UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL DEGREE THESIS

BACHELOR'S DEGREE IN BIOMEDICAL ENGINEERING

BACHELOR'S DEGREE IN INDUSTRIAL ELECTRONICS AND AUTOMATIC CONTROL ENGINEERING

MOTOR IMAGERY-BASED BRAIN-COMPUTER INTERFACE BY IMPLEMENTING A FREQUENCY BAND SELECTION

Final Report and Annexes

Student: Ali Abdul Ameer Abbas
Director: Herminio Martínez García
Call: June 2022

Abstract

Motor Imagery-based Brain-Computer Interfaces (MI-BCI) are a promise to revolutionize the way humans interact with machinery or software, performing actions by just thinking about them. Patients suffering from critical movement disabilities, such as amyotrophic lateral sclerosis (ALS) or tetraplegia, could use this technology to control a wheelchair, robotic prostheses, or any other device that could let them interact independently with their surroundings.

The focus of this project is to aid communities affected by these disorders with the development of a method that is capable of detecting, as accurately as possible, the intention to execute movements (without them occurring) in the upper extremities of the body. This will be done through signals acquired with an electroencephalogram (EEG), their conditioning and processing, and their subsequent classification with artificial intelligence models. In addition, a digital signal filter will be designed to keep the most characteristic frequency bands of each individual and increase accuracy significantly.

After extracting discriminative statistical, frequential, and spatial features, it was possible to obtain an 88% accuracy on validation data when it came to detecting whether a participant was imagining a left-hand or a right-hand movement. Furthermore, a Convolutional Neural Network (CNN) was used to distinguish if the participant was imagining a movement or not, which achieved a 78% accuracy and a 90% precision. These results will be verified by implementing a real-time simulation with the usage of a robotic arm.

Resumen

Las interfaces cerebro-computadora basadas en imaginaciones motoras (MI-BCI) son una promesa para revolucionar la forma en que los humanos interactúan con las máquinas o el software, realizando acciones con tan solo pensar en ellas. Los pacientes que sufren discapacidades críticas del movimiento, como la esclerosis lateral amiotrófica (ALS) o la tetraplejía, podrían usar esta tecnología para controlar una silla de ruedas, prótesis robóticas o cualquier otro dispositivo que les permita interactuar de manera independiente con su entorno.

El objetivo de este proyecto es ayudar a las comunidades afectadas por estos trastornos con el desarrollo de un método que sea capaz de detectar, con la mayor precisión posible, la intención de ejecutar movimientos (sin que se produzcan) en las extremidades superiores del cuerpo. Esto se hará mediante señales adquiridas con un electroencefalograma (EEG), su acondicionamiento y procesamiento, y su posterior clasificación con modelos de inteligencia artificial. Además, se diseñará un filtro de señal digital para mantener las bandas de frecuencia más características de cada individuo y aumentar significativamente la exactitud del sistema.

Después de extraer características estadísticas, frecuenciales y espaciales discriminatorias, fue posible obtener una exactitud del 88% en los datos de validación a la hora de detectar si un participante estaba imaginando un movimiento con la mano izquierda o con la derecha. Además, se utilizó una red neural convolucional (CNN) para distinguir si el participante estaba imaginando un movimiento o no, lo que logró un 78% de exactitud y un 90% de precisión. Estos resultados se verificarán implementando una simulación en tiempo real con el uso de un brazo robótico.

Resum

Les interfícies cervell-ordinador basades en imaginacions motores (MI-BCI) són una promesa per a revolucionar la manera com els humans interactuen amb les màquines o el programari, realitzant accions només amb el pensament. Els pacients que pateixen discapacitats de moviment crítiques, com l'esclerosi lateral amiotròfica (ALS) o la tetraplegia, podrien utilitzar aquesta tecnologia per controlar una cadira de rodes, pròtesis robòtiques o qualsevol altre dispositiu que els permeti interactuar de manera independent amb el seu entorn.

L'objectiu d'aquest projecte és ajudar les comunitats afectades per aquests trastorns amb el desenvolupament d'un mètode que sigui capaç de detectar, amb la màxima precisió possible, la intenció d'executar moviments (sense que es produeixin) en les extremitats superiors del cos. Això es farà mitjançant senyals adquirits amb un electroencefalograma (EEG), el seu condicionament i processament, i la seva posterior classificació amb models d'intel·ligència artificial. A més, es dissenyarà un filtre de senyal digital per mantenir les bandes de freqüència més característiques de cada individu i augmentar significativament l'exactitud del sistema.

Després d'extreure les característiques estadístiques, freqüencials i espacials més discriminatòries, va ser possible obtenir una exactitud del 88% en les dades de validació a l'hora de detectar si un participant estava imaginant un moviment de la mà esquerra o de la dreta. A més, es va utilitzar una xarxa neuronal convolucional (CNN) per distingir si el participant estava imaginant un moviment o no, la qual cosa va aconseguir una exactitud del 78% i una precisió del 90%. Aquests resultats es verificaran mitjançant la implementació d'una simulació en temps real amb l'ús d'un braç robòtic.

Acknowledgement

I wholeheartedly thank everyone who has encouraged me to pursue this project, especially my family, friends, and professors. I also want to thank my astounding tutor Herminio Martínez García for supervising me through this thesis; his exceptional assistance always enlightened me with new ideas.

Last but not least, I want to thank my lovely partner Eva Deltor Cortés for constantly believing in my capabilities and cheering me up when I needed it the most. I am extremely grateful to share my life with another engineer who can comprehend the technical matters of this work; thank you for all your support.

Glossary

BCI: Brain-Computer Interface.

MI: Motor Imagery.

MI-BCI: Motor Imagery-based Brain-Computer Interface

EEG: Electroencephalogram.

MEG: Magnetoencephalography.

fMRI: functional Magnetic Resonance Imaging.

fNIRS: functional Near-Infrared Spectroscopy.

ECoG: Electrocorticogram.

ALS: Amyotrophic Lateral Sclerosis.

SCI: Spinal Cord Injury.

AI: Artificial Intelligence.

FIR: Finite Impulse Response.

IIR: Infinite Impulse Response.

SMA: Supplementary Motor Area.

SMR: Sensorimotor Rhythm.

ERS: Event-Related Synchronization.

ERD: Event-Related Desynchronization.

ERDS: Event-Related Desynchronization and Synchronization.

ERP: Event-Related Potentials.

EPSP: Excitatory Postsynaptic Potentials.

IPSP: Inhibitory Postsynaptic Potentials.

SNR: Signal-to-Noise Ratio.

ADC: Analog-to-Digital Converter.

BW: Bandwidth.

CMRR: Common-Mode Rejection Ratio.

PSD: Power Spectral Density.

SSVEP: Steady-State Visually Evoked Potential.

eGUI: experimental Graphical User Interface.

ICA: Independent Component Analysis.

CSP: Common Spatial Patterns.

STFT: Short-Time Fourier Transform.

DFT: Discrete Fourier Transform.

PDF: Probability Density Function.

LDA: Linear Discriminant Analysis.

SVM: Support Vector Machines.

RF: Random Forest.

ANN: Artificial Neural Network.

CNN: Convolutional Neural Network.

SGD: Stochastic Gradient Descent.

TN: True Negative.

TP: True Positive.

FN: False Negative.

FP: False Positive.

Index

1. Preface

1.1. Origin of the project

The brain is constantly controlling all movements that an individual is planning to carry out; however, it is impossible to execute them if the peripheral nervous pathways are damaged. This project started to find out how to keep using the signaling generated by the motor cortex to extract useful information and perform an activity (control a wheelchair, etc.), thus enhancing the life quality of people with paralysis.

1.2. Motivation

Not being able to move nor interact with one's surroundings can devastate the life of patients suffering from disabilities such as quadriplegia. Based on personal experiences from paralytic interviewees [1], the hardest difficulties they face are *"the loss of independence and the struggle to overcome physical barriers in their environment."*. After learning from those testimonials and empathizing with their stories, the author of this thesis wanted to alleviate their situation.

Furthermore, the foundation and milestones of *Neuralink*, a neurotechnology company, increased the enthusiasms of the author in the area of Brain-Computer Interfaces, which made him see it as a viable option for aiding people with spinal cord injuries.

1.3. Previous requirements

For achieving the objectives of this thesis, the author must have advanced knowledge in the following fields:

- <u>Biomedical Engineering:</u> To understand the functioning of the biomedical devices used to collect data from the brain (electroencephalogram). To comprehend the physiology of the motor cortex. To get familiarized with different techniques used to preprocess biomedical signals.
- <u>Electronics and Automatic Control Engineering:</u> To correctly implement a digital signal processing algorithm. To understand and design digital filters and interpret Bode magnitude and phase plots.

- Artificial Intelligence: To understand Machine and Deep Learning architectures for creating, designing, and optimizing different classification models.

Hence, this 48 ECTS project covers the competencies of the Bachelor's degree in Biomedical Engineering and the Bachelor's degree in Industrial Electronics and Automatic Control Engineering. Some of the subjects that were useful for developing this thesis are:

- Statistics (ES – 820002, EIA and BIO).
- Digital Electronics (ELDI – 820224, EIA).
- Analogue Electronics (EAEIA – 820222, EIA).
- Control Techniques (TCEIA – 820230, EIA).
- Industrial Computer Science (IIEIA – 820226, EIA).
- Biology (BB – 820021, BIO).
- Physiology (FIB – 820026, BIO).
- Biomedical Signal Processing (PSB – 820027, BIO).
- Sensors and Signal Conditioners (SCSB – 820030, BIO).
- Monitoring, Diagnostic and Therapeutic Equipment (EMDTB – 820025, BIO).
- Biostatistical Learning (AB – 295601, BIO).

2. Introduction

Spinal cord injuries (SCIs) occur when there is significant damage to the nerve pathways that interconnect the brain with peripheral nervous systems, impeding the correct transmission of neural impulses [2]. If the harm is severe, SCIs can produce a complete movement loss, resulting in paraplegia or tetraplegia. According to the World Health Organization (WHO), there are from 250 thousand to 500 thousand new patients worldwide suffering from SCIs yearly who often need constant assistance from familiars, friends, or caretakers [3]. These concerning numbers make it essential to seek a technology that could allow them to be more independent.

A Brain-Computer Interface (BCI), also known as Brain-Machine Interface, is a system that could efficiently aid these patients. It brings together hardware and software for capturing, preprocessing, extracting, and interpreting valuable features from brain signals with the purpose of controlling a device, for instance, a robotic prosthesis [4]. In other words, BCIs provide a direct communication pathway between the brain and a computer regardless of the presence of a SCI or a neurodegenerative disease such as amyotrophic lateral sclerosis (ALS) [5].

Numerous experiments have been implemented to prove the reliability and accuracy of BCIs when it comes to detecting the movement intentions of an individual, known as motor imagery BCI (MI-BCI). One relevant example is Elon Musk's company Neuralink, which has implanted an array of 1024 miniaturized electrodes into the motor cortex of a monkey and demonstrated that the animal was able to precisely play "Pong" by just imagining hand gestures [6]. However, implanted BCIs are impossible to implement in a final degree thesis due to their high complexity and the nonexistent publicly available signal data. Furthermore, this advanced technology is extremely dangerous since it needs surgery and hardly gets FDA approval.

As an alternative, an electroencephalogram (EEG) can be used since the electrodes are superficially attached to the person's scalp. Nevertheless, non-intrusive BCIs present drawbacks that must be addressed for proper system operation. First and foremost, the signals captured from an EEG have an amplitude of microvolts, which make it highly noise susceptible, especially if there are no active electrodes [7]. Secondly, it is challenging to find the source of the neural activity if there is not a sufficient number of electrodes [4]. Thirdly, owing to the poor conductivity of the skull, there is a substantial spatial resolution loss [8]. Lastly, since the electrodes are manually placed, their position and quantity of hydrogel used can slightly vary from session to session, which can affect the results of the BCI [4]. Sophisticated mathematical techniques must be carried out to solve these complications, thus considerably increasing the performance of the system.

2.1. Objectives of the project

The main purpose of this project is to implement a non-invasive MI-BCI, with data captured from an EEG, and design a multitude of algorithms and methods that will be capable to distinguish whether the user has the intention (just by thinking of it) to move the right or the left hand. From the outputs, it is intended to control a robotic arm in real-time.

EEG signals must be understood in the context of MI to painstakingly choose which signal processing procedures are more suitable for better extraction of pertinent features. These characteristics will act as inputs to train various Artificial Intelligence (AI) models that will be evaluated with certain metrics to discriminate which one performs the best. An appropriate tuning and optimization of AI models are of great importance in this work.

Parallelly, finite impulse response (FIR) filters will be carefully studied and analyzed to exclude insignificant spectral information. Hence, this project also aims to design a digital filter in a real-time operation, considering the pros and cons when working with higher or lower order filters.

Due to the inability of the author to gain access to a high-quality EEG unit with a sufficient number of electrodes, all the objectives proposed will be fulfilled using a large dataset that includes numerous sessions of patients performing MI tasks with an EEG [9].

3. Definitions and basic concepts

In this section, there will be a summary of important notions necessary to understand basic theory and all the procedures done to accomplish the goals of this thesis. Initially, there will be an explanation of the nervous system to explain how the brain works. Subsequently, the electroencephalography technique is described to comprehend how the signals are collected with the best quality possible. The chapter will end by introducing BCIs.

3.1. Brain physiology

3.1.1. The neurons

The nervous system is essential for the correct functioning of the human body thanks to its ability to rapidly control, regulate, sense and communicate a wide range of biological processes, such as maintaining homeostasis or activating muscular fibers [10]. All of this is achieved mainly with specialized cells that respond to the signals of the extracellular space: the **neurons**.

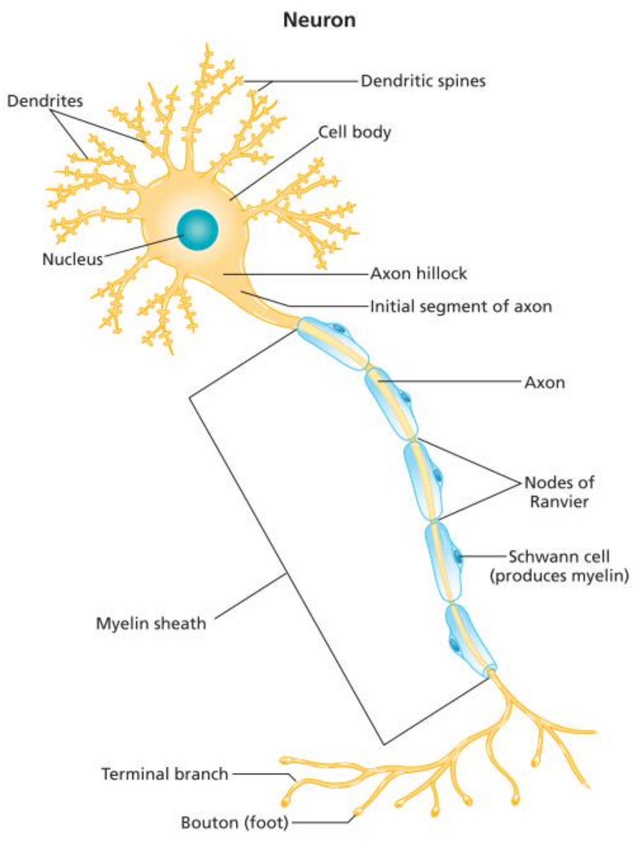


Figure 3.1.- The structure of a neuron. Figure extracted from [11].

These cells constitute the functional part of the nervous system as they can transmit information to other neurons/cells by producing action potentials and conducting them through its membrane. In this manner, other interconnected neurons may be triggered or inhibited, causing a cascade of neural impulses to achieve the desired action [12].

Neurons have distinct parts (Figure 3.1.) that characterize them and play indispensable roles in the well-functioning of the cell. The **cell body**, or soma, contains the nucleus and most organelles, such as the mitochondria. It is the control center of the neuron, where the metabolism and the nerve impulses are regulated, and most proteins are synthesized. The **dendrites** are responsible for receiving information from outside the neuron and transmitting it to the cell body. Notice that these structures usually have many branches and subbranches (dendritic spines) because a nerve cell can communicate with thousands of other neurons. The **axon** is the elongated component of the neuron that carries the triggered impulses from the soma to the **axon terminal**, where many **neurotransmitters** are produced, stored in vesicles, and released [11][13].

It is important to mention that the axon terminals meet with the dendrites of other neurons in the **synaptic cleft**. The synapse is a gap of approximately 20 nm in which chemical molecules (neurotransmitters) travel from the presynaptic to the postsynaptic neuron (target cell) to excite or inhibit it.

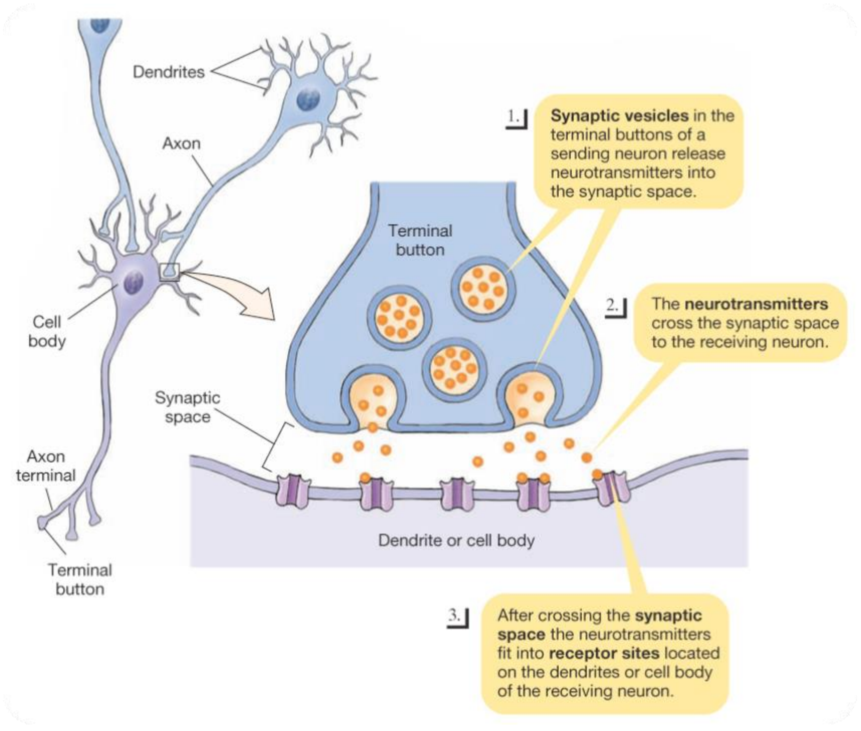

Dendrites
Axon
Cell body
Synaptic space
Axon terminal
Terminal button
Terminal button
Dendrite or cell body
1. Synaptic vesicles in the terminal buttons of a sending neuron release neurotransmitters into the synaptic space.
2. The neurotransmitters cross the synaptic space to the receiving neuron.
3. After crossing the synaptic space the neurotransmitters fit into receptor sites located on the dendrites or cell body of the receiving neuron.

Figure 3.2.- Transmission of neurotransmitters between two neurons through the synapse. Figure extracted from [14].

There can be up to ten thousand neurons types, which can be classified based on their structure or function [11]. In Figure 3.3. it is possible to distinguish the different morphologies that these cells can have: **unipolar**, **pseudounipolar**, **bipolar**, and **multipolar**. Regarding its functionality, there are **sensory neurons**, which are activated by stimuli from the environment; **motor neurons**, which send commands to move muscles or secrete substances from glands; and **interneurons**, which transmit nerve impulses exclusively to other neurons.

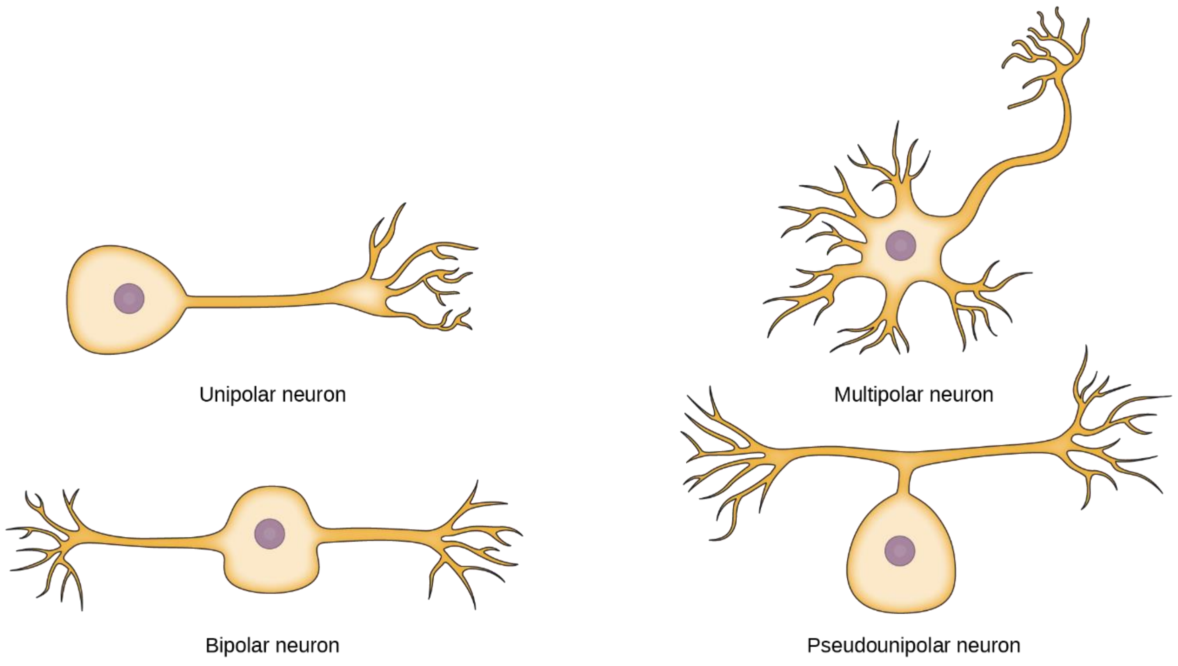


Figure 3.3.- The different structure types of neurons. Figure extracted from [15].

