UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Low cost Brain Computer Interface system for AR.Drone Control

Dissertação de mestrado.

Rafael Mendes Duarte

Orientador: Prof. Alexandre Trofino Neto, PhD.

Florianópolis, May, 2017.

Rafael Mendes Duarte

LOW COST BRAIN COMPUTER INTERFACE SYSTEM FOR AR.DRONE CONTROL

Dissertação submetida ao Programa
de Pós-Graduação em Engenharia de
Automação e Sistemas da Univer-
sidade Federal de Santa Catarina
para a obtenção do Grau de Mestre
em Engenharia de Automação
de Sistemas

Universidade Federal de Santa Catarina – UFSC

Departamento de Automação e Sistemas

Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Orientador: Prof. Dr. Alexandre Trofino Neto

Florianópolis(SC) - Brasil

2017

Duarte, Rafael Mendes
Low cost Brain Computer Interface system for AR.Drone Control / Rafael Mendes Duarte ; orientador, Alexandre Trofino Neto, 2017.
102 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2017.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Brain Computer Interface. 3. Drone. 4. Electroencephalography. I. Trofino Neto, Alexandre. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

Rafael Mendes Duarte

LOW COST BRAIN COMPUTER INTERFACE SYSTEM FOR AR.DRONE CONTROL

Este trabalho foi julgado aprovado como **DISSERTAÇÃO DE MESTRADO** no Curso de Pós-Graduação em Engenharia de Automação e Sistemas e aprovada em sua forma final pela Banca Examinadora designada.

ABSTRACT

This work presents the design, implementation, and testing of a Brain-Computer Interface (BCI) system based on μ-waves to control the navigation of a drone. BCI systems perform the translation of brain signals into commands to communicate with external applications. The μ rhythm is a type of brain signal response to motor activity which can be easily measured by electroencephalography (EEG). For this reason, μ-waves based BCI systems have been extensively explored in the literature as a way of enabling patients with compromised neuromotor functions to interact with the outside world. To implement the signal processing and application interface routines, a software platform was built based on well-established filter and classification techniques, such as the Common Spatial Patterns (CSP) and the Linear Discriminant Analysis (LDA). For interfacing with the drone, an algorithm for translating the classifier outputs into drone commands was proposed. In addition, the acquisition of brain waves was performed by a low-cost and open-hardware EEG amplifier called OpenBCI. The validation of the designed system was performed using public and an acquired motor imagery EEG datasets, which were supplied to the platform to simulate the real-time performance of the system. The tests, conducted in a drone simulator, demonstrated the correct operation of the proposed methodology and the designed system.

Keywords: Brain-Computer Interfaces, Drone, Electroencephalography, EEG

RESUMO

Este trabalho apresenta o projeto, implementação e teste de um sistema de Interface Cérebro Máquina (BCI) baseado em ondas μ para o controle da navegação de um drone comercial. Sistemas BCI realizam a tradução dos sinais cerebrais em comandos que podem ser usados para ativação e controle de aplicações externas. O ritmo μ é um tipo de resposta cerebral que é modulado através da atividade motora e pode ser facilmente medido através de eletroencefalografia (EEG). Por este motivo, sistemas BCI baseados em ondas μ tem sido extensivamente explorados na literatura como uma forma de permitir que pacientes com sistema neuromotor comprometido interajam com o ambiente externo. Neste trabalho, uma plataforma de software foi desenvolvida para implementar as rotinas de processamento de sinal e de interface com a aplicação. Ténicas bem estabelecidas como o filtro espacial CSP e o classificador LDA foram utilizadas para realizar a detecção dos padrões cerebrais. Além disso, é proposta uma metodologia para traduzir o sinal de saída do classificador em comandos que podem ser diretamente enviados para o drone. Para aquisição dos sinais de EEG, um amplificador de baixo custo e *open-source* chamado OpenBCI foi utilizado. A implementação do sistema foi validada através de um conjunto de dados público, que foram utilizados na plataforma como forma de simular o comportamento em tempo-real do sistema. Os testes de aplicação foram conduzidos em um simulador do drone, o que demonstrou o correto funcionamento da metodologia proposta e do sistema desenvolvido.

Keywords: Interface Cérebro Máquina, Drone, Eletroencefalografia, EEG

RESUMO EXPANDIDO

INTRODUÇÃO

De acordo com organização mundial da saúde (OMS), aproximadamente quinhentas mil pessoas sofrem lesões na medula espinhal todo ano. Em muitos desses casos, a lesão causa comprometimento das funções motoras do paciente ao afetar a comunicação entre o sistema nervoso central (SNC) e o sistema motor periférico. Sistemas de interface cérebro-máquina (BCI) surgem como uma alternativa para restabelecer a mobilidade de tais pacientes, criando um canal alternativo entre o cérebro e um sistema motor artificial.

Em geral, aplicações BCI utilizam a dessincronização do ritmo μ como padrão a ser detectado para a geração dos sinais de controle dos dispositivos externos. A onda μ é um sinal oscilatório, de frequência predominante na banda alfa (8 a 12 Hz), que pode ser medido sobre a área do córtex motor. Tais potenciais aparecem quando o indivíduo não está realizando nenhuma tarefa motora, em estado de relaxamento. Ao realizar algum tipo de movimento, as ondas μ são atenuadas e não podem mais ser detectadas.

Dentre as possíveis aplicações de tais sistemas, destacam-se as que objetivam o controle de cursores em até 3 dimensões, de cadeira de rodas, robôs e carros. Nestes casos, são abordados diversos aspectos do controle de tais dispositivos como usabilidade, praticidade. Além disso, com o intuito de restaurar a autonomia de usuários, estudos demonstraram a capacidade de controle de veículos aéreos não tripulados (Drone), através de BCIs.

OBJETIVOS

O objetivo deste trabalho é permitir o controle de navegação de um veículo aéreo não tripulado através de comandos gerados a partir dos sinais de eletroencefalografia do usuário. O foco do projeto é desenvolver métodos de geração de sinais de controle para navegação do drone, excluindo-se considerações a respeito de seu comportamento dinâmico em resposta a estes sinais.

METODOLOGIA

No sistema desenvolvido, um amplificador de biosinais coleta os dados EEG do usuário e os envia para a interface de processamento, imple-

mentada em Python. Esta, por sua vez, realiza os cálculos necessários para a geração de um sinal de controle, que será enviado para o AR.Drone 2.0. A interface escrita em Python gerencia o fluxo de amostras vindo do amplificador e realiza a filtragem e classificação dos dados EEG. As rotinas de envio de comandos para o Drone também estão presentes neste bloco.

O amplificador utilizado para aquisição do sinal de EEG foi o OpenBCI, de baixo-custo e *open-source*. O OpenBCI amostra os sinais biológicos a uma taxa de 250 amostras por segundo e os envia em tempo real para a plataformas.

Para o processamento dos sinais, as amostras são filtradas em uma banda de interesse através de um filto passa-banda do tipo IIR. Em seguida, as características relevantes do sinal são extraídas e, posteriormente, fornecidas ao algoritmo de classificação conhecido como Análise de Discriminantes Linears (LDA). Este algoritmo gera um rótulo de classificação de acordo com a intenção do usuário, presente no sinal EEG.

Neste trabalho, foi proposta uma metodologia para a tradução dos rótulos emitidos pelo classificador em comandos de controle que podem ser diretamente enviados ao drone. Esses rótulos são armazenados em memória e, através de um estimativa baseada na média de classificação, um comando de alteração de direção é enviado para o drone. Dessa forma, foi possível controlar o drone através dos sinais de EEG coletados do usuário.

O sistema e rotinas desenvolvidas foram testadas através de um simulador de drone e de dados previamente coletados e salvos em memória. Nos testes, o objetivo foi controlar a direção do drone ao longo de um caminho pré-determinado. Durante os testes, foram medidos o tempo total de execução do percurso e o erro entre o caminho ótimo e o caminho percorrido.

RESULTADOS OBTIDOS

Os resultados demonstraram o correto funcionamento das metodologias propostas para controle de um drone ao longo de um caminho pré-determinado. A plataforma desenvolvida foi utilizada para controlar a direção da aeronave e guiá-la através do caminho. As taxas de erro e desvio do caminho ótimo foram aceitáveis.

O sistema BCI implementado foi testado utilizando dados públicos

da competição BBCI IV. A principal contribuição deste trabalho foi o detalhamento prático da implementação completa de um sistema BCI baseado em rítmos μ. Apesar de não terem sido realizados testes de controle do drone utilizando dados coletados em tempo-real de usuários, foi desenvolvido um sub-sistema para simular o comportamento de um usuário real. Dessa forma, foi possível inferir a performance do sistema e dos métodos através desses testes simulados.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

According to the World Health Organization, around 500 thousand people suffer from spinal cord injuries every year [49]. In many of these cases, the injury results in motor functions impairment by affecting the communication between the central nervous system and the motor system. Brain-Machine Interface (BCI) systems come as an option to improve life quality of such patients by enabling an alternative path between the brain and an auxiliary artificial motor system. For this reason, the research of new applications and the general improvement of BCI technologies have been pursued by several groups worldwide [13] [41].

These interfacing systems rely basically on the information stored in neural signals to extract the user's intent and, hence, to communicate with the outside world. In typical non-invasive BCI systems, the neural signal is acquired through electroencephalography (EEG), where sensors are placed on the scalp of the user to measure the neural activity [45]. The obtained signal is then filtered to remove noise components, before being digitally processed to identify pre-defined EEG patterns. The detected patterns can then be mapped into commands which are sent to a target application [50].

BCI systems can be built based on several kinds of brain patterns, including μ-wave oscillatory signals, P300 responses [15], steady state visual evoked responses (ssVEP) [27] [14] and so on. Each of these systems presents pros and cons regarding complexity, usability, and possible applications. In this sense, μ-wave based BCI systems have been the focus of a number of BCI research groups due to its direct correlation with motor functions and easy detection when compared to other types of brain responses [47] [41].

The μ-wave is an oscillatory response related to the neural motor activity which can be detected when the user is in a relaxed state [34]. As soon as the user engages in a motor task, the response vanishes and can no longer be measured. This control mechanism has made possible the implementation of μ-wave based BCI systems for controlling external applications through the modulation of motor activity [35]. Besides, one key advantage of this wave pattern is that it can be controlled by imagining motor activity, without the need of actually executing it [34]. This makes the μ-wave rhythms, often called motor imagery responses, specially interesting for BCI which focus on applications for patients with neuromotor impairments [28] [20] [9].

Despite the advantages of BCI systems, the task of acquiring brain waves, in general, imposes critical challenges on the practical implementation of these systems. The low amplitudes and low signal-to-noise ratio (SNR) of this type of biological signal require acquisition equipment with high precision and, as a consequence, high-cost [45]. In addition, the robustness requirements tend to compromise the portability and usability of this equipment. Moreover, since the EEG signal recorded by the electrodes is strongly affected by different noise sources, the task of detecting and extracting the relevant information becomes arduous. As a result, the signal processing stages in BCI systems must incorporate complex filtering and machine learning techniques for cleaning and accurately detect the neural activity patterns [25] [31]. This issue becomes even more emphasized as simpler, and more compact EEG acquisition equipment are used [18].

The mentioned characteristics impose severe limitations to the diffusion of BCI systems among society, significantly restricting its use to a controlled medical environment or research laboratories. Furthermore, these aspects are often neglected in the literature, which tends to focus more on the theoretical aspects of BCI systems. In this context, the investigation of BCI systems implemented with simpler and less costly equipment consists in a significant step for the propagation of these systems. In addition, the testing of the current state-of-the-art filtering and classification techniques with signals acquired from such equipment is also an important aspect which needs to be addressed.

Another important barrier frequently faced by BCI systems developers is the lack of complete reports about the implementation of a BCI system, from EEG signal acquisition to application control. In general, research groups tend to focus on modular parts of the system, such as filtering or classification techniques [5] [7], without adequately addressing the links between these submodules. More specifically, issues such as translating the output of classification algorithms to actual application commands are commonly omitted in the literature. This, as a result, imposes considerable challenges for the full implementation and suppresses the main contribution of BCI systems, which is the final control of an application.

The purpose of this work is to fully implement a μ-wave based BCI system for controlling an application with high degrees of freedom. As a byproduct, the issues found in the design of such system will be discussed and fully documented. The implementation will be based on well-known algorithms [30], which are commonly used in the BCI field for filtering and classifying EEG patterns. With the aim of addressing the issues cited above, the system will be implemented based on a low-cost and open-source EEG amplifier called OpenBCI. The idea is also to infer the applicability of new EEG devices for the design of complex and complete BCI systems. The application to be controlled by the system will be a commercial drone, which will allow to thoroughly assessing the performance of the system operating with a very flexible application.

In short terms, a drone presents many challenges regarding movement dynamics: it can move in 3 dimensions and presents very rapid flight dynamics [1]. For this reason, by implementing a BCI system in charge of controlling such target application, it is possible to test the dynamic and usability boundaries imposed by these systems. As BCI technologies evolve to control more complex applications, such as a drone, the control of more simple devices will become more robust and reliable. Also, as reported in the literature [26] [12], a BCI-controlled drone can unravel new frontiers for patients with severely compromised motor functions by enabling a new way of interacting with the outside world.

Although there are many BCI research projects aiming to control more complex applications [47] [20] [26] [8] [28], such as the one proposed in this work, the signal processing and classification algorithms used are in general simplistic. This, as a result, tends to overload the user training stages before the system use [11]. Usually, techniques based directly on energy levels of μ-waves are used in such applications, neglecting well-established processing techniques [5]. Although usually effective, these methods tend to compromise the generalization of the system and to demand extra training sessions for model adaptations and calibrations. This, in turn, results in a longer preparation time for the use of the system and, as a consequence, an even more challenging diffusion of designed system in practical applications.

The project presented here was validated in two ways. First, a public EEG dataset containing data from imagery motor tasks was used for simulating a user controlling the drone on a simulator. The EEG data from this dataset was rearranged to create a new dataset with the goal of controlling the drone across a pre-defined path. Next, the same procedure was performed using EEG data acquired using the OpenBCI amplifier. Although due to time limitations, it was not possible to test the system operating with the user in real-time, the performed tests show that the proposed methodology for connecting the BCI blocks managed to successfully translate the EEG data to commands which are sent to the drone. In addition, all the practical details of the work were fully documented, enabling the develop and testing of the system in real-time in the near future.

Chapter 2 covers the human brain physiological structure and the signals generated by it. Next, the characteristics of a BCI system will be discussed in Chapter 3, including all its submodules and their function. Chapter 4 covers the application proposed in this work, discussing the common concepts involved in a drone flight, its flight dynamics and, also, the drone model chosen for this work. The mechanisms used to link the BCI system output and the application, called Computer Machine Interface (CMI), will be explained in Chapter 5. Next, the software platform developed for this work will be presented. The goal of the platform is to implement all the practical requirements of a BCI system, as it will be seen later. The results obtained in this work are then presented in Chapter 7. Finally, the last chapter provides a few conclusions and future perspectives obtained throughout this work.

CHAPTER 2

Brain Activity Signals

The brain is one the most complex and diverse organ of the human body. To understand how the mechanisms and outputs of the brain can be explored when implementing a Brain Computer Interface system, a brief review of its structure and physiology is required. The goal of this chapter is to provide a general overview of the brain structure and dynamics which are commonly explored in BCI applications. The first section presents the different regions and tissues of the brain as well as their roles in the central nervous system. The mechanism involved in generating the neural signals from these tissues are also reviewed. Subsequently, the discussion focus on the properties of the signals acquired from these tissues. More specifically, the properties of electroencephalography (EEG) are described.

2.1 Brain Structure

The brain is divided into subregions and areas which play different roles in the control of the human body activities [37]. Each of these sub-areas is responsible for performing various tasks and, therefore, generating different signals which can be diversely explored by BCI systems.

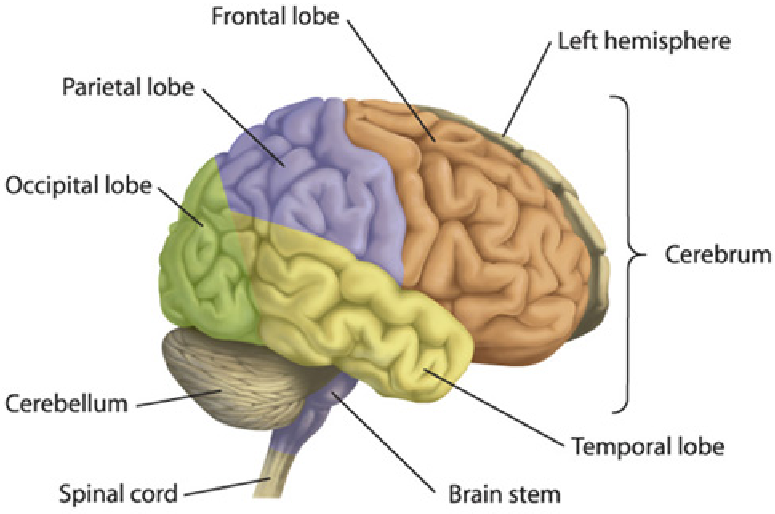


Figure 2.1: The different regions and parts of the human brain. The cortex is located at the topmost part of the brain and is of crucial importance for BCI systems[Adapted from [37]]

Superficially, the brain regions are organized as depicted in Figure 2.1. The longitudinal fissure divides the brain structure into two symmetrical hemispheres (left and right). On top of each hemisphere, there is a thick layer of gray matter called cortex. Structures such as the brain stem, thalamus and basal ganglia, called subcortical areas, are positioned beneath the cortex layer. The cortex can be divided into five major parts or lobes: frontal (orange), parietal (purple), occipital (green), temporal (yellow) and insula (not shown in the picture). These subregions are mainly determined by small grooves (gyri) and ridges (sulci) along the cortex [37].

The brain cortex is of great importance for BCI systems since it is associated with a significant part of functions and neurological responses. For instance, the temporal lobe is related to auditory information processing whereas the parietal lobe is linked to the somatosensory information. One region of the cortex which is regularly explored by BCI implementations due to its direct relationship with motor functions is the Primary Motor Cortex (M1) [50] [19].

The M1 region is located in the frontal lobe along the central sulcus, in a strip going approximately from one ear to another along the surface of the cortex. Each sub-region along the M1 is responsible for controlling the motor activity of different parts of the human body. Figure 2.2 is known as the Wilder Penfield Homunculus and shows the map of the portions of the M1 responsible for controlling a different muscle of the human body. The limbs that require a more sophisticated controlling, such as hands and feet, have a larger area of the M1 for information processing and, therefore, are depicted on a larger scale in the diagram. It is important to notice that the Homunculus diagram only displays half of the human body. Also, because of how the paths that carry neural information to the muscles are arranged, all the information processed at one side of the brain is sent to limbs on the opposite side of the body. For instance, a hand movement command generated from the left hemisphere will be forwarded to the right-hand [37].

Also, another important characteristic of the cortex is that it is located at the uppermost part of the brain. For this reason, signals generated there are less attenuated by the organic structures of the human head and easier to capture by non-invasive methods of signal acquisition. This feature makes possible the measurement of cortex signals through sensors positioned on the surface of the head [45]. This non-invasive procedure is called electroencephalography and will be further explained in the next section.

2.2 Electroencephalography Signals

When information is transmitted and processed by the brain, chemical interactions, which take place at synapses, occur between the neurons (high-specialized brain cells). These chemical reactions open and close ion channels of the neuron's membrane, altering the potential difference between the intracellular medium and the extra-cellular medium. Eventually, the overall change in the potential reaches a particular limit and triggers the transmission of an electrochemical pulse along the neuron (action potential), hence transmitting information across the neural tissue. The firing of action potentials influences the flow of charges inside

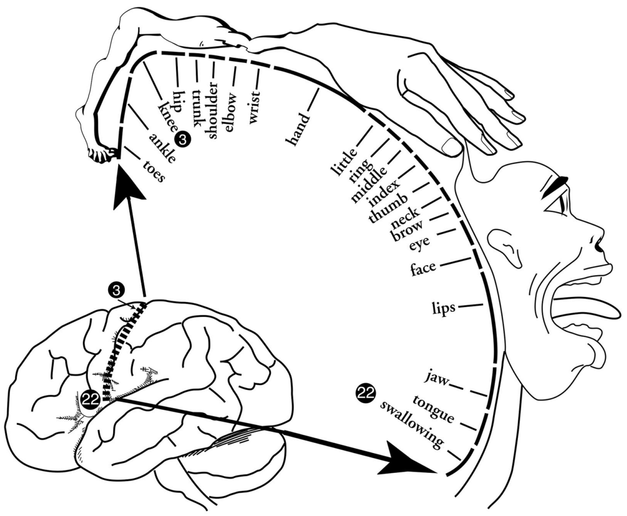


Figure 2.2: The Wilder Penfield Homunculus: the drawing of the limbs along the M1 region shows the regions responsible for generating a motor control signal to different parts of the body.[Adapted from [19]]

the brain tissues and, therefore, generates signals that can be measured which are directly related to the brain activity [37].

Although there are methods to directly measure the change in potential generated by the firing of action potentials by single neurons, these methods usually involve surgical procedures which are too complex and risky for BCI applications. However, the action potential generated by groups of neurons create a response potential strong enough to be measured by sensors positioned on the scalp, avoiding the need for such invasive procedures [50]. One of them is called electroencephalography and is continually explored by BCI applications due to its simplicity and non-invasive characteristics.

