline (from original script) ---")
    sr_offline = 250
    # Example file path - replace with an actual path if you want to run this
    example_filepath = "eeg_data/trial_eeg_data_placeholder.csv" # Placeholder

    # This part will likely fail if the placeholder file doesn't exist or has wrong format.
    # It's here for structural reference from the original script.
    try:
        df_example = pd.read_csv(example_filepath)
        df_example = df_example.iloc[:, :-11] # Adjust slicing as per your CSV
        eeg_data_offline_example = df_example.values.T

        freqs_to_test_offline = [8.57, 12] # Example frequencies
        true_labels_offline = [0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1] # Example labels

        main_epochs_offline = extract_trials_from_data_offline(eeg_data_offline_example, sr=
            sr_offline)

        # For offline, you might run ICA on each epoch or concatenated epochs
        # cleaned_epochs_offline = np.array([ICA_offline(epoch)[0] for epoch in
            main_epochs_offline])

        band_filters_offline = setup_band_filters(sr=sr_offline)

        # Offline FBCCA would iterate through epochs:
        all_preds_offline = []
        all_scores_offline = []
        print("\nSimulating offline FBCCA on extracted epochs:")
        for i_epoch, epoch_data in enumerate(main_epochs_offline): # Using non-ICA'd for this
             example
            pred_idx, pred_val, conf = fbcca_classify_core(
                epoch_data, freqs_to_test_offline, band_filters_offline, sr_offline
            )
            all_preds_offline.append(pred_idx)
            all_scores_offline.append(conf) # Storing only max score for simplicity here
            print(f"Epoch {i_epoch+1}: Pred Idx: {pred_idx}, Pred Freq: {pred_val:.2f} Hz,
                Confidence: {conf:.4f}, True Label Idx: {true_labels_offline[i_epoch]}")

        if all_preds_offline:
            correct_offline = sum(p == t for p, t in zip(all_preds_offline,
                true_labels_offline) if p != -1) # Count correct only if a prediction was
                made
            valid_predictions = sum(1 for p in all_preds_offline if p != -1)
            accuracy_offline = correct_offline / valid_predictions if valid_predictions > 0
                else 0
            print(f"\nOffline Accuracy: {correct_offline}/{valid_predictions} = {
                accuracy_offline:.2%}")

            cm_offline = confusion_matrix([true_labels_offline[i] for i,p in enumerate(
                all_preds_offline) if p!=-1], # only use true labels where a prediction was
                made
                                          [p for p in all_preds_offline if p!=-1])
            if cm_offline.size > 0:
                sns.heatmap(cm_offline, annot=True, fmt='d', cmap='Blues',
                            xticklabels=freqs_to_test_offline, yticklabels=
                                freqs_to_test_offline)
                plt.xlabel("Predicted Frequency")
                plt.ylabel("True Frequency")
                plt.title("Offline FBCCA Confusion Matrix (Simulated)")
                plt.show()
        else:
            print("No offline predictions made.")

    except FileNotFoundError:
```

```
        print(f"Offline example data file not found: {example_filepath}. Skipping offline
            pipeline test.")
    except Exception as e:
        print(f"Error in offline pipeline test: {e}")


    print("\n\n--- Testing Live Processing Function ---")
    test_sr = 250  # Hz
    test_duration = 1  # seconds
    test_num_channels_raw = 8 # Total channels in the raw mock data
    test_num_samples = int(test_sr * test_duration)
```

A.4. Plots Code

A.4.1. FFT

```
import matplotlib.pyplot as plt
import numpy as np

def plot_all_main_trial_spectra(epochs, labels, fs=250, max_freq=30):
    n_trials = len(epochs)
    cols = 5
    rows = int(np.ceil(n_trials / cols))
    fig, axes = plt.subplots(rows, cols, figsize=(18, 10))
    axes = axes.flatten()

    for i, trial in enumerate(epochs):
        # Average over channels
        signal = trial.mean(axis=0)
        fft_vals = np.abs(np.fft.rfft(signal))
        fft_freq = np.fft.rfftfreq(len(signal), 1/fs)

        axes[i].plot(fft_freq, fft_vals)
        axes[i].axvline(labels[i], color='r', linestyle='--', label='Target Freq')
        axes[i].set_xlim(0, max_freq)
        axes[i].set_title(f"Trial {i+1} (label={labels[i]} Hz)", fontsize=9)
        axes[i].set_xlabel("Frequency (Hz)")
        axes[i].set_ylabel("Amplitude")
        axes[i].legend(fontsize=6)

    # Hide any extra subplots
    for i in range(n_trials, len(axes)):
        fig.delaxes(axes[i])

    fig.tight_layout()
    plt.suptitle("Power Spectrum of Main Trials (averaged across channels)", fontsize=14, y
        =1.02)
    plt.show()