The **glia** are another group of crucial cells in the nervous system that are in charge of providing the structure of the nerves together with supporting, isolating, protecting, and nurturing the neurons. For instance, some neuroglia such as the **oligodendrocytes** and **Schwann cells** are in charge of sheathing regions of the axon with **myelin**. By doing so, the uncovered parts, known as the **nodes of Ranvier** (Figure 3.1.), enable a faster and further propagation of the action potential [16]. Other glia cells are **astrocytes**, **ependymal cells**, and **microglia**.

3.1.2. The action potential

The membrane of a neuron plays a fundamental in encapsulating most cell substances and interchanging ions with the extracellular fluids to achieve polarization. In the lipidic bilayer of the membranes, which insulates the cells thanks to its hydrophobic properties, there are plentiful incrusted proteins that act as channels or pumps to enable passive or active transportation of ions, especially potassium (K^+), sodium (Na^+), calcium (Ca^{2+}), and chloride (Cl^-).

When the cell is not excited, there is a higher concentration of K^+ ions inside than outside the cell; the opposite effect happens with Na^+ and Cl^- ions. This result is achieved thanks to mechanisms such as the sodium-potassium pump that ejects three Na^+ ions per two K^+ ions injected inside the cell. The procedure requires energy since there is a natural tendency for K^+ ions to leave the cell through the potassium channels due to the diffusion gradient caused by a concentration imbalance. Consequently, the interior of the cell becomes negatively charged relative to the extracellular fluid with a voltage drop that typically ranges from -60 to -70 mV, called the **resting potential** [13][17].

In contrast with other cells, neurons can change their resting potential and trigger an electric impulse that travels through their membrane. The process starts when neurotransmitters released from the presynaptic neurons interact with proteins found in the dendrites of the target cell, allowing the activation of ionic channels and a slight change in the membrane potential. All the potential changes, which can be excitatory or inhibitory, arrive at the **axon hillock** and will be summed up to check if the voltage level surpasses the threshold (usually between -50 and -55 mV) for generating the **action potential**. If the charge summed comes from different synapses, there is a **spatial summation,** and if it is added from a train of fast pulses from the same synapse, there is a **temporal summation** [11][13][18].

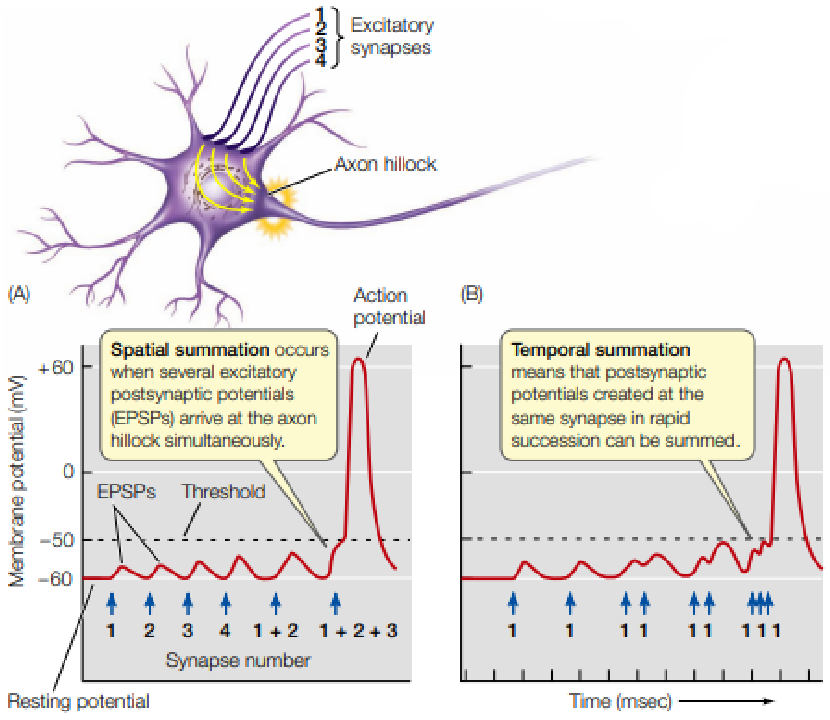

1
2
3
4
Excitatory synapses
Axon hillock
(A)
Action potential
(B)
Spatial summation occurs when several excitatory postsynaptic potentials (EPSPs) arrive at the axon hillock simultaneously.
Temporal summation means that postsynaptic potentials created at the same synapse in rapid succession can be summed.
Membrane potential (mV)
+60
0
–50
–60
EPSPs
Threshold
1 2 3 4 1 + 2 1 + 2 + 3
Synapse number
1 1 1 1 1 1 1 1 1 1
Resting potential
Time (msec)

Figure 3.4.- This figure illustrates how an action potential may be triggered in the axon hillock. And also shows how the potentials are summed. Figure extracted from [13].

When the threshold gets exceeded, a massive opening of the **activation gate** of voltage-gated sodium channels quickly **depolarizes** the neuron, making the inside positive relative to the outside. This spike is counteracted rapidly by the closure of the **inactivation gate** of the sodium channels and the opening of voltage-gated potassium channels. Notice that sodium channels, as well as other ones, have two controllable gates. On the one hand, the activation gate is usually closed and only opens when the neuron is excited. On the other hand, the inactivation gate is always opened and only closes for a brief period after the action potential.

While the inactivation gates of the sodium channels remain closed before naturally opening again, an **absolute refractory period** occurs to transmit the potential just in one direction. Subsequently, the potassium channels close, and the membrane potential **hyperpolarizes**, which means that the voltage gets lower than the resting potential, making it harder to fire another impulse (**relative refractory period**). In this way, the action potential travels until the axon terminal, where it activates voltage-gated calcium channels that enable the entrance of Ca2+ ions to liberate neurotransmitters deposited in vesicles to the synaptic cleft. Eventually, the neuron reaches the resting potential again [11][13][19][20].

The following illustrations summarizes all the concepts explained visually so that the reader can understand the process with more clarity:

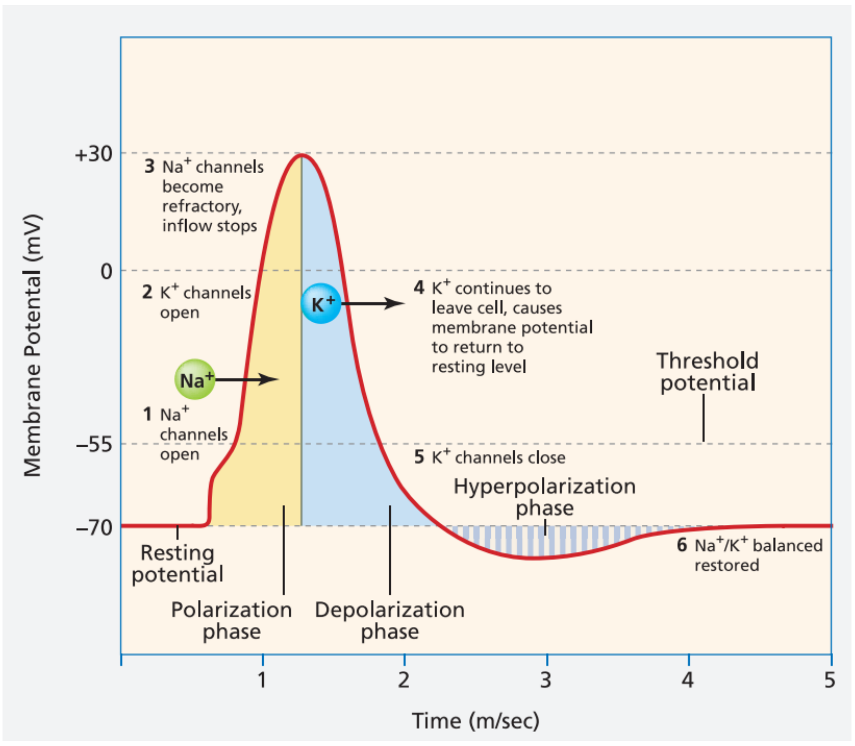


Figure 3.5.- The action potential. This figure illustrates the procedure in which an action potential occurs. Figure extracted from [11].

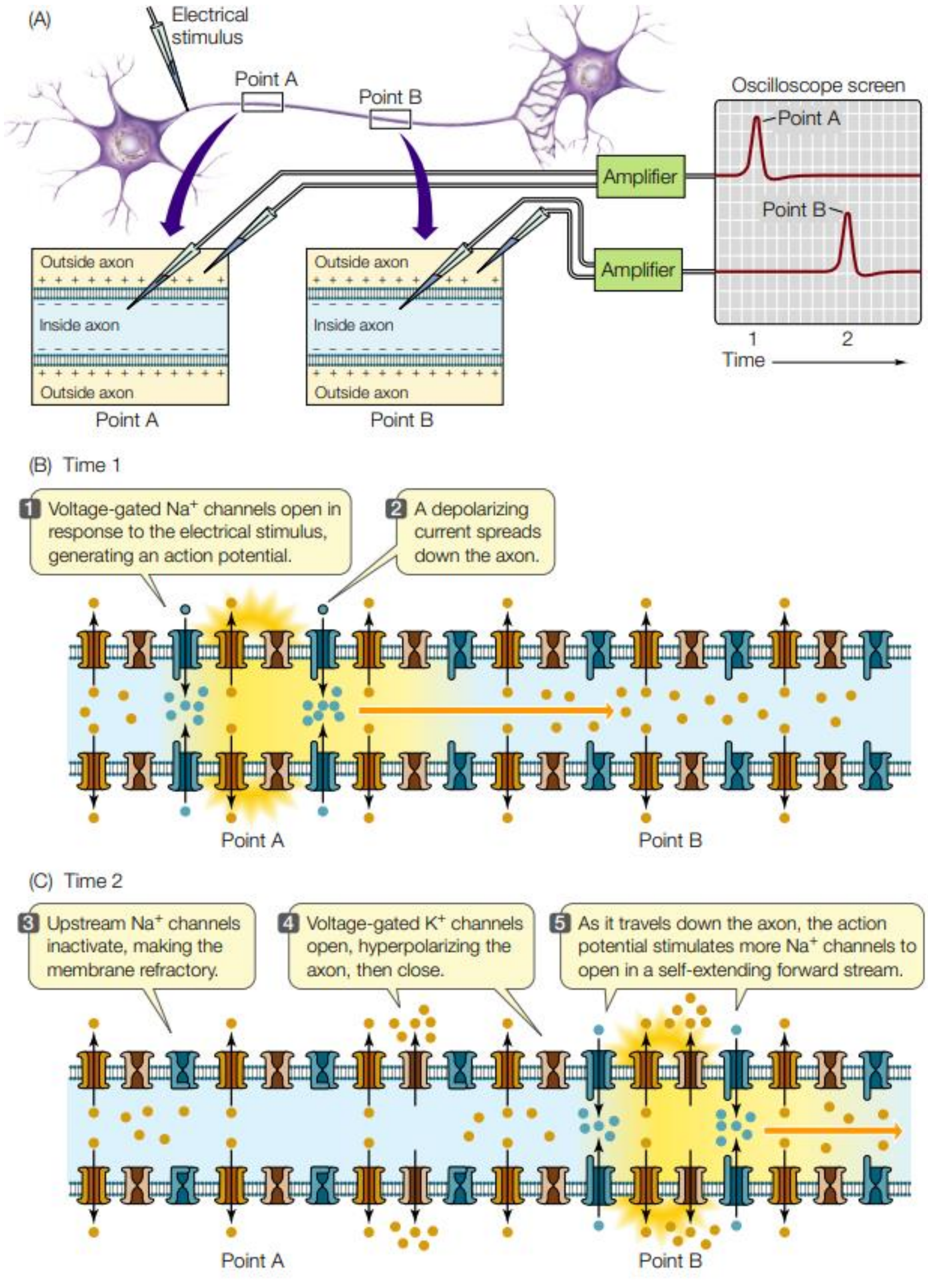

(A)
Electrical stimulus
Point A
Point B
Oscilloscope screen
Point A
Point B
Amplifier
Amplifier
Outside axon
Inside axon
Outside axon
Point A
Outside axon
Inside axon
Outside axon
Point B
1
2
Time
(B) Time 1
1 Voltage-gated Na+ channels open in response to the electrical stimulus, generating an action potential.
2 A depolarizing current spreads down the axon.
Point A
Point B
(C) Time 2
3 Upstream Na+ channels inactivate, making the membrane refractory.
4 Voltage-gated K+ channels open, hyperpolarizing the axon, then close.
5 As it travels down the axon, the action potential stimulates more Na+ channels to open in a self-extending forward stream.
Point A
Point B

Figure 3.6.- The action potential propagation. The behavior of the voltage-gated sodium and potassium channels when stimulated electrically. Figure extracted from [13].

3.1.3. Brain structures

The brain is the center of control of the organism, where millions of neurons organize themselves into dense and complex networks to get specialized in multiple tasks. This organ is responsible for processing stimuli, executing voluntary and involuntary actions, developing feelings and consciousness, among many others. It is distributed in four major subgroups: the **brainstem**, the **cerebellum**, the **diencephalon**, and the **cerebrum** [11][13][21].

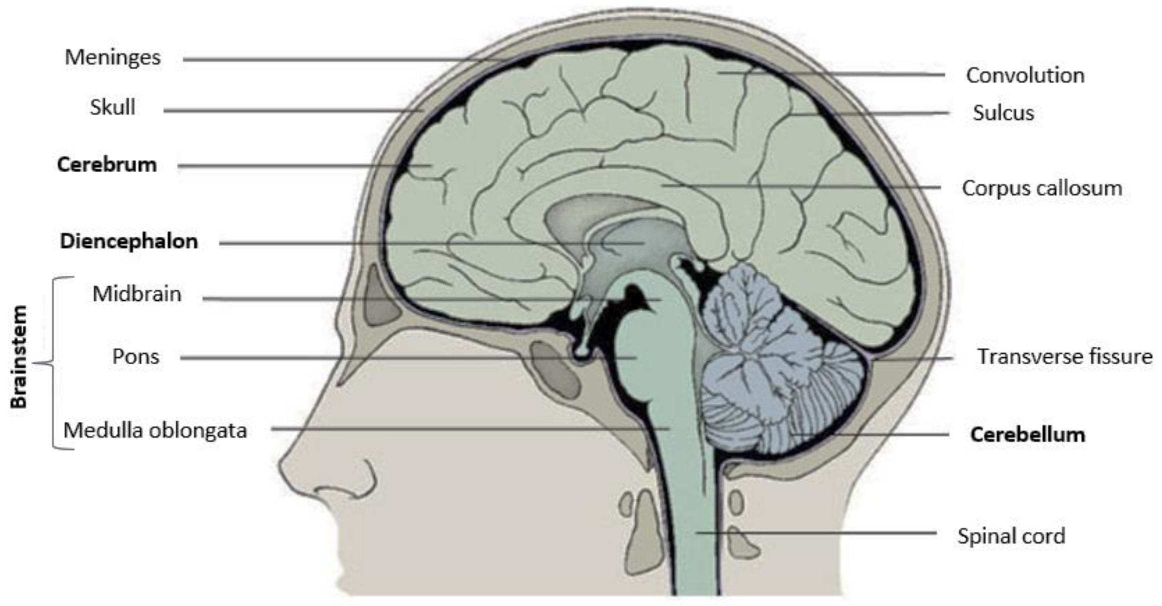


Figure 3.7.- Major portions of the brain. Figure extracted from [22].

The brainstem is the intermediary between the spinal cord and the other brain structures. It controls the information that enters and exits the brain, and, additionally, it is in charge of many physiological processes such as respiration, heartbeat, digestion, sternutation, swallowing, etc. This region is subdivided by the **medulla oblongata**, the **pons**, and the **midbrain** [23].

The cerebellum is located behind the brainstem and below the diencephalon. It is needed to process the commands going to muscles as well as the information coming from them; therefore, it synchronizes movements and upholds the body posture and balance [24].

The diencephalon is positioned underneath the two hemispheres of the cerebrum, at the brain's core. Fundamentally, it consists of the **thalamus** and the **hypothalamus**. The former is indispensable in transmitting and relaying sensory stimuli to the cortex [25]. The latter principally manages the production and secretion of hormones [26].

Last but not least, the cerebrum, also known to as the **telencephalon**, is the largest and uppermost part of the brain. It starts around the diencephalon, where the **limbic system** is found, which involves emotions, behavior, memory, arousal, fear, learning, space perception, navigation, among others [13]. Furthermore, the telencephalon presents two hemispheres that are connected by the **corpus callosum**. On the surface of the cerebrum, the **brain's cortex** is present, in which rugosities and sulcus are abundant. This superficial layer has a greyish color due to the limited myelin surrounding their neurons. Beneath this layer, there is a white matter that mainly connects the different parts of the cortex. The whiteish color derives from the myelin sheaths that enable a faster and further nerve impulse [27]. The cerebral cortex possesses four lobes on each halve of the brain [13]:

- **Frontal lobe**: This region, located in the front of the cerebrum, allows the planning of future events and actions, permits self-recognition, and defines the personality and social skills of the individual. With the help of the limbic system, it also controls sentiments and attentiveness. Additionally, the **Broca area** of this lobe has a relation with language expression. Lastly, the posterior zone of the lobe englobes the **premotor cortex**, the **supplementary motor area** (SMA), and the **primary motor cortex**, which altogether contribute to the preparation and execution of simple or complex movements [28].
- **Parietal lobe**: This segment, situated after the central sulcus, behind the frontal lobe, is capable of processing tactile feelings from all the body (in the **primary somatosensorial cortex**), feeling physical pain, detecting the position of the articulations, learning and reading Braille language, and focusing on complex stimuli or tasks. Intelligence, especially in mathematics, is also related to this lobe.
- **Temporal lobe**: It is found in the laterals of the telencephalon, and it receives and deals with sounds. This region enables the person to recognize objects, human faces or voices, animals, or plants and comprehend speeches (in the **Wernicke area**). In addition, it is extremely important for maintaining equilibrium.
- **Occipital lobe**: The lobe is placed at the back of the brain. Its principal function is to handle visual information and perceive motion.

After understanding how the brain functions, it is clear that the motor cortex is decisive for detecting the imagery of left or right-hand movements, which is the purpose of this work. Several studies have shown that motor imagery is linked to the same cortical regions as motor execution, especially in the primary motor cortex [29]–[32]. The specific muscles controlled in this area can be represented through the motor homunculus [33] (Figure 3.8.), thus mapping the source of each motor activity. Notice the enormous hand size, highlighting its abundant neural activity in comparison with other parts of the body. Furthermore, it is interesting to mention that, as happens to the somatosensorial cortex, **the left hemisphere manages the right side of the body and vice versa** [34]; this will be proven in upcoming sections of the thesis.

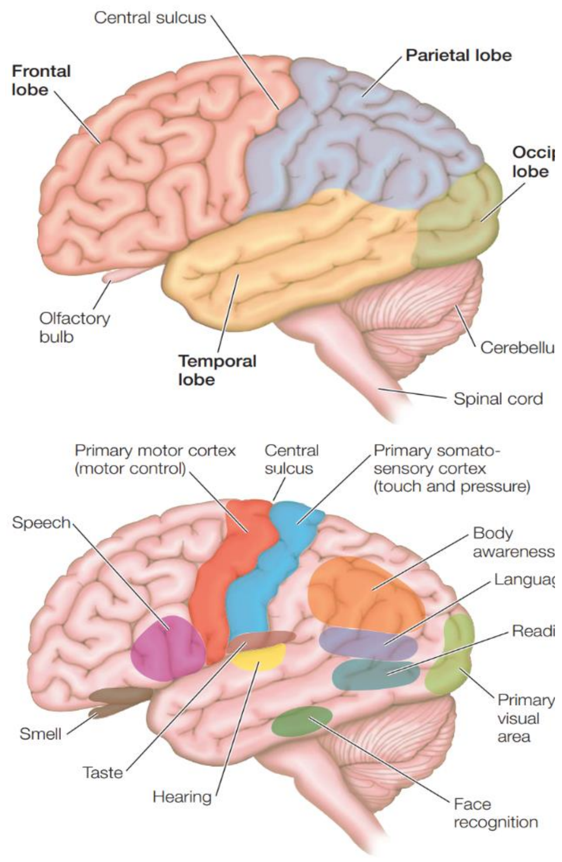


Figure 3.9.- The upper illustration contains a distribution of the different lobes of the brain. The lower one highlights some of the functions of the cortex. Figure extracted from [13].

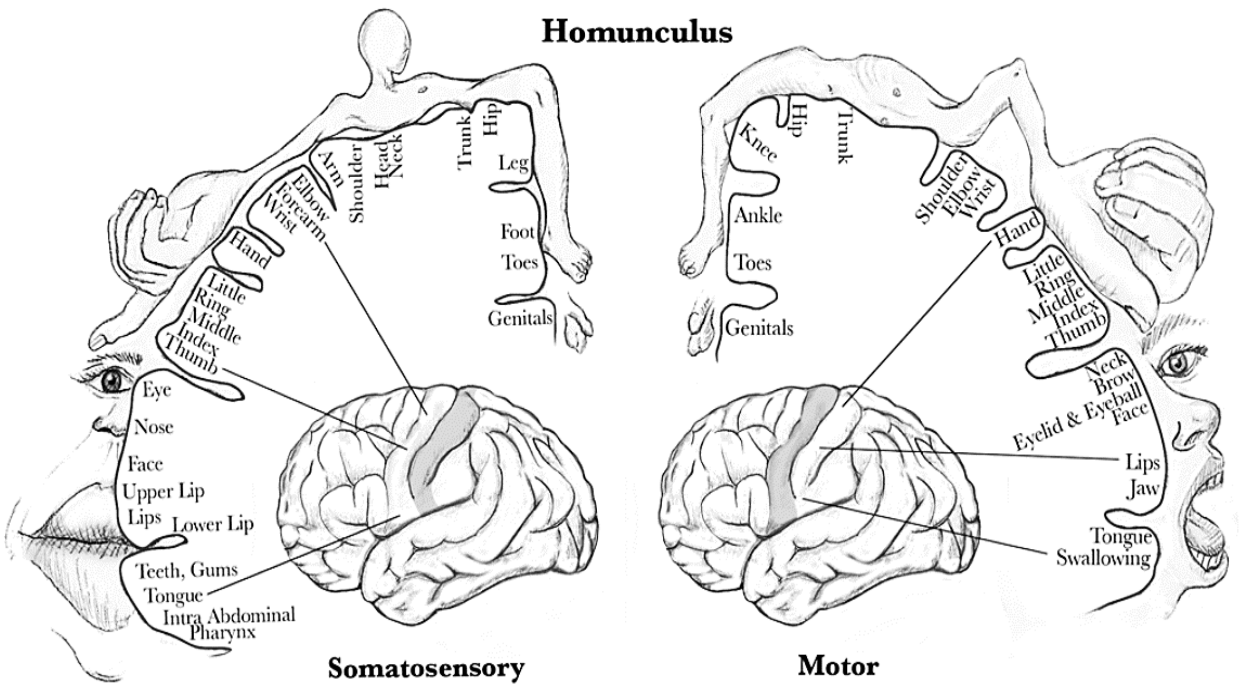


Figure 3.8.- The cortical homunculus. It highlights the relevance of the different parts of the body in the motor and somatosensory cortex. Figure extracted from [33].