In Figure 2.3, the EEG signals are plotted in the time domain for different electrodes. In the frequency domain, the EEG signals are subdivided into frequency bands which are related to various kinds of neural responses. Mainly, the frequency spectrum of EEG signals is

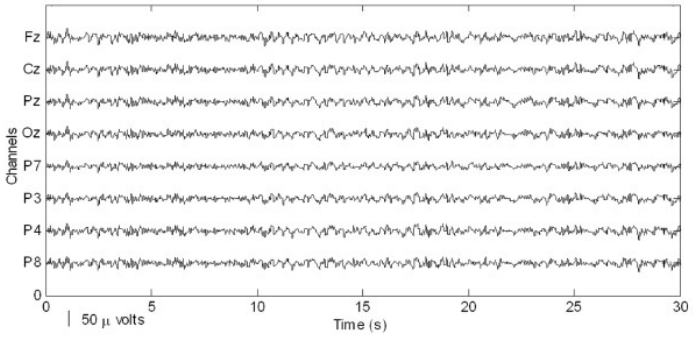


Figure 2.3: Plot of EEG signals along time for different channels positioned according to the 10-20 system.

divided into the following denominations and ranges [50]:

- Delta: 1-3Hz
- Theta: 4-7Hz
- Alpha: 8-12Hz
- Beta: 13-30Hz
- Gamma: 31-50Hz

The alpha band, for instance, is more present in the occipital and central area of the cortex. It is related, among others, to motor control activities. For this reason, this work will frequently discuss the signal energy concentrated in the alpha band.

The EEG signals are collected by positioning electrodes on the scalp of the user. But, as discussed, the region at which each sensor is placed is of great importance for the subsequent analysis of the acquired signals. Therefore, to place the electrodes over critical and pre-defined regions of the brain, a positioned system called 10-20 is often used for EEG data acquisition [45]. The 10-20 system marks the position of the EEG electrodes relatively to the dimensions of the head of the user. Figure 2.4 shows the organization of the electrodes on the scalp according to the 10-20 system. Each electrode is labeled according to the cortex region which it is measuring (example: O - Occipital region).

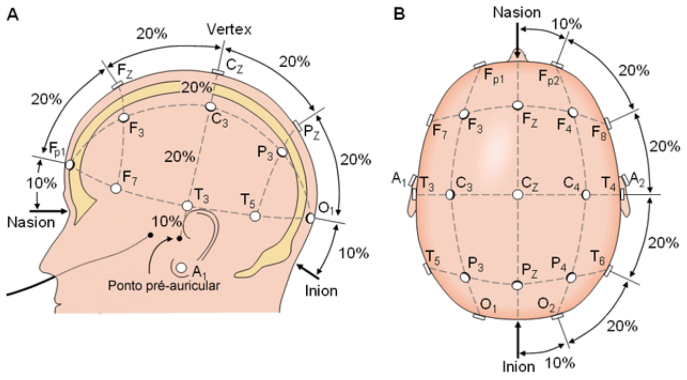


Figure 2.4: The 10-20 positioning system: the electrodes are separated between distances of 10 and 20 percent of the dimensions of the head. [Adapted from [40]]

The even-numbered electrodes denote the right hemisphere whereas the odd-numbered electrodes denote the left hemisphere of the brain. The percentages are about the distance between the Nasion (upper part of the nose) and Inion (back of the head). The 10-20 system is an important standard for EEG data acquisition and was the standard adopted in this work.

2.3 Artifacts

Although the acquisition of EEG signals is simplified using non-invasive procedures, the fact that the signals are read relatively far from their sources imposes severe limitations to the overall quality of the acquired signals. The presence of signals not originated from the brain, commonly denoted artifacts, considerably deteriorates and suppresses the information related to neural activities [22]. These unwanted signals can be generated from multiple sources, such as [50]:

- *Electromyography (EMG)*: electrical activity generated by muscles surrounding the head region (neck, face, etc...)
- *Electroculography (EOG)*: generated by eye movements
- *60 Hz*: Noise generated by the power supply lines. It is one of the main noise sources in EEG acquisitions. EEG amplifiers are

usually equipped with filters to strongly filter out this kind of noise

- *Poor connection between electrodes and skin*: Poorly connected electrodes generate high-impedance connections which increase 60 Hz contamination
- *Electrocardiography (ECG)*: In some patients, the heart electrical signals can influence the EEG signals.

The presence of artifacts in EEG signals is reduced by standard procedures during the acquisition sessions or by implementing sophisticated filtering algorithms. However, the complete elimination of such noise sources if often not possible, making artifacts inherently present in any EEG signal.

2.4 Motor Imagery in EEG signals

In general, EEG signals encompass the overall activity of the brain during the acquisition sessions. However, specially for BCI applications, it is not possible to analyze all the brain responses at once and, hence, it is more interesting to explore specific responses of the brain to particular types of stimuli or even to specific behaviors of the patient. For this purpose, many responses have been explored in BCI systems such as P300 responses [15], steady-state visual evoked potentials (ssVEP) [14] [9], μ-waves [43] and so on. Each of these reactions is suitable for different kinds of BCI applications. Due to its simplicity and relatively easy detection, mu-wave based BCI systems have been extensively studied and implemented [7] [11]. Hence, this work will focus mainly on this kind of response.

The μ-wave is an oscillatory signal with frequency spectrum within the alpha-band (9-13 Hz) [34]. The mu-waves can be measured mainly over the motor cortex, being directly related to the motor activity being executed by the brain [37]. These rhythms are found in EEG signals due to the synchronized firing of pyramidal neurons in the cortex (synchronization). This coordinated firing happens when the brain is not engaged in any motor task or a relaxed state. When motor activities are being executed, such as moving hands, this synchronization is interrupted, and the mu-waves are attenuated (desynchronization) [11].

In Figure 2.5, the synchronization and desynchronization stages are depicted in details. The top figures show the topographic map of a human head with the colors representing the energy of the signal in different parts of the brain (red represents the areas with the highest magnitude). The plot A accounts for the magnitudes when the user is in a relaxed state whereas plot B shows the energy map when the user is engaged in a motor task. In the bottom curve, the frequency spectrum of the two scenarios is depicted. When the user is relaxed, the energy presented over the motor cortex area is high (red area). Also, for this case, the frequency spectrum shows an amplitude peak at around 10 Hz. However, when the user is engaged in the motor activity, the plot B shows no relevant difference in the energy magnitudes throughout the brain. Also, in the scenario shown in plot B, the amplitude peak is strongly attenuated. In fact, the μ-waves are also modulated by imagining the movement of the limbs, without actually executing the action itself [34]. This particularly characteristic makes μ-wave based BCI especially suited for patients with compromised motor functions but with healthy brain functions, such as those affected by Amyotrophic Lateral Sclerosis (ALS).

Another important fact illustrated in figure 2.5 is that there is also synchronization information in the beta frequency range, around 20 Hz. The amplitude peak shown at this frequency is also directly related to motor activity and is modulated according to the same behavior described above for the amplitude peak at 10 Hz [6]. Consequently, it is common in μ-waves based BCI system to include the beta frequency range in the signal processing stages for the detection of synchronization and desynchronization patterns. As it will be shown, this is accomplished by implementing a wider bandpass filter at the beginning of the processing data flow, with a frequency band of around 8 to 30 Hz.

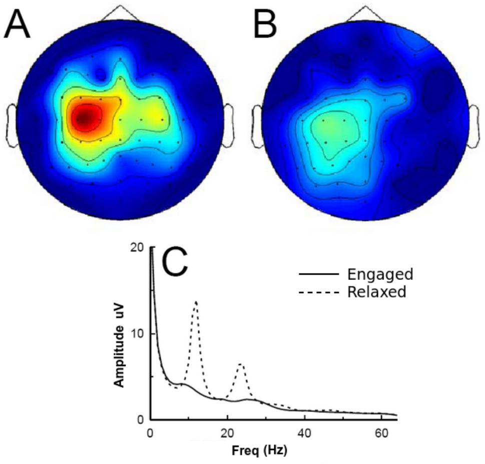


Figure 2.5: Plot A: topographic map displaying the signals energy when the user is not performing a motor task. Plot B: same topographic map when the user is engaged in a motor task (left hand movement). Plot C: Frequency Spectrum of channel C3 for scenarios A (dashed line) and B (line)[Adapted from [23]]

CHAPTER 3

BCI Systems

The previous chapter described the basic behavior of the brain and how this behavior affects the biological signals that can be measured by electroencephalography. This section describes how a system that can take advantage of such signals works. These systems are called brain-computer interfaces since they take the information stored in neural signals and translate it into useful commands which are sent to an application. Thus, a BCI system provides an interface between the brain and external devices [50].

Figure 3.1 shows a functional diagram of a generic BCI system. The first stage encompasses the brain signal acquisition, at which the analog data generated by the brain activity is digitized and delivered to a digital processing system. The signal is then processed using filtering techniques to attenuate noise and enhance specific characteristics. At the processing stage, these characteristics, often called features, are classified based on pre-defined patterns. Finally, the patterns detected are transformed into commands that are transmitted to a target device, responsible for executing a given task [19]. Although this last stage, which generates the commands to the external application, is described in the literature as a submodule of the BCI system, in this work it will

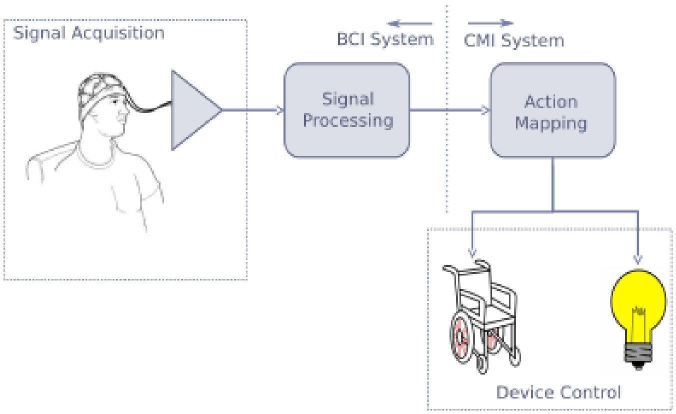


Figure 3.1: General diagram of a Brain-computer interface system. The brain signal is acquired and digitized by an amplifier and sent to the signal processing module. The data is then processed to detect and classify discriminant patterns. Based on this classification, the CMI system can compute command and send them to the target application.

be treated as an additional interface system called Computer-Machine Interface System (CMI). The CMI system comprises all the routines responsible for computing the command based on the classification outputs. For simplicity, however, the present work shall often refer to a BCI system as the complete architecture.

In the next sections, the signal acquisition and the processing stages will be discussed, providing an overview of the most important aspects involved in the design of a BCI system. The CMI system and, hence, the generation of action commands will be covered later in Chapter 5, which is dedicated to the interfacing techniques between BCI systems and their applications.

Finally, although the techniques explained in the first sections of this chapter are crucial to building a complete motor imagery-based BCI system, sometimes it might be unclear how these submodules are connected to building an entire BCI system. There are important practical considerations that must be taken into account for the implementation, which is not covered in details in the literature. For this reason, the last two section will provide an analysis of two important practical parts of BCI systems: the model calibration and the real-time or online operation.

3.1 EEG Signals Acquisition

When working with BCI systems, one of the main concerns is to make sure the signal acquisition stage is reliable and well performed [45]. The measurement of EEG signals consists in analyzing the dynamic and spatial behavior of the current sources located at cortical areas, as explained in the previous chapter. A well-performed signal acquisition session implies in signals with high quality and with a tolerable presence of artifacts. These characteristics, as will be shown in the subsequent stages, result in a BCI system with higher accuracy rates and better usability.

In BCI systems, the EEG signal digital conversion is performed by an EEG amplifier, which takes as inputs the signals captured by the electrodes, performs basic analog filtering and amplification and send the digitized version of the signals to a digital processing system. The EEG amplifiers can be found in the market with different specification and features. The specifications can vary regarding the number of input electrodes, analog-to-digital resolution (in bits), frequency bandwidth and so on. Since this equipment requires very precise electronic components and robust safety circuitry, the cost tends to be greater when compared to the prize of conventional analog amplifiers [18]. As a consequence, the budget needed to use this equipment tend to limit the spread of BCI research activities [9]. This work will explore the utilization of a low-cost and open-source EEG amplifier called OpenBCI, which will be presented in Chapter 6.

3.2 Signal processing

Although the EEG amplifier plays a significant role in the overall quality of the implementation of the BCI system, the subsequent stage, which is responsible for digitally processing the acquired signal, is also crucial to increase the accuracy and usability of the system [30]. In this section, the signal processing mechanisms applied to the incoming signal from the brain are discussed.

In Figure 3.2, a detailed block diagram of the signal processing stage

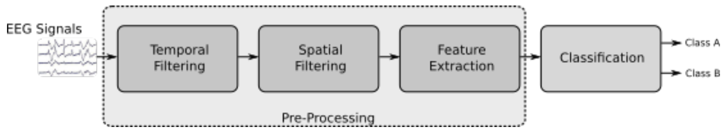


Figure 3.2: Signal processing unit of a BCI system. The EEG signal is digitized and filtered. Next, the spatial filter performs a linear transformation of the signal to enhance discriminative features of the signal. These features are then extracted and delivered to the classifier, which outputs a label for the input signal.

implemented in a BCI system is shown. In general, the signal processing can be divided into two submodules: pre-processing and classification. The first is responsible for filtering out noise components as well as mathematically rearranging the signal to improve certain properties [48]. Also, in the preprocessing stage, specific parameters are extracted from the filtered signal to, finally, be used as separable information by the classifier afterwards [5]. These steps will be discussed in more depth in the following subsections.

3.2.1 Preprocessing

The raw EEG signal contains the recorded potential in each channel as a function of time. Together with the signals related to brain activity, noise, as well as unwanted signals such as artifacts [22], can be found. The goal of the preprocessing stage is to perform mathematical transformations on the input signal to extract or highlight only the relevant information.

The preprocessing stage can be subdivided into smaller modules which are responsible for transforming the signal in different ways and, as result, filter out different kinds of noise or artifacts. Two important filtering techniques commonly used in BCI systems are temporal and spatial filtering. The former seeks to attenuate noise sources in the time domain, which can also be seen as eliminating noise components in the frequency domain. The latter is used to transform the signals to a different subspace and, hence, to highlight specific characteristics there [48]. When used together, these two techniques, which will be discussed in detail in the sequel, can significantly increase the accuracy rates of the BCI system [30].

3.2.1.1 Temporal Filtering

The first preprocessing stage consists of applying a bandpass filter to the signal. The goal is to select only the frequency band that contains the desired information, excluding noise and other undesired information stored in other frequencies ranges. Finite or Infinite Impulse Response (FIR and IIR) type filters are usually chosen to implement this stage due to their simplicity and effectiveness [30]. A notch filter to remove 60 Hz noise components (Brazilian standard) is also implemented when the EEG amplifier does not provide such implementation in hardware (analog filtering at the input of the amplifier).

To provide a brief overview of the filtering process, consider a full rank matrix $Z \in \mathbb{R}^{n \times q}$ representing a raw EEG segment with n channels e_k, where $k = 1,...,n$, and q samples per channel acquired with a sampling frequency f_s. For each channel e_k, the following equation describes the filtering process which outputs a filtered signal x_k [36]:

$$x_k[\hat{n}] = \frac{1}{a_0}\left(\sum_{i=0}^{P} b_i e_k[\hat{n}-i] - \sum_{j=1}^{Q} a_j e_k[\hat{n}-j]\right), \tag{3.1}$$

where $\hat{n} = 0,1,2,3...q$ represents the sample index. The parameters b and a are called the filter coefficients and have dimensions defined by the filter orders P and Q. Applying the z-transform to equation 3.1 and rearranging the result, the filter transfer function becomes:

$$H(z) = \frac{\sum\limits_{k=0}^{M} b_k z^{-k}}{1 + \sum\limits_{k=1}^{N} a_k z^{-k}}. \tag{3.2}$$

Hence, the coefficients b perform a feed-forward filtering while coefficients a provide a feedback loop. Based on equation 3.2, these parameters can be easily selected using standard filter design techniques according to the filter specifications and constraints [5]. In FIR filters, the coefficients a are equal to zero, canceling the feedback component and yielding inherently stable filters. However, FIR implementations require higher order filters (more coefficients) when compared to IIR implementations to achieve similar filtering specifications. For this reason, to simplify the structure of the filtering stage, an IIR filter was used in this work.

The temporal filtering stage significantly enhances the quality of the input signal, reducing noise and eliminating unwanted frequency components. In most EEG signals used as inputs to BCI systems, however, there is also crucial information stored in the spatial distribution of the signals recorded throughout the scalp [48] [50]. Therefore, the extracted features can also be improved in the spatial sense, as will be seen in the next section.

3.2.1.2 Spatial Filtering

When applying temporal filtering techniques to the signal, the samples acquired along time are combined to generate a filtered version of the input the signal. Therefore, the transformation is performed along time (or frequency). When applying spatial filtering techniques, the samples of different channels are combined to generate an enhanced version of the signal [42]. This means that, when designing and applying the spatial filter, the spatial distribution of the signal is the core information to generate the filter response. Note that this is specially attractive for EEG signals and BCI systems since the location of the signals sources are directly related to the type and properties of brain activity being recorded. In other words, there is also crucial information stored in the spatial distribution of the EEG signals and that is the main reason why this stage is so important for BCI systems.

There is a number of different techniques for enhancing the spatial characteristics of EEG signals such as Common Average Reference (CAR), Laplacian Reference (LAR), Principal Components Analysis (PCA), Independent Components Analysis (ICA) [19], Common Spatial Patterns [32] and so on [5] [30]. The choice of which method to use depends on the type of the EEG response which is targeted. In this work, since the focus is on applications based on motor imagery responses such as μ-waves, only the Common Spatial Patterns (CSP) algorithm will be discussed in detail.

As discussed previously, the preprocessing stage is crucial for enhancing specific characteristics or features of the signal. In general, these characteristics are directly related to the patterns which are to be detected by the system. For motor imagery BCI systems, these patterns might include hand movement (left or right), feet, tongue and so on [35]. These different patterns, in machine learning denotation, are often called classes. This nomenclature will frequently be used in this work.

The CSP seeks a linear projection of the filtered EEG signals in a subspace in which the characteristics that define each class are highlighted. The method relies, basically, on the fact that, for different motor imagery tasks, there is a significant difference between the energy (variance) recorded in the channels from one brain hemisphere when compared to the energy found in the opposite hemisphere [48]. This energy is quantified through the covariance matrices of the signals from each class. The algorithm then computes different directions which maximize the discrepancy between the energy of the signals of both classes. The data is then projected onto these directions to ease the job at the classification stage, which takes place subsequently.

To begin the mathematical analysis of the CSP algorithm, let $Z \in \mathbb{R}^{n \times q}$ be a full rank matrix representing a raw EEG segment with n channels and q samples per channel acquired with a sampling frequency f_s. Also, after applying the filtering technique described in subsection 3.2.1.1, let $X \in \mathbb{R}^{n \times q}$ be a full rank matrix representing the samples of the filtered EEG signal obtained from a band pass filter with desired frequency range, $\Delta f \triangleq f_u - f_l$, where f_l and f_u are the lower and upper cutoff frequencies, in *Hertz*, respectively.

The algorithm is computed based on a dataset containing EEG data from when the user was executing two different motor tasks. This dataset is used to train a mathematical model which describes the distribution of these two groups of data. By using this model, the CSP seeks a linear projection matrix $W \in \mathbb{R}^{n \times n}$ to project the data onto a new subspace which highlights the discrepancies between the two groups of data. This linear transformation can be written as [48]:

$$Y = WX \tag{3.3}$$

where $Y \in \mathbb{R}^{n \times q}$ is the spatially filtered signal. The covariance matrix of a signal can be written as (notice that since the signal mean is zero due to the temporal filtering, the covariance and correlation matrices are equal):

$$\mathbf{cor}\{X\} = XX^T. \tag{3.4}$$

Hence, applying equation 3.4 to equation 3.3 yields:

$$\mathbf{cor}\{Y\} = WXX^TW^T = W\mathbf{cor}\{X\}W^T, \tag{3.5}$$

which is a direct form of defining the correlation matrix of the projected matrix Y.

The goal now is to define the projection W which maximizes the distinctions between the two classes. To compute the CSP model, the complete EEG dataset is segmented into smaller portions of EEG signals which surround time events marked along the signal acquisition, defined as X_i. These time events highlight the time frames at which the user was executing a particular activity. The extracted segments are often referred to as epochs and, therefore, this procedure is commonly denoted as *epoch extraction*. For instance, consider a dataset with two different tasks $\mathscr{A}$ and $\mathscr{B}$ being executed during the experiment (left and right-hand movement, for instance). The dataset is segmented into n_a and n_b epochs, which correspond to the EEG activity recorded during the execution of task $\mathscr{A}$ and $\mathscr{B}$, respectively.

Supposing that $\mathscr{A} \cap \mathscr{B} = \oslash$ and considering the correlation matrix $\mathbf{cor}(X_i) \in \mathbb{R}^{n \times n}$ of a given EEG sample segment, the CSP algorithm first computes the mean correlation matrices of both classes, defined as C_a and C_b, respectively.

$$C_a = \frac{1}{n_a} \sum_{i=1}^{n_a} \mathbf{cor}\{X_i\} \tag{3.6}$$

and

$$C_b = \frac{1}{n_b} \sum_{i=1}^{n_b} \mathbf{cor}\{X_i\}. \tag{3.7}$$

Next, the matrix M is defined as the sum of the mean covariances of both classes, which can also be represented by its spectral decomposition, as follows [17]:

$$M = C_a + C_b = U\Lambda U^T, \tag{3.8}$$

where Λ, U are respectively the diagonal matrix with eigenvalues their respective eigenvectors ($UU^T = I$). Since the matrix M is computed from the sum of the two covariance matrices C_a and C_b, it is reasonable to assume that $M > 0$ (positive definite matrix), since the EEG channels are inherently independent signals. Based on this assumption, the square root of M can then be computed by $M^{-\frac{1}{2}} = U\Lambda^{-\frac{1}{2}}U^T$. Hence, equation (3.8) can be normalized and rearranged resulting in:

$$M^{-\frac{1}{2}}C_aM^{-\frac{1}{2}} + M^{-\frac{1}{2}}C_bM^{-\frac{1}{2}} = S_a + S_b = I_n \tag{3.9}$$

From equation (3.9), we conclude that matrices S_a and S_b have a common basis of eigenvectors, i.e. $S_a = V\Lambda_aV^T$, $S_b = V\Lambda_bV^T$, where $V^TV = I_n$. Moreover, their eigenvalues are complementary, in the sense that $\Lambda_a + \Lambda_b = I_n$. This implies that the eigenvector associated with the largest eigenvalue of S_a is the eigenvector associated with smallest eigenvalue of S_b. The CSP projection matrix is then defined as:

$$W = V^TM^{-\frac{1}{2}} \tag{3.10}$$

and the filtered version of the signal can be found using equation 3.3.