# Call it
plot_all_main_trial_spectra(main_epochs, labels=[6.66]*3 + [7.5]*3 + [8.57]*3 + [10.0]*3 +
    [12.0]*3)
```

A.4.2. Correlation plot

```
import pandas as pd
import numpy as np
from sklearn.cross_decomposition import CCA
from scipy.signal import butter, filtfilt
import matplotlib.pyplot as plt

def load_single_csv_trial(filepath, sr=250):
    """
    Load a single 10s EEG trial from a CSV file with 8 channels.
    Returns: (channels, samples)
    """
    df = pd.read_csv(filepath, skiprows=1, header=None)
    data = df.iloc[:, :8].astype(float).values.T  # shape: (8, samples)
    return data

```

```

def generate_reference(f, N, fs, nh=3):
    t = np.arange(N) / fs
    ref = np.vstack(
        [np.sin(2 * np.pi * k * f * t) for k in range(1, nh + 1)] +
        [np.cos(2 * np.pi * k * f * t) for k in range(1, nh + 1)]
    ).T
    return ref

def setup_band_filters(sr):
    subbands = [(5, 45), (10, 45), (15, 45), (20, 45), (25, 45)]
    filters = []
    for low, high in subbands:
        b, a = butter(4, [low / (sr / 2), high / (sr / 2)], btype='band')
        filters.append((b, a))
    return filters

def fbcca_debug_single_trial(epoch, freqs, band_filters, sr=250):
    n_bands = len(band_filters)
    n_freqs = len(freqs)
    N = epoch.shape[1]
    ref_signals = {f: generate_reference(f, N, sr) for f in freqs}
    rho_squared = np.zeros((n_bands, n_freqs))

    for j, (b, a) in enumerate(band_filters):
        X_filt = filtfilt(b, a, epoch, axis=1).T  # (samples × channels)
        for k, f in enumerate(freqs):
            Y = ref_signals[f]
            cca = CCA(n_components=1)
            cca.fit(X_filt, Y)
            X_c, Y_c = cca.transform(X_filt, Y)
            rho = np.corrcoef(X_c[:, 0], Y_c[:, 0])[0, 1]
            rho_squared[j, k] = rho ** 2

    return rho_squared


# === Parameters ===
filepath = r"C:\Users\Adam\OneDrive\Documenten\Bep\BCI\Signal␣Acquisition\recordings\
    Recordings␣21-05\ch1␣10Hz␣sine␣wave.csv"
sr = 250
freqs_to_test = [7, 8.0 ,8.5,9.0, 9.5, 10.0, 10.5, 11, 11.5, 12.0, 20.0, 30]  # test around
    the known signal

# === Load & prepare ===
epoch = load_single_csv_trial(filepath, sr=sr)  # (channels, samples)
band_filters = setup_band_filters(sr)

# === Run FBCCA debug ===
rho_sq = fbcca_debug_single_trial(epoch, freqs_to_test, band_filters, sr)

# === Print scores ===
total_scores = rho_sq.sum(axis=0)
for i, f in enumerate(freqs_to_test):
    print(f"Freq␣{f}␣Hz␣→␣Total␣FBCCA␣score:␣{total_scores[i]:.4f}")

# === Plot per-band scores ===
bands = [f"{low}-{high}Hz" for (low, high) in [(5, 45), (10, 45), (15, 45), (20, 45), (25,
    45)]]
plt.figure(figsize=(8, 4))
for i, f in enumerate(freqs_to_test):
    plt.plot(range(len(bands)), rho_sq[:, i], marker='o', label=f"{f}␣Hz")
plt.xticks(range(len(bands)), bands)
plt.ylabel(" ²␣Score")
plt.title("Per-Band␣FBCCA␣ ²␣Scores␣(10␣Hz␣Recording)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

A.4.3. Confusion matrix

```
#true_labels = [0, 0, 0, 1, 1, 1]  # Adjust based on your ground truth
true_labels = [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]

# === Run full pipeline ===
main_epochs = extract_trials_from_csv(filepath, sr=sr)
band_filters = setup_band_filters(sr=sr)
preds, scores = fbcca_classify(main_epochs, freqs_to_test, band_filters, sr=sr)

# === Evaluation ===
print("Frequencies tested:", freqs_to_test)
print("Predicted indices: ", preds)
print("Predicted freqs:    ", [freqs_to_test[i] for i in preds])
print("True freqs:        ", [freqs_to_test[i] for i in true_labels])

correct = sum(p == t for p, t in zip(preds, true_labels))
print(f"\nAccuracy: {correct}/{len(true_labels)} = {correct / len(true_labels):.2%}")

# === Optional: Confusion Matrix ===
cm = confusion_matrix(true_labels, preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=freqs_to_test, yticklabels=freqs_to_test)
plt.xlabel("Predicted Frequency")
plt.ylabel("True Frequency")
plt.title("FBCCA Confusion Matrix")
plt.show()
```