3.2. Electroencephalogram (EEG)

The electroencephalogram (EEG) is a non-invasive device used to record the brain's neural activity, especially from the cortex, and provide relevant information for carrying out several applications, like diagnosing medical disorders, evaluating cognitive states, or operating a BCI.

Initially, the biosignals, which are in the orders of microvolts, are captured from an array of electrodes placed meticulously in the patient’s scalp. Subsequently, they must be conditioned, with an amplification and filtration stage, to increase the signal-to-noise ratio (SNR), enhance the signal quality, and utilize the appropriate input range of the analog-to-digital converter (ADC) for maximizing the resolution.

3.2.1. The source of the EEG signal

Pyramidal neurons are widely spread around the cerebral cortex and provide the principal signal of interest acquired with the EEG. These elongated cells, which have a pyramidically shaped soma and a multipolar morphology, are critical for the functioning of many cognitive processes [35].

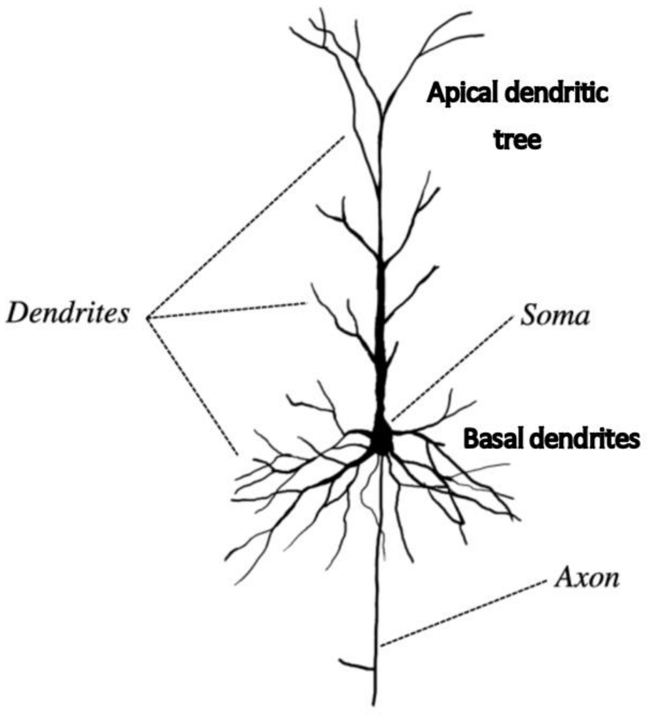


Figure 3.10.- Different parts of the pyramidal neuron. Figure extracted from [36].

Their multiple dendritic trees interact with other neurons that may trigger **excitatory postsynaptic potentials** (EPSPs) or **inhibitory postsynaptic potentials** (IPSPs), which will change the ionic charge of the cell and its surroundings. If it is excitatory, the extracellular fluid in the active synaptic areas will become more negative since the neuron depolarizes. As a result, this local exterior area is negative relative to other extracellular zones of the neuron. If it is inhibitory, the same effect occurs, but with inversed polarity because the neuron hyperpolarizes. In other words, either potential will initiate an extracellular charge imbalance, producing a **dipole** [37].

In the following image, there are two neurons forming dipoles. The left neuron could have gained that polarity either by EPSPs in the top dendrites (**apical dendrites**) or by IPSPs in dendrites close to the pyramidal body (**basal dendrites**). Regarding the right neuron, the polarity could have been achieved by IPSPs in apical dendrites or EPSPs in basal dendrites. The circles around each neuron represent an electrode, with the sensed signal deflection drawn inside it.

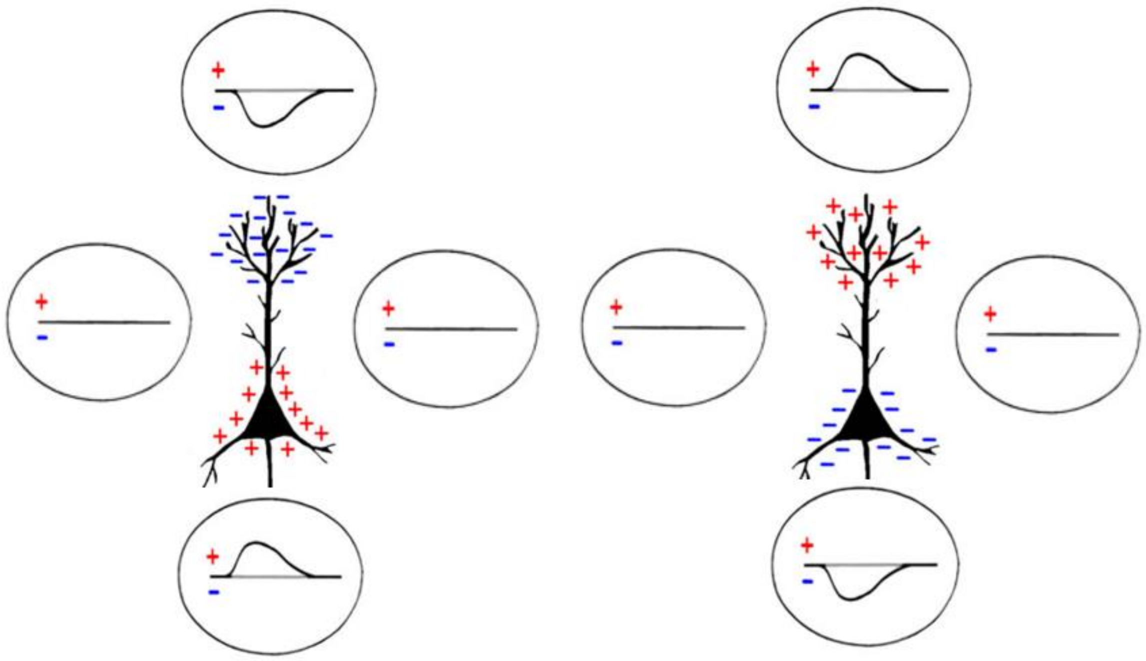


Figure 3.11.- Dipoles formed in pyramidal neurons. Figure extracted from [37].

However, it is impossible to measure the deflection of just one neuron's dipole with an EEG; the signal would be undetectable due to its low amplitude and the attenuation with other biological tissues. Hence, the EEG will perceive the cumulative dipole effect of numerous neurons. For optimal detection, neurons should be parallel, fire synchronously, and possess the same polarity. Otherwise, the net sum of all the dipoles will tend to be null [37][38].

Once the dipoles originate, there is a transmission of electric and magnetic fields through the brain tissue in a process known as **volume conduction** [39]. This effect provokes a repulsion of ions with the same charge until they reach and accumulate at the edges of the conductive volume (the brain). Now, the signal must overcome multiple layers that will act as conductive volumes separated by insulators (operating as dielectrics), such as the dura, the skull, the scalp, the electrolytic gel, and the electrode. Therefore, for going from one layer to the other, the potentials will be transmitted capacitively (**capacitive conduction**), thus arriving at the electrode [37].

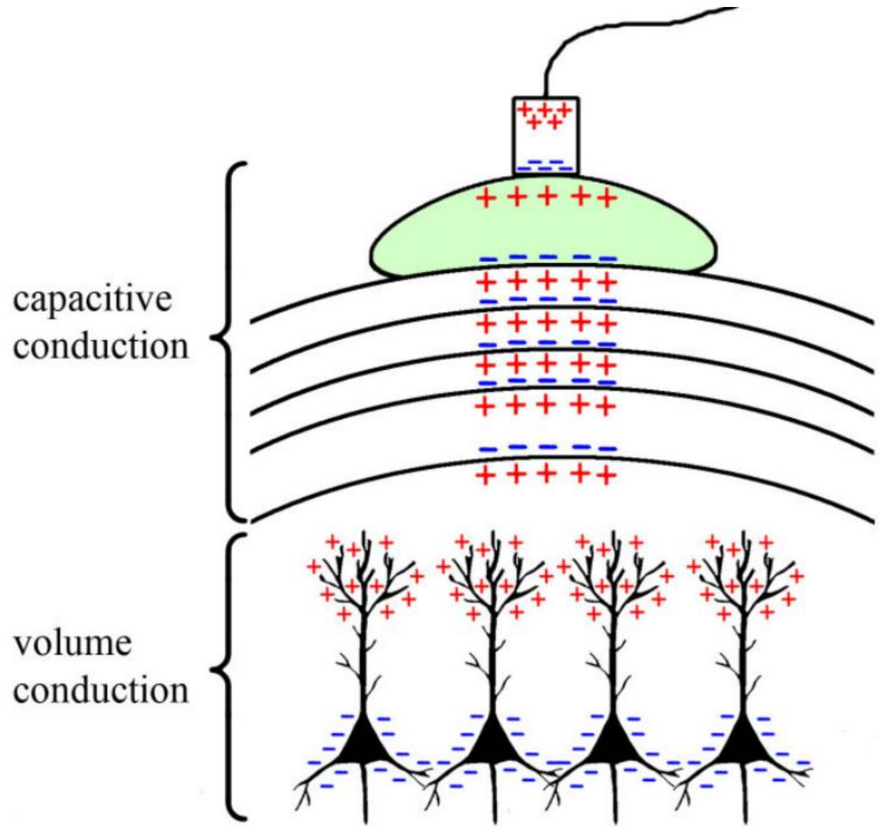


Figure 3.12.- Capacitive and volume conduction. Propagation of a signal from the pyramidal cells to the electrode. Figure extracted from [37].

All this process of signal propagation presents clear limitations. Firstly, a dipole emanates an electric field in most directions (Figure 3.13.), thus gradually spreading the scalp's charge in the surrounding areas located above the intracranial source. This behavior, referred to as spatial smearing, is aggravated by the volume conduction of the tissues (especially the skull) and is the principal reason for the EEG's low spatial resolution [40]. Another problem is the interaction of groups of dipoles positioned in different parts of the brain, which may alter the expected reading from the desired source [41]. Lastly, the capacitive effect and the soft tissues of the stacked layers attenuate cortical activity with frequencies higher than 30 Hz, restricting the bandwidth (BW) of the signals [42].

These issues can be mitigated if, instead of an EEG, an electrocorticogram (ECoG) is used. These devices serve the same function as an EEG, but they are implanted surgically inside the skull. Since their electrodes are in direct touch with the cerebrum, this apparatus enhances the spatial resolution, reduces sources interactions, and significantly increases the BW [43]. Nevertheless, the invasiveness of the ECoG makes this technology risky to use and less appealing to the user.

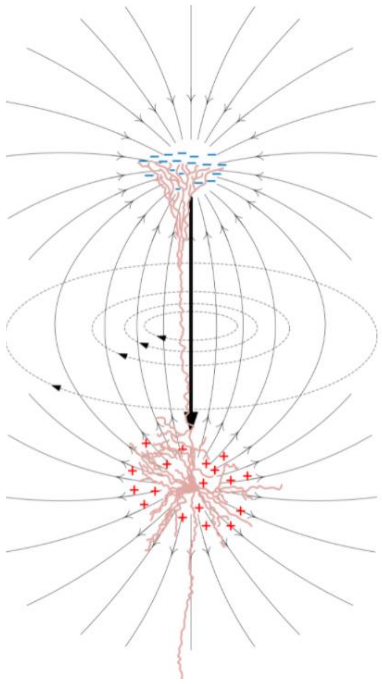


Figure 3.13.- Magnetic field (dashed line) and electric field (solid line) of the dipole produced in a pyramidal cell. Notice that the electric field emanates to all directions. Figure extracted from [44].

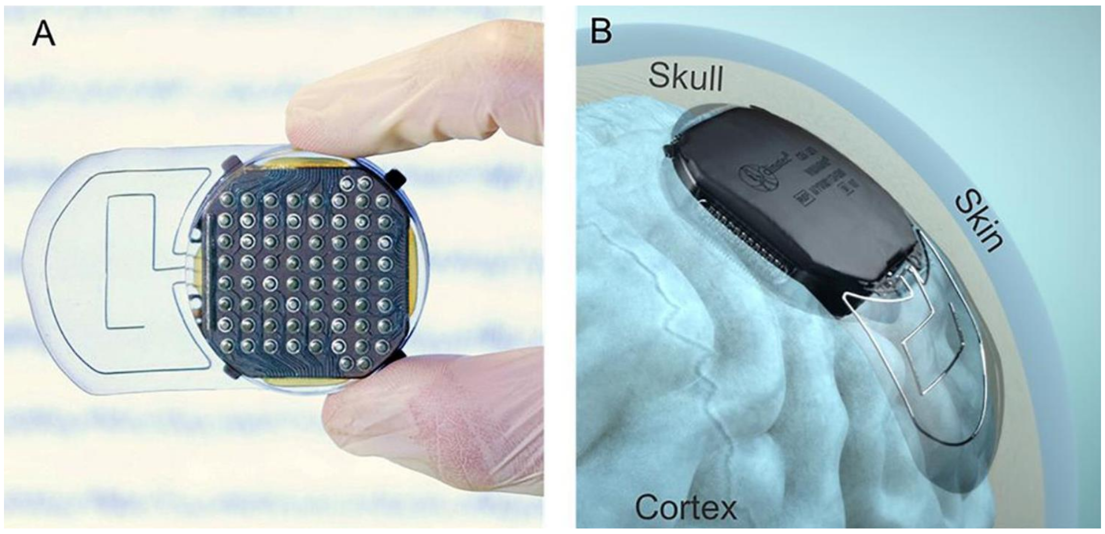


Figure 3.14.- ECoG with 64 channels. Picture A shows the electrodes of the device. Illustration B displays how the device is implanted. Figure extracted from [43].

3.2.2. Technical characteristics

The internal electronics of an EEG device must be designed adequately to acquire relevant signals with the maximum quality possible. Therefore, the user must understand key technical characteristics that differentiate a first-class EEG from a mediocre one. In this section, there will be a brief introduction of meaningful features that should not be overlooked, especially if the application is a MI-BCI. Most of this information will be extracted from an article written by Bitbrain [45], a neurotechnology company that, among other things, develops competent EEG equipment.

3.2.2.1. Sampling frequency and filters

The sampling rate is the number of samples that an ADC is capable of capturing in a certain time lapse, usually expressed in Hertz (Hz). While working with any signal that requires a digitalization process, it is extremely important to take into consideration the **Nyquist-Shannon theorem**, which states that the sampling frequency of the converter must be equal or superior to the double of the highest frequential component of the signal:

$$\boldsymbol{f_s \geq 2 \cdot f_{MAX}} \qquad \textbf{(Eq. 3.1)}$$

Where $\boldsymbol{f_s}$ is the sampling frequency and $\boldsymbol{f_{MAX}}$ is the maximum frequency of the signal. If this condition is not met, it is impossible to reconstruct the analog signal again due to an undesired effect referred to as **aliasing**. The following graphic illustrates this phenomenon, where the sampling frequency (black dots) is smaller than twice the frequency of the signal (red); hence, the reconstructed signal (blue) does not match with the original one (red).

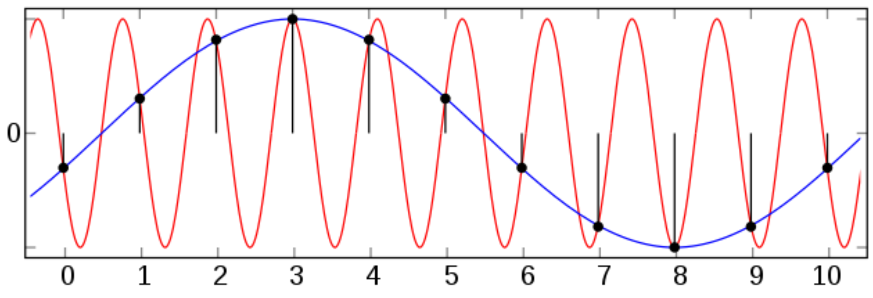


Figure 3.15.- Aliasing effect produced by choosing a wrong sampling rate. Figure extracted from [46].

Since most EEG signals cannot reach frequencies greater than 80 Hz, the sampling rate should be at least 160 Hz. However, the sampling frequency is usually set to a higher value for better rebuilding, typically 256 Hz. If the rate is too large, the system will have to process much more data points that may not be necessary.

Lastly, it is important to remark that the EEG will have a low-pass filter, or **anti-aliasing filter**, to restrict the maximum frequency of the signal and satisfy the Nyquist criterion. Additionally, there is a high-pass filter to eliminate the DC component, thus erasing the offset and avoiding a possible saturation of the electronic components.

3.2.2.2. Resolution

The ADC carries out the digitalization process of the analog signal, translating voltage intervals into binary code. For narrowing the intervals and having a more defined output two factors must be taken into consideration.

First and foremost, the number of bits available in the converter. This parameter specifies the resolution of the ADC and, therefore, how many unique binary outputs there are, which can be calculated using the following formula:

$$\boldsymbol{N = 2^b} \qquad \textbf{(Eq. 3.2)}$$

Where $\boldsymbol{N}$ is the number of quantization levels and $\boldsymbol{b}$ the bits of the ADC. Good quality EEGs usually have a 24 bits ADC, providing around 16 million quantization levels; however, 16 bits is also acceptable.

The other important aspect is the overall voltage measurement range that can be introduced to the ADC, which will define the resolution of the acquisition system expressed in volts. Hence, the voltage resolution is described as following:

$$\boldsymbol{Q = \frac{FS}{2^b}} \qquad \textbf{(Eq. 3.3)}$$

Where $\boldsymbol{Q}$ is the resolution (in volts), $\boldsymbol{FS}$ is the full scale voltage range, and $\boldsymbol{b}$ the bits of the ADC.

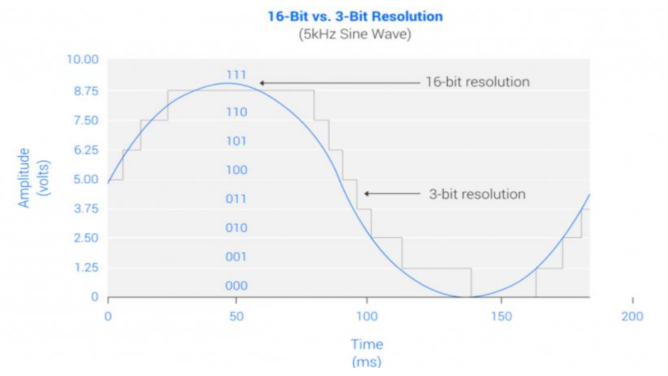


Figure 3.16.- Difference between a 16 bit and a 13 bit resolution ADC with *FS = 10 V*. Figure extracted from [45].

3.2.2.3. Input impedance

When the signal is introduced into the amplifying stage of the EEG, it is of the utmost importance to have the highest input impedance possible. By doing so, the signal of interest is conserved without significant attenuations from other resistive sources of the system, such as the electrode/skin impedance and the wire resistance, since most of the voltage drop will occur on the high input impedance of the amplifiers. Furthermore, this parameter contributes to the maximization of the SNR, tolerating better the electrical noise [37].

According to [47], the skin impedance can reach up to 1 MΩ; therefore, an appropriate input impedance should be not less than one hundred times bigger (100 MΩ).

3.2.2.4. Common-mode rejection ratio (CMRR)

While performing an EEG session, the user may attract unwanted interferences, predominantly from the 50/60 Hz powerline, that can be amplified altogether with the brain's signals. These artifacts are introduced at both of the input terminals of the amplifier, establishing an inconvenient common mode voltage. The common-mode rejection ratio (CMRR) is a property that defines the ability of the amplifier to reject this common mode voltage and magnify the differential mode voltage, as shown in the following illustration:

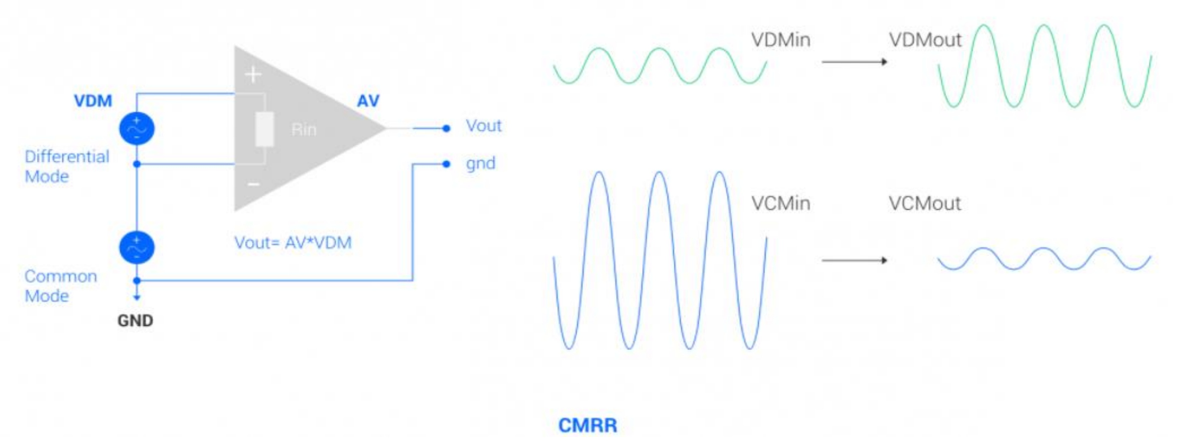


Figure 3.17.- The effect of the CMRR in an amplifier. The green signal represents the differential voltage, which is amplified. The blue signal represents the common voltage, which is attenuated. Figure extracted from [45].