The projection matrix W can be interpreted as a spatial transformation to the signal X that renders diagonal and complementary (with respect to the identity matrix) the mean correlation matrices of the two classes. The idea is that the transformed signals WX_i will lead the mean correlation matrices C_a and C_b to be diagonal and complementary.

The interesting aspect of the spatial transformation in (3.10) is that it allows establishing properties of the classes from the eigenvalues of the mean correlation matrix (the eigenvectors are the same). The largest eigenvalues of one class will correspond to the smallest ones of the other class, leading to very different properties. However, the eigenvalues close to 0.5 are not useful for class discrimination in the sense that they will be practically the same for both classes. For this reason, the eigenvectors, which are often referred to as *neighbors*, associated with eigenvalues close to 0.5 are discarded from (3.10). This leads to a projection matrix, namely $W \in \mathbb{R}^{2r\times n}$, with reduced dimension when compared to the number of channels, where r is the number of pairs of eigenvectors selected and $2r \leq n$ [48]. This advantage will be further explored in the feature extraction stage next.

3.2.1.3 Feature Extraction

After applying the CSP transformation, the discriminative characteristics of the output signal must be extracted to be classified [30]. In the machine learning field, these characteristics are commonly known as features and, therefore, this stage is called feature extraction.

As discussed previously, the CSP seeks a linear projection which maximizes the energy of the signals from a given class while minimizing the energy of the data of the other class. Hence, it makes sense to consider the energy of the spatially filtered signals (components) as features. Besides, since components with eigenvalues close to 0.5 offer no discriminative advantages, it is a common practice when using the CSP method to pick only the components with eigenvalues close to 1 or 0.

In this case, the feature vector $\boldsymbol{p}_i$ of a given element X_i is chosen as the diagonal entries of the projected correlation matrix, which is defined in equation 3.5 (the diagonal elements of the correlation matrix represents the variance or energy of each component), and can be expressed as [51]:

$$\boldsymbol{p}_i = \mathrm{vdiag}(\mathbf{cor}\{WX_i\}) = \mathrm{vdiag}(W\mathbf{cor}\{X_i\}W^T) \tag{3.11}$$

where vdiag(.) represents a vector whose entries are the diagonal elements of matrix in (.).

The feature vector above will be utilized by the classifier, in the subsequent stage of signal processing. It is important to choose the right set of features taking into account the characteristics of the signal to be classified and, more importantly, the mechanisms used for classification, which will be presented next.

3.2.2 Pattern Detection and Classification

After extracting the relevant information from the EEG signal, the BCI system needs to detect and classify the data based on the features arrays [19]. This is accomplished by the classifier, which translates the features into a labeled output of one of the original classes. The accuracy and performance rates of the classification model, which is considered to be one of the cores of a BCI system, directly impacts the usability and communication speed of the system. For this reason, the selection and design of such stage must take into account the characteristics of the input features and how they are distributed in the feature space [25].

Many types of classifiers have been implemented in BCI applications such as Support Vector Machine (SVM) [24], Fuzzy classifiers [29], Neural Networks [21], Regression [44], based on Riemann Metrics [4], Linear Discriminant Analysis (LDA) [51] and so on. Purportedly, each of these methods offers advantages and disadvantages regarding complexity, computational cost, speed, generalization, and accuracy. Also, more importantly, the properties of the feature vector are critical information to be taken into account when selecting the correct classifier. In this work, the LDA classifier will be used to translate the classes labels.

The LDA is a simple yet accurate method for classifying linear separable data, in which data points from different classes can be separated by a line or a plane [50]. The output characteristics of the pre-processing performed by the CSP makes the LDA classification mechanism ideal to separate the data points of motor-imagery data, as it will be seen next. Other methods such as SVM and Neural Networks can also achieve similar results in classifying this kind of data. However, these methods tend to present a high complexity with no significant improvement in the accuracy rates, making it not suitable for more simple applications such as the one explored in this work [5] [31].

A classifier consists of a mathematical model which can be applied to new (but similar) information and, hence, be used to detect patterns in the data. To obtain it, the model must be trained based on a set of examples. Depending on the classifier algorithm, the training sessions can also be used to tune the model to yield maximum accuracy rate and generalization. As a requirement for the training, a dataset containing the input and correct outputs of the system must be supplied to the model. This data set is called the training dataset. Ideally, the training dataset must be large enough to encompass the variations in the data of each class and, hence, to yield a model with good generalization.

In Figure 3.3 it is depicted a typical classification problem where the data points from different classes, in practice, are said to be linearly separable. All the data points in Figure 3.3 are composed by a feature vector of 2 dimensions (p_1 and p_2) and belong to either class $\mathscr{A}$ (red crosses) or $\mathscr{B}$ (green circles). Now, consider the projection of the data points of both classes onto a line in a way that reduces the superposition of data points from different classes in the projected space. The projection of the data points onto this line will make it easier to classify the points based on their location along this line. In Figure 3.4, two possible line choices are shown (blue line 1 and yellow line 2).

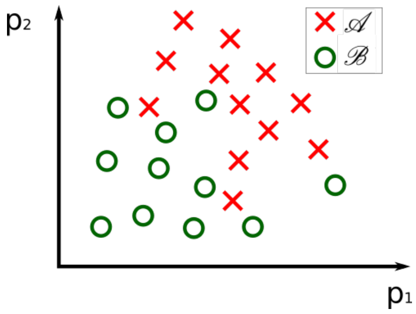


Figure 3.3: Data points distribution as a function of the features p_1 and p_2. The goal of the classifier is to learn a model to identify the data points from the two different classes $\mathscr{A}$ and $\mathscr{B}$.

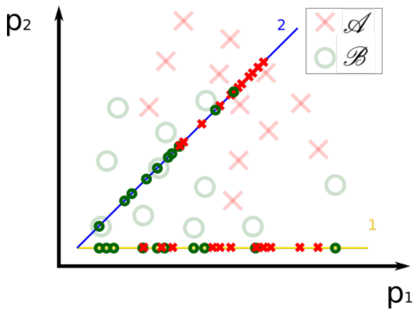


Figure 3.4: Lines 1 and 2 show the possible subspaces to project the data onto aiming to maximize the separability.

To choose the line which yields the best separability, a method to infer the separation of the projected data is required. Defining μ_A and σ_A as the mean and variance of the projected data of class $\mathscr{A}$ and μ_B and σ_B as the average and variance of the projected data of class $\mathscr{B}$, one possible solution is to take into account two metrics: *1 - the distance between μ_1 and μ_2* (how far apart the means from each class are) and *2 - the dispersion of the data points around their class mean* (how large σ_A and σ_B are). These two metrics are shown in Figure 3.5 for the two possible lines cited above. By looking at Figure 3.5 and taking into account the two metrics discussed, it is clear that line 2 yields the best separability in the projected subspace. Line 2 is the best choice because the projected data points of each class have means relatively well separated and, also, there is no significant overlap of the distribution (small variances or dispersion) when compared to the results obtained by line 1. This is the basis of how the LDA works. The LDA seeks the best line or hyperplane (for multi-dimensional feature vectors) to project the data points onto and, therefore, transform the classification problem into a comparison between scalars (one-dimensional classification).

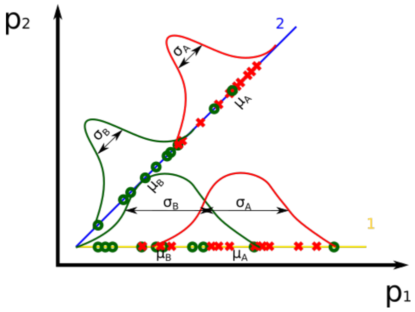


Figure 3.5: The figure shows the Gaussian distribution of projected data points onto lines 1 and 2. Clearly, line 2 yields the best separability since the means are more distant apart, and the distribution range of each class is narrower when compared to the data projected onto line 1. The goal of the LDA is to find the line which yields the best separation results.

The previous paragraphs were important to give the general view and goal of the LDA. Next, the mathematical analysis of the method will be discussed. Since in this work we are using the CSP as a pre-processing stage of the classifier, it is more interesting to link the two approaches during this mathematical analysis. Hence, from now on, the feature vectors which will be supplied to the LDA are output vectors of the CSP, as discussed in the subsection 3.2.1.3.

The LDA algorithm takes as input the feature vector shown in equation 3.11: $\boldsymbol{p}_i = \mathrm{vdiag}(WC_iW^T)$. Hence, for convenience, in this section the elements of the classes $\mathscr{A}$,$\mathscr{B}$ are $\boldsymbol{p}_i^A$ and $\boldsymbol{p}_i^B$. The mean elements of each class are:

$$\mu_A = \text{vdiag}(WC_aW^T) \tag{3.12}$$

and

$$\mu_B = \text{vdiag}(WC_bW^T) \tag{3.13}$$

where C_a and C_b are the mean correlation matrices of each class, as defined in equations (3.6) and 3.7 and W is the CSP projection matrix, defined in equation (3.10).

The LDA algorithm finds the optimum direction ϕ^* to project the data resulting in the best separability in the projected space. Similarly to the CSP algorithm, the LDA seeks a linear projection in the form:

$$L = \phi^* \boldsymbol{p}_i \tag{3.14}$$

where L is called the linear scores and it usually a scalar (the LDA can also be used for dimensional reduction and, hence, L can vary in dimension according to ϕ). After projecting the data onto ϕ^*, the classification can be performed by checking if the data point to be classified is above or below the mean distance between the classes mean [30].

The core information computed by the LDA is the projection vectors ϕ^*. These optimal vectors are the ones who maximize the Fisher criterion, which is defined as [19]:

$$R(\phi) = \frac{\phi^T S_M \phi}{\phi^T S_W \phi} \tag{3.15}$$

Matrices S_M and S_W are the between-class scattering matrix and within-class scattering, respectively, and can be written as:

$$S_M = (\mu_A - \mu)(\mu_A - \mu)^T + (\mu_B - \mu)(\mu_B - \mu)^T \tag{3.16}$$

$$S_W = \sum_{\boldsymbol{p}_i \in \mathscr{A}} \frac{(\boldsymbol{p}_i - \mu_A)(\boldsymbol{p}_i - \mu_A)^T}{n_a} + \sum_{\boldsymbol{p}_i \in \mathscr{B}} \frac{(\boldsymbol{p}_i - \mu_B)(\boldsymbol{p}_i - \mu_B)^T}{n_b} \tag{3.17}$$

where n_a, n_b are the number of elements in $\mathscr{A}$,$\mathscr{B}$ and $\mu = (\mu_A + \mu_B)/2$ represents the global mean of the data set. The vector $\boldsymbol{p}_i$, as discussed in the previous section, is the feature vector.

Notice that maximizing (3.15) implies reducing the dispersion of data points of the same class while increasing the distance between the data points of different classes. The solution of this maximization problem can be found based on the Rayleigh coefficient, resulting in [30]:

$$\phi^* = S_W^{-1}(\mu_A - \mu_B) \tag{3.18}$$

The classification can be performed by projecting the data onto (3.18) and comparing the position in the projected space to the mean between the two classes. Observe that larger values of $R(\phi)$ are obtained when μ_A is far from μ_B and the elements of the classes $\mathscr{A}$,$\mathscr{B}$ are close to their respective mean values (low dispersion).

3.3 Performance Assessment

The performance of a BCI system can be quantified through several methods, depending on its application and the architecture of the system. For instance, one possibility is to take into account the performance or accuracy of the classifier. Although this method provides a way of quantifying how well the EEG patterns are detected by the system, it does not provide any information about how fast these detections are performed. For this reason, in some cases, a complete analysis of the system requires the use of a combination of criteria.

In this section, a few options for performance assessment of BCI systems will be discussed. Although many alternatives are being presented in the literature [19] [50], the focus here will be on the metrics that will be used to infer the performance of system design in this work. For this purpose, three metrics were chosen: the accuracy rate of the classifier, the task execution rate, and the error compared to the optimal execution. The first one will be used to assess how well the EEG patterns are being detected by the classifier. The second parameter gives an estimation of how fast the users can execute a given task when using the system. These two factors will be briefly discussed next.

In machine learning, the accuracy of a classifier is inferred by presenting a new set of data with known labels (true labels) to the trained model. The algorithm then classifies each sample, which in this case are EEG segments or epochs, of this dataset and outputs the predicted classes. The outputs are then compared to the true labels, and the accuracy is computed by taking into account the number of correct predictions performed by the algorithm and the total number of samples, as defined in:

$$accuracy = \frac{N_{hit}}{N}, \tag{3.19}$$

where N_{hit} is the number of samples correctly classified (hits), and N is the total number of samples in the test dataset ($N = n_a + n_b$).

As mentioned, the accuracy rate might sometimes not be able to provide complete information about the performance of the system. In fact, the computed accuracy assess the performance of one submodule of the system, which is the classifier. To quantify the effectiveness of the system as a whole, metrics which are specific for the application of the system must be taken into account. In this work, the task execution rate and the error compared to the optimal execution task were additionally used.

The task execution rate is defined as the required time for the user to complete the given task. A BCI experiment usually consists of a predefined task which has to be performed by the user. The task execution rate measures the time between the start and the end of the task. In general, the lower this rate is, the faster the BCI system can exchange information between the brain and the external application. For some applications, sometimes it makes sense to express this quantity as a rate of bits per second being output by the system. In this case, the rate is called Information Transfer Rate (ITF) [19]. Since the ITF in bits/seconds does not make much sense in the application discussed in this work, the execution time rate will be expressed simply in seconds.

Some BCI systems have as target application an external device which has to perform an action through a specific path, or according to a specific metric. In this case, another useful way of inferring the performance of the system is to compare the execution results achieved by the BCI system to a gold-standard or an optimal reference of the task execution. The performance of a given task realized by the system can then be quantified by calculating the error between this standard and the result achieved during the execution. Low error rates imply that the system is operating near to its maximum capability.

3.4 Model Calibration

In section 3.2, the CSP and LDA techniques for spatially filtering and detecting patterns in EEG data were covered. However, as mentioned, these algorithms require a training dataset to compute the mathematical model which will be used on new EEG data later. So, an important practical consideration when implementing a BCI system is how to obtain this training dataset.

In Figure 3.6, a top-level block diagram of an implementation of a BCI system is depicted. The system can be divided into two different operation modes: model calibration and online operation. The first one is dedicated to the computation of the preprocessing (CSP) and the classification (LDA) models. The online mode then uses these models to process the EEG data in real time and, hence, to interact with the application.

In the calibration mode, the EEG data is collected while instructions are presented to the user, and their timestamps and labels are recorded. The instructions simply guide the user to execute the different tasks in specific moments. They also notify the user when there is a break interval for relaxing and so on. Each pair of timestamp and label is denoted an event. Subsequently, the event timestamps are used to segment the dataset into epochs and to compute the CSP and the LDA mathematical models. Finally, if the algorithms allow it, the models can be tuned to achieve better performance rates.

For motor imagery (MI) based BCI systems, the instructions presented to the user are a set of cues indicating when the user has to perform a motor imagery task (classes) such as left/right-hand movement, feet movement and so on [34]. As detailed in Section 3.2, for the computation of the CSP model, the EEG dataset is segmented into epochs based on the time information stored in the events of the dataset. Thus, the sequence and timing of the guiding information have to be precisely controlled to avoid miss training the model with wrong information.

A typical instruction scheme used in MI-based BCI systems is shown in Figure 3.7. The execution of the sequence presented in Figure 3.7 is called an experiment *trial*. During the first two seconds, a circle is presented to the user indicating the beginning of the trial. This warning instruction helps the user to prepare and focus for the task execution.

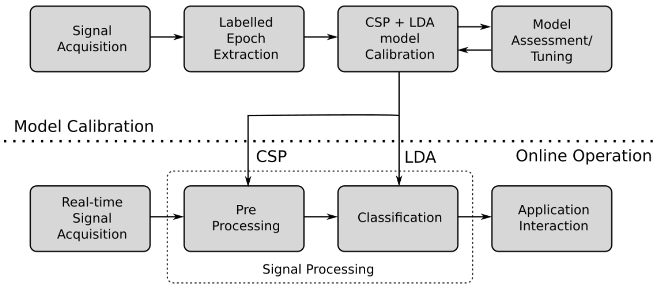


Figure 3.6: Top level overview diagram of an implementation of a complete BCI system. During the model calibration stage, the EEG signal is acquired while instructions are presented to the user, and timestamps of each executed task are recorded. This information is used to compute the preprocessing and classification models for the Online operation mode. When operating online, the system also acquires EEG data in real time, but lively processes it and interacts with the application.

After that, a cue indicating which task is to be executed is displayed (the gray arrows are examples of cues). The user then has around 5 seconds to execute the indicated task while the screen displays a square. A full signal acquisition for BCI model calibration can have many runs (set of trials) and many trials per run depending on the requirements for the computation of the model.

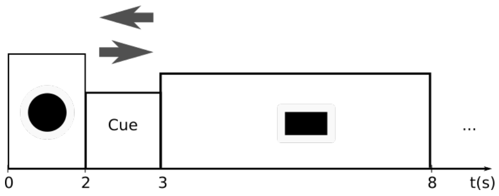


Figure 3.7: A typical stimulus presentation scheme used for training set generation in MI-based BCI systems. The gray arrows show possible cues which indicate left or right hand movement.

Using the scheme presented in Figure 3.7, the user is theoretically engaged in the motor task during the 5 seconds when the square picture is being displayed. However, the μ-waves are considered to be a *quasi*-stationary pattern (the signal state is preserved only for short periods of time) and are strongly dependent on the user's attention and engagement levels. As a result, it is highly unlikely that the μ-wave pattern will be preserved during this whole 5 seconds period. For this reason, for training the CSP and LDA models, an EEG epoch of approximately 2 seconds is usually extracted after each cue event [19].

After computing the CSP filter and LDA classifier parameters, the system is ready to operate in online mode, where the classification is performed continuously as new data is acquired by the amplifier. This operation will be covered next.

3.5 Online Operation

The second branch shown in Figure 3.6 is also known as the real-time operation mode, where the obtained model is applied to the incoming EEG data (stream). The collected data is then continuously processed based on the computed models and commands are dispatched to the application. In Figure 3.8, the block diagram of this mode is shown, detailing the signal acquisition stage shown in Figure 3.6. The amplifier acquires, digitizes and send the EEG data to the system. At each sampling period, which is dependent on the amplifier characteristics, a new array of samples is pushed up to the stream. In online mode, however, a typical structure called *Circular Buffer* stores a segment of EEG data which is used by the subsequent processing stages of the system.

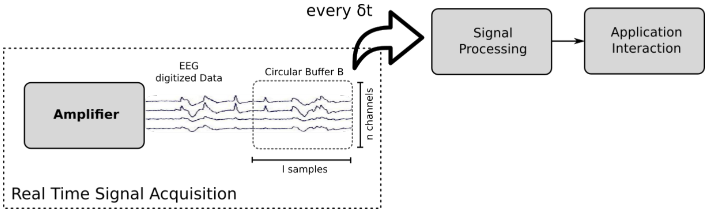


Figure 3.8: An overview of the operation in online mode. The amplifier pushes EEG samples upstream. A circular buffer stores the most recent samples and is accessible by the signal processing modules. At every δt seconds, these modules process the data in the buffer and perform a classification.

This buffer structure can be seen as a window where the EEG signal is passing through. Mathematically, the buffer is defined as a matrix $B \in \mathbb{R}^{n \times l}$, where n represents the number of channels, and l is the buffer size. The buffer is updated with a new sample at every sampling period and, when it is full, old samples are discarded to accommodate new ones (hence the *circular* connotation). Since the classifier is trained with epochs with a given time length, the length l of the buffer is usually set equal to the length of the epochs used for training the models. This way, the preprocessing and classification models operate to identify patterns in the same conditions as they were trained, increasing the accuracy rates.

The signal processing blocks have permanent reading access to this buffer to perform the required mathematical operations in the incoming data. This process of reading and operating on the data of the buffer is made at every δ_t seconds, where δ_t is a multiple of the sampling period and can also be expressed in terms of samples as $\delta_s = \delta_t/f_s$. After processing the signals, an output command is generated and sent to the application.

The δ_s parameter defines the window overlap between each filtering and classification performed by the Signal Processing modules. A $\delta_s = 1$ implies maximum data overlap between consecutive classifications. At every new sample collected by the amplifier, the whole signal processing cycle is performed. However, a large $\delta_s = l$ results in no data overlap between consecutive classifications and the signal processing cycle is being performed with completely new samples every time.

It is important to notice, however, that the processing interval δ_s has to be set taking into account the time required to perform all the mathematical operations. If the interval is too short, the system will not have sufficient time to process the data before being called again, which might lead to an overload of the processing hardware unit. Also, if the interval is too large, EEG samples are skipped and not used by the classifier, which might lead to problems to correctly map the classifications into commands, as it will be discussed in chapter 5. This parameter is usually customizable, and the user has to take into account these factors when choosing the right processing interval δt.

CHAPTER 4

BCI System for drone control

So far, BCI systems were discussed without defining the device to be controlled. The possible outputs of the classifier were set as the classes presented to it during the training stage, but without specifying the action which will be generated based on each class detection. In this chapter, this target application and its details will be discussed.

The goal of this work was to design a BCI system to control the flight directions of a drone. Aiming to provide a better understanding of a drone's dynamics and control, this chapter focuses on the core concepts involved in drone applications. The first section covers the main definitions and terminology related to drones. Section 4.2 describes in detail the particular drone model used in this work. The last two sections are dedicated to the routines used to move and rotate the drone based on the commands generated by the BCI system.