B

B Supplementary Material: Extended Results and Visualizations

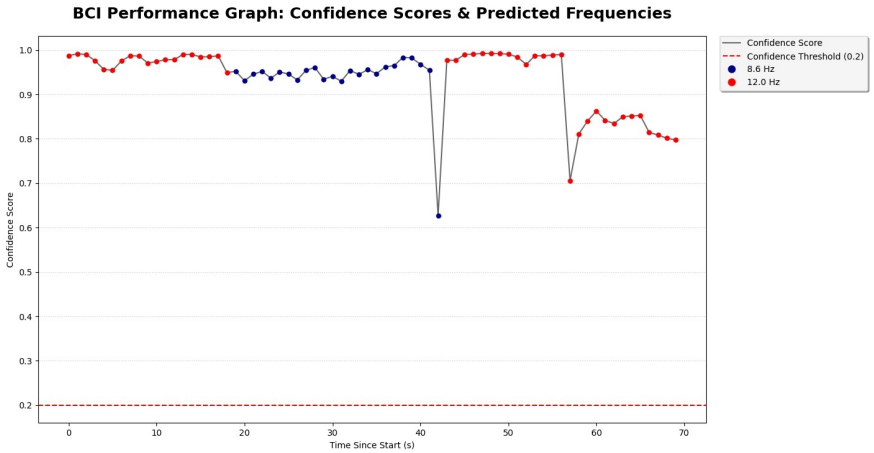


Figure B.1: Time- and frequency-domain SNR from an average accuracy recording. A 1 V sine wave was induced using a function generator at the electrodes of the EEG headset, with 6 trials per target. The two target frequencies were 8.57 Hz and 12 Hz.

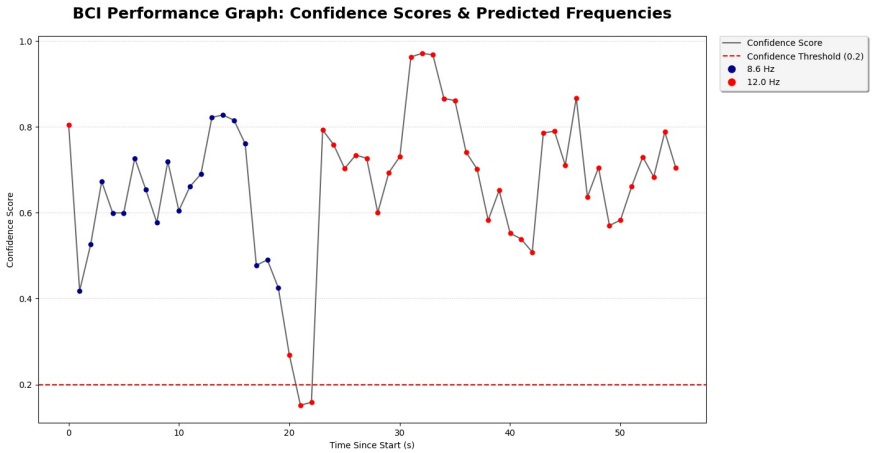


Figure B.2: Time- and frequency-domain SNR from an average accuracy recording. A 100 μV sine wave was induced using a function generator at the electrodes of the EEG headset, with 6 trials per target. The two target frequencies were 8.57 Hz and 12 Hz.

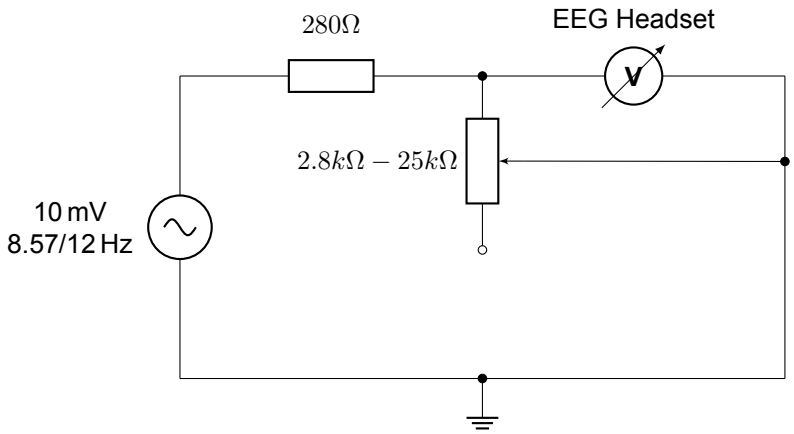


Figure B.3: Measurement setup with function generator and tunable voltage divider

C

C Statement use of AI

AI tools were used to support the writing of this Bachelor thesis by summarizing papers, clarifying methods, improving grammar and vocabulary, and refining self-written text. All AI-assisted content was reviewed to ensure it met academic standards, and all analysis and conclusions are the authors' own work.