The CMRR can be computed with following formula, expressed in decibels (dB) [48]:

$$\boldsymbol{CMRR\ (dB) = 20 \cdot \log_{10}\left(\frac{A_d}{|A_{cm}|}\right)(dB)} \qquad \textbf{(Eq. 3.4)}$$

Where A_d is the differential gain and A_{cm} is the common-mode gain. In addition, the CMRR also varies depending on the frequency range, decreasing at higher frequencies [49].

Ideally, the CMRR should have an infinite value at all the BW. A CMRR of at least 100 dB (at the powerline frequency) is considered suitable for a commercial EEG.

3.2.3. Electrode configurations

EEG electrodes are primarily responsible for capturing signals from the brain for subsequent conditioning. They are usually made of metals with no reactivity (gold, platinum, silver/silver chloride, etc.) placed on a conductive gel, previously applied to the skin to reduce the electrode-scalp impedance [50].

As it has been remarked during this thesis, EEGs present poor spatial resolution and require advanced signal processing techniques to enhance the signal quality by focalizing the potential fields right above their electrocortical sources, reducing as much as possible the spatial smearing effect. These procedures need a considerable number of electrodes placed all around the scalp; therefore, the more electrodes available, the better the localization of the brain's signals will be. Furthermore, an adequate number and placement of electrodes are crucial when it comes to detecting and erasing problematic artifacts such as blinking, muscular contractions, or cardiac activity [51].

For that reason, there are worldwide accepted standardizations that define how the electrodes should be placed, depending on how many electrodes the EEG has. The **international 10-20 system** is used for 21 electrodes, where there is a 10% or a 20% separation between two adjacent electrodes relative to the total distance of the skull. The electrodes are generally named after the brain lobes located beneath them: **F** and **Fp** electrodes are above the frontal lobe, **T** above the temporal, **P** above the parietal, and **O** in the occipital. **C** electrodes are in the center of the scalp, where the motor cortex is found. **A** electrodes are attached to the earlobes for reference. The numbers' parity indicates the side of the head where they are encountered; even numbers are found on the right hemisphere and odd ones on the left hemisphere. The letter **z** specify that electrode are placed on the midline sagittal plane of the skull. Other international standards, such as the **10-10** or the **10-5 system**, follow the same structure as the **10-20** but use more electrodes [50][52].

It is essential to place the electrodes in the same position between sessions, especially for BCI applications. Thus, the electrodes can be settled with the help of a head cap that will guide the user on the exact location where the electrodes need to be [53]. Additionally, these accessories will decrease the risk of gel dispersion on the scalp, which can be an issue if the hydrogel contacts other electrodes.

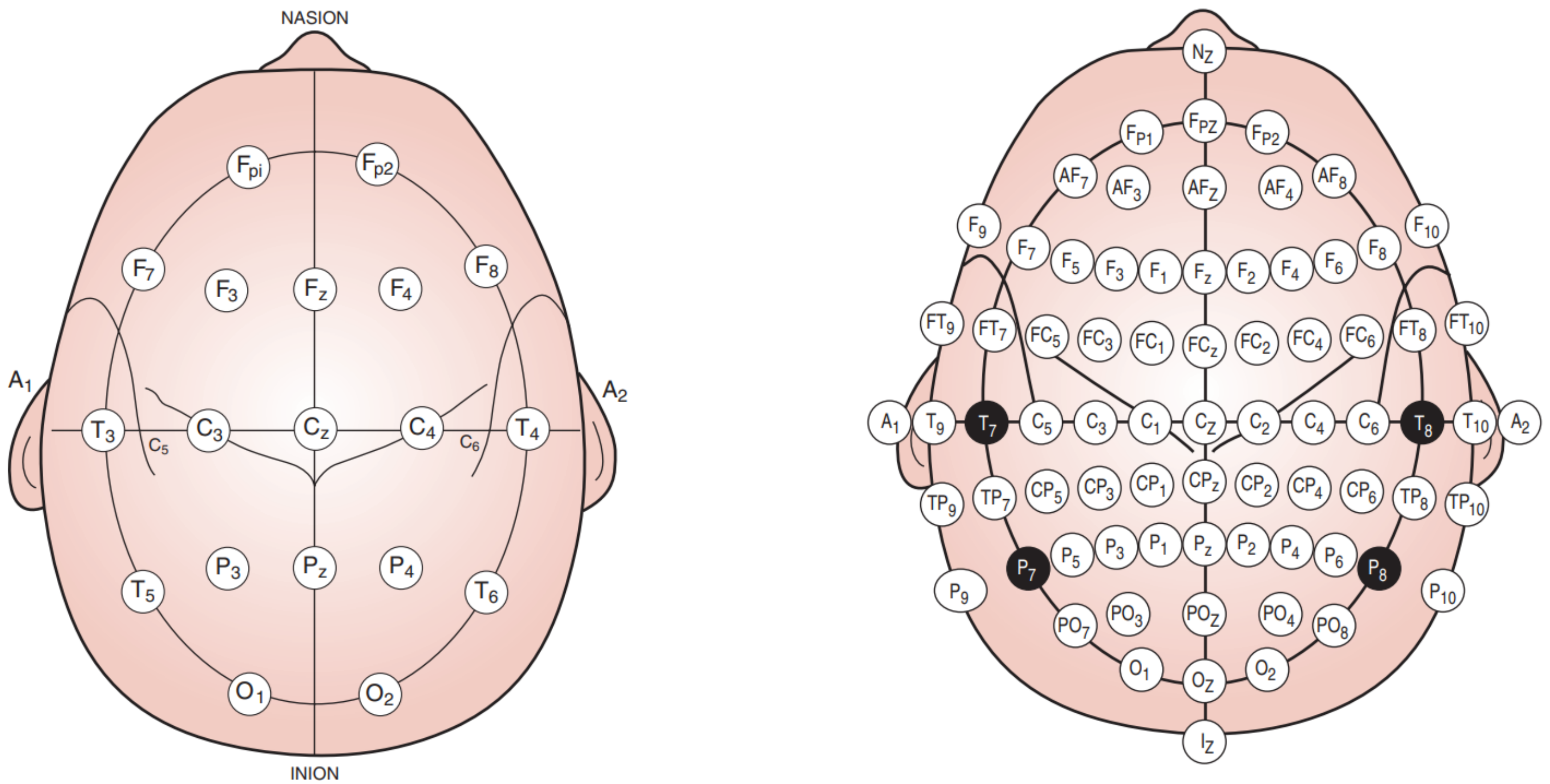


Figure 3.18.- Electrode name and placement standardization. Left image: the international 10-20 system. Right image: the international 10-10 system. Figure extracted from [50].

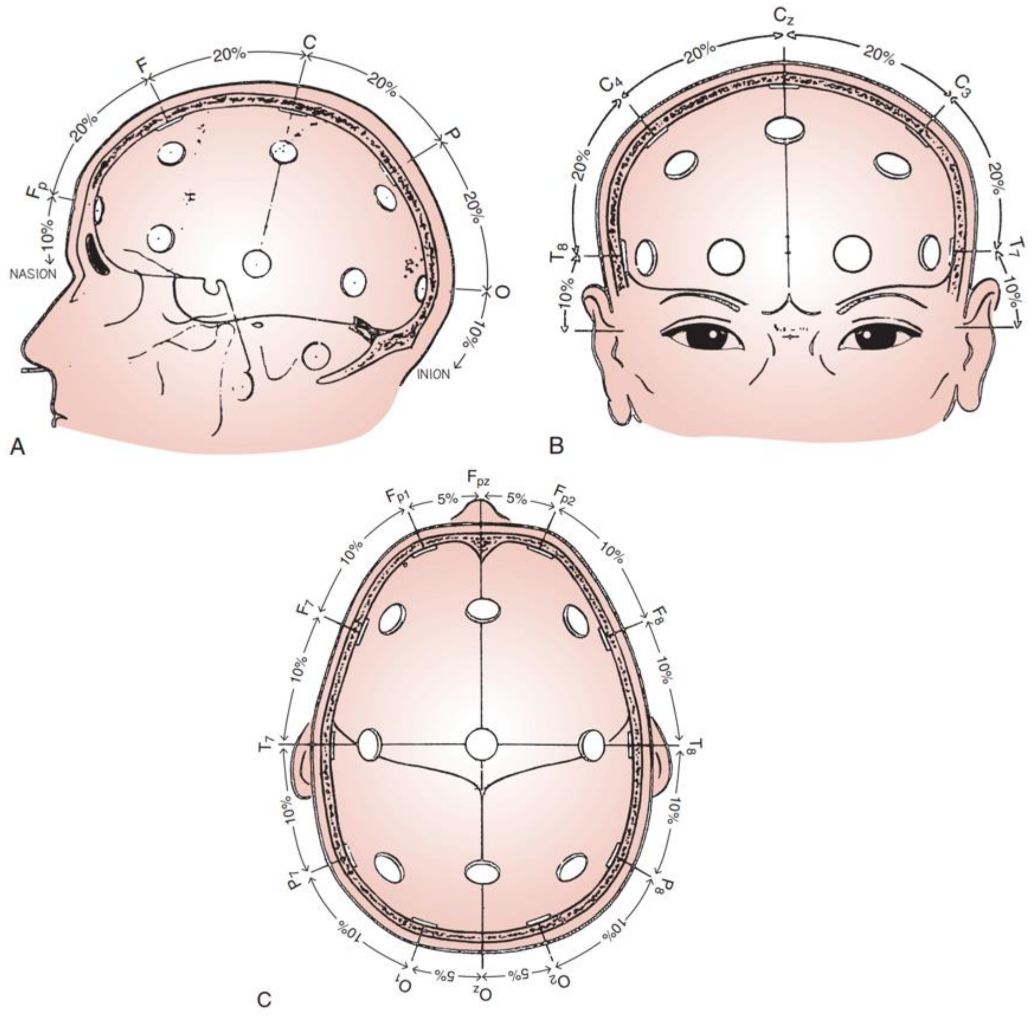


Figure 3.19.- The exact position of each electrode in the international 10-20 system. Sagittal plane (A), coronal plane (B), and horizontal plane (C). Figure extracted from [50].

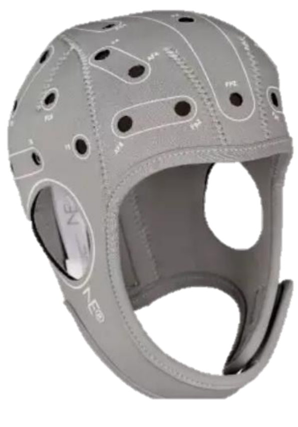


Figure 3.20.- Neuroelectrics EEG head cap. Figure extracted from [53].

3.2.4. EEG frequency bands

The brain is constantly generating waves due to the synchronized firing of cortical neurons, which produces rhythmic patterns that can be detected with an EEG. These neural oscillations reveal characteristic information about the individual's mental state. Brainwaves can be separated into five major frequency bands, which will be rapidly overviewed.

The slowest one is the **Delta** band (0.5 Hz – 4 Hz) which is mainly present in the occipital regions of the EEG when the user is sleeping; in this band, the brain intensifies the healing and restoring process of the body's tissues. The next one is the **Theta** band (4 Hz – 8 Hz) which appears when the person is sleepy. **Alfa** activity (8 Hz – 13 Hz) happens when the person is feeling relaxed and calmed. When the individual starts focusing on a subject, **Beta** waves (13 Hz – 30 Hz) are abundant. Lastly, the fastest oscillations take place in the **Gama** band (> 30 Hz), which involves high concentration and problem-solving [50][54][55].

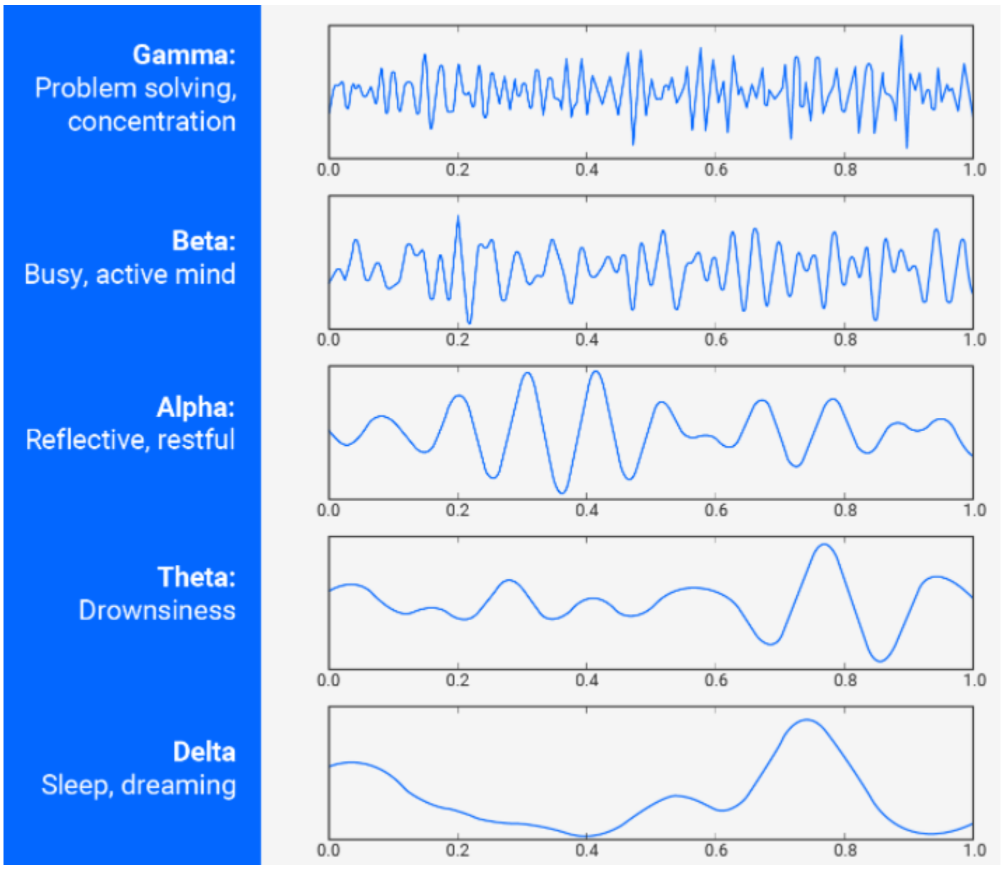


Figure 3.21.- A brief description of the brainwaves. Figure extracted from [56].

3.3. Brain-Computer Interfaces (BCIs)

A Brain-Computer Interface (BCI), as its name suggests, is a system that enables direct communication between the brain and a computer to accomplish the desired goal. In other words, by using BCIs, it is possible to detect and interpret cerebral impulses in real-time without minding if the neural activities reach or not the spinal cord or other peripheral nerves [57]. This technology often requires a cycle of six steps in order to make it function correctly, which are described below [58]:

1) **Signal acquisition of brain activity:** The brain's signals are captured with an appropriate device. The most usual techniques for the acquisition are electroencephalography (EEG), electrocorticography (ECoG), magnetoencephalography (MEG), functional magnetic resonance imaging (fMRI), and functional near-infrared spectroscopy (fNIRS).
2) **Preprocessing:** The acquired signals are refined to attenuate undesired attributes and enhance their quality.
3) **Feature extraction:** The relevant information of the signals (contained inside a time window) is extracted. This step may require a transformation process of the raw data to obtain other values that still describe the initial signals.
4) **Classification:** The features are introduced into a classification algorithm that can distinguish the class type of the original signals, for instance, if the user has imagined a right or a left-hand movement.
5) **Application Interface:** The classifier's output is translated into a command to control another device or software.
6) **Feedback:** The user is constantly identifying how the target device/software is behaving and changes his the mental activity to regulate the overall system.

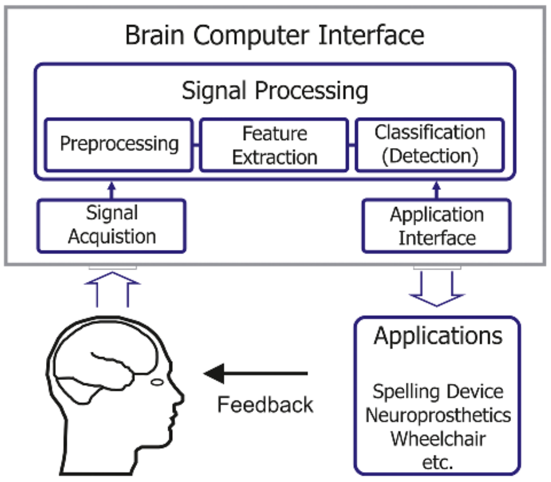


Figure 3.22.- The closed loop system of a BCI. Figure extracted from [59].

3.3.1. Classification of BCIs

BCIs can be categorized depending on various factors that define the nature of the system. In this subsection, the four major BCI classification criteria will be introduced. First and foremost, the BCI can be classified based on how the brain's signal is produced, which can be **active**, **reactive**, or **passive**. The first one implies that the user is the main responsible for performing an intentional mental task for controlling the system, such as imagining the movement of an arm. On the other hand, the reactive BCI needs an external intervention to stimuli the brain in a certain way, evoking a cerebral reaction that can be subsequently detected. If the BCI does not require any action or reaction from the user and only analyzes the brain's activity, it is a passive approach.

Secondly, it can be grouped concerning the period in which the user can interact with the BCI. A **synchronous** BCI does not enable continuous cooperation with the system; the user must wait for an indication to perform the intended mental activity. On the flip side, if the user can carry out the task at any moment that he wishes, it is an **asynchronous** BCI. Synchronous BCIs are commonly employed for creating and evaluating datasets that are posteriorly used to design and implement asynchronous BCIs. It is important to note that asynchronous BCIs need to identify the passive mental state occurring when the user is not performing any action [60].

The third categorization distinguishes between **hybrid** and **non-hybrid** BCIs. It is a hybrid BCI if at least one of the following conditions are met; otherwise, it is non-hybrid:

- The system also uses information of other biosignals, mainly from muscular activity.
- The system uses different BCI techniques simultaneously, for example, a combination of active and reactive BCI.

Finally, as it has been mentioned through the work, the signal acquisition device used in the BCI can be **invasive** or **non-invasive**. The former involves direct implantation of the equipment inside the skull, requiring medical intervention. The latter does not require any surgical procedure.

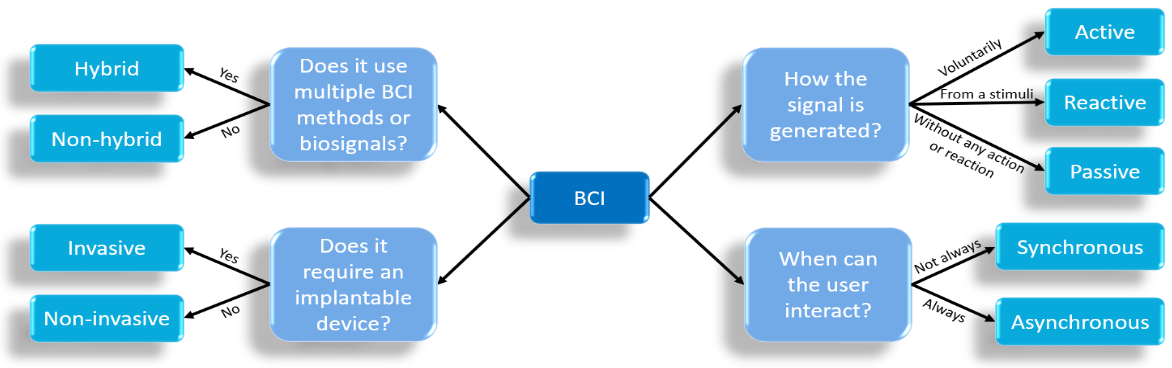

Hybrid
Non-hybrid
Yes
No
Does it use multiple BCI methods or biosignals?
Invasive
Non-invasive
Yes
No
Does it require an implantable device?
BCI
How the signal is generated?
Voluntarily
From a stimuli
Without any action or reaction
Active
Reactive
Passive
When can the user interact?
Not always
Always
Synchronous
Asynchronous

Figure 3.23.- Classification of BCIs. Figure created by the author.

3.3.2. Motor Imagery BCIs (MI-BCI)

Motor imagery (MI) occurs when someone has the intention to perform a voluntary muscular movement but at no moment executes the action; therefore, the individual only imagines the motor activity that wants to perform. Many researchers have proven that MI tasks can stimulate the motor cortex in similar ways as an actual motor execution, producing brainwave oscillations known as **sensorimotor rhythms** (SMRs) [61][62].

SMRs have two frequency bands that must be considered: the **Mu** band (8 Hz – 13 Hz), which reflects alpha fluctuations located in the sensorimotor region of the brain, and the **Beta** band (13 Hz – 30 Hz). Both have shown that they are relevant for identifying the activity of neural groups while imagining or performing a movement [63]. The spectral information of these bands is essential for the MI-BCI system to perceive **event-related synchronizations** (ERS), occurring when neurons produce oscillations that increase in amplitude after the imagery task, and **event-related desynchronizations** (ERD), in which there is an amplitude decrease after the MI. As their name suggests, ERS and ERD are **event-related potentials** (ERP), defined as a cerebral response in front of a produced event, in this case, the MI.