4.1 Drone Dynamics

As discussed previously, one of the main challenges involved in designing a BCI system having a drone as target application is the high degree of freedom inherently present in a drone's movement. For this reason,

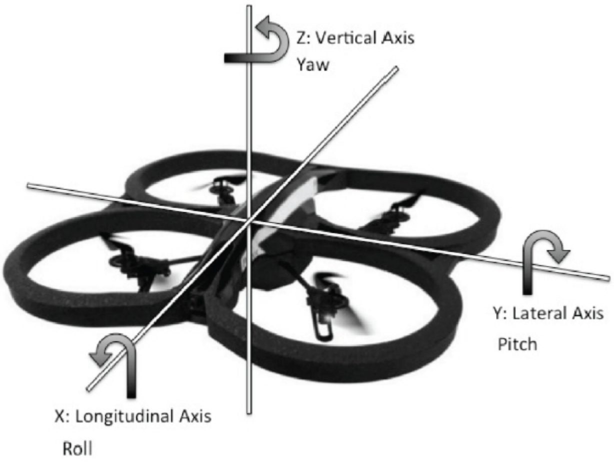


Figure 4.1: Movement or principal axis of a drone. The drone is facing the x-axis direction and presents high levels of movement freedom.

to better understand this challenge and also to provide some background about the movement mechanisms of this type of aircraft, this section will review the basic concepts related to the drone navigation methods.

In Figure 4.1, a diagram of the drone rotational movement is depicted. The rotation around the principal axis of an aircraft (z - vertical, y - lateral and x - longitudinal) are respectively known as yaw, pitch, and roll. These terms are often used to describe the position and orientation in degrees of the drone in the three-dimensional space.

In modern drone models, the pitch and roll velocity are stabilized by robust control techniques which make use of onboard sensors. The commands passed to the drone are responsible for controlling the yaw velocity and, hence, the longitudinal direction (forward direction) of the aircraft. Besides, the drone can also have lateral movement (Y-axis direction) and vertical movement (Z-axis direction).

4.2 AR Drone 2.0

In order to decide which drone model would be used for this work, a few key-points were taken into account: *1 - flight stability*, *2 - available*

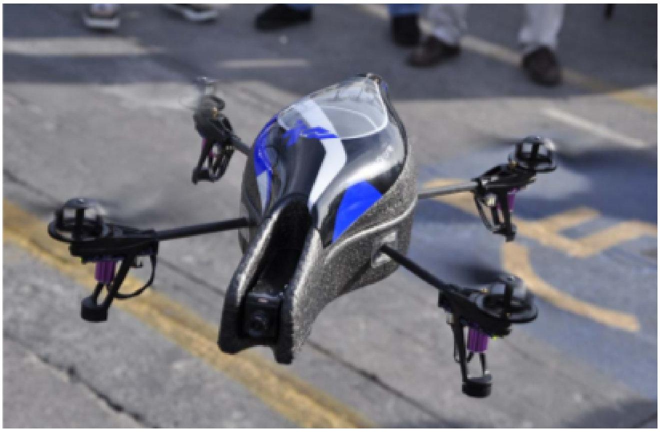


Figure 4.2: Picture of the Parrot AR Drone 2.0

documentation for interfacing with the device, *3 - cost*. The first critical point is related to how steady the drone is during flights. Usually, cheaper and simpler drones tend to have poor onboard sensors and control systems, which lead to significant oscillations in the position of the aircraft even when no command is sent. The second key point is associated with the documentation release by the manufacturer to send commands and receive data from the drone. Without the proper API (application programming interface), it would be impossible to link the designed BCI system to the drone. The last key point affects the overall cost of the system. Since the purpose of this work is to develop a simple, low-cost yet efficient BCI system, a drone with moderate cost had to be chosen. Taking all these points into account, the drone AR.Drone 2.0 from Parrot was selected.

The AR.Drone 2.0 shown in Figure 4.2 is a low-cost quad-copter (approximately \$ 300) which presents high flight stability and control precision [1]. The device is well documented and, in addition, there is a large community working with it to build customized applications such as the one proposed in this work.

The drone flight system is composed by 4 propulsion motors which control the movement speed and direction. To control these motors, the drone is shipped with a stability control system called AutoPilot by the manufacturer.

To compute the right command to send to the motors, the central processing unit relies on information provided by several onboard sensors. This sensory system is composed of three sensors: a 3-axis gyroscope, an altitude sensor, and a camera constantly pointed to the ground. The first collects information about the attitude of the aircraft (orientation and inclination). The second reads the current altitude, and the camera is used to monitor the drone's position, allowing it to stay still in a given area.

Using all this information, the AutoPilot system uses inertial sensors and computational visual techniques to maintain a stable and reliable flight. All the processing is performed by an embedded microprocessor running Linux. The total flight time can reach up to 12 minutes.

4.3 Drone Navigation Strategy

The motion dynamics of a drone, as seen above, is very diverse and can involve a series of different control routines and techniques. The information throughput of a BCI system, though, is significantly limited and, hence, the control of all these movement axes is not feasible or may result in a system too complex to be managed by the user in real time. For this reason, in this work, the goal of the BCI system is to control only the yaw rotation position of the drone. In addition, the longitudinal forward velocity will be automatically controlled based on the commands triggered by the BCI system.

In Figure 4.3, the proposed method for navigation control is illustrated. At the start of the simulation, the drone has a constant longitudinal velocity v_{long}. When a new turn command is triggered by the BCI system, the longitudinal velocity is reduced, and the yaw rotation takes place, altering the direction of the drone (indicated by the red arrow). After the rotation is completed, the velocity v_{long} is reestablished to its initial constant value.

This methodology was chosen based on several other works with BCI which targeted dynamic applications such as the one proposed in this work [20] [26] [12]. This approach of deriving other control signals from an independent and more critical task avoids the need for multiple and simultaneous inputs from the user. By limiting the number of parameters that the user needed to control, the user can focus better

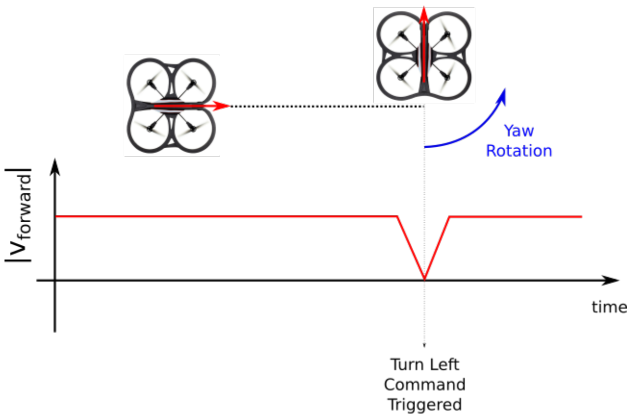


Figure 4.3: Drone Navigation Strategy. The drone is automatically accelerated to a constant longitudinal velocity. When a turn command is triggered by the BCI system, the rotation around the yaw is initiated to alter the direction of the aircraft. After that, the longitudinal velocity is restored to the constant initial value.

on the primaries and mandatory tasks, which lead to an increase in the accuracy rates and better adaptability to the BCI system.

4.4 Drone Control

The previous section provided details about the drone navigation strategy. The control of the longitudinal velocity completely relies on the onboard control system of the drone. However, the control of the yaw rotation position required a more sophisticated approach. The control system has to take into account the current angular position of the drone and the targeted position. Based on this information, the angular rotation speed needs to be set to alter the final pointing direction of the aircraft correctly. To do that, a proportional control system was implemented to lock the direction of the drone to a reference which is determined by the BCI system.

In Figure 4.4, the drone control system is depicted. The commands are sent aiming to rotate the drone around the yaw axis. The aircraft is modeled as a vector (red arrow in Figure 4.4) in a two-dimensional space determined by the pitch and roll axis. The angle θ denotes the

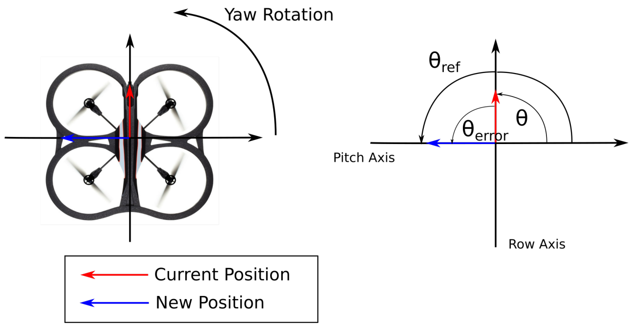


Figure 4.4: Drone Yaw Rotation Control System. The BCI system sets a θ_{ref} reference angular position for the drone. Based on the current position θ, an error signal θ_{error} is computed and used to correct the yaw speed rotation of the drone. As the drone achieves the desired position θ_{ref}, the error θ_{error} becomes smaller and the rotation speed is restored to zero.

position of the vector in this space. At the same time, the system can determine a new position by altering the reference vector (blue arrow). The angle of the new rotation position is defined as θ_{ref}.

If a left-turn command is triggered, the reference θ_{ref} is decreased by 90 degrees. Likewise, if a right-turn command is detected, 90 degrees are added to θ_{ref}. The control system then begins to increase the yaw rotation velocity of the drone towards the new angular reference. The equation which describes this behavior is defined as [17]:

$$v_{yaw} = K_p \times \theta_{ref} \tag{4.1}$$

where K_p is the proportional gain and is defined based on the drone dynamics. The yaw rotation velocity of the drone is updated in very short periods to ensure a stabilization fast enough compared to the system timing requirements.

Although the proportional control system described by equation (4.1) imposes significant limitations on the speed and accuracy of the direction of the drone, these restrictions are not critical for the application. The inherent error presented by the system is significantly low to affect the trajectory of the drone. The limitations and advantages of such approach for the drone piloting system will be further discussed in the chapters about the implementation and achieved results of this work.

4.5 Drone Command Interface and Simulator

So far, the routines and strategies controlling the drone linear and angular speeds were reviewed. However, the practical implementation of the communication between the computer and the drone itself has not been covered. In this section, this interface will be explained. In addition, the drone simulator used in this work is also presented.

To manage the commands and transfer the flight instructions to the drone, the Robot Operating System (ROS) framework [38] was used. The ROS framework was officially launched in 2007 and offers a complete set of open-source tools and libraries for building robot applications. The goal of ROS is to integrate a series of sensors, motors and robot components in general and treat each part of the robot as a node. These nodes can communicate with each other using a simple publisher and subscriber methodology, where the information is not exchanged directly between the nodes, but through topics in ROS. Each topic, which is dedicated to a given parameter of the system, has publishers, which send data to the topic, and subscribers, which consume the published data.

In Figure 4.5 the block diagram of the interface to send commands to the application (drone or simulator) is depicted. Each block of the chart can be seen as a node in ROS. The control commands are published to the topic related to the speed and orientation of the AR Drone. Inside ROS, the mathematical model of the drone is implemented in order to simulate the real flight behavior of the aircraft in the simulator. The ROS Core is responsible for managing the messages flow within the operating system.

Although the utilization of ROS for this project introduces extra complexity to the system, the use of such framework makes it easier to manage how the commands are sent to the drone. This is because all the AR Drone models are already implemented in ROS, and all the routines to receive and send data and commands to the drone are also fully implemented and tested. Also, the ROS framework is shipped with Gazebo, which is a set of tools to simulate robots within ROS. Hence, due to all these advantages, the application interface system was built based on ROS.

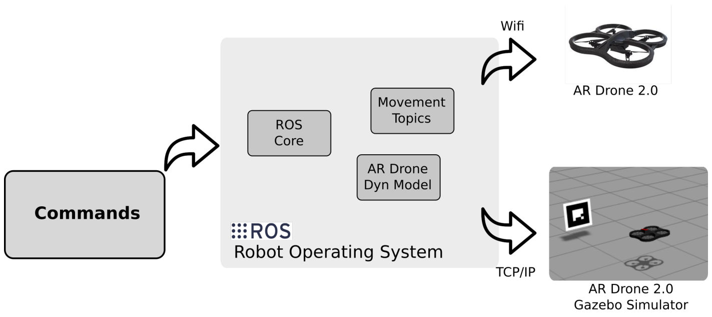


Figure 4.5: Interface diagram for sending commands to the drone and simulator via ROS.

CHAPTER 5

Computer-Machine Interface

In chapter 3, the technical aspects of a BCI system were discussed in details. Nonetheless, the analysis did not go into the details about the application end of the system. To cover this part, chapter 4 provided an overview of drones, including the flight mechanisms and, also, the specific aspects of the drone model used in this work. To fully implement the system, however, the connection structure between the system and the application must also be considered. For this reason, this chapter focus on the link required between these two sub-blocks of the system: the BCI system and the application. Since the architecture discussed here implements a link between a computer and an external device or machine, this system is called Computer Machine Interface (CMI). This chapter includes a discussion about strategies and algorithms used to translate the output of the classifier into an actual pilot command which will be sent to the drone. The theory and implemented structures required for this translation will be covered.

The first section discusses the structures required to link the two blocks mentioned above. The following section is dedicated to explaining the command generation approach used in this work. The algorithm behind the command mapping will be detailed, providing a set of practical examples directly related to the drone control system.

5.1 Online operation Scenario

In chapter 3, the signal flow inside a BCI system was analyzed. This flow included the signal acquisition, preprocessing and classification of EEG data. In the latter, the characteristics of the filtered EEG signal are classified by a machine learning algorithm which outputs a class label. The definition presented in Chapter 3, though, did not include the transcription of the output class label provided by the classifier to a command which can be sent to an external device.

In general, the μ-wave based BCI systems reported in the literature implement routines to directly map the power in the alpha band into commands which are sent to application [20] [26] [12]. This is usually performed by autoregressive methods or by computing the spectral power over the alpha band and linearly transforming the calculated values into commands to the application. Although effective, the robustness to noise signals, which are commonly present in EEG signals, tend to compromise the usability of such systems. Also, there is a major effort in the BCI field to improve the quality of filtering and classification routines such as the CSP and LDA approach. Hence, a system based on these well-established techniques for detecting the user's intent and producing an application instruction can probably achieve the best results regarding usability and precision. In this section, a procedure is presented to compute a command signal based on the classification labels provided by the LDA model. The generated command is a basic signal which changes the flight direction of the drone, enabling the user to control the flight path of the aircraft.

In Figure 5.1, an overview of the BCI system operating in real-time is depicted for the case when only EEG data from a left-hand class is being acquired by the EEG amplifier. This specific scenario will be useful to understand the mechanisms that underline the online classification problem in a BCI system. The first part to consider is how the classifier operates to classify the incoming signal and, hence, to output the class labels [30]. As explained in Section 3.5, in online mode, this operation is performed at every δ_t interval, as new data is acquired and sent for classification. In Figure 5.1, this behavior is

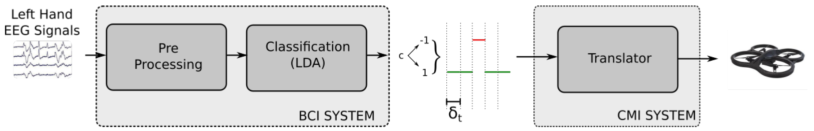


Figure 5.1: A simplified diagram of a BCI system connected to a Computer Machine Interface System (CMI). The CMI translates the classifier outputs into commands which are sent to the application (drone).

illustrated by the continuous output c being generated by the classifier sub-block.

Mathematically, in a time period of t seconds, the number of new classifications can be computed by:

$$n(t) = int(\frac{t}{\delta_t}), \tag{5.1}$$

where the $int(.)$ operator extracts only the integer part of the result, t denotes the continuous time variable and δ_t specifies the classification time interval discussed in Section 3.5. The value of the output c along time can then be written as:

$$c[n] = \begin{cases} -1, & \text{if } lefthand \\ 1, & \text{if } righthand \end{cases} \tag{5.2a}$$

It is important to notice that, in the scenario depicted in Figure 5.1 where only EEG data from left-hand imagery movement is being acquired, the label c is equal to 1 in a given time instant. Ideally, since only data from one class is being supplied to the classifier, it would be expected that only output labels $c = -1$ would be generated at the output. However, since the classifier is not an ideal model, miss classifications can occur. The rate at which the classifier miss classify a given EEG segment, as it will be seen shortly, is directly dependent on the validation accuracy of the model. Taking into account this effect, the labels c generated must then be converted into a command that the application (drone) can understand and execute. This translation mechanism is performed by the Computer Machine Interface (CMI) system shown in Figure 5.1, which implements a translator to map the classifier outputs into flight commands for the drone. The translation method proposed in this work will be presented in the next section.

5.2 Command Generation in Motor Imagery BCI Systems

The proposed approach for control signal generation is based on an integrator: at each time interval δ_t, for each class, an increment variable u_j is computed and added to a sum U_j, where $j \in [1,2]$ for the two-class problem explored here. When U_j reaches a predefined limit in a given time length, a new control command is generated and sent to the application (drone). In this case, the increment variables for each class are defined based on c, as follows:

$$u_1[n] = \begin{cases} 1, & \text{if } c = -1 \\ 0, & \text{if } c = 1 \end{cases} \tag{5.3}$$

$$u_2[n] = \begin{cases} 0, & \text{if } c = -1 \\ 1, & \text{if } c = 1 \end{cases} \tag{5.4}$$

At each δ_t interval, the increment values are computed and added to their respective sum U_j, according to the following equation:

$$U_j[n] = \sum_{k=0}^{k=n} u_j[k] \tag{5.5}$$

In Figure 5.2, the dynamics of the equations discussed above is illustrated based on an ideal case, where the classifier outputs class labels from only one class for a considerable period. The integration interval time δ_t was defined as 50 ms, and the values of U_1 and U_2 are plotted against the continuous time t instead of n to allow a more clear vision of the behavior of the function. During the first 10 seconds, the classifier identifies only patterns from left-hand movement ($c = -1$). As a consequence, during this time, the accumulated value of U_1 increases linearly while the value of U_2 remains constant and equal to 0. In the next 10 seconds, the situation is inverted: the classifier detects only right-hand movements ($c = 1$), leading to a linear increase in U_2 while U_1 remains constant. Using this methodology, it is possible to convert the class labels provided by the classifier into mathematical functions

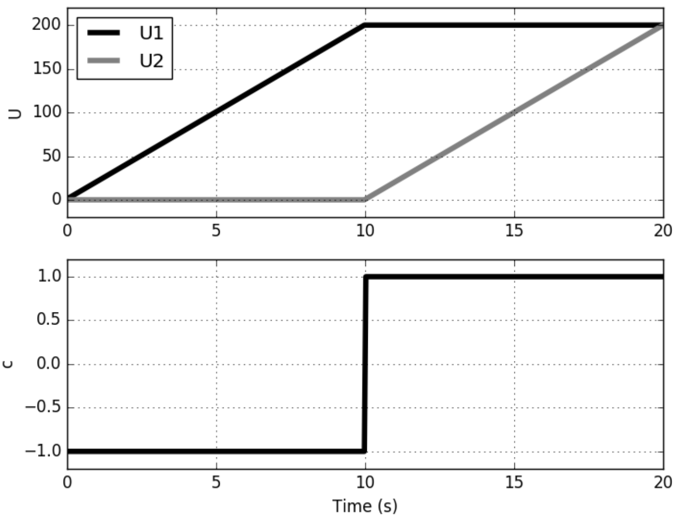


Figure 5.2: The curves show the ideal case behavior of equation 5.5 based on the output of the LDA classifier. The classifier outputs $c = -1$ during the first 10 seconds, leading to an increase in the accumulated value U_1. The opposite occurs during the next 10 seconds: only $c = 1$ is detected by the classifier, forcing an increase in U_2.

dependent on time. The next important step is to devise a way to trigger a command generation based on the dynamics of the curves presented in Figure 5.2.

The most basic mechanism for triggering a command is by monitoring the global value of U_j and regularly checking if a pre-defined limit U_{max} has been reached. Also, in order to discard old samples and emphasize the user's intent in a fairly recent time period, the behavior of U_j is monitored restricting the number of past samples.

In Figure 5.3, this process is illustrated. The curves shown are similar to the ones shown in Figure 5.2, the major difference is that now the classifier alternates the output between the classes, which corresponds to a closer scenario to the real case. The samples of U_j are stored a the circular buffer $A_j \in \mathbb{R}^{1 \times w}$, where w is the buffer size in samples. At each new processing cycle (when n is incremented), a new sample of U_j is stored in buffer A_j.

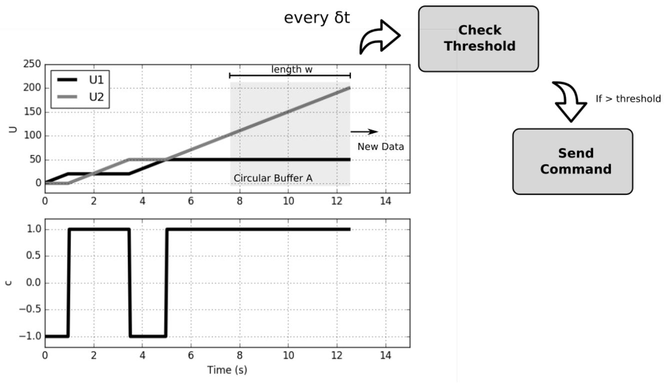


Figure 5.3: General diagram of the method for generating commands by periodically checking the slope of curves U_1 and U_2. The curves show the behavior of equations 5.5 based on the output of the LDA classifier. The circular buffer A stores the most recent samples of U_1 and U_2. The slope of the data inside the buffer is computed at every δ_t interval to check if the threshold has been reached. If that is the case, a command is sent to the application.

To send a command to the drone, the algorithm computes and monitors the variation ΔA_j, which corresponds to the variation of U_j inside the buffer A_j, according to the following equations:

$$\Delta A_j[n] = A_j[w] - A_j[0] = U_j[n] - U_j[n-w] \tag{5.6}$$

where n is the time variable defined in equation (5.1) and w is the buffer length. The expressions $A_j[0]$ and $A_j[w]$ denote the oldest and the newest samples in the buffer, respectively. When this variation reaches the pre-defined limit Δ_{th}, the respective command is generated and dispatched. The threshold check is performed at every δ_t interval, since it is the rate at which the buffers are updated.

In order to understand better how the proposed method, it is important to comprehend the behavior of function (5.5) and how it evolves throughout the online operation. With this objective, the next section will discuss the modeling and prediction of this accumulated value.

5.3 Modeling the accumulated values

In the proposed methodology, the value of U_j is directly related to the number of classifications n and can be seen as the number of times that patterns from a given class were identified by the classifier. Additionally, the increment u_j is always equal to 1 or 0 (see equations 5.3 and 5.4). So, assuming an EEG segment with only data from the left-hand class, as shown in Figure 5.1, and, given that the classifier is ideal ($u_1[n] = 1 \; \forall n$), the value of U_1 then becomes:

$$U_1^*[n] = n \tag{5.7}$$

where the superscript $*$ is used to highlight that the value of U_1 is valid only for the optimum scenario described above. Replacing n in equation (5.7) by its definition expressed in equation (5.1) yields the ideal value of U_1 as a function of time:

$$U_1^*(t) = \frac{t}{\delta_t} \tag{5.8}$$

So, for instance, for the idealistic example shown in Figure 5.2, the value of U_1 after 10 seconds of only data classified as class 1 can be computed using equation (5.8). Since $\delta_t = 50$ ms, the result yields $U_1 = 10/0.05 = 200$, which is in accordance to the value displayed in the curve.

Equation (5.8) computes the number of left-hand class detections in a given period of time t when the classifier is ideal. However, the mathematical models used in BCI systems can rarely operate in such optimal conditions. Since the accuracy of the trained model is limited, for an EEG dataset containing data from only one class, the percentage of classifications which will be correctly identified is limited by the accuracy rate m_{acc} of the model, which is obtained through validation. Under online operation, this implies that for a given number of consecutive classifications, in average, m_{acc} percent of the classifications will be correct. In this case, the classifier will correctly detect a left-hand class pattern ($u_1 = 1$) only m_{acc} percent of the times. As a result, the number of times the increment will be added to U_1 will be limited by m_{acc} and, so, equation (5.8) becomes:

$$U_1[n] = m_{acc}n \tag{5.9}$$

which can be also be expressed as a function of time using the definition of n provided in equation (5.1):

$$U_1(t) = m_{acc}\frac{t}{\delta_t} \tag{5.10}$$

This result can be extended to the other classes by using the subscript j:

$$U_j(t) = m_{acc}\frac{t}{\delta_t}. \tag{5.11}$$

Hence, the validation accuracy of the trained model dictates the slope of the curves U_j. High accuracy values imply that a large number of increments will be summed and the respective curve U_j will increase fast. For low accuracy rates, a null increment will be frequently added to U_j, resulting in a limited slope.

In Figure 5.4, the modeling presented above is demonstrated. Similarly to what was presented in Figure 5.2, the bottom curve shows the output c of the classifier, while the top curves show the behavior of the accumulated values U_1 and U_2 as a function of time together with their estimated values computed using equation 5.11. The main difference from Figure 5.2 is that the classifier here detects the classes with an accuracy of $m_{acc} = 0.9$. So, assuming the scenario where only data from left-hand class is provided to the system, the classification algorithm will correctly identify the label only 90% of the times, which is demonstrated by the spikes with $c = 1$ shown in the bottom curve. This, purportedly, alters the inclination of curves U_1 and U_2 as described by equation 5.11.

In the next section, the configuration parameters of the method will be discussed.

5.4 Parameters definition

A crucial aspect in BCI is the usability of the system and how well the user can adapt to its control routines and mechanisms. In this sense, the timing characteristics, such as the time for sending a command, are important factors when designing a BCI system. In the proposed

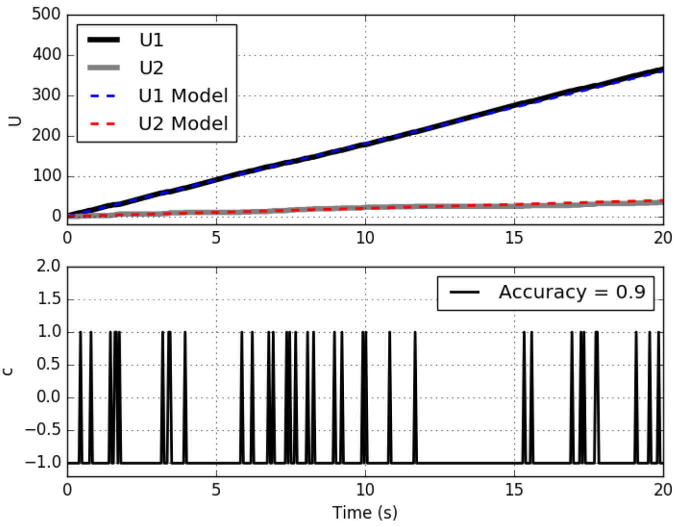


Figure 5.4: Modeling of the sum variables U_1 and U_2 according to equation 5.11. The bottom curve shows the output c of the classifier with accuracy of 90% for input data only from class 1. The spikes represents the 10% miss-classifications. The top Figure shows the behavior of U_1 and U_2 as a function of the input c and time. The curves computed based on equation 5.11 demonstrate a good fit to the actually computed curves.

method, the command generation timing is directly related to the value of ΔA_j (equation (5.6)) and also by the definition of the threshold Δ_{th}, which has to be set according to the user's capabilities. The following section will discuss how to define an adequate threshold value for a given user.

The variation ΔA as a function of time can be computed substituting equations (5.9) on equation (5.6), and replacing n by equation (5.1), which yields:

$$\Delta A_j = m_{acc}(\frac{t}{\delta_t}) \tag{5.12}$$

In order to define a variation limit to trigger a command, it is necessary to specify a time length in which the user is expected to reach this limit. In this method, this time length is called time to action (t_{ta})

and is set according to the application and its dynamics. For instance, for the drone application used in this work, the allowed time for the system to generate a drone control command is dependent on the linear and angular speed of the drone. Monitoring the U_j variation within this time frame implies in setting the buffer A_j length in seconds equal to t_{ta}. Applying this substitution to equation (5.12) results in:

$$\Delta A_j = \frac{m_{acc}\, t_{ta}}{\delta_t} \tag{5.13}$$

and the buffer length in samples can be computed using:

$$w = \frac{t_{ta}}{\delta_t} \tag{5.14}$$

The threshold to trigger a command is then simply the theoretical variation achieved by the model of U_j after t_{ta} seconds, or:

$$\Delta_{th} = \frac{m_{acc}\, t_{ta}}{\delta_t} \tag{5.15}$$

In Figure 5.5, it is possible to visualize the process of defining the threshold for a given application. Based on the accuracy of the model (m_{acc}), the threshold can be easily computed as the expected variation during the t_{ta} period of time.

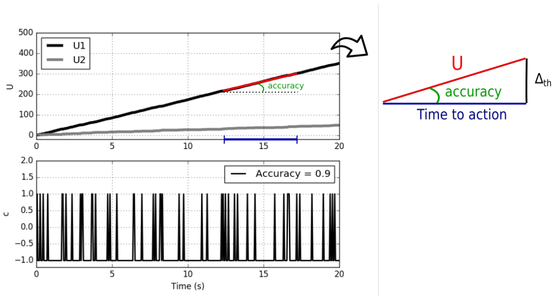


Figure 5.5: Threshold definition illustration: based on the model accuracy, the threshold level is defined as the expected variation in a given time to action period of time.

So, for instance, assuming a $t_{ta} = 10$ seconds and a $\delta_t = 50$ ms, the length of the buffer is $w = 10/0.05 = 200$. This implies that the circular buffer A will accumulate 200 past samples of U_j, which will be used to compute the value of ΔA_j defined in equation (5.6).

The examination of curves in Figure 5.2 and in Figure 5.6 together with the triggering mechanism explained above demonstrates valuable insights about the methodology proposed. To trigger a command response, the classifier must output consecutively class labels from just one class. This mechanism, at the EEG-side and the user-side of the system, implies that the user must be generating EEG data from just one class for a reasonable amount of time. In other words, to change the direction of the drone, the user must be engaged in a motor task activity (left or right-hand movements) for a given period. The length of this period is determined by the definition of the threshold Δ_{th} and, also, by the length of the circular buffer A_j. These parameters can be used to tune the system and, hence, to adapt better to the user's ability to control the synchronization of μ waves.

CHAPTER 6

Developed Platform

To implement the routines and interfaces described in the previous chapters, a software platform was developed. In short terms, this platform is responsible for communicating with the signal acquisition equipment, processing the data according to Chapter 3 and, finally, communicating with the target application as described in Chapter 5. Also, all the techniques were implemented allowing real-time operation. Although there a few good options available for complete BCI platforms such as BCILAB [23] and OpenVIBE [39], due to specific requirements of the application proposed in this work, it was considered adequate to develop a custom software. The mechanisms used to build such platform is discussed in this chapter. Although the discussion in general does not go into specific details of the implementation, the description of each module of the platform gives a reasonable idea of how the BCI system was implemented.

The first section provides an overview of the software architecture, providing block and data flow diagrams of the system. Subsequently, the real-time or online implementation techniques are covered. Finally, the last section gives an overview of how the software communicates with the application, including the case when this application is a drone simulator.

6.1 Software Architecture and Implementation

The software was developed using *python* [16] and was based on the *kivy* framework [10] for building the graphical interface. The produced code is completely open-source and is available on *GitHub* [1]. For implementing the data handling and signal processing modules, several external *python* packages were used and are listed on the *GitHub* page. In Figure 6.1, the block diagram of the implementation of the platform is depicted. According to section 3.5, in general, mu-wave based BCI systems have to operate in two distinct modes: calibration and online. This subdivision in the software is shown in Figure 6.1. The Data Handler (DH) is responsible for communicating with the EEG amplifier and for managing the acquired samples. When calibrating the model, the DH gets the samples from the amplifier and stores it on disk. In addition, the Display Controller (DC) organizes the instruction visual presentation to the user. The DH and the DC then send information about the events to the Event Marker, which stores on disk the event information. The stored dataset and the events are then used by the Model Trainer to compute the CSP and LDA models as described in Section 3.2. In this mode, there is no real-time signal processing being applied to the signals; the collected data is recorded in the raw form. As expected, in Online Operation mode, the architecture is significantly different to accommodate the real-time signal processing stages and also the command generation and to dispatch to the drone.

Table 6.1: Function of each module in the platform.

Module	Function
Data Handler (DH)	Calibration:Store the acquired data on disk Online: Arrange the acquired data in the circular buffer
Display Controller (DC)	Control the display of instructions as described in section 3.5
Event Marker (EM)	Get the event information from the DH and the DC and stores it on disk
Model Trainer (MT)	Use the EEG dataset and the events information to compute the CSP and LDA models
Signal Processor (SP)	Process the EEG data in the circular buffer continuously using the computed CSP and LDA models
Command Dispatcher (CD)	Receive the classification outputs from the SP and generate a command do the drone based on the technique explained in Section 5.2

In online operation, the DH continually deals with new samples and

[1] https://github.com/rafaelmendes/bcitp

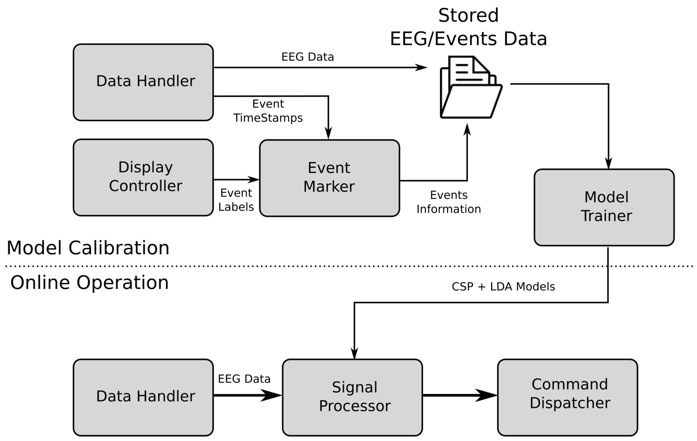


Figure 6.1: Overview of the architecture of the platform. The software can operate in Calibration and Online modes, as described in section 3.5. In calibration mode, the EEG data and event information are stored on disk and used to compute the LDA and CSP models. Later, on online operation, the models are used to continuously classify real-time acquired EEG Data and to send a command to the application.

arranges them in a circular buffer which is accessible by the Signal Processor (SP) module. The SP preprocesses and classifies the data in the buffer using the filtering techniques described in Section 3.2. The CSP and LDA models used are provided by the computation performed in the calibration mode. Subsequently, the classification results are delivered to the Command Dispatcher to be mapped into Drone commands and dispatched to the drone's piloting unit. In Table 6.1, the functions of each module are summarized. The role of the DH module is described for both Calibration and Online modes. A more detailed description of each implementation will be discussed in the following sections.

6.2 Data Handler

The Data Handler is responsible for managing the samples sent by the EEG amplifier and storing it on the local disk. The DH module also organizes the distribution of EEG data across the software, sending the EEG data to the signal processing modules and providing timing information to the Event Marker. Since the DH structure is present in both operation modes, as shown in Figure 6.1, it will be covered before the other modules which were specifically implemented for each mode.

The internal structure of the Data Handler is depicted in Figure 6.2. The platform was designed to work acquiring data directly from the amplifier or in a simulated acquisition mode, where the data is read from a stored dataset. For the former, the EEG amplifier (OpenBCI) acquires the EEG data and send it to the Data Handler to be stored or processed. The data is saved in the disk as a matrix of n channels per q samples. In the simulation mode, the data is extracted from a dataset stored in the local disk and sent to the platform at a speed rate based on the original sampling rate of the data. This way, it is possible to simulate the operation of the software in real-time using a playback strategy based on saved datasets.

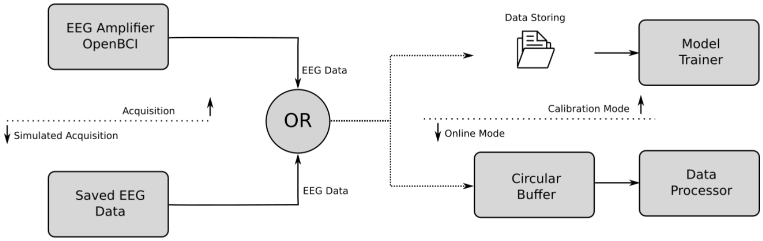


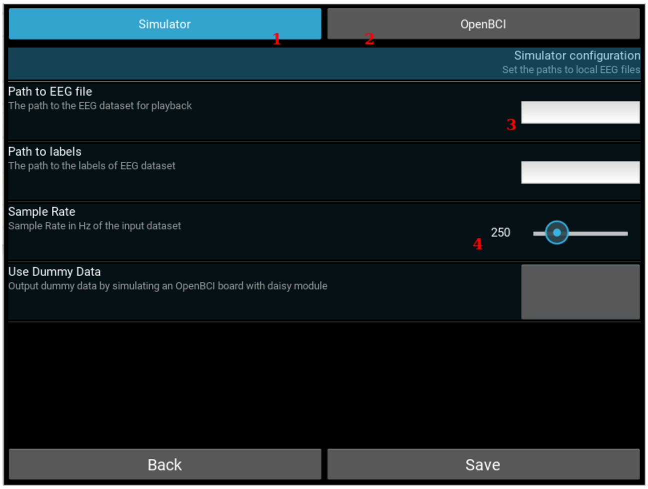
Figure 6.2: The Data Handler can operate by communicating directly with the EEG amplifier or by accessing a saved dataset and simulating an EEG amplifier. In both cases, the DH also delivers the samples to all the other signal processing modules in the platform.

In Figure 6.3 a screen shot of the acquisition configuration screen of the platform is depicted. The user can select between the two acquisition modes (playback or OpenBCI) and configure a few basic parameters such as the path to the local EEG file and the frequency sampling rate of the data.

The following subsections will briefly discuss the EEG amplifier chosen and how the simulated acquisition scenario works in practice.

6.2.1 OpenBCI EEG Amplifier

The first stage of any BCI system is dedicated to the signal acquisition, which is performed by the EEG amplifier in EEG-based systems. In this work, the amplifier OpenBCI was chosen for this task. The OpenBCI



Simulator configuration
Set the paths to local EEG files

Path to EEG file
The path to the EEG dataset for playback

Path to labels
The path to the labels of EEG dataset

Sample Rate
Sample Rate in Hz of the input dataset
250

Use Dummy Data
Output dummy data by simulating an OpenBCI board with daisy module

Figure 6.3: Acquisition configuration screen of the developed platform. The user selects the acquisition mode: (1) - playback simulation using stored dataset, or (2) - Using the OpenBCI amplifier. The screen shows the configuration fields for the playback option. Field (3) is used to set the path to the EEG data file. The sampling rate of the dataset is adjusted using the field (4). If the OpenBCI mode is enabled, a screen with basic configuration items is shown.

project started in 2013 with the purpose of making BCI technologies affordable and more accessible to small research groups and hobbyists. Also, the company provides a broad and complete documentation for the development of applications using the amplifier, including the fostering of a very active community for related information exchange. For these reasons, the OpenBCI was considered the most suitable choice for this project.

The OpenBCI board is depicted in Figure 6.4. The board is composed by an analog front-end (ADS1299 from Texas Instruments) to sample the biologic data. The sampling rate is 250 *samples/sec*, when collecting date from 8 EEG channels and 125 *samples/sec* when all the 16 channels are enabled. The data is sent to a computer via a *bluetooth* serial bridge (using the USB dongle shown on the right-hand side of Figure 6.4). The electrodes are connected to the pin headers shown in the top-most part of Figure 6.4. A simple microprocessor is also included in the board for developing low-level hardware applications. The board is powered by batteries to reduce 60 *Hz* noise contamination.

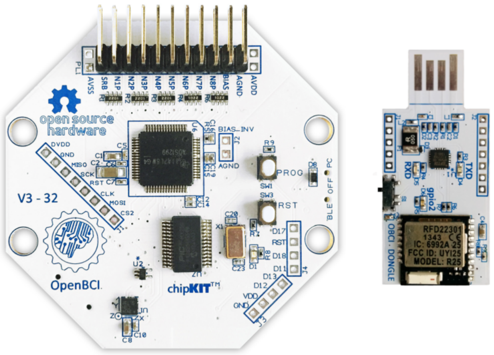


Figure 6.4: Top view of the OpenBCI board. The EEG electrodes are connected to the pin headers and the signal is transmitted via *bluetooth* to a computer using the USB dongle shown on the right-hand side of the picture.

The USB dongle shown in Figure 6.4 allows the communication with the main board through a serial port. To get the EEG data from the board, the manufacturer provides a *python* library with high-level functions. At each sampling interval, the dongle outputs an array with the EEG samples of each enabled channel. This array is then fed to the next stage, which stores the acquired data and process it.

6.2.2 Playback EEG Data

As described in previously, the platform can also be used to playback an EEG file obtained in a previous experiment. This feature allows the testing of the platform and its algorithms without the need of performing a real EEG acquisition experiment, which is usually an arduous and longstanding process. In this section, the mechanisms used to implement this feature will be briefly discussed.

The goal of this module is to simulate an EEG amplifier feeding data to the platform. For this reason, the sampling rate of the saved dataset has to be taken into account when sending the data to the platform. In Figure 6.5, the core diagram of the solution is shown. After every sampling rate interval, which is pre-configure on the platform, the Data

Handler pops an array of data samples from the EEG file and send it to a subsequent stage (MT or SP). This process is repeated until all the samples saved in the archive are popped, ending the simulation.

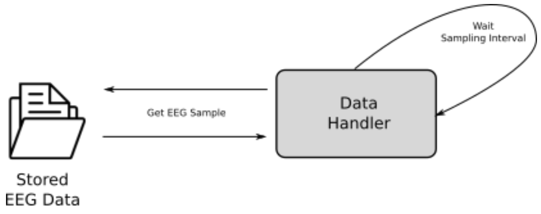


Figure 6.5: EEG dataset playback mechanism.

It is important to notice that the simple process described above works very similarly to the operation of the OpenBCI in the online mode. This compatibility enables the use of both the OpenBCI and a generic stored EEG data without the need of performing adaptations in the rest of the software. Next, the distinct aspects of the two operation modes of the platform will be covered.

6.3 Model Calibration

As discussed in chapter 3, the CSP and the LDA models need to be trained based on a series of pattern examples which are labeled according to the events of tasks realized during the acquisition. From this training, the mathematical models are extracted and evaluated to then be used for detecting mu-waves patterns in new EEG data. Therefore, for the final objective of controlling the AR Drone through neural signals, the platform must include a complete routine for the acquisition of these training datasets. This includes interfaces for collecting the EEG data while presenting different types of instructions to the user and recording the time and label at which this guidance information were presented. This section explains how these features were incorporated into the platform.

In Figure 6.6, the diagram illustrates the workflow of the signal acquisition part for model calibration in the software. The implementation can be divided into two independent threads, which are run in parallel on the platform. In the data storing thread, the Data Handler gets the EEG samples from the EEG amplifier and stores it in the disk. At the same time, in the event processing thread, the Display Controller defines which task will be executed by the user and controls the graphical interface to present the corresponding instruction (as described in section 3.4). The event information (timestamp and type) is then stored also in the disk by the Event Marker.