At first, when the user is resting, no ERS nor ERD is encountered, the motor cortex remains neutral. When the user starts imagining or executing a motor movement, an ERD starts forming in the Mu and the Beta band to indicate the brain's preparation and readiness to act. Under this condition, the mental activity increases while the rhythmic activity attenuates as neurons work independently and asynchronously. The ERD is followed by an ERS predominantly in the Beta band, representing reduced excitability in the sensorimotor areas of the brain even though there is a rise in the frequential amplitude. In the end, Mu and Beta bands return to the resting state [64][65]. This procedure can be visualized in Figure 3.24., representing the mental activity at different frequencies (Y-axis) and through the time (X-axis) after a motor event.

It is important to note that ERS and ERD may appear in a slightly different bandwidth depending on the subject [66]; this complicates the implementation of a generic MI-BCI. Hence, for the correct functioning of the system, this thesis proposes to adjust the BCI so that there is always an appropriate frequency band selection, filtering out irrelevant spectral information.

Furthermore, it is optimal to mentally train the user to enhance his motor and MI skills for having better results and more pronounced ERS/ERD (Figure 3.24.). This improvement can be achieved with hours of practice or neurofeedback [67][68].

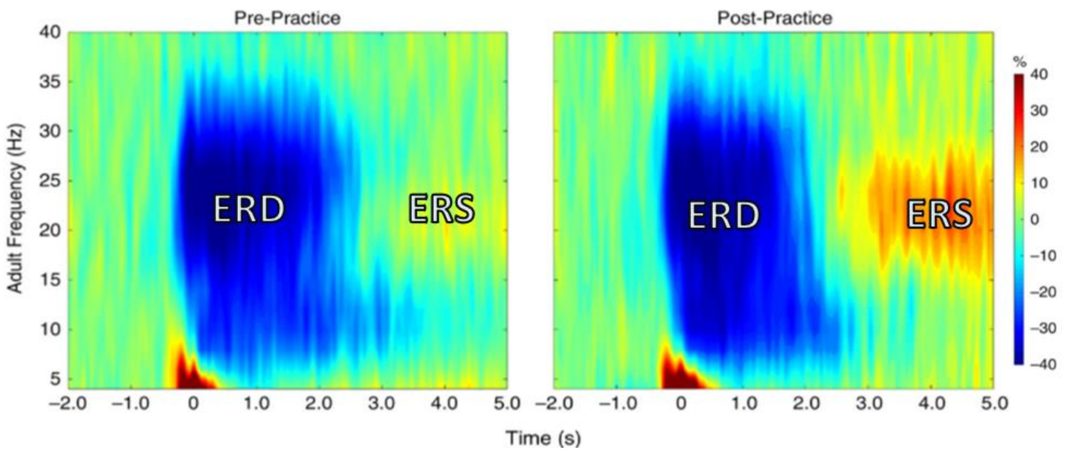


Figure 3.24.- Time-frequency spectrogram of ERS and ERD appearing after a motor event. The graph was created after averaging multiple motor events from adult volunteers, and it also demonstrates that, after practicing the motor task, there is an improvement in the ERPs. The X-axis represents the time (1 second per division), the Y-axis represents the frequency (5 Hz per division), and the Z-axis represents the relative power change of the mental activity (red tonalities show an increase and blue tonalities a decrease). Figure extracted from [69].

Since MI ERD and ERS occur in the motor cortex, they can be measured with the **C3** and **C4** EEG electrodes (10-20 system). For instance, while imagining hand movements, the C3 electrode, located on top of the left hemisphere, detects the imagery of the right hand; conversely, the C4 electrode, positioned above the right half of the brain, detects the imagery of the left hand. However, the remaining electrodes must not be ignored as they are necessary to carry out all the signal processing techniques needed for extracting the best features possible, used to classify the different MI actions.

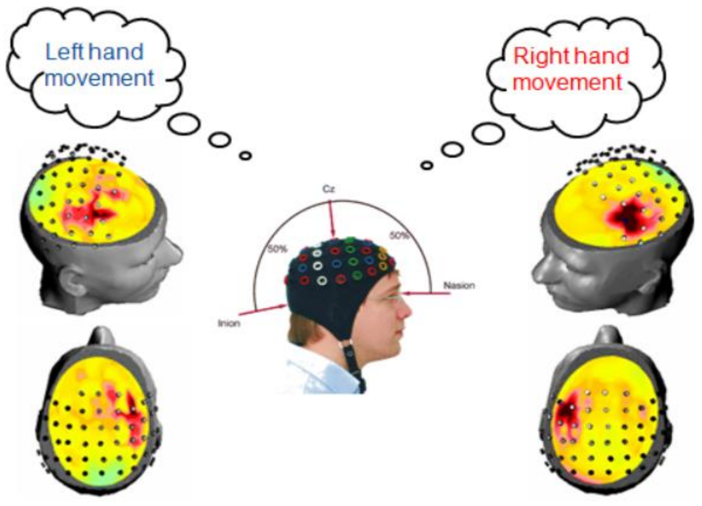


Figure 3.25.- Mental activity while performing or thinking a left or right-hand movement. Notice that the activated hemisphere reflects the contralateral side of the body. Figure extracted from [70].

3.3.3. Steady-state visually evoked potentials BCIs (SSVEP-BCI)

Besides MI, which is an active BCI technique, it is possible to use another approach that can potentially aid people with SCI, ALS, or any other movement dysfunctionality. This alternative method is a reactive BCI based on **steady-state visually evoked potentials** (SSVEP-BCI). An evoked potential is a reaction from the brain after sensory stimulation, which can be traced and processed to evaluate the cerebral response of the organism [71].

Regarding the SSVEPs, the evoked potentials are produced due to a visual excitement coming from an object flickering at an established frequency. The cerebral response appears in the occipital lobe, where it is possible to detect oscillations with a frequency equal to or harmonic to that of the blinking visual stimuli. Despite not having a general agreement, many studies indicate that the stimulant should not have frequencies below 3 Hz nor above 40 Hz [58].

In most BCIs based on SSVEPs, the person sits in front of a screen displaying animated figures flashing at different frequencies and is asked to focus visually at any of them. In this way, the BCI can recognize the stared object since the brain is triggered differently for each figure [72]. One example that could be used for controlling a wheelchair is a monitor showing a black background with four separated pink squares appearing and disappearing with a frequency of 9 Hz, 12 Hz, 15 Hz, and 17 Hz, respectively. Depending on the square that the participant concentrates on, the wheelchair acts differently:

- 9 Hz: it turns right.
- 12 Hz: it rotates left.
- 15 Hz: it goes forward.
- 17 Hz: it stops.

The processing carried out in the SSVEP-BCI is more straightforward than the MI-BCI since the frequencies of interest are clearly highlighted in the power spectrum. Moreover, it presents much lower inter and intra-individual variability as the visual stimuli produces a more robust response from the brain than the MI. Additionally, there is a lower influence of artifacts, such as eye-blinking, since the occipital lobe is located at the back of the head [58].

Nevertheless, despite all the benefits described, this thesis does not pursue a SSVEP-BCI approach. To begin with, the system always requires an external device (for example, a monitor or virtual reality glasses) that the user must always focus on, hindering his ability to interact with his environment. Furthermore, this technology does not take into account individuals with visual disabilities. In addition, many SSVEP-BCI participants complained of having fatigue after a time of seeing flickering lights [4].

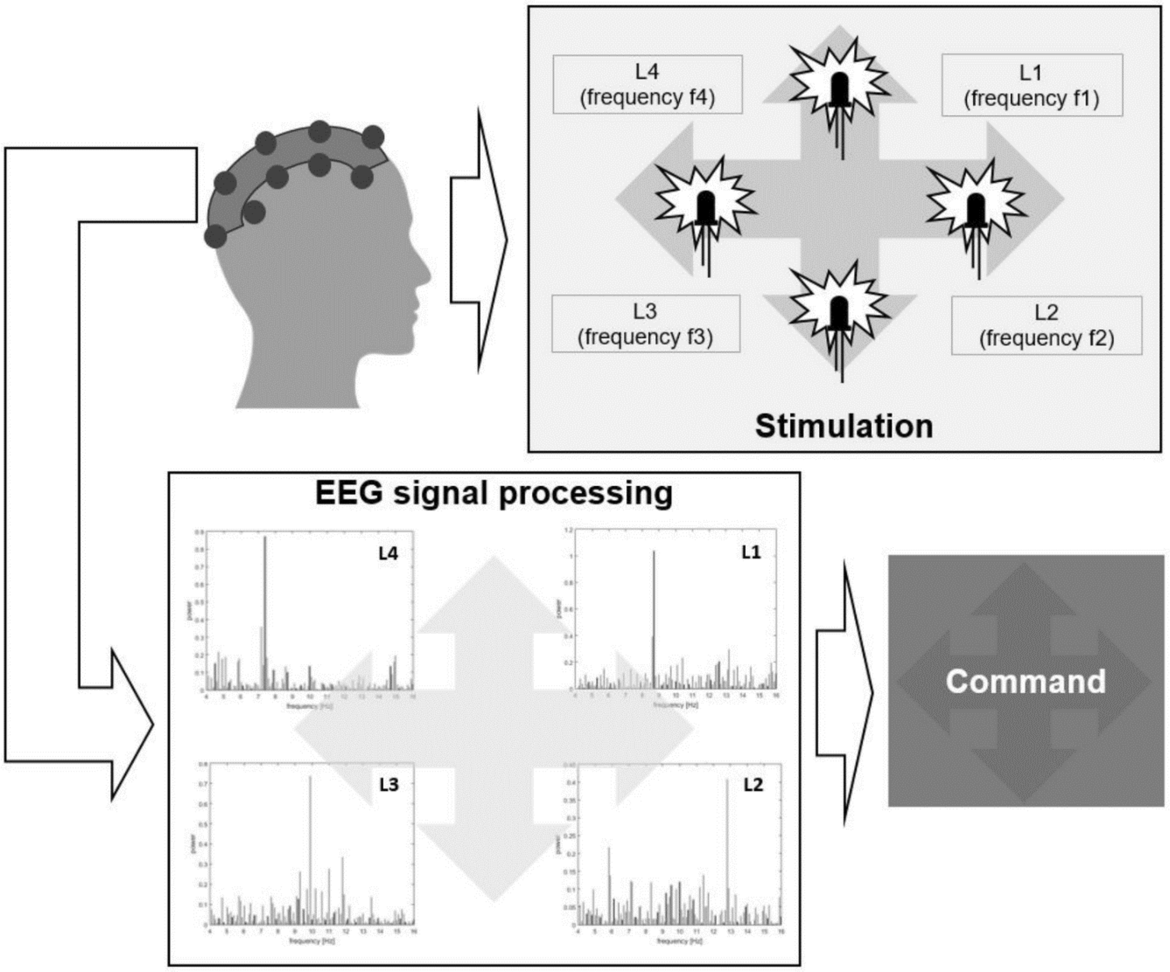


Figure 3.26.- SSVEP-BCI. The user focuses on a flickering light and, based on its frequency, sends a command to an external device. Figure extracted from [73].

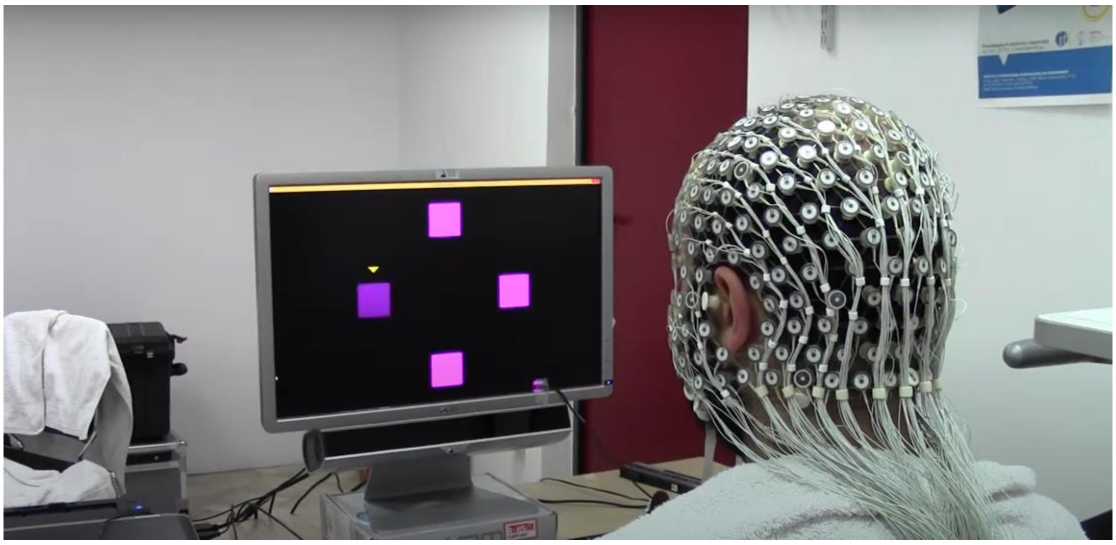


Figure 3.27.- SSVEP-BCI user staring at a blinking pink square. Figure extracted from [74].

4. Methodology

4.1. Overview of all the procedures carried out

Implementing a MI-BCI is not a simple task as it requires meticulous processes to treat the signals properly for extracting the most discriminant information from them. In this chapter, there will be a clear explanation of the followed methodology to ensure the fulfillment of the thesis' objectives.

As the author had no access to a high-quality EEG device, he worked on one of the most extensive publicly available MI datasets for designing and training the MI-BCI classifier. The dataset, described in the following subsection, contains EEG recording sessions where different participants had to imagine left and right-hand movements. Each applicants' data is analyzed to understand the signals, visualizing the effect of ERS and ERD in a spectrogram. By seeing the most relevant frequency bands from each subject, the signals were bandpass filtered digitally to exclude unnecessary information that may disturb the artificial intelligence (AI) models. Subsequently, the data was prepared for training the classifiers, including artifact removal (eye-blinking) and feature extraction with advanced methods.

From there, two AI models were designed; the first one to classify the hands imagery (left and right-hand) (**Model LR**) and the second to distinguish between an imagery event and the resting state of the user (**Model IR**). It must be highlighted that the AI models will be trained with data from multiple subjects; hence, the inter-subject variability in the ERS/ERD frequency bands must be taken into account. Before choosing the cut-off frequencies and the window's length that will be used for the model, each participant must undergo a calibration process to understand his frequential information in order to implement a broader band pass filter that covers the essential spectral behavior of all the subjects collectively (see Figure 4.3.).

Once the models were trained and evaluated, the MI-BCI was implemented on an hour MI session (from the dataset) unseen by the AI models. That was done by coding an algorithm that simulates a real-time signal acquisition, where the signals of the session are introduced inside a time window the same way that they would do if the data were obtained live from an EEG. Afterward, the signals are filtered, and the features are extracted. Notice that this real-time execution requires a proper digital filter design, avoiding significant delay between the mental imagery and its detection. Next, the correspondent features are introduced to Model IR. If the prediction is a resting state, no action will be taken. Otherwise, if it is an imagery prediction, the appropriate features will be introduced to Model LR, which will classify if the signal present in the time window is a left or right-hand imagery. By doing so, it is possible to control a robotic arm directly from the mental activity of the subject.

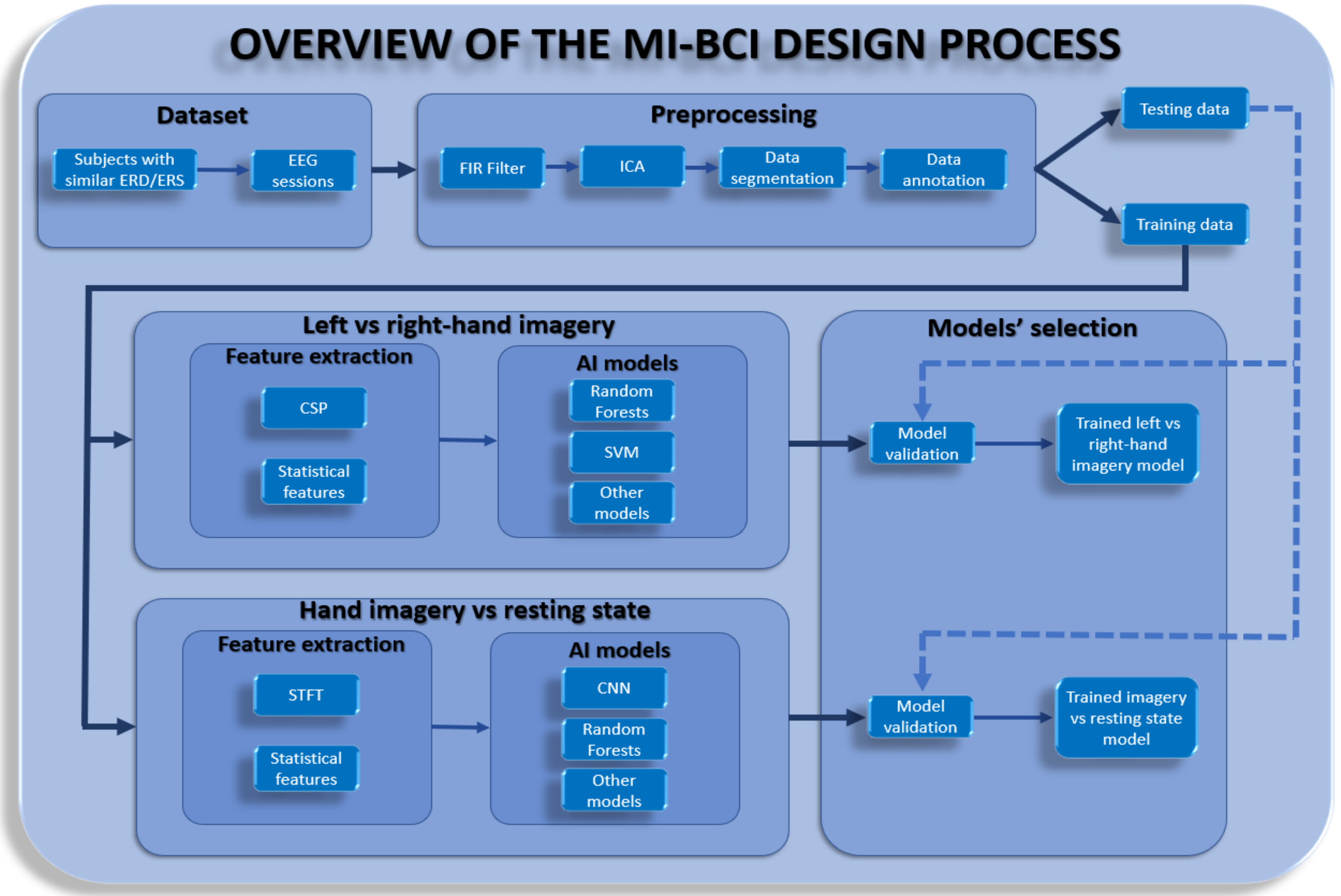


Figure 4.1.- This diagram shows an overview of all the processes that must be carried out for designing the MI-BCI. Figure created by the author.

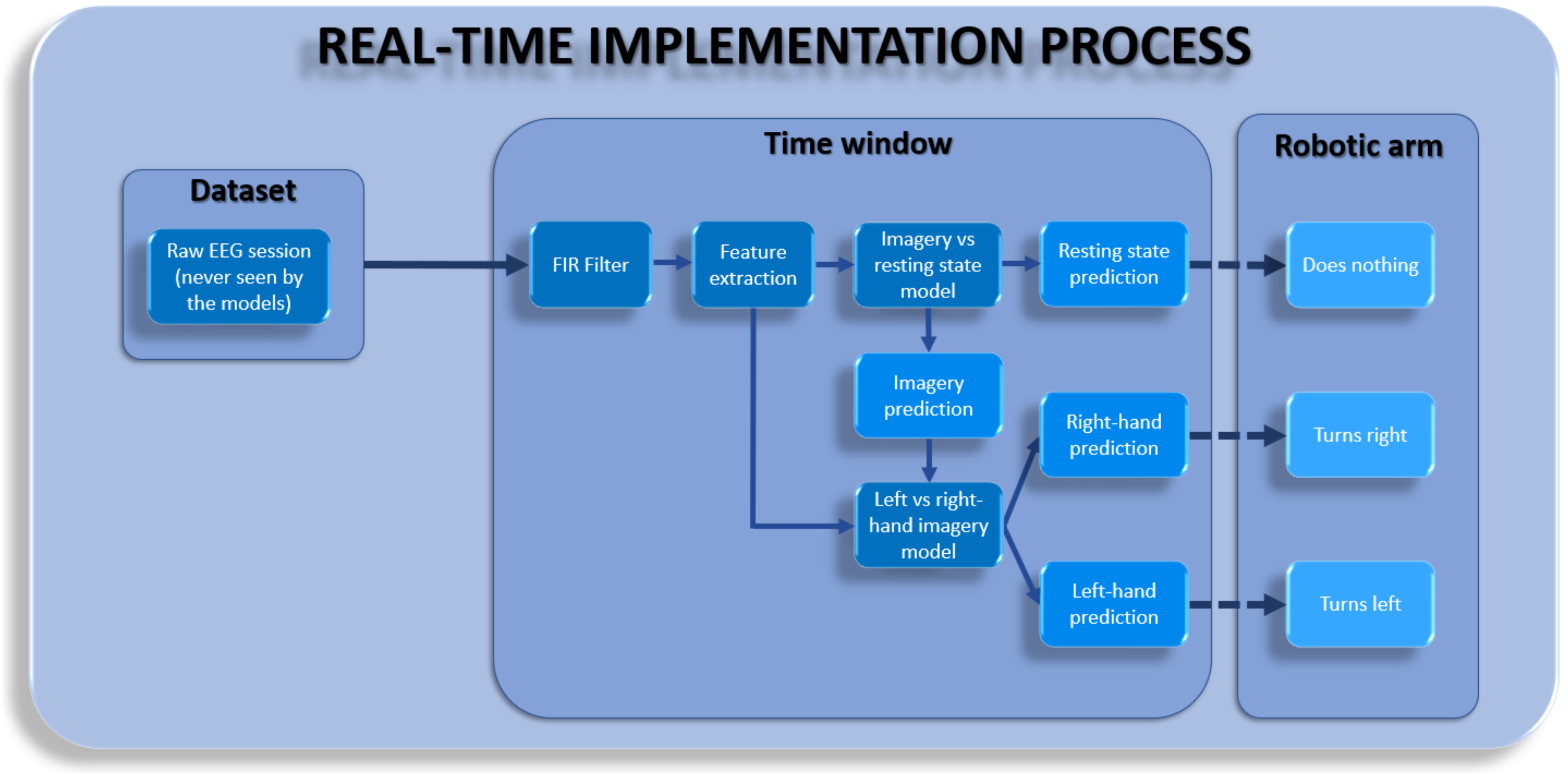


Figure 4.2.- This diagram shows how to implement the real-time simulation and implementation of the MI-BCI. Figure created by the author.