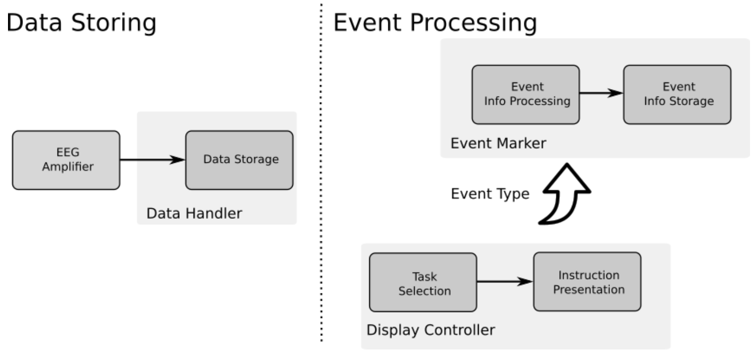
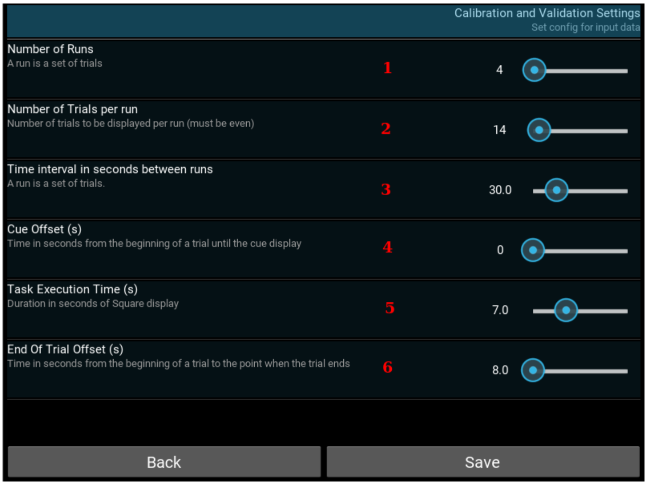


Figure 6.6: Block diagram of the training dataset acquisition for model calibration. The Data Storing and Event Processing threads are run simultaneously. In the former, the Data Handler communicates with the amplifier and stores the collected EEG samples. The Event Processing thread is responsible for displaying the instructions to the user and, at the same, store the event information based on timing and label information provided by the other modules.

The graphical interface implemented in the platform for configuring this acquisition stage is depicted in Figure 6.7. The interface allows the configuration of parameters such as the number of runs and the number of trials per run in the experiment. This will define the number of epochs which will be used to train the model in the subsequent stage by the Model Trainer. Furthermore, parameters related to the timing characteristics of the instruction ordering and display are also configurable in the screen shown 6.7. The timing configuration is set based on the diagram shown in Figure 6.8.

Similarly to what was discussed in Section 3.4 and Figure 3.7, the scheme shown in Figure 6.8 describes the characteristics of the instruction presentation during the training dataset acquisition. In the developed platform, the parameter *Cue Offset* denotes the time in seconds at which the screen will display the circle, indicating the beginning of a new trial. Next, a cue (left or right) is presented for a fixed time of 1


Calibration and Validation Settings
Set config for input data

Number of Runs A run is a set of trials	1	4	
Number of Trials per run Number of trials to be displayed per run (must be even)	2	14	
Time interval in seconds between runs A run is a set of trials.	3	30.0	
Cue Offset (s) Time in seconds from the beginning of a trial until the cue display	4	0	
Task Execution Time (s) Duration in seconds of Square display	5	7.0	
End Of Trial Offset (s) Time in seconds from the beginning of a trial to the point when the trial ends	6	8.0	

Figure 6.7: Calibration configuration screen of the developed platform. The first three fields (1,2 and 3) are used to set the number of runs, number of trials per runs and the pause interval between consecutive runs. The remaining fields (4,5 and 6) define the instruction presentation timing characteristics, as illustrated in Figure 6.8

second, indicating which task the user should execute. The user then can perform the task while a square is displayed in the screen during *Task Execution Time* seconds. Finally, the user has a few seconds to rest and prepare for the next trial (*End of Trial Offset*).

The ordering of the instructions and coordination of the timing properties discussed above is performed by the Display Controller, which will be debated next.

6.3.1 Display Controller

The main task of the Display Controller is to select which instruction will be displayed to the user. For a left and right-hand imagery movement experiment, the DC has to generate a sequence of trials which will collect EEG data from these two different tasks. In the developed platform, the user sets the number of runs n_{runs} and the number of trials per run $n_{trials\_per\_run}$ (parameters 1 and 2 shown in Figure 6.7),

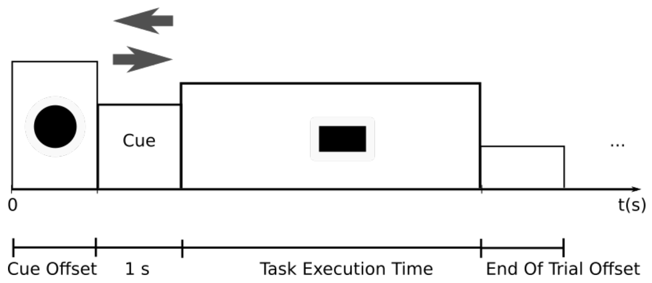


Figure 6.8: Illustration of timing characteristics for the instruction presentation. The diagram shows the meaning of parameters 4, 5 and 6 displayed in Figure 6.7. The configuration of these parameters determine the timing properties of the instruction presentation during the training dataset acquisition.

which in turn defines the total number of trials in the experiment as:

$$n_{trials} = n_{runs} \times n_{trials\_per\_run} \tag{6.1}$$

The DC controller then creates a random sequence of cues with an equal number of trials from each class:

$$n_{trials}[LEFT\ HAND] = n_{trials}[RIGHT\ HAND] = \frac{n_{trials}}{2} \tag{6.2}$$

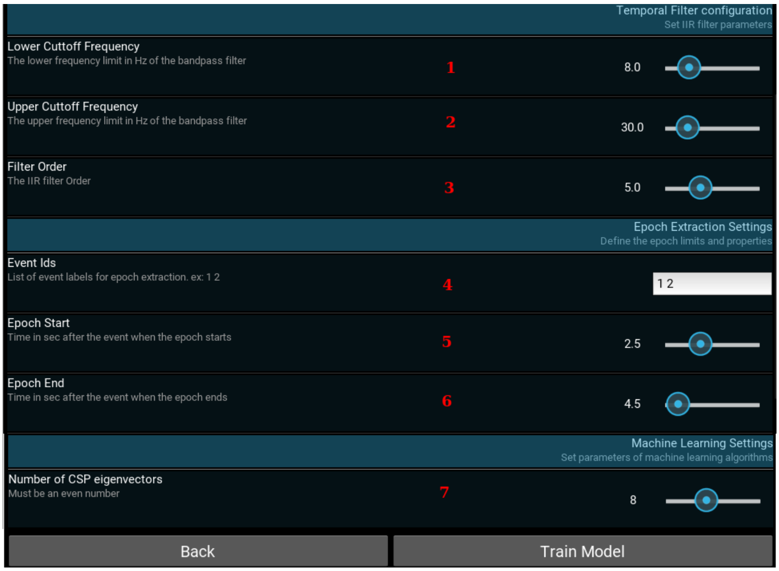
As the instructions are shown to the user, the trial label information is sent to the Event Marker.

6.3.2 Event Marker

The Event Marker received the timing and label information for each event and stores it in memory. Each event is stored as a tuple ($[time, label]$) and can be imported to the platform by the Model Trainer. The time for each event marks the exact moment when the cue instruction was displayed to the used. For instance, in Figure 6.8, the event timestamps is marked right after the end of the *Cue Offset* period.

6.3.3 Model Trainer

After the acquisition of the training dataset, the CSP and LDA models can be trained on the platform by configuring a few parameters such as channels, filter bandwidth and order, the number of CSP *eigenvectors* and time interval around events for epoch construction. In Figure 6.9, the screen to configure these parameters is illustrated. The mathematical procedure used to train the model is the one described in Section 3.4 and shown in the top part of Figure 3.6. First, the epoch is extracted based on the event information stored by the Event Marker. Then, the CSP and LDA models are trained based on the mathematical techniques described in Section 3.2.



Temporal Filter configuration
Set IIR filter parameters

Lower Cuttoff Frequency
The lower frequency limit in Hz of the bandpass filter
1
8.0

Upper Cuttoff Frequency
The upper frequency limit in Hz of the bandpass filter
2
30.0

Filter Order
The IIR filter Order
3
5.0

Epoch Extraction Settings
Define the epoch limits and properties

Event Ids
List of event labels for epoch extraction. ex: 1 2
4
1 2

Epoch Start
Time in sec after the event when the epoch starts
5
2.5

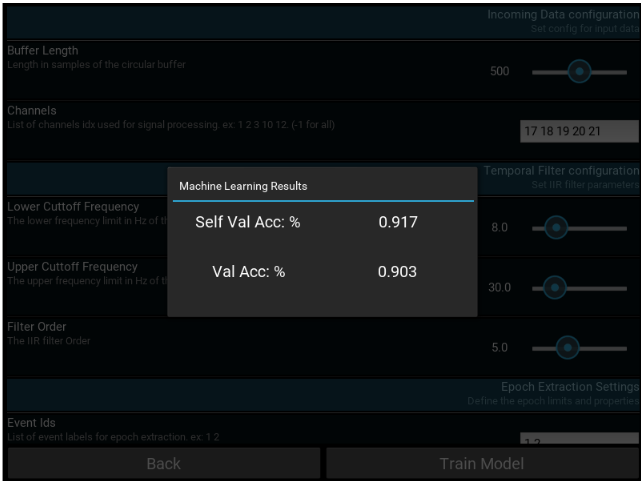
Epoch End
Time in sec after the event when the epoch ends
6
4.5

Machine Learning Settings
Set parameters of machine learning algorithms

Number of CSP eigenvectors
Must be an even number
7
8

Figure 6.9: Model Trainer configuration screen. Parameters 1,2 and 3 define the bandpass filter characteristics. Fields 4, 5 and 6 are used to select the epoch extraction properties. The field 4 define the labels which will be used to extract the correspondent epoch. The time settings 5 and 6 define the time range in seconds around each event. This will determine the time length of the extracted epochs. Finally, parameter 7 determines the number of CSP eigenvectors used in the spatial filtering stage (see Section 3.2.1.3)

After computing the mathematical models, the classification performance is assessed through self-validation and validation, as described in section 3.3. The results are displayed on the platform as depicted in Figure 6.10, after pressing the "Train Model" button. The model can be tuned by altering the parameters in Figure 6.9 and interactively checking the resultant accuracies. The final model is stored in the disk to be used in the following online processing mode.



Incoming Data configuration
Set config for input data

Buffer Length
Length in samples of the circular buffer
500

Channels
List of channels idx used for signal processing. ex: 1 2 3 10 12. (-1 for all)
17 18 19 20 21

Temporal Filter configuration
Set IIR filter parameters

Lower Cuttoff Frequency
The lower frequency limit in Hz of th
8.0

Upper Cuttoff Frequency
The upper frequency limit in Hz of th
30.0

Filter Order
The IIR filter Order
5.0

Epoch Extraction Settings
Define the epoch limits and properties

Event Ids
List of event labels for epoch extraction. ex: 1 2
1 2

Machine Learning Results

Self Val Acc: %	0.917
Val Acc: %	0.903

Figure 6.10: Model Trainer performance assessment screen. The dialog box shows the self validation and validation results. These metrics can be used to tune the models to achieve best classification accuracy rates.

6.4 Online Operation

After the model is tuned and trained, the online operation mode can be used to process the EEG data and control the drone in real-time. In Figure 6.11, the block diagram, and workflow of this mode are shown. The system is divided into three major blocks: the Data Handler, the Data Processor, and the Command Dispatcher. The Data Handler was already covered previously in section 6.2 and works by forwarding samples to other components of the software. This is accomplished by appending every new sample in a circular buffer. The other blocks implemented for the online operation of the platform will be covered next.

6.4.1 Data Processor

The circular buffer has to be processed periodically, aiming to identify known patterns in the EEG data and, as a consequence, to detect the

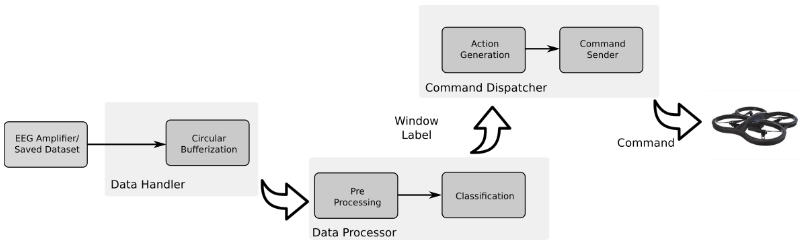


Figure 6.11: Diagram of the workflow of the platform operating in online mode: the Data Handler communicates with the EEG acquisition system (or the stored EEG file) and manages the data inside the circular buffer. The buffer is filtered and classified by the Data Processor to detect the patterns present in the EEG segment. The classifier's output is then sent to the Command Dispatcher to be mapped into a command and sent to the drone.

user's intent to trigger an action. The module responsible for this task in the platform is the Data Processor. In Figure 6.12, the method used by this module is shown. As described earlier, the buffer, which can be seen as a window with dimensions $n \times l$, is regularly updated with new EEG samples by the Data Handler. The Data Processor also periodically applies the filtering and classification algorithms to the buffer, generating a class label for the EEG data present in the buffer. This label is then used by the Command Dispatcher at the next stage to map the identified classes into commands and dispatch it to the drone, following the approach discussed in chapter 5.

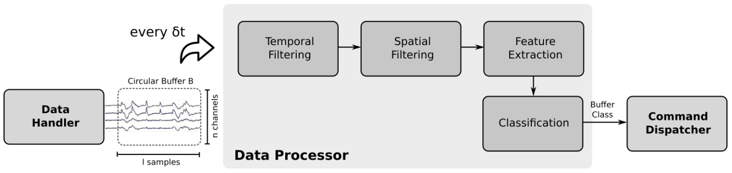


Figure 6.12: Block diagram of the Data Processor module.

The processing interval, shown in Figure 6.12 as δ_t, has to be set taking into account the time required to perform all the mathematical operations depicted in the Data Processor module. If the interval is too short, the processor will not have sufficient time to process the data before being called again, which might lead to the overload of the processing unit. Also, if the interval is too large, EEG samples are skipped and not used for class detections, which might lead to problems to correctly map the classifications into commands, as discussed in chapter 5. In the platform, this parameter is customizable, and the user has to take into account these factors when choosing the right processing interval δt.

6.4.2 Command Dispatcher

The last stage of online processing, called the Command Dispatcher, is responsible for getting the buffer class labels and map them to commands to be sent to the drone. These two steps are shown in Figure 6.11, in the highlight area entitled Command Dispatcher. The Action Generator sub-block gets the labels from the classifier and maps them into commands as described in Section 5.2. In addition, the Command Sender block interfaces with the drone's API in ROS (see Section 4.5) to send commands and also receive sensor data such as position, altitude, and velocity.

In the previous sections, the internal procedures run by the platform for acquiring, processing and dispatching the EEG signals and drone commands were discussed. However, no insights about the interface with the user were provided. To trigger a command via the methods explained, the user has to be continuously informed about the parameters discussed in Section 5.2 such as the Δ_A slope in the circular buffer. However, this information needs to be presented in a friendly and unpolluted way to avoid disturbing the user's attention to the main goal of the experiment, which is controlling the drone. In this subsection, the mechanism for visually communicating with the user will be explained. The text will be focused on the interface used by the user to trigger the left or right-hand command and, therefore, to change the drone's direction.

In section 5.2, the mechanisms for generating a command based on the user inputs were described in detail. In summary, for triggering a command, the user has to focus on a specific class for a pre-defined period. The classifier will then identify consecutive patterns from that given class and, when the variation in the cumulative sum U reaches a pre-defined threshold within a period, a command is sent to the application. The developed user interface has to incorporate all these details and present them to the user in a simple way. In Figure 6.13, a screen-shot of this user interface implemented for this purpose is shown.

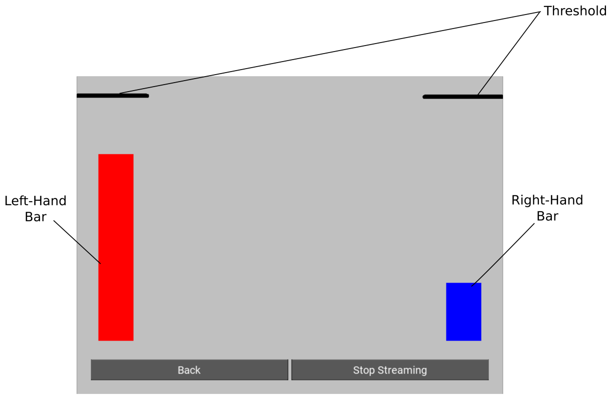


Figure 6.13: Interface diagram for sending commands to the drone and simulator via ROS.

The fundamental goal when using the interface is to control the level of the red and blue bars to reach the threshold levels. Each bar corresponds to a different class: left or right-hand imagery movement. The level of the bars represents a percentage of the targeted threshold variation, defined in 5.2. When the bar is full, $\delta_A = \delta_{th}$, the limit has been reached, and the respective command can be sent to the drone. For instance, assuming that the user wants to turn the direction of the drone left, he has to execute left-hand imagery movements. As a result, the red bar then will start to rise until the point the threshold is reached, and the "turn left" command is dispatched. This underlying mechanism allows the implementation of the technique explained in Section 5.2 through a simple graphical interface.

CHAPTER 7

Results

In this chapter, the results obtained in this work will be presented. The first section starts with two sub-tests performed on the developed platform. These tests will assess the pipeline for filtering and classifying the acquired EEG data, as well as the performance of the control system designed to change the direction of the drone. As the software built will be used to achieve the remaining results presented throughout this chapter, the validation of its submodules consists in an important step.

After testing the essential features of the platform, the complete flow of the BCI system is assessed using a public EEG data. The goal is to infer the performance of the system when operating with a well-known and well-tested dataset for BCI applications. Also, the chosen dataset allows for testing the system when working with users with reduced yet reasonable accuracy rates.

The last section shows the results obtained with the EEG data acquired using the OpenBCI hardware and the developed platform. By using this data, it is possible to test the behavior and performance of the system when operating with less reliable data and, also, to validate further the complete system designed in this work.

When testing the complete BCI system, the metrics presented in

Table 7.1: BBCI IV Dataset trial timing characteristics based on Figure 6.8.

Parameter	Value
Cue Offset (s)	2
Task Execution Time (s)	3
End of Trial Offset (s)	1.5

Section 3.3 will be used to quantify the performance of the system.

7.1 General Platform Tests

With the aim of testing the built software, two key points are assessed in this section: the μ-wave pattern classification dataflow and the drone control routines. By briefly analyzing the performance of the platform regarding these two important submodules, it is possible to assess the functionality of the BCI system in a modular way. Each of these mentioned tests will be discussed separately next.

7.1.1 Motor Imagery Classification

The first step was to test the pre-processing and classification routines implemented on the platform. For this purpose, a public data, available online for download [1], was used. The data set IIa from the BBCI competition IV includes EEG signals from 9 subjects who performed four different mental tasks: left (LH) and right (RH) hand, feet (FE) and tongue (TO) motor imagery. The signals were recorded using a professional EEG amplifier with 22 channels at a sampling rate of 250 Hz. Two datasets were available for training and testing, both containing each 72 trials for each one of the cited tasks. Three electrodes for measuring electrooculography (EOG) activity are also included in the dataset but were not considered in this work. The trials were acquired with the instructions timing parameters shown in Table 7.1, which are based on the scheme shown in Figure 6.8.

The processing flow applied to the data was described in section 3.2 and chapter 6. First, the data was filtered using an IIR bandpass Butterworth filter. Next, the training dataset was segmented into epochs

[1] www.bbci.de/competition/iv/

Table 7.2: Parameters set for processing and classification.

Parameter	Value
Bandpass Filter Frequency Range	8 to 30 Hz
Bandpass Filter Order	5^{th}
Epoch Time range	0.5 to 2.5 sec
CSP *eigenvectors*	6

Table 7.3: Evaluation results of the proposed and classical methods.

Subj.	1	2	3	4	5	6	7	8	Mean
[32]	88.89	51.39	96.53	70.14	54.86	69.4	81.25	92.4	75.60
This Work	90.3	50.0	97.2	70.8	50.7	69.4	79.4	92.4	75.02
Error	-1.41	-1.39	0.67	-0.66	4.16	0.00	1.85	0.00	0.58

of left and right-hand movement, discarding the epochs labeled as foot and tongue movement. The epochs were then used to train the CSP and the LDA models. The same pre-processing flow was applied to the validation dataset, but, in this case, the epochs were classified by the computed model and used to calculate the validation accuracy rates. All the processing was performed *offline*.

In Table 7.2, the parameters set for the IIR bandpass filter as well as for the CSP configuration are displayed. The configuration list includes the bandpass filter cutoff frequencies and order, the time range of each epoch using the event timestamps as the reference, and the number of CSP eigenvectors used in the preprocessing stage. To allow an effective comparison, these parameters were set according to [32], which also performed the same tests on the same dataset used here. The goal of this section is to compare both achieved results to assess whether the platform signal processing is operating properly.

In Table 7.3, the classification accuracy rates for each of the subjects are depicted. The Table shows the results obtained using the developed platform and the results achieved by [32]. The global mean across all subjects for each work is also shown in the last column. The performance of the algorithm was assessed, in both cases, through validation. The last row of the Table shows the absolute error between the achieved results in both works (the difference between accuracy attained by this work and by [32]).

The low error rates demonstrate that, compared to a well-established reference, the routines for filtering and classifying the μ-waves patterns are working properly. The small variations presented in Table 7.3, as for instance for the classification rates of subject 4, can be explained based on the implementation variations of the processing methods and, also, due to subtle numerical precision discrepancies. Since the error is low compared to the classification rates, these differences do not compromise the system designed in this work.

The parameters shown in Table 7.2 were set according to [32] to allow a proper comparison. However, it is important to notice that using the same configuration for classifying EEG patterns from different users is not an optimal procedure to maximize the classification rates. Since the brain structure and μ-wave generation mechanism can be very distinct from one subject to another, it is usually better to tune the signal processing modules for each user. The tuning procedure, though, can be complex due to the number of parameters to be configured and their impact on the final classification rate. Although not implemented in this work, one possible solution to this problem is to devise an algorithm for automatically tuning the pre-processing parameters of the models, targeting a maximum accuracy rate for each specific subject.

7.1.2 Drone Control System

The module implemented to control the direction of the drone in the simulator also required dedicated testing. This module was discussed in Section 4.4 but without providing any practical examples. The tests were performed using the drone simulator described in Section 4.5. Since the simulator incorporates the physical and dynamic model of the drone model utilized in this work, the results obtained in this section can be reasonably extrapolated to the real behavior of the aircraft in a real environment. For the test, the following procedure was used:

(i) Measure the initial angular position θ of the drone

(ii) Define an angular position θ_{ref} as a target direction for the drone.

(iii) Measure the behavior of the control signal described in equation 4.1 (yaw rotation velocity).

(iv) Measure the dynamics of the $\theta(t)$ in response to the control signal changes.

(v) Assess the final position θ of the drone and the resultant error signal.

The obtained results are shown in Figure 7.1. The gain K_p in equation 4.1 was set to 0.05. The top-most curve shows the angular position of the drone θ. The θ_{ref} is also displayed in the same plot. The graph in the middle depicts the error signal $\theta_{error} = \theta_{ref} - \theta$. Finally, the last curve shows the value of the control signal, or the yaw rotation velocity, of the drone in m/s. All values are plotted as a function of time. For generating the curves, the drone was initially at the position $\theta = 90^o$, the target position was set $\theta_{ref} = 270^o$, and the maximum rotation velocity was $v_{yaw}^{max} = 1$ m/s.

At the beginning of the experiment shown in Figure 7.1, the drone is stable at position $\theta = 100^o$. As soon as the control system starts to act (when the new θ_{ref} is set), the error becomes different than zero, and the yaw velocity is updated according to equation (4.1). The yaw velocity reaches its maximum value initially since the error is relatively large. As θ gets closer to θ_{ref}, at approximately $t = 7$ s, the error becomes small, and the yaw velocity is rapidly decreased to zero. The drone is then locked in the new angular position or direction. The control system then stabilizes the drone in this new θ value, while waiting for an update in θ_{ref}. When it happens, the process repeats until the new position is achieved.

7.2 Drone Control using Public EEG Data

This experiment was used to assess the procedure for drone control using EEG signals and the devised motor imagery BCI system. The EEG dataset from the BBCI competition IV was used to simulate a BCI user, serving as input to the signal processing and control stages. The data was created by appending epochs from two classes (left and right-hand imagery-motor) and generating an artificial dataset which simulated a user piloting the drone along a predefined track. The performance was measured based on the three metrics presented in Section 3.3 adapted to the application proposed in this work: the accuracy rate

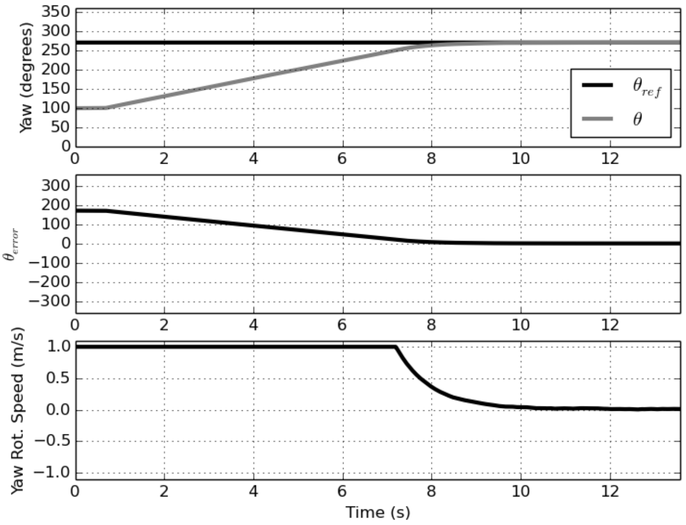


Figure 7.1: Top: Angular position θ of the drone and target position θ_{ref}. Middle: Error signal $\theta_{error} = \theta_{ref} - \theta$. Bottom: Yaw rotation velocity as described in equation 4.1 with $K_p = 0.05$. The results obtained using the drone simulator. The experiment starts with the drone at $\theta = 100^o$ and $\theta_{ref} = 270$. At the beginning, the error is large and the yaw velocity is maximum. As θ converges to θ_{ref}, the error signal becomes small and the yaw velocity is reduced. The drone is stabilized at the new position as the error signal and the velocity becomes very close to zero.

of the classifier model, the overall time to complete the track and the error between the optimal path and the path traveled by the drone when being controlled by the BCI system.

The datasets from subjects 7 and 8 were used to assess the performance of the system. The data from Subject 7 demonstrated a relative average validation accuracy and allowed to evaluate the behavior of the system when operated by a user with limited model accuracy. The data from subject 8 assessed the performance of the system when the user is well-trained, and the trained CSP+LDA model is highly accurate (see Table 7.3 for accuracy rates achieved by subject).

For each subject, ten runs were performed. In each run, the system was responsible for randomly selecting epochs labeled as left and right-

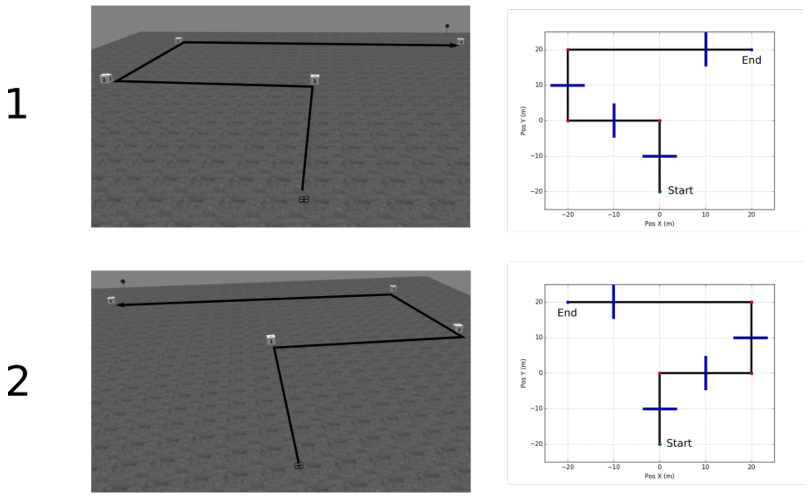


Figure 7.2: Simulation scenarios for drone control. In each scenario, a sequence of numbered targets indicated the path which the drone had to follow.

hand imagery-movement from the validation dataset, processing the incoming signal and generating the control signal to pilot the drone in a simulated environment. To test the generalization capability of the system, the simulations were run in the two scenarios shown in Figure 7.2. Both scenarios consisted of a plane with numbered targets (turning points) that indicated the path that should be followed by the drone. The plots on the right-hand side of Figure 7.2 highlight the optimal path considered in this experiment. The sequence of turn directions at each turning point was configured in the platform, enabling the system to concatenate the EEG epochs according to these predefined directions.

To define the moment at which the algorithm should start appending epochs from a given class, an action limit distance d_{act} was defined. This limit is shown in Figure 7.2 as blue lines perpendicular the drone's goal path. When the drone is moving before this line, the algorithm appends epochs to the dataset looking to balance the command bars described in 6.4.2 and, hence, not triggering any command. When the drone crosses this limit line, the algorithm starts concatenating EEG data from a given class (left or right) to turn the direction of the drone towards the predefined path configured in the platform. This mechanism was based on the behavior of a real user operating the platform:

Table 7.4: Parameters set for processing and classification.

Parameter	Value
Bandpass Filter Frequency Range	8 to 30 Hz
Bandpass Filter Order	5^{th}
Epoch Time range	0.5 to 2.5 sec
CSP *eigenvectors*	6
Drone longitudinal Velocity v_{drone}	1 m/s
d_{act}	10 m
δ_t	100 ms
Δ_{th} Subject 7	79.4
Δ_{th} Subject 8	92.4

when the user does not want to change directions, he will focus on equalizing the two bars by executing tasks from both classes interchangeably. As he gets closer to the turning point, he will begin to focus on one single class, which will perform the required change in the direction.

All the drone dynamics and control algorithms were similar to the real characteristic of the AR.Drone 2.0. In runs in which the drone significantly deviated from the correct path (for instance, when the drone gets 40 meters away from the center of the scenario), the simulation was interrupted and not included in the final results. In Table 7.4, the general configuration parameters of the system set for the simulations are depicted.

The signal processing parameters for filtering and classifying the EEG data were set as described in Table 7.2. The drone longitudinal (forward) velocity was set to 1 m/s. The distance between the blue lines shown in Figure 7.2 and the turning point d_{act} was 10 meters, exactly as shown in the scenarios graphs in Figure 7.2. Hence, the time available to take an action is:

$$t_{ta} = \frac{d_{act}}{v_{drone}} = \frac{10}{1} = 10s \tag{7.1}$$

Next, substituting the result above in equation 5.15 and taking into account the validation accuracies for each subject shown in Table 7.3, the threshold variation for subject 7 ($\Delta_{th} = 79.4$) and for subject 8

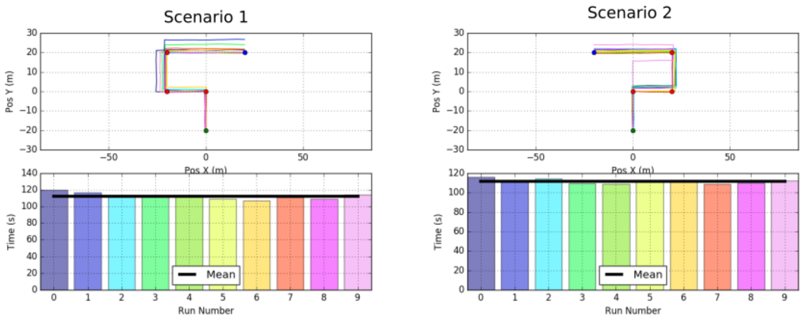


Figure 7.3: Simulation path for EEG data from subject 8 for both simulated scenarios. Top: path traveled by the drone in each run. Bottom: respective time required to complete the track in each run.

($\Delta_{th} = 92.4$) were computed. These results are also shown in Table 7.4.

On top of Figure 7.3, the path traveled by the drone in each run is shown with different colors for the case where the dataset used was from subject 8. The figure shows the results for both simulated scenarios shown in Figure 7.2. On the bottom, the time required to complete the track is displayed for the runs performed, matching the color patterns shown on the top curve. The line in black highlights the mean execution time across all runs.

The curves displayed in the top part of Figure 7.3 demonstrate the correct operation of the designed system. In the first scenario, all the paths trajectories are grouped together, indicating that the algorithm correctly generated the commands for changing the direction of the drone based on the artificially generated EEG data.The fact that the drone was able to modify the course almost at the exact position of the numbered squares shows that the methodology for calculating and estimating the behavior of U is valid. This is also indicated by the relatively constant time across all runs required to complete the track. For the simulations in the second scenario, although the majority of the runs still performed well and exhibited a concise group trajectory, the result obtained in run number 9 significantly deviated from the optimal path.

The effect observed in run number 9 on the right-hand side of Figure

7.3 is because the proposed methodology tends to penalize classification errors drastically when the model accuracy is high (close to 100 %). For models with m_{acc} close to one, the estimated value of U along time by equation 5.11 becomes close to the ideal value described in equation 5.8. As a consequence, a command is triggered only when the variation across the circular buffer A is close to its maximum ideal value (when the buffer is filled with $u = 1$). However, assume the case where a wrong classification (False negative) occurs. This miss-classification will push a sample $u = 0$ to the most recent position of the buffer A. Since the command is only triggered when the variation inside the buffer is nearly ideal, this error will reduce the value of Δ, preventing it from reaching the optimum target value Δ_{th}. This limitation will persist until new correctly classified samples are pushed to the circular buffer, and the miss-classification is removed from it. This explains why in run number 9 of the second scenario, the drone missed by far the first turning point. A miss-classification right before reaching the turning point can prevent the system from generating a turning command for approximately the time length of the buffer. Since, generally, in models with very high accuracy, it is very unlikely that miss-classifications will happen, this limitation of the proposed methodology does not impose severe issues. However, more complex and robust implementations of this method can help mitigate such issue.

Similarly, in Figure 7.4, the same simulation results are shown for the case the dataset from subject 7 was used. In this case, specially in the first scenario, the trajectory curves are more dispersed than presented in Figure 7.3. This can be explained based on the model accuracy m_{acc}. Since, for the model trained with data from subject 7, the validation accuracy reached was $m_{acc} = 79.4\%$, the system becomes more susceptible to miss-classifications.

The rate m_{acc} measures the capability of the model incorrectly identifying the epochs in the validation dataset. Hence, an accuracy of 80 % implies that the LDA model correctly identified 80 % of the epochs in the validation dataset. However, in *online* mode and as explained in Section 5.2, the classification is performed based on the incoming EEG data at every δ_t milliseconds. This implies that the classification is performed mostly in overlapped data, the same chunk of EEG data is reclassified with δ_t milliseconds of new data. Therefore, the

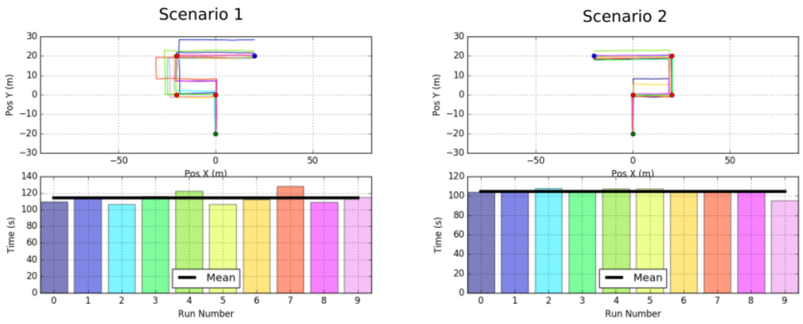


Figure 7.4: Simulation path for EEG data from subject 7 for both simulated scenarios. Top: path traveled by the drone in each run. Bottom: respective time required to complete the track in each run.

real-time classification scenario is significantly different from the one used to compute the accuracy rate, where a complete 2-seconds epoch is classified. Besides, the problem is aggravated since the generation of the dataset in this simulation is accomplished by appending complete epochs and sending it to processing unit. For instance, considering the case where an epoch of 2 seconds with bad EEG data (an epoch which was wrongly classified by the model in the validation stage) is appended to the artificial dataset and sent to the signal processing module. This epoch will be buffered and classified by the system $2/\delta_t$ times or, in this specific case $2/0.1 = 20$ times. This will fill the circular buffer A with classifications mistakes (False Negatives), significantly delaying the triggering of a command to the application.

The limitation described above can cause the path fluctuations shown in Figure 7.4. This effect is not relevant when the model accuracy is high since the probability of appending a bad epoch to the artificial dataset is very low. By devising new and more complete methods to assess the performance of the classification model, the described issue can be significantly mitigated. These methods have to address, specially, the operation conditions of the classifier in *online* mode, where the data overlap between consecutive classifications is considerably high.

Finally, to extend the discussion about the limitations of the proposed methodology, consider the proposed methodology for the case where the classification accuracy is relatively low, close to the chance-level (50 % for a two-class classification problem), and all the other parameters displayed in Table 7.4 are kept unchanged. Since the drone speed is the same and the distance d_{act} remained constant, the time to action parameter t_{ta}, calculated using equation 7.1, remains $t_{ta} = 10$ s. In this case, according to equation 5.15, the computed threshold will also be low. As a consequence, the limit required for dispatching a command is lower. In the proposed methodology, this works as a mechanism to compensate the poor behavior of the classification model: for low accuracies rates, the computed threshold is set to low values to help the system to achieve this limit. Setting low values for the threshold, though, tends to make the system more susceptible to miss-classifications and more prone to false triggering commands. This is specially a problem since the classifier itself, which already achieve poor classification accuracy rates, is very likely to miss-classify the incoming data.

To avoid the problem described above, two options can be considered: 1- reduce the drone speed, 2- Increase the validation model accuracy. By reducing the drone speed, the t_{ta} parameter and, as a consequence, the limit Δ_{th} become higher. This measure will prevent the system from false triggering commands but, in turn, will make the system slower and reduce its usability. The second option is straightforward: the better the validation model, the better the system will translate the EEG signals into commands [50]. However, this option is not always available since the accuracy is dependent on the user, the processing parameters choice and so on. Also, it is worth to mention that there is obviously a lower limit for the allowed accuracy of the model. As this accuracy gets closer to its chance-level, the classifier can no longer identify the patterns in the EEG data, and the system becomes inoperable.

Although the designed system and specially the proposed methodology present the limitations cited above, the results shown in Figures 7.3 and 7.4 demonstrate its proper operation in the two distinct scenarios proposed. In addition, the obtained results also inferred the operation of all the signal processing and drone interface modules of the designed BCI system. The next important step is to validate also the signal acquisition module and the operation of the system when a less reliable source of EEG signals is being used. This will be explored in the next section.

7.3 Drone Control using Acquired Data

In this section, the testing method performed in the last section based on a public EEG data will be repeated using the EEG data acquired by the OpenBCI amplifier and the developed platform. The procedure will be similar to the one presented before; the dataset will be segmented into epochs to generate an artificial dataset which simulated the user behavior when operating the system.

The first testing step included the analysis of the signal acquisition or the Data Handler, as described in section 6.2. The test assessed the reliability of the communication between the developed platform and the OpenBCI amplifier. For this experiment, a volunteer was asked to sit comfortably in a chair in front of a monitor which displayed the graphical interface of the designed platform. The EEG electrodes were positioned according to the $10-20$ system. Since the OpenBCI amplifier only allows the acquisition of 16 electrodes simultaneously, the $10-20$ system locations shown in Figure 7.5 were selected for acquisition based on the information about the brain characteristics and μ-waves described in chapter 2. These locations in general provide better results for the detection of μ-waves [19].

During the acquisition stage, a series of a instructions was presented to the user. The presentation scheme followed was similar to the one described in section 3.4 and in section 6.3, the timing parameters of are shown in Table 7.5. The time stamp and label were recorded by the Event Maker at the exact moment when the cue was displayed to the user. Two datasets were recorded, one for training and other for validation of the classification model. For each dataset, 45 trials were acquired per class in 3 runs separated by pause intervals of 15 seconds (15 trials per run).

After obtaining the data following the procedure discussed above, the segments of EEG data were extracted around the marked events to form the training epochs. In Figure 7.6, the EEG signals of a left-hand movement epoch, acquired from channel C3 and C4, are depicted as a function of time. The signals displayed in Figure 7.6 were filtered through a bandpass filter with the frequency range of 0.5 to 50

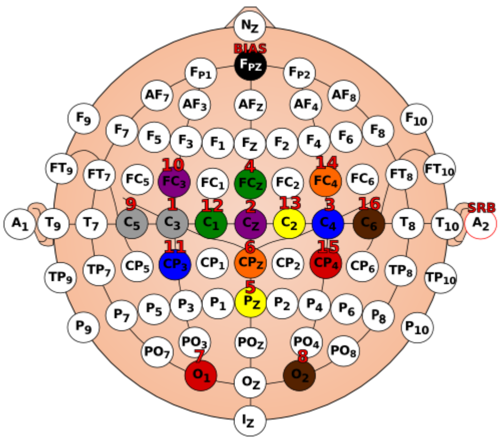


Figure 7.5: lectrodes positions according to the $10-20$ system for signal acquisition with OpenBCI. The locations were chosen based on the brain physical characteristics and the generation properties of μ-waves.

Table 7.5: Timing characteristics used to EEG data acquisition using the devised platform (based on Figure 6.8).

Parameter	Value
Cue Offset (s)	1
Task Execution Time (s)	7
End of Trial Offset (s)	1

Hz. According to section 2.4, when the user is performing a left-hand movement, the neurons from the right hemisphere of the motor cortex (channel C4, for instance) are desynchronized and, therefore, the alpha energy or the μ-waves are expected to have a low amplitude compared to the left hemisphere (channel C3). However, the displayed curves do not present these discriminative characteristics. To highlight these properties, it is common to analyze the average of the spectrum of the measured signals for each class. In Figure 7.7 the average fast Fourier transforms (FFT) of the epochs from left, and right movement task is displayed.

To obtain the curves, the following procedure was applied to the EEG signal:

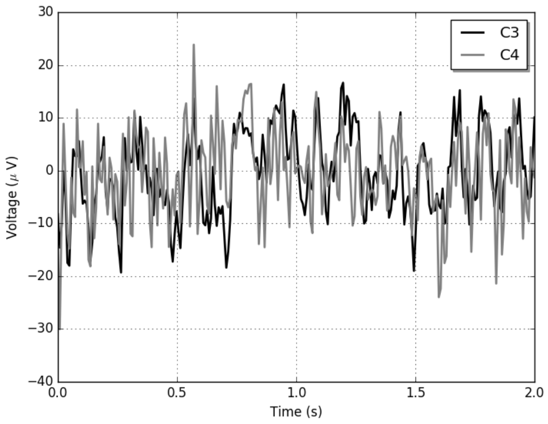


Figure 7.6: EEG signals acquired using the platform and the OpenBCI amplifier. The curves show the voltage signal as a function of time acquired at electrode C3(black) and C4(gray).

(i) Filter the EEG signals with bandpass filter with a frequency band of 0.5 to 50 Hz

(ii) Extract the epochs from each class: left-hand and right-hand movement.

(iii) Compute the FFT of each extracted epoch.

(iv) Compute the average of the FFT signal across all epochs from each class.

(v) Plot the result for channel C3 from 0 to 30 Hz.

The average FFT shown in Figure 7.7 demonstrates the differences between the brain signals when the user is performing a left and right imagery-motor task. The amplitude peak shown in 7.7 at around 10 Hz for the case when the user is engaged in a left-hand imagery movement is compatible with the description of μ-waves. When the user is executing a right-hand imagery task, a desynchronization in the neurons of the left hemisphere of the brain occurs. As a result, the power concentrated over this band in channel C3 gets significantly attenuated. This effect can be seen in the curves of Figure 7.7.