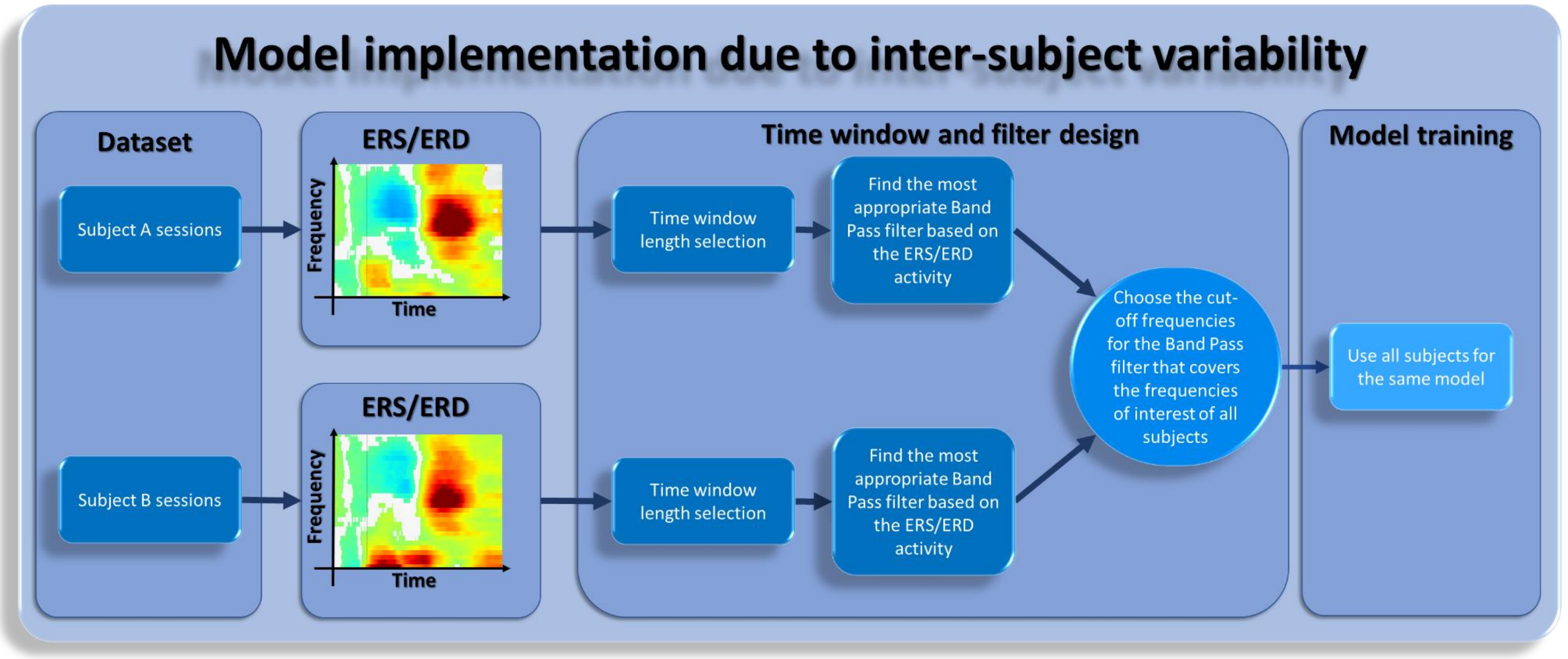

Model implementation due to inter-subject variability
Dataset
Subject A sessions
Subject B sessions
ERS/ERD
Frequency
Time
ERS/ERD
Frequency
Time
Time window and filter design
Time window length selection
Find the most appropriate Band Pass filter based on the ERS/ERD activity
Time window length selection
Find the most appropriate Band Pass filter based on the ERS/ERD activity
Choose the cut-off frequencies for the Band Pass filter that covers the frequencies of interest of all subjects
Model training
Use all subjects for the same model

Figure 4.3.- This diagram shows how to understand the data and implement the models based on inter-subject variability. Figure created by the author.

4.2. Software used

Everything was coded in Python, using libraries that ease all the processes previously mentioned. The following list indicates the most important ones, detailing the main functionalities used:

- **MNE**: Library specialized in treating brain signals captured with an EEG, a MEG, ECoG, or any other device. It includes many functions that enable optimal preprocessing and processing of EEG signals. Furthermore, it has many visualization tools that help to understand the data.
- **NumPy**: It provides data structures that allows the analysis and exchange of data between different algorithms. It mainly uses multidimensional vectors and arrays.
- **Pandas**: Used to organize the data into tables, also referred to as DataFrames. By doing so, the data is more organized and manageable.
- **SciPy**: Scientific library used to carry out simple and advanced statistical and mathematical operations, for example, the short-time Fourier transform. It is widely used for designing digital filters.
- **PyWavelets**: Scientific library used to carry out wavelet transforms.
- **Matplotlib**: It is a data visualization library used for the creation and customization of various types of graphics.
- **Scikit-learn**: This library, focused on Machine Learning, includes classifiers, data preparation tools, evaluation metrics, and other features. It also enables the creation of pipelines, allowing sequential execution of steps, for example, normalizing the data and then introducing it inside a Random Forest model.
- **TensorFlow**: It is mainly used for designing deep neural networks, convolutional neural networks, and other deep learning architectures. It also possesses plentiful normalization, optimization, and regularization tools.
- **Pickle**: It is useful for saving and loading the AI models.
- **Pymatreader**: Since the dataset's file has a MATLAB format, this library adapts it to an appropriate Python structure.
- **Streamlit**: It is employed for designing local web applications and graphical user interfaces.
- **PySerial**: It enables a serial communication between Python and Arduino.

4.3. The dataset

4.3.1. How to create a dataset?

Even though the EEG recordings used in the project were available online, the author wanted to design an intuitive application (using the Python library Streamlit) to guide future participants on the tasks they must perform to collect information for a MI-BCI dataset.

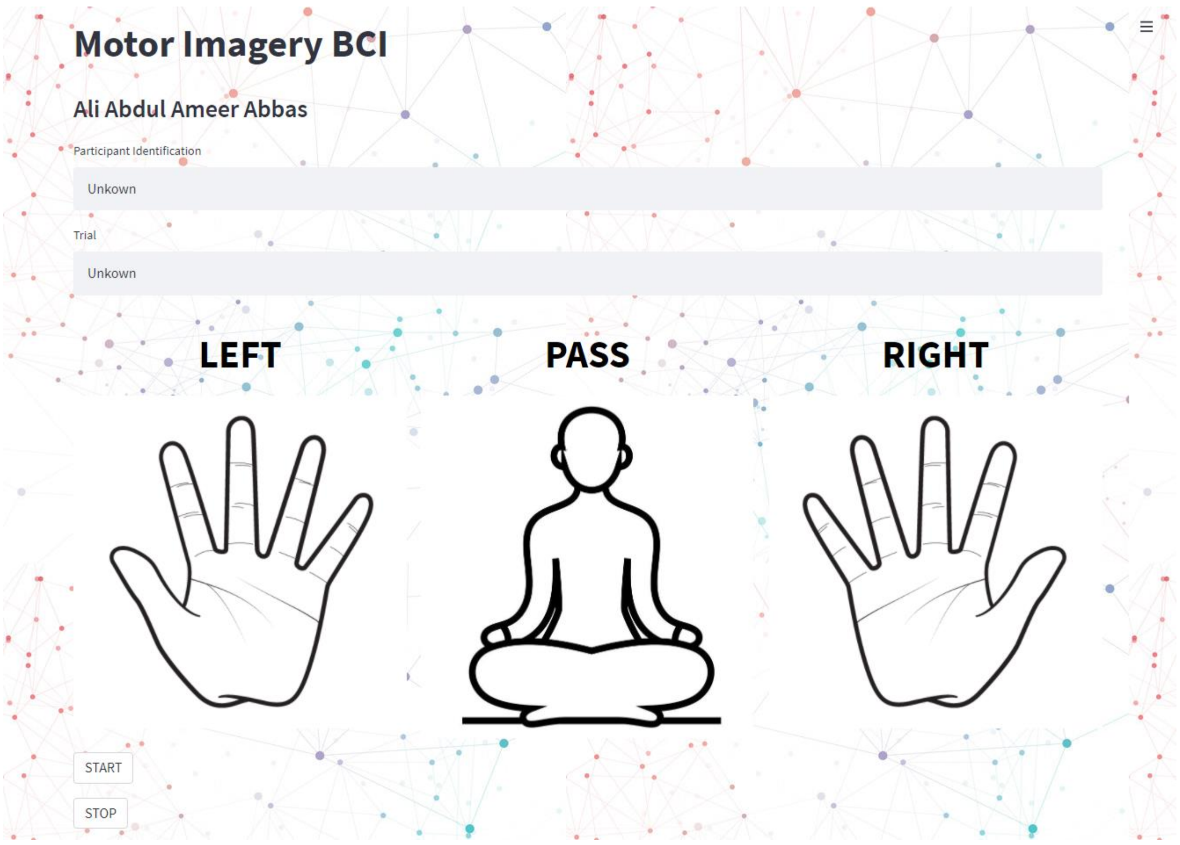


Figure 4.4.- Application created by the author that could be used to collect a dataset.

Before recording a session, the instructor must register the applicant's identification and the session number. Afterward, he must ensure that the participant is ready, looking at the application's interface and with all the EEG electrodes placed correctly; if everything is fine, he presses the start button, and the EEG starts capturing the signals. Simultaneously, one of the three figures turns green randomly for one second, indicating that the applicant must carry out one of the following tasks:

- **Left**: Imagine a left-hand movement.
- **Right**: Imagine a right-hand movement.
- **Pass**: Remain in a resting state without any motor imagery.

The task must be carried out until the figure changes to the original color. If no action is marked, the applicant must be in a passive state, waiting for another random figure to turn green. This process is repeated cyclically and only finishes when the instructor presses the stop button. Once the session is over, a CSV file is generated with all the markers needed to identify the time intervals in which the participant performed one of the described tasks; this is useful for segmenting the data. Notice that there is an indicator for each sample of the signal. The markers are labeled as follows:

- **Number 0**: No task was asked to the participant.
- **Number 1**: The participant imagined a left-hand movement.
- **Number 2**: The participant remained in a resting state without any motor imagery.
- **Number 3**: The participant imagined a right-hand movement.

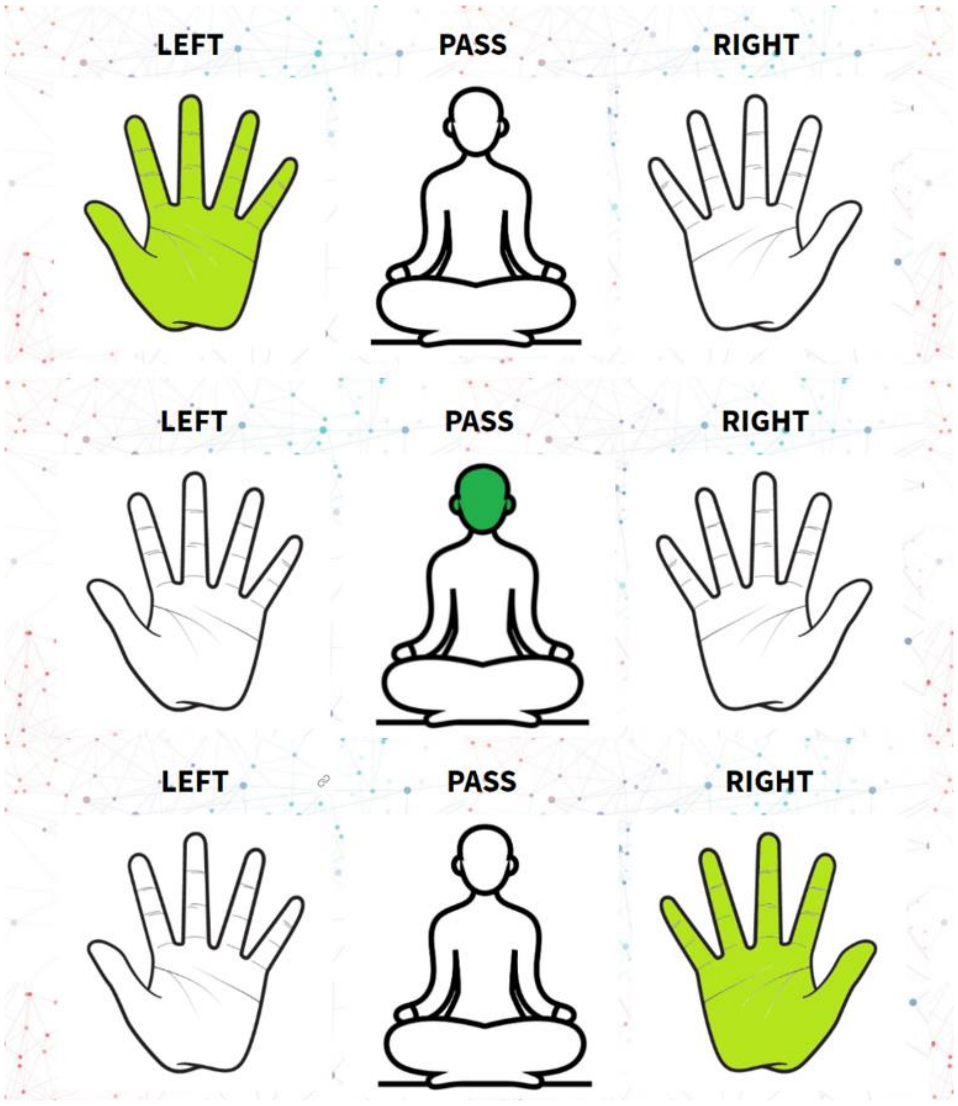


Figure 4.5.- The different tasks turning green. Figure extracted from the author's application.

4.3.2. The dataset used

In October 2018, some researchers from the Mersin University (Turkey) published one of the largest MI-BCI datasets freely available online [9], summing a total of sixty hours of EEG recordings and sixty thousand mental imageries from thirteen subjects. All the participants (60% males and 40% females between the ages of twenty and thirty-five) were healthy students without any psychiatric disorder diagnosed; they are nicknamed as "Subject X", being X a letter from A to M.

The protocol followed by the participants was strictly supervised by the instructor, avoiding any problems while recording the data. They had to sit in front of a screen that displays an experimental graphical user interface (eGUI) that guides the user in the imagery task that he must carry out during the session, in a similar way to that explained in the previous subsection. At the beginning of the session, the participant had two and a half minutes to relax and focus solely on the experimental process. Subsequently, there are three intervals of fifteen minutes (separated by break times of two minutes) in which the eGUI showed which imagery must be done. During this time, the eGUI selects one random action for one second (in which the user must remain to do the imagery), followed by a pause of approximately two seconds (in which the user must return to the resting state); this is repeated periodically until the break time. The EEG recorded the session uninterruptedly until the termination of the last fifteen-minute interval, lasting around one hour.

The medical device used to acquire all the signals was the EEG-1200 JE-921A EEG system from the company Nihon Kohden. Its datasheet [75] mentions that it works with a CMRR of 105 dB (at 60 Hz), an input impedance of 100 MΩ, a 16 bit ADC, and a sampling frequency that reaches up to 1000 Hz. These parameters are more than adequate for a MI-BCI application (see section 3.2.2.). The EEG is carefully placed (with the help of a head set) on the participant's scalp using the international 10-20 system, cleaning beforehand the area where the electrodes are placed to reduce the skin's impedance. In addition, multiple impedance checks are performed to ensure that all the electrode/skin impedances are below 10 kΩ. The sessions were acquired using a 200 Hz sampling rate, using a bandpass filter with cutoff frequencies of 0.53 Hz and 70 Hz. In this way, it is possible to remove the DC component and avoid the aliasing effect (meeting the Nyquist-Shannon theorem). Furthermore, there is a Notch filter in the 50 Hz to attenuate the powerline interferences.

The database contains four different paradigms; however, this thesis will only work with one of them: the classical (CLA) paradigm. It consists of the imagery of left and right-hand movements and the resting state; hence, this paradigm is almost identical to the one designed by the author in section 4.3.1. The following table shows all the participants that took part in this paradigm, detailing the subject's nickname, the file name of each recording, and the date of execution of the session. Notice that, in the case that the participants had more than one session, the dates are different; this is important to assess the intra-variability effects.

Table 4.1.- Table showing all the participants, sessions, and dates in the CLA paradigm.

Nickname	Number of sessions	File	Date (dd/mm/yyyy)
Subject A	1	CLASubjectA1601083StLRHand.mat	08/01/2016
Subject B	3	CLASubjectB1510193StLRHand.mat	19/10/2015
		CLASubjectB1510203StLRHand.mat	20/10/2015
		CLASubjectB1512153StLRHand.mat	15/12/2015
Subject C	3	CLASubjectC1511263StLRHand.mat	26/11/2015
		CLASubjectC1512163StLRHand.mat	16/12/2015
		CLASubjectC1512233StLRHand.mat	23/12/2015
Subject D	1	CLASubjectD1511253StLRHand.mat	25/11/2015
Subject E	3	CLASubjectE1512253StLRHand.mat	25/12/2015
		CLASubjectE1601193StLRHand.mat	19/01/2016
		CLASubjectE1601223StLRHand.mat	22/01/2015
Subject F	3	CLASubjectF1509163StLRHand.mat	16/09/2015
		CLASubjectF1509173StLRHand.mat	17/09/2015
		CLASubjectF1509283StLRHand.mat	28/09/2015

Inside each file of the CLA paradigm, multiple markers are helpful to understand all the events that occurred during the session. From them, the signals can be segmented and annotated, which are essential steps to understand the data for posterior training and testing of the AI models that will classify the imageries. The markers mean the following:

Table 4.2.- Marker's meaning in the sessions.

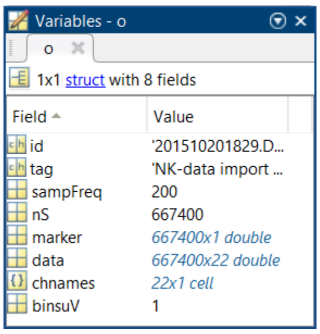
Marker Identification Number	Meaning
0	No task was asked to the participant.
1	The participant imagined a left-hand movement.
2	The participant imagined a right-hand movement.
3	The participant remained in a resting state.
91	The session break time.
92	The end of the session.
99	The initial relaxation of the participant.

4.4. Understanding the data

The first step taken in every signal processing project is to understand the data as much as possible, carrying out different visualization methods necessary to take further actions to enhance the overall performance of the AI models.

4.4.1. Data importation

In the explanatory paper that described the dataset (mentioned in subsection 4.3.2.), the authors indicate that all the files were saved as MATLAB files (with the extension ".mat"). Thus, for importing one participant's session of an EEG recording, it is required to convert the data into a Python-compatible format, avoiding any information leakage. This can be achieved with the library *Pymatreader*. It is important to note that inside each MATLAB file all the data is saved as a structure array (*struct*), which can save collections of associated information into organized containers (known as fields). The following *struct* named "o" is an example of a EEG recording session:



1x1 struct with 8 fields

Field	Value
id	'201510201829.D...
tag	'NK-data import ...
sampFreq	200
nS	667400
marker	667400x1 double
data	667400x22 double
chnames	22x1 cell
binsuV	1

Figure 4.6.- MATLAB's *struct*, corresponding to an EEG recording from a participant. Figure from the author.

As it can be seen, there are containers that, besides providing the EEG signal data, also give relevant information such as the ID, the sampling frequency (200 Hz), the number of samples, the channel names, and the markers (explained in Table 4.2.). Notice that the marker's length and the number of samplings each channel has (shown in the data container) are the same, with 667 400 samples. Moreover, it is possible to see that there are 22 channels; however, one of them (the X3 channel) will not be considered as it was used exclusively for signal acquisition purposes.

After the data is imported in Python, the EEG signals' data should be passed to microvolts multiplying by $1{\cdot}10^{-6}$, this is done because MNE library works by default in volts. Furthermore, all that *struct* containers must be saved into a MNE data type known as *info*, which is used to define basic information of the EEG recording.

Additionally, it is crucial to define the EEG configuration used, which is the international 10-20 system. By doing so, all the electrodes' positions and the distances between them are introduced to the program, enabling any spatial signal processing. The following illustrations show 2D and 3D head models with the electrodes drawn on them:

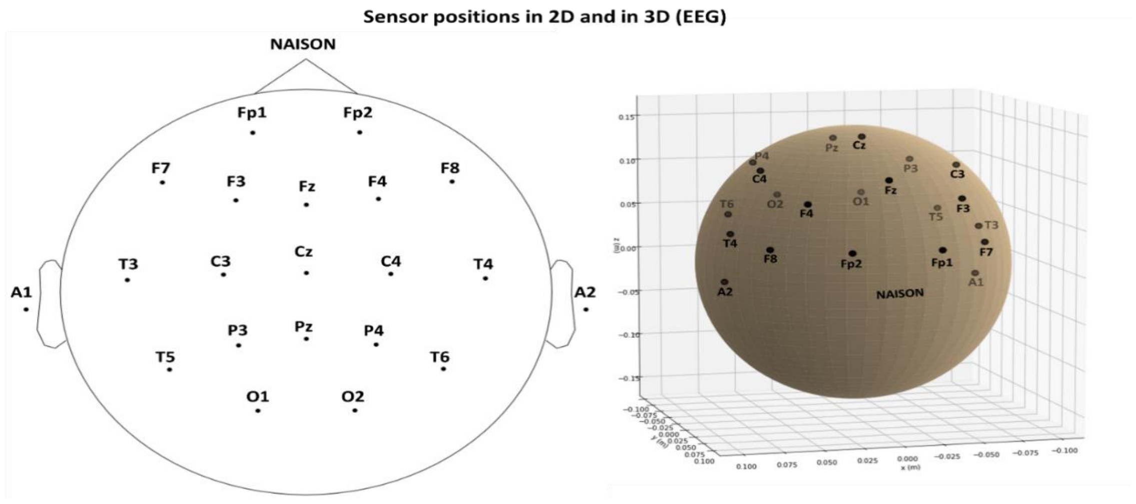


Figure 4.7.- Electrodes' position and distances in the international 10-20 system used in the project. The left image shows a 2D electrode configuration. The right image represents a 3D head (the brownish sphere) with the precise position of the electrodes drawn (X and Y-axis with a scale of 0.025 meters per division, and the Z-axis with 0.05 meters per division). The NAISON corresponds to the nose. Figures designed by the author.

The last step before visualizing the EEG signal for the first time is to set an appropriate common reference to the electrodes. According to the dataset explanatory paper [9], the acquisition system (EEG-1200 JE-921A) references all the electrodes to the value obtained from the following formula:

$$\boldsymbol{CommonReference = 0.55 \cdot (C3 + C4)\ (V)} \qquad \textbf{(Eq. 4.1)}$$

Where C3 and C4 are the voltage value in those electrodes. However, many researchers recommend re-referencing all the channels to the average value of all the electrodes (**common average referencing** (CAR)) [76][77]. This technique reduces considerably noisy sources that may be problematic. It also acts as a spatial filter by reducing the presence of shared activity between electrodes [78]. Therefore, this method was implemented to improve the signals' quality by applying the following formula:

$$\boldsymbol{x_i^{CAR}(t) = x_i(t) - \frac{1}{C}\sum_{j=1}^{C} x_j(t)\ \ (V)} \qquad \textbf{(Eq. 4.2)}$$

Where $\boldsymbol{C}$ is the number of the total electrodes used, $\boldsymbol{x_i}$ is the value of the electrode **i**, $\boldsymbol{x_i^{CAR}}$, is the value of the electrode **i** after the CAR, and $\boldsymbol{x_j}$ the value of the electrode **j**. After this, the EEG signal can be visualized:

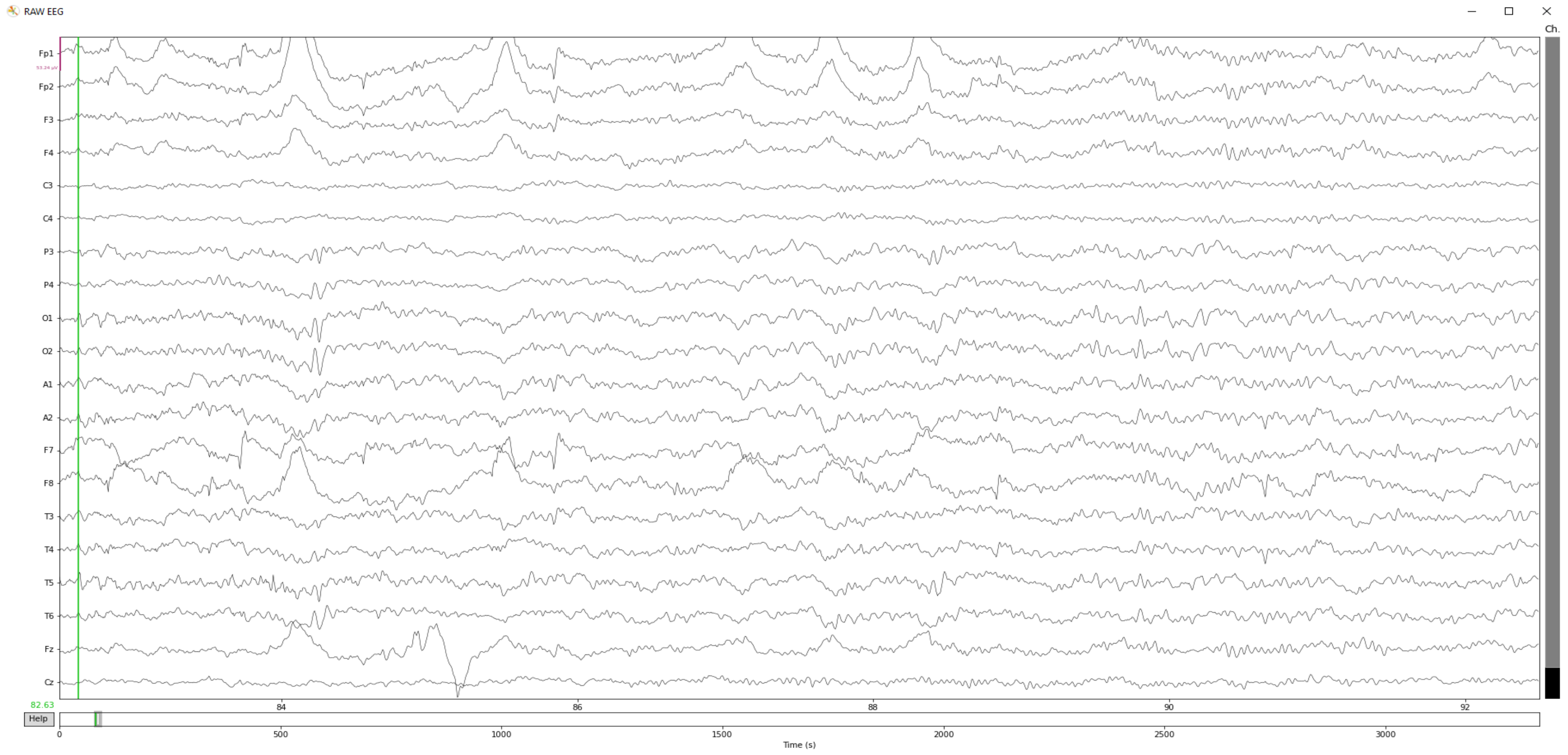


Figure 4.8.- EEG visualization of the raw signals of each electrode of the EEG. The bar below enables one to scroll the window and visualize the whole EEG session. At the top left, there is a purple bar that indicates the amplitude scale of the signals (53.24 µV). The X-axis shows the time in seconds. Figure created by the author.

4.4.2. Power spectral density (PSD)

When working with signals, it is highly recommended to check their spectral information as it contains valuable information about their nature. After carrying out a discrete Fourier transform (DFT), which decomposes the discrete signals into a spectrum of frequencies, it is possible to apply numerous spectral calculations to better interpret the data, for instance, the power spectral density (PSD).

The PSD indicates the strength or power of the signals by looking at their squared amplitudes in the frequency domain, which is a useful indicator to assess the relevance of the different frequency bands [79][80]. The units used to quantify the spectral power in the EEG signals are squared microvolts divided per Hertz ($\mu V^2/Hz$). The MNE library calculates the PSD with the Welch's segment averaging estimator [81] and displays the power spectrum of each channel as shown below:

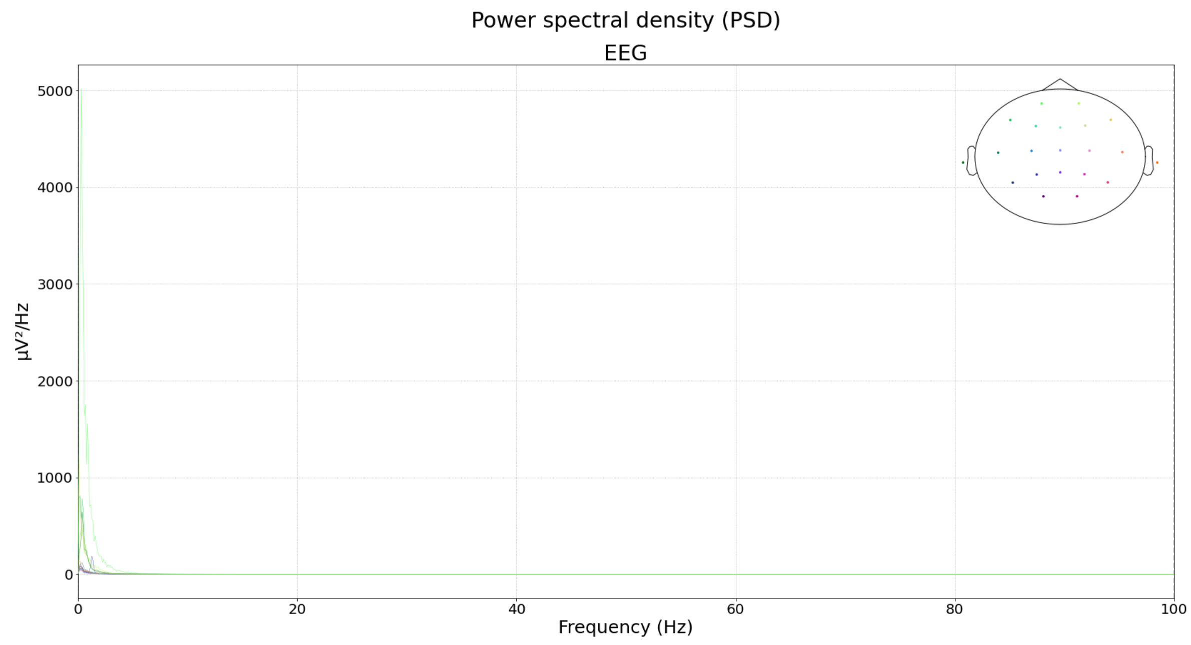


Figure 4.9.- PSD of the signals of each EEG channel used. Each color represents a channel (see the top-right 2D head model). X-axis shows the frequency in Hz (20 Hz per division) and the Y-axis shows the $\mu V^2/Hz$ (1000 $\mu V^2/Hz$ per division). Figure created by the author.

At first sight, it seems that the graph does not provide significant information because the low frequencies have pronounced power in comparison with higher frequencies. Therefore, to enhance the visualization, it is convenient to work in dB as it allows the researcher to perceive small values as well as large ones. That is achieved by passing the Y-axis from linear to logarithmical, using the following formula:

$$\boldsymbol{Y_{dB} = 10 \cdot log(Y_{lin})\ (dB)}$$ **(Eq. 4.3)**

Where $\boldsymbol{Y_{dB}}$ is the result in dB and $\boldsymbol{Y_{lin}}$ is the linear value. As it can be seen in the following illustration, now it is simpler to interpret the results:

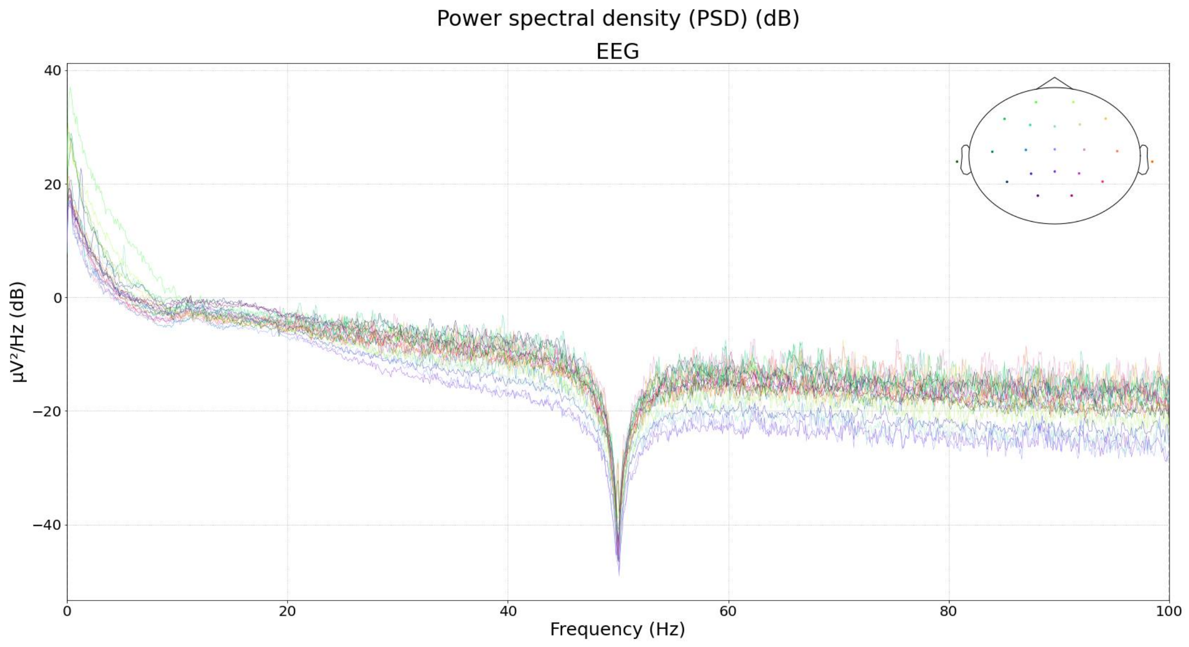


Figure 4.10.- PSD of the signals of each EEG channel used, in dB. Each color represents a channel (see the top-right 2D head model). X-axis shows the frequency in Hz (20 Hz per division) and the Y-axis shows the µV^2/Hz in dB (20 dB per division). Figure created by the author.

Thanks to this spectral analysis, it is possible to confirm the effect of the EEG's Notch filter at the 50 Hz component, corresponding to the power line frequency, is remarkably attenuated; this is important to check out, as the signals might be contaminated with undesired electromagnetic interferences.

Additionally, there is high power near the DC component, which is a normal behavior in EEG signals. Furthermore, notice that the frontal channels, especially Fp1 and Fp2 (in green), have more amplitude at frequencies between 0.1 Hz and 10 Hz; this can be associated with the blinking artifact, which is a low-frequency response.

Finally, it can be seen that, after the 10 Hz, the power of all the electrodes is constantly decreasing as the frequency increases. Later than 40 Hz, almost all the channels are below -10 dB.

4.4.3. Event related synchronization and desynchronization (ERS/ERD)

As it has been remarked in subsection 3.3.2., plotting the ERD and ERS maps is extremely helpful to understand the frequential behavior of the left and right-hand imagery over time. With this information, the relevant frequency bands can be detected in order to posteriorly design the cut-off frequencies of the band-pass filter, attenuating insignificant bands from the spectrum for better performance.

Furthermore, it is possible to check the most recurrent reaction time (of each participant) after the eGUI's indications to perform imageries. In addition, the time window length is estimated based on the time in which the ERD/ERS activity lasted in the map. These procedures are necessary to later segment the data for training the AI models.

The ERDS maps were drawn following instructions described in scientific papers and the MNE documentation [82][83][84]. They are computed by clustering and averaging the time-frequency representations of all the periods that contain an imagery event (i.e., right-hand imagery). These periods start one second before the instruction to perform the MI task (represented as a dashed line) and end four seconds later. It is important to remember that, as most of the MI events occur in the motor cortex, the ERDS will be plotted in the central electrodes of the EEG (C3, C4, and Cz).

The blue tonalities of the maps represent a low activity (normally an ERD) and the red ones a higher activity (usually an ERS). The darker the color is, the more mental responsiveness there is. Therefore, it is optimal to detect ERS and ERD with this property as it means that the participant has an outstanding ability to control the MI tasks.

In this section, the ERDS graphics will be generated for each participant (from Subject A to F) using their first EEG recording session for representing the ERDS in the right and left-hand imageries. With this, the author can assess whether the participants' cortical activity behaves as expected or not. This phase is critical to discard subjects without any clear mental pattern or with uncommon events; as it has already been explained, MI tasks are not easy to perform at the beginning and, on most occasions, require hours of practice before mastering them.

Finally, there will be a brief commentary on how to treat the sessions of the participants depending on their ERS/ERD activity.

4.4.3.1. ERD/ERS (Subject A)

Subject A presents remarkably noisy maps; no clear patterns are defining the ERS and the ERD. Both left and right-hand imageries have very high activity in frequencies above the Beta band, which is unusual and hardly corresponds to an ERS. Moreover, slightly above the Mu band of the left-hand imagery at the C4 channel (2.2 s – 3.5 s), there is a mild activity that could indicate a MI event. However, if this was true, this same pattern should appear for the right hand at the C3 electrode; instead, it is repeated at the C4 (0.5 s – 1.8 s), which is unexpected. For these reasons, **this participant will be discarded from the study**.

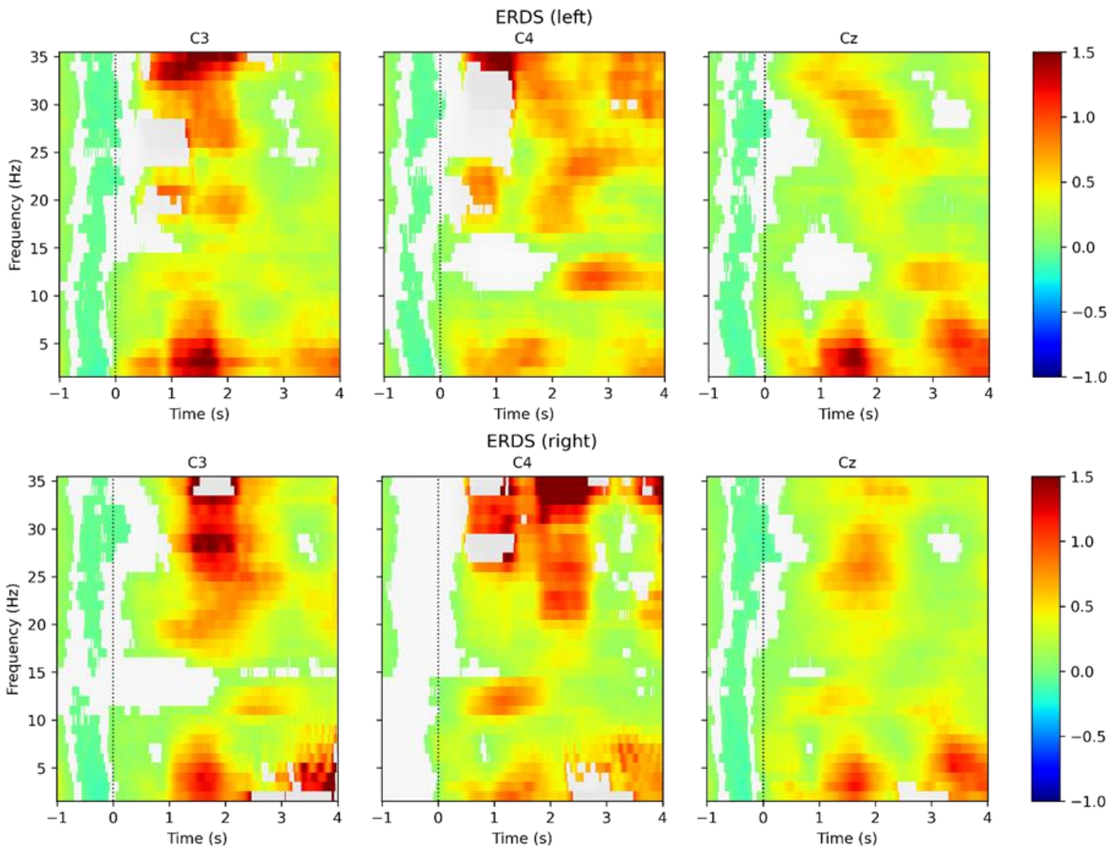


Figure 4.11.- Subject A. ERD and ERS plots of the C3, C4, and Cz channels of the EEG. The upper graphs represent the time-frequency response occurring while imagining a left-hand movement, and the lower ones the right-hand movement. For each plot, the X-axis represents the time in seconds (1 s per division) and the Y-axis represents the frequency in Hz (5 Hz per division). The colors represent the mental activity (blue for low activity and red for high activity). The dashed line represents the indication to perform the imagery task. The circles represent areas of interest. Figure created by the author.

4.4.3.2. ERD/ERS (Subject B)

The overall activity of subject B in electrodes C3, C4, and Cz is not substantial; nevertheless, there is a visible change in the Beta and Mu bands after the event occurs. For the left-hand imagery, channel C4 seems to have an ERS between 20 Hz and 25 Hz (1.5 s – 2.8 s), and, at the same time, an ERS appears at the Mu band (10 Hz – 13 Hz); this happens while channel C3 records neutral activity. Conversely, the right-hand imagery has a similar effect in the C3 electrode, but more powerfully (20 Hz – 26 Hz, 2 s – 3 s; 10 Hz – 15 Hz, 2 s – 2.8 s); meanwhile, channel C4 remains with low activity. This response indicates a MI detection, especially after seeing that the activated brain hemisphere reflects the contralateral side of the body. However, as the ERS have low amplitudes, this might indicate that from all the trials, only a few of them were indeed imageries. After evaluating Subject B, it was concluded that it perform poorly due to the reason mention before. Therefore, **this subject will not be further considered in the study**.

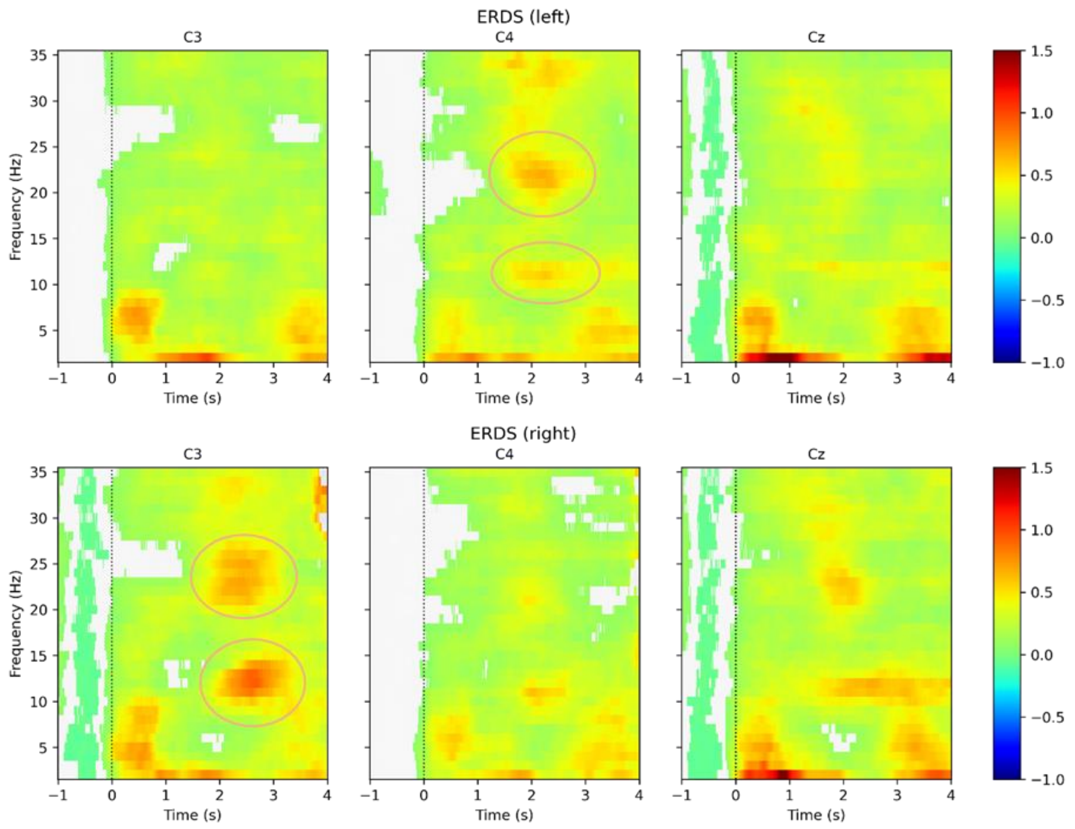


Figure 4.12. Subject B. ERD and ERS plots of the C3, C4, and Cz channels of the EEG. The upper graphs represent the time-frequency response occurring while imagining a left-hand movement, and the lower ones the right-hand movement. For each plot, the X-axis represents the time in seconds (1 s per division) and the Y-axis represents the frequency in Hz (5 Hz per division). The colors represent the mental activity (blue for low activity and red for high activity). The dashed line represents the indication to perform the imagery task. The circles represent areas of interest. Figure created by the author.

4.4.3.3. ERD/ERS (Subject C)

Of all the candidates, subject C is the one that has the most perceptible ERD and ERS patterns, approaching the theoretical behavior that these maps should have. For the left-hand imagery, there is a dominant activity in channel C4, where an ERD in the Beta band (17 Hz – 26 Hz, 0.1 s – 1 s) is followed by an ERS in the Beta (15 Hz – 21 Hz, 1.8 s – 2.8 s) and the Mu band (10 Hz – 13 Hz, 2 s – 3 s). Notice that a similar pattern occurs in electrode C3 but with much less power. For the right hand, channel C3 shows the same pattern as the C4 electrode of the left-hand imagery, with ERDs and ERSs in the same frequency bands and times. Thus, **this participant will be considered in the study**.

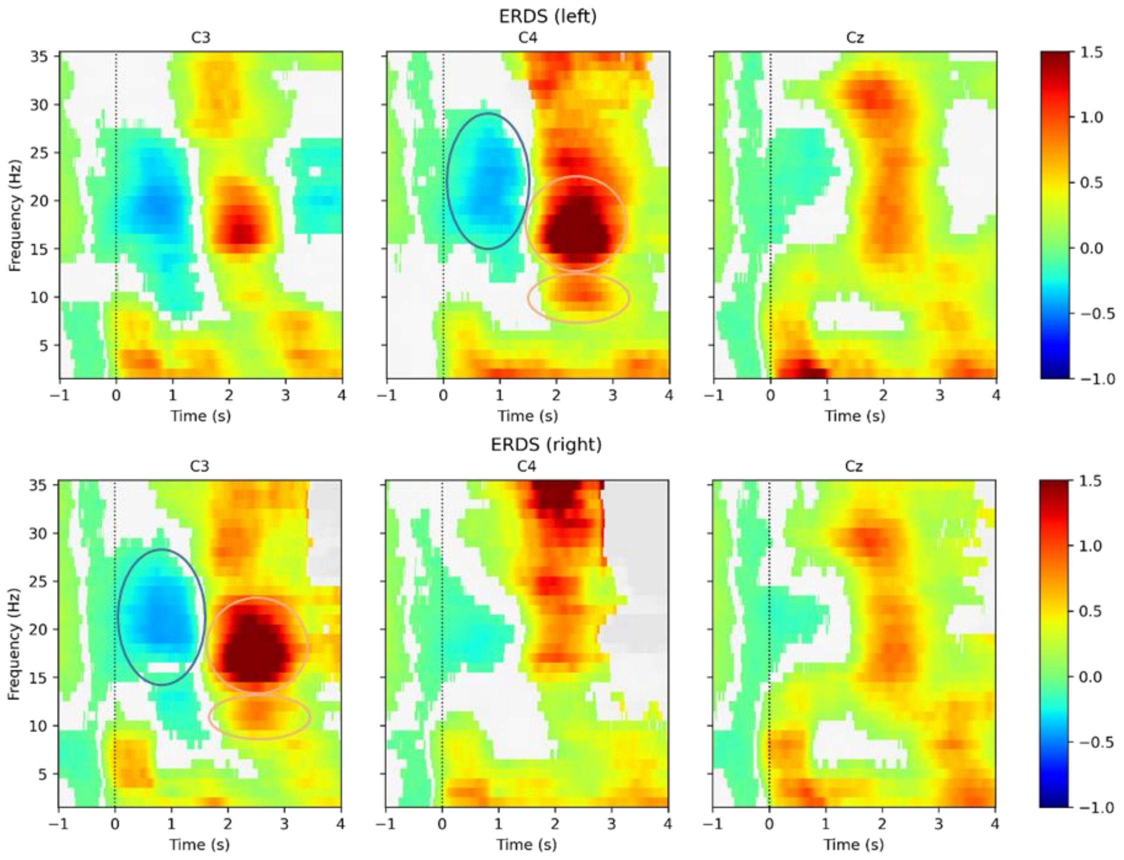


Figure 4.13. Subject C. ERD and ERS plots of the C3, C4, and Cz channels of the EEG. The upper graphs represent the time-frequency response occurring while imagining a left-hand movement, and the lower ones the right-hand movement. For each plot, the X-axis represents the time in seconds (1 s per division) and the Y-axis represents the frequency in Hz (5 Hz per division). The colors represent the mental activity (blue for low activity and red for high activity). The dashed line represents the indication to perform the imagery task. The circles represent areas of interest. Figure created by the author.

4.4.3.4. ERD/ERS (Subject D)

There are no clear ERD/ERS patterns in the left-hand imagery of participant D. Electrodes C3 and C4 show a similar activity distribution. Moreover, electrode Cz is more weighty than both of them, which more likely indicates that left-hand imagery is hardly appreciable. The right-hand imagery presents an ERS in the Beta band (15 Hz – 21 Hz, 1.2 s – 2.5 s) and neutral activity in the C4 electrode. Nonetheless, due to the problematic detection of the left-hand imagery, this subject **will be discarded from the study**.

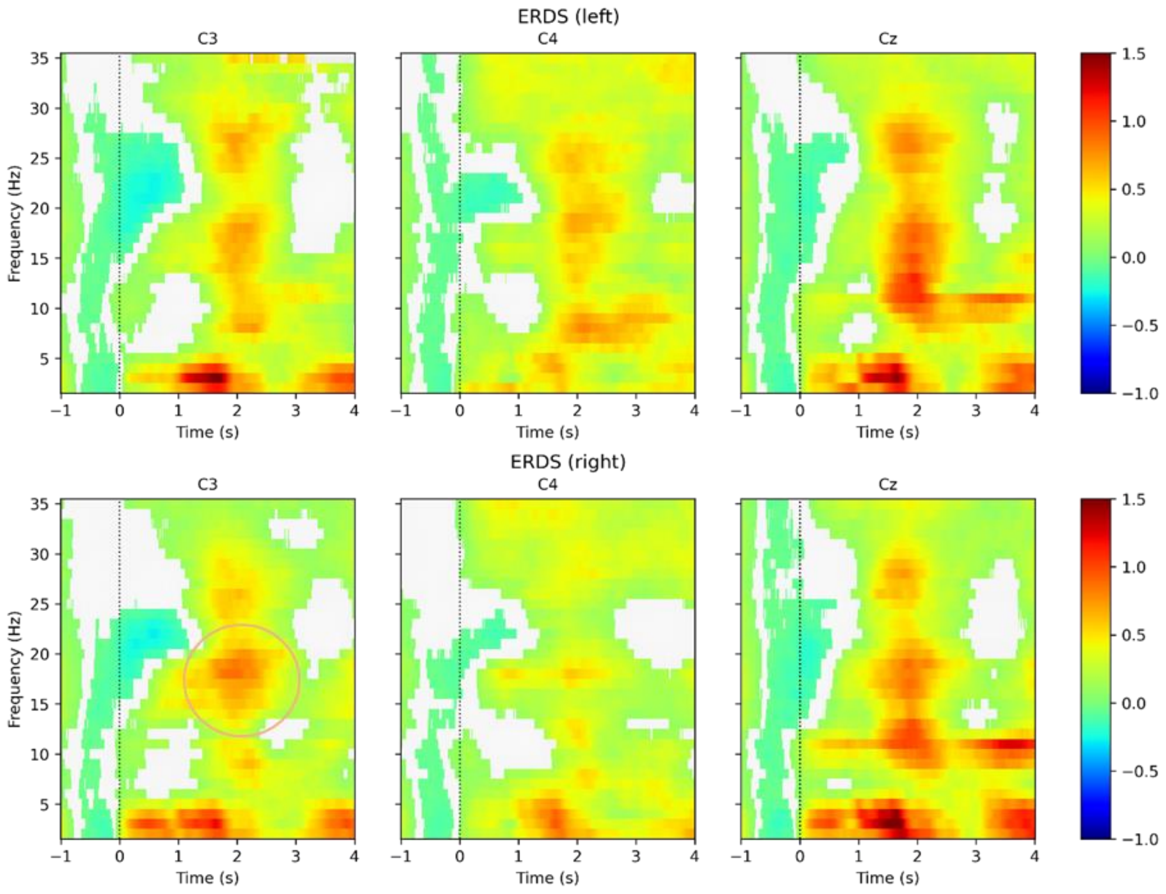


Figure 4.14. Subject D. ERD and ERS plots of the C3, C4, and Cz channels of the EEG. The upper graphs represent the time-frequency response occurring while imagining a left-hand movement, and the lower ones the right-hand movement. For each plot, the X-axis represents the time in seconds (1 s per division) and the Y-axis represents the frequency in Hz (5 Hz per division). The colors represent the mental activity (blue for low activity and red for high activity). The dashed line represents the indication to perform the imagery task. The circles represent areas of interest. Figure created by the author.

4.4.3.5. ERD/ERS (Subject E)

Subject E has an ERS in the Beta band at the C4 electrode of the left-hand imagery (18 Hz – 23 Hz, 2.3 s – 3.2 s) and the C3 electrode of the right-hand imagery (18 Hz – 24 Hz, 2 s – 3 s); the remaining channels remain neutral. Even though the ERDS maps do not present any ERD or activity at the Mu band, the ERS described at the beginning of the paragraph could signify a MI; consequently, **this subject will be considered in the study**.

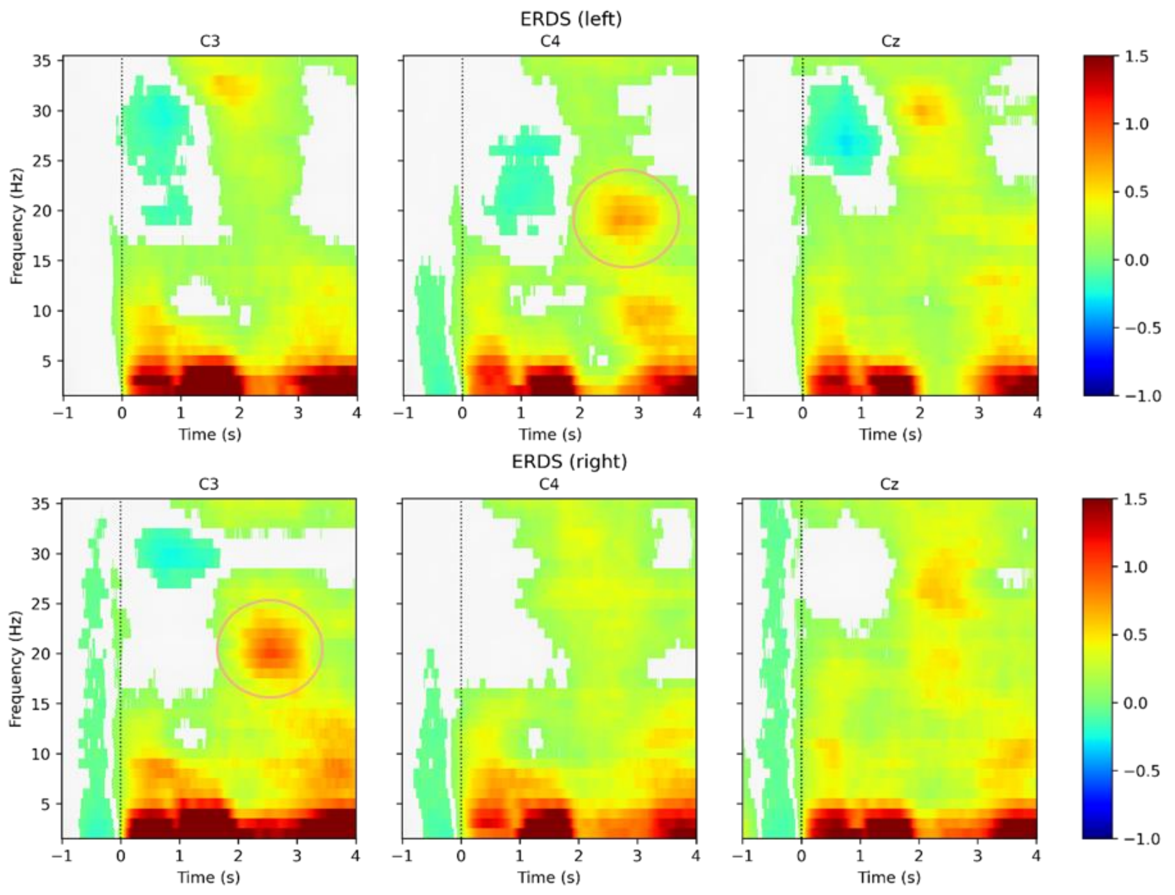


Figure 4.15. Subject E. ERD and ERS plots of the C3, C4, and Cz channels of the EEG. The upper graphs represent the time-frequency response occurring while imagining a left-hand movement, and the lower ones the right-hand movement. For each plot, the X-axis represents the time in seconds (1 s per division) and the Y-axis represents the frequency in Hz (5 Hz per division). The colors represent the mental activity (blue for low activity and red for high activity). The dashed line represents the indication to perform the imagery task. The circles represent areas of interest. Figure created by the author.

4.4.3.6. ERD/ERS (Subject F)

The left-hand imagery of Subject F has high activity in the upper Beta band in all the electrodes represented, which is strange as it should only appear at the C4 channel. Furthermore, there is an ERS at the Mu band exclusively at C3. Regarding the right-hand imagery, C3 has the most significant intensity in the Beta band while C4 is lower, which is expected. However, the Mu band is visible at C4 instead of being active at the C3 electrode. Despite the odd behavior of these responses, the high intensities in the ERS bands could potentially indicate hand imageries; hence, **this subject will be considered in the study**.

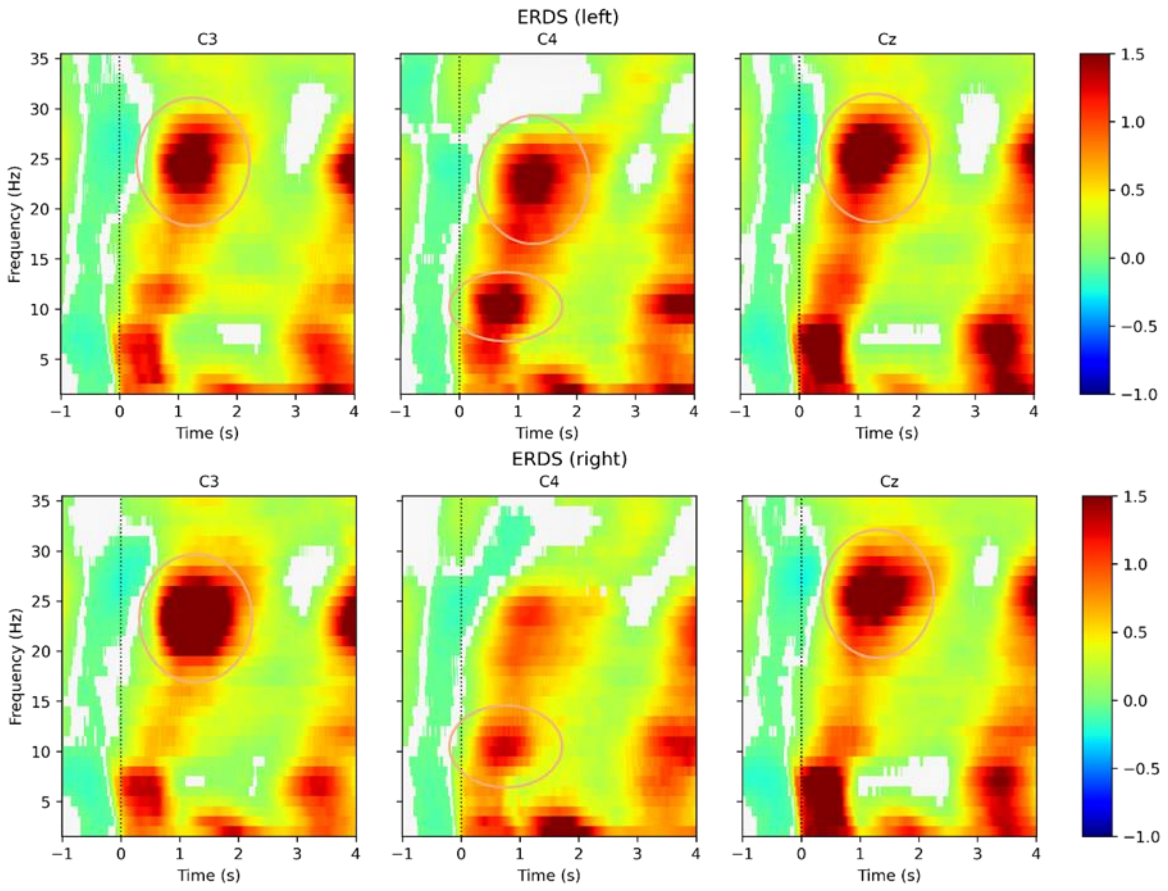


Figure 4.16. Subject F. ERD and ERS plots of the C3, C4, and Cz channels of the EEG. The upper graphs represent the time-frequency response occurring while imagining a left-hand movement, and the lower ones the right-hand movement. For each plot, the X-axis represents the time in seconds (1 s per division) and the Y-axis represents the frequency in Hz (5 Hz per division). The colors represent the mental activity (blue for low activity and red for high activity). The dashed line represents the indication to perform the imagery task. The circles represent areas of interest. Figure created by the author.

4.4.3.7. ERD/ERS commentary

After analyzing all the ERDS maps, many aspects must be pointed out before segmenting the data and training the BCI's AI models.

First and foremost, from all the dataset's subjects, three participants were discarded from the study as they had abnormal behaviors that could preclude the distinction between the left and right-hand imagery. That could be caused due to the lack of experience while carrying out MI tasks, which could be mastered by practicing this mental chore for hours. The only subjects with fair quality recordings are **subjects C, E, and F**.

Secondly, only Subject C presented ERD, while the remaining participants jumped from a neutral state to an ERS directly. Hence, from all the time-frequency representations, only the ERS will be considered for training the AI models. Furthermore, it has been seen that the activity in the Mu band was not present in all the subjects, and, when it did, its intensity was noticeably lower than the ERS appearing in the Beta band. That is to say that the band-pass filter that will be implemented should maintain the ERS of the Beta band. Moreover, the time window length used for segmenting the data should be sufficient to cover the ERS activity so that the model can process it correctly.

Finally, it is necessary to assess the inter-subject variability in order to know how to train and implement the BCI. As it can be seen in Figure 4.3., after obtaining the ERS/ERD maps, the researcher must decide which time window length and cut-off frequencies should be used for each participant to highlight the ERS. Subsequently, all the subjects will be compared to check if their ERS have similarities. The following table will compare the selected subjects to determine the most appropriate frequency band for the final model:

Table 4.3.- Inter-subject variability evaluation. Table created by the author.

Subject	Time window length	Frequency band	Frequency band of the final model
C	0.9 s	**15 Hz** – 21 Hz	**15 Hz – 26 Hz**
E	0.9 s	18 Hz – 24 Hz	
F	0.9 s	20 Hz – **26 Hz**	

To conclude, subjects C, E, and F will be trained in the same AI model with a **window length of 0.9 seconds** and a band pass filter with lower and higher **cut-off frequencies of 15 Hz and 26 Hz**, respectively.

4.5. Digital filtering

After deciding which frequency bands are suitable, a bandpass digital filter must be designed to remove undesired frequencies; this is an important step in this MI-BCI design as a well-designed filter can eradicate noise, interference, artifacts, or mental activities that are not related to MI. Indeed, after several trials, the author found out that there is a substantial accuracy and precision increase in the AI predictions after having a more selective BW. There are two main types of digital filters: the **finite impulse response** (FIR) and the **infinite impulse response** (IIR) filters [85][86][87].

FIR filters have a non-recursive nature and lack feedback. That is to say that, assuming causality, it only uses the present and a finite number of past samples of the input. Furthermore, designing its parameters is relatively straightforward, making it