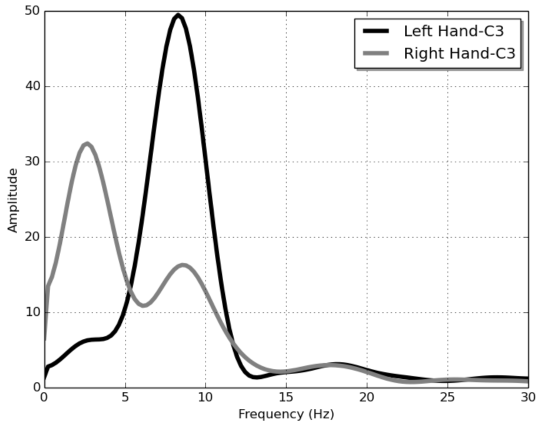


Figure 7.7: Average Fast Fourier Transform of channel C3 of Left movement (black) and Right movement (gray) epochs. The curves were obtained following the procedure described in the text. Since analyzed channel is in the left hemisphere, its energy in the α band is expected to be higher when the right-side members of the body are in a relaxed state. This can be seen by the amplitude peak found at around 10 Hz in the black curve shown.

To train the CSP and LDA models, the parameters depicted in Table 7.6 were used for configuring the signal processing stages. The parameters were chosen empirically in order to maximize the validation accuracy while maintaining reasonable conditions for the drone control simulation. With the given parameters, the validation accuracy rate achieved was 72.5 %. When acquiring the signal from all 16 channels of OpenBCI, the sample rate is reduced to 125 Hz, as described in section 6.2.1.

Although the accuracy rate achieved was not high (i.e.,. > 90 %), the previous tests performed with the public dataset demonstrated that the drone control at average accuracy rates, such as the one achieve by the dataset from subject 7, is feasible. Also, the model classification parameters were chosen manually, without a robust method for optimizing their choice targeting a maximum accuracy rate. The configuration of the filter and classifier in an automatic way can save considerable time while ensuring the operation of the system in its optimum state.

Finally, the trajectory of the drone in the simulated environment

Table 7.6: Parameters set for processing and classification.

Parameter	Value
Bandpass Filter Frequency Range	8 to 30 Hz
Bandpass Filter Order	5^{th}
Epoch Time range	0.6 to 2.6 sec
CSP eigenvectors	8
Drone longitudinal Velocity v_{drone}	1 m/s
d_{act}	10 m
δ_t	100 ms
m_{acc}	0.725
Δ_{th}	73

using the dataset acquired with the OpenBCI amplifier is shown in Figure 7.8. The paths traveled by the drone resemble the ones achieved by the simulation performed for the dataset from subject 7, in the previous section. Although the majority of lines is concentrated close to the optimal path, specially in scenario 1, the drone significantly diverged from the optimal path in some runs (9,6 and 4 in scenario 1). The resemblance of both results can be explained by the close validation accuracy rates achieved by both models. Hence, the same conclusions drawn from the results shown in Figure 7.4 can be extended to the ones displayed in Figure 7.8.

The signals used in this section were acquired by a simpler EEG equipment, with smaller acquisition rate capability and fewer channels than the one used to acquire the public dataset described in the previous section. Nevertheless, the results obtained with this open-source equipment are comparable to the ones achieved with the more robust device, as shown in Figure 7.8. Using the acquired data to simulate a user operating the developed platform, the system was able to correctly identify the μ-wave synchronization and desynchronization events and, also, to properly map the output of the classifier into commands to change the direction of the drone. Furthermore, even for the model with the lowest accuracy rates, the speed of the drone remained unchanged without significantly impacting the overall quality of the results. Hence, the achieved results shown in this section successfully

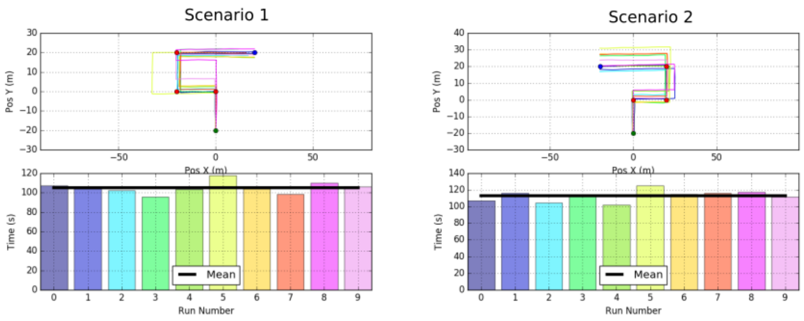


Figure 7.8: Simulation path for EEG data acquired with the OpenBCI amplifier for both simulated scenarios. Top: path traveled by the drone in each run. Bottom: respective time required to complete the track in each run.

demonstrate the operation of the platform when working with a less robust, cheaper and more practical EEG source.

CHAPTER 8

Perspectives and Conclusions

In this work, a BCI system for controlling a drone was designed and tested using public EEG data from the BBCI competition IV and private data acquired using the OpenBCI amplifier. The main contribution of this work was the practical evaluation and report of a complete implementation of a BCI system based on motor imagery and μ rhythms. Although it was not possible to assess the performance of the system in a complete real-time environment due to time constraints, the proposed method of testing the system with previously acquired EEG data has successfully allowed the testing and the performance assessment of the designed BCI system.

The proposed system focused on a BCI system based on μ waves for command generation. Although this approach has demonstrated several advantages concerning detection and acquisition complexity, it is worth to note that the system can be expanded to include other kinds of neural responses as well. In this sense, the research of BCI systems based on combinations of brain signals, often called hybrid systems [3], can also produce considerable improvements in the overall quality of the system. These hybrid systems present the advantages of combining different kinds of brain responses to enable the user to control the application using multiple channels of communication. Therefore, the flow of information becomes closer to the flow found in daily social and motor interactions, allowing a more immersible and beneficial experience for the user [27] [14]. These features and improvements in the designed system can be pursued shortly.

The implementation of a BCI system using a low-cost and open source amplifier enabled the analysis of these systems when operating with simpler equipment, which is not approached in literature [18] [9]. The classification rates obtained when applying the CSP and LDA approach to the collected data showed reasonable accuracy when compared to the ones presented in similar works [32], but with implementations based on more robust and expensive EEG apparatus. Nevertheless, the lack of a similar research work, with similar characteristics and application has made impossible to perform a complete and parametric comparison between low-cost systems and the ones based on well-established technologies. Such comparison, with a deeper analysis, can be pursued in future works, as new technologies such as the OpenBCI hardware become available.

The use of a drone as the target application also enabled an in-depth analysis of a BCI system operating for controlling an application with significant levels of movement freedom. Besides, the flight dynamics and timing characteristics of such application provided a valuable tool for evaluation the performance of the BCI system. The most significant contribution in this sense was the technique proposed for translating the output of the classifier into useful commands which could be sent to the drone. This particular part of BCI systems, as explained, is poorly discussed or entirely omitted in BCI research papers or, in other cases, is based on more simplistic approaches [26] [20]. However, without a proper conversion between identified classes to application instructions, the designed system becomes incomplete and fails to achieve the goals initially targeted. The results obtained when testing the techniques have reasonable validated the proposed method by demonstrating the capability of piloting the drone across a defined path. This application control occurred while receiving pure and raw EEG data at the signal acquisition stage. The complete signal flow implemented in this work can be helpful as a basis for designing new BCI systems with diverse and useful applications. This expansion can also be explored in future works.

As discussed in the previous Chapter, the method proposed for translating the classifier output into drone commands was based on a very simple approach of an integrator. Further improvements in the usability of the system can still be achieved by considering more sophisticated techniques for taking into account the sequence of outputs provided by the classifier. The idea is to capture the user intent to generate a command to change the direction of drone when other parameters, in addition to the value of U, also suggest that the user is trying to reach this goal. This improved algorithm, of course, has to take into account noisy events that might change the output of the classifier and lead to a wrong classification. In this sense, although not discussed here, a few initial tests were performed including also a derivative and proportional control system in this translation algorithm, suggesting a slight improvement in the usability of the system. Further improvements in this regard are currently being pursued.

Also, regarding the practical details of a BCI system, the use of comprehensive frameworks for communicating with target applications can be a major advance towards the diffusion of implementations of complete BCI systems. For this purpose, the use of ROS as the interface layer between the BCI and the drone demonstrated that this might be an interesting way of abstracting the development of more sophisticated routines for dealing with issues specifics to each application. In fact, the use of ROS as a common platform for controlling applications through BCI systems is already being discussed in the literature [46]. The generality and robustness of frameworks for the development of robot applications such as ROS constitute in useful tools for the dissemination of BCI technologies.

As a final point regarding the drone application, the testing of the platform in a real scenario where the actual drone device is being controlled can also pose new challenges to the approach proposed here. Although the simulator models implemented in ROS and Gazebo used in this work can reliably replicate the flight dynamics of the drone, the use of the real device will inherently pose practical issues to test the complete workflow of the system. For instance, a large open space is required to construct the real scenarios where the drone will navigate (similar to the virtual ones displayed in Figure 7.2). Since the acquisition of EEG requires well-controlled environments, the tests performed in these open locations constitute in an important issue to be addressed in the future. Moreover, the use of a real aircraft and, considering the BCI control being executed by a real user, can also lead to problems involving the concentration of the user during the experiment. As previously reported [33], the use of real environments and applications can disturb the user's concentration significantly and, consequently, impact the accuracy of the BCI system. These issues will need to be inherently addressed as this work evolves to more practical and complete implementations.

Another important aspect which can be further explored in future works is the use of more complex algorithms for reliably classifying the μ wave synchronization and desynchronization as, for instance, presented in [2] and [32]. Although this work focused only on well-established, simpler yet robust techniques for this task, it is well reported in the literature that slightly more sophisticated algorithms can significantly improve the overall quality of the system regarding usability, data transmission speed, and accuracy [19]. These features have also been explored by other members of our research group [43] [42], but the derived routines for signal filtering and classifications could not be incorporated into this work in able time. However, this should be one of the main focus of the group presently.

A simpler approach for improving the overall quality of the system is to include some optimizer for choosing the parameters for the model calibration stage. In this work, these parameters were set according to previous references, for the public datasets, or empirically, for the dataset acquired. Since the model accuracy imposes severe consequences on the speed and precision of the application control, a method for automatically tuning the configuration parameters can be useful to achieve a better usability.

Another vital contribution provided here is the platform for implementing and testing BCI applications from EEG data acquisition to application controlling. Because of the modular approach adopted, the code developed here can be used as a basis for the development of new BCI systems which focus on completely different scenarios and applications. In this sense, the inclusion of new signal processing techniques, calibration routines and application interfaces in the platform consist in an important improvement which can be carried on in future works. The integration of the developed platform with current state-of-the-art BCI software such as [39] and [23] can also be pursued in this regard.

Although this topic was not frequently covered in this report, the procedures for the acquisition of EEG signals consisted in one of the greatest challenges faced throughout the development of this work. The EEG signal acquisition is, in general, a complex technique. The lack of experience in the assembly and positioning of the EEG electrodes consumed a significant amount of time, often without producing concrete results which could be based on for generating conclusions and insights. An important point noticed during this signal acquisition stages it that, for conducting top-level BCI research, it is highly recommended to have an expert for going through all the procedures and methods for EEG signal acquisition. The research on the development of new equipment or techniques to overcome the basic yet arduous issues found in this stage consists in a necessary improvement for the diffusion of BCI technologies in non-scientific environments. Initiatives such as the OpenBCI have been significantly thrusting the BCI field in this direction. However, there is still a lot of development required to achieve better usability and portability results for BCI systems.

Also regarding the acquisition of EEG signals, the test and validation of the platform with a more robust and complete protocol for signal acquisition are also important. To assess the performance of the signal acquisition module in this work, the signals from only one user were acquired and supplied to the platform. However, due to the diversity and variability of this type of biological signal, a complete study of a larger user population is crucial to assess how well the platform and the proposed methodology can adapt to different EEG sources.

The purpose of this dissertation is to provide general guidelines for the design of a complete MI-based BCI system. The practical details as well as the problems faced and reported here will hopefully help new researchers to implement better and more robust BCI systems.

References

[1] *The Navigation and Control Technology Inside the AR.Drone Micro UAV*, Milano, Italy, 2011.

[2] M. H. Alomari, A. Samaha, and K. AlKamha. Automated classification of L/R hand movement EEG signals using advanced feature extraction and machine learning. *CoRR*, abs/1312.2877(6), 2013.

[3] S. Amiri, R. Fazel-Rezai, and V. Asadpour. A review of hybrid brain-computer interface systems. *Advances in Human-Computer Interaction*, (187024), 2013.

[4] A. Barachant, S. Bonnet, M. Congedo, and C. Jutten. Multiclass brain–computer interface classification by riemannian geometry. *Biomedical Engineering, IEEE Transactions on*, 59(4):920–928, 2012.

[5] A. Bashashati, M. Fatourechi, R. K. Ward, and G. E. Birch. A survey of signal processing algorithms in brain–computer interfaces based on electrical brain signals. *Journal of Neural Engineering*, 4(2):R32, 2007.

[6] L. Brinkman, A. Stolk, H. C. Dijkerman, F. P. de Lange, and I. Toni. Distinct roles for alpha- and beta-band oscillations dur-

ing mental simulation of goal-directed actions. *Journal of Neuroscience*, 34(44):14783–14792, 2014.

[7] N. Brodu, F. Lotte, and A. Lecuyer. Comparative study of band-power extraction techniques for motor imagery classification. In *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on*, pages 1–6. IEEE, April 2011.

[8] Y. Chae, J. Jeong, and S. Jo. Noninvasive brain-computer interface-based control of humanoid navigation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 685–691, Sept 2011.

[9] B. Choi and S. Jo. A low-cost eeg system-based hybrid brain-computer interface for humanoid robot navigation and recognition. *PLoS ONE*, 8(9):e74583, 09 2013.

[10] K. Community. Kivy: Crossplatform framework for nui, 2012.

[11] E. A. Curran and M. J. Stokes. Learning to control brain activity: A review of the production and control of eeg components for driving brain–computer interface (bci) systems. *Brain and Cognition*, 51(3):326 – 336, 2003.

[12] A. J. Doud, J. P. Lucas, M. T. Pisansky, and B. He. Continuous three-dimensional control of a virtual helicopter using a motor imagery based brain-computer interface. *PLoS ONE*, 6(10):e26322, 10 2011.

[13] T. Ebrahimi. Recent advances in brain-computer interfaces. In *Multimedia Signal Processing, 2007. MMSP 2007. IEEE 9th Workshop on*, pages 17–17, 10.1109/MMSP.2007.4412807, Oct 2007.

[14] G. Edlinger and C. Guger. A hybrid brain-computer interface for improving the usability of a smart home control. pages 182–185, July 2012.

[15] R. Fazel-Rezai, B. Z. Allison, C. Guger, E. W. Sellers, S. C. Kleih, and A. Kübler. P300 brain computer interface: current challenges and emerging trends. *Frontiers in Neuroengineering*, 5(14), 2012.

[16] P. S. Foundation. Python language reference, version 2.7.

[17] G. F. Franklin, D. J. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition, 2001.

[18] J. Frey. Comparison of an open-hardware electroencephalography amplifier with medical grade device in brain-computer interface applications. *CoRR*, abs/1606.02438, 2016.

[19] B. Graimann, B. Z. Allison, and G. Pfurtscheller. *Brain-Computer Interfaces: Revolutionizing Human-Computer Interaction.* Springer Publishing Company, Incorporated, 2013.

[20] D. Göhring, D. Latotzky, M. Wang, and R. Rojas. Semi-autonomous car control using brain computer interfaces. In S. Lee, H. Cho, K.-J. Yoon, and J. Lee, editors, *Intelligent Autonomous Systems 12*, volume 194 of *Advances in Intelligent Systems and Computing*, pages 393–408. Springer Berlin Heidelberg, 2013.

[21] X. Hong, S. Chen, A. Qatawneh, K. Daqrouq, M. Sheikh, and A. Morfeq. A radial basis function network classifier to maximise leave-one-out mutual information. *Applied Soft Computing*, 23:9 – 18, 2014.

[22] M. A. Khan, M. R. Islam, and M. K. I. Molla. Artifact suppression from electroencephalography signals using stationary subspace analysis. In *2016 19th International Conference on Computer and Information Technology (ICCIT)*, pages 252–256, Dec 2016.

[23] C. A. Kothe and S. Makeig. Bcilab: a platform for brain–computer interface development. *Journal of Neural Engineering*, 10(5):056014, 2013.

[24] D. H. Krishna, I. A. Pasha, and T. S. Savithri. Autonomuos robot control based on eeg and cross-correlation. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–4, Jan 2016.

[25] D. J. Krusienski, M. Grosse-Wentrup, F. Galán, D. Coyle, K. J. Miller, E. Forney, and C. W. Anderson. Critical issues in state-of-the-art brain–computer interface signal processing. *Journal of neural engineering*, 8(2):025002–025002, 04 2011.

[26] K. LaFleur, K. Cassady, A. Doud, K. Shades, E. Rogin, and B. He. Quadcopter control in three-dimensional space using a noninvasive motor imagery-based brain–computer interface. *Journal of Neural Engineering*, 10(4):046003, 2013.

[27] Y. Li, J. Pan, F. Wang, and Z. Yu. A hybrid bci system combining p300 and ssvep and its application to wheelchair control. *Biomedical Engineering, IEEE Transactions on*, 60(11):3156–3166, Nov 2013.

[28] J. Long, Y. Li, H. Wang, T. Yu, and J. Pan. Control of a simulated wheelchair based on a hybrid brain computer interface. In *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, pages 6727–6730, Aug 2012.

[29] F. Lotte. The use of fuzzy inference systems for classification in eegbased brain-computer interfaces. In *The use of fuzzy inference systems for classification in eegbased brain-computer interfaces*, pages 12–13.

[30] F. Lotte. A tutorial on eeg signal processing techniques for mental state recognition in brain-computer interfaces. Springer, 2014.

[31] F. Lotte, M. Congedo, A. Lecuyer, F. Lamarche, and B. Arnaldi. A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 4(2):R1, 2007.

[32] F. Lotte and C. Guan. Spatially regularized common spatial patterns for eeg classification. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, 1051-4651, pages 3712–3715, 10.1109/ICPR.2010.904, Aug.

[33] D. J. McFarland, W. A. Sarnacki, and J. R. Wolpaw. Electroencephalographic (eeg) control of three-dimensional movement. *Journal of Neural Engineering*, 7(3):036007, 2010.

[34] M. L.-V. T. W. J. McFarland, DennisJ. Mu and beta rhythm topographies during motor imagery and actual movements. *Brain Topography*, 12(3):177–186, 2000.

[35] G. Pfurtscheller and F. H. Lopes da Silva. Event-related eeg/meg synchronization and desynchronization: basic principles. *Clinical Neurophysiology*, 110(11):1842–1857, 2015/10/13.

[36] J. G. Proakis and D. K. Manolakis. *Digital Signal Processing (4th Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[37] D. Purves, G. J. Augustine, D. Fitzpatrick, L. C. Katz, A.-S. LaMantia, J. O. McNamara, and S. M. Williams. *Neuroscience*. Sinauer Associates, 2 edition, 2001.

[38] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[39] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand, and A. Lécuyer. Openvibe: An open-source software platform to design, test, and use brain–computer interfaces in real and virtual environments. *Presence: Teleoper. Virtual Environ.*, 19(1):35–53, Feb. 2010.

[40] L. R. L. H. N. M. P. T. Sharbrough F, Chatrian G-E. American electroencephalographic society guidelines for standard electrode position nomenclature. volume 8. Journal of Clinical Neurophysiology, 1991.

[41] D. J. K. Shih, Jerry J. and J. R. Wolpaw. Brain-computer interfaces in medicine. *Mayo Clinic Proceedings*, 87 (3):268–279, 2012.

[42] C. Silva, R. Duarte, R. Goulart, and A. Trofino. Towards a lmi approach to feature extraction improvements and classification by riemann distance. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*, pages 990–995, June 2016.

[43] C. Silva, R. Duarte, and A. Trofino. Feature extraction improvements using an lmi approach and riemannian geometry tools: An application to bci. In *2016 IEEE Conference on Control Applications (CCA)*, pages 966–971, Sept 2016.

[44] M. T. F. Talukdar, S. K. Sakib, N. S. Pathan, and S. A. Fattah. Motor imagery eeg signal classification scheme based on autoregressive reflection coefficients. In *2014 International Conference on Informatics, Electronics Vision (ICIEV)*, pages 1–4, May 2014.

[45] M. Teplan. Fundamentals of eeg measurements. In *In: Measurement Science Review, Volume 2, Section*, page 2002.

[46] L. Tonin, A. Cimolato, and E. Menegatti. On the use of ros as a common infrastructure for robotic bci driven applications. In D. S. Gernot R. Muller-Putz, Jane E. Huggins, editor, *Brain-Computer Interfaces*. Verlag der Technischen Universitat Graz, 2016.

[47] M. van Vliet, A. Robben, N. Chumerin, N. Manyakov, A. Combaz, and M. Van Hulle. Designing a brain-computer interface controlled video-game using consumer grade eeg hardware. In *Biosignals and Biorobotics Conference (BRC), 2012 ISSNIP*, pages 1–6, Jan 2012.

[48] Y. Wang, S. Gao, and X. Gao. Common spatial pattern method for channel selection in motor imagery based brain-computer interface. pages 5392–5395, Jan 2005.

[49] WHO and ISCOS. *International Perspectives on Spinal Cord Injury*. 2013.

[50] J. Wolpaw and E. W. Wolpaw. *Brain–Computer Interfaces: Principles and Practice*. Oxford Scholarship, 1 edition, 2012.

[51] S. L. Wu, C. W. Wu, N. R. Pal, C. Y. Chen, and S. A. Chen. Common spatial pattern and linear discriminant analysis for motor imagery classification. In *2013 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB)*, pages 146–151, April 2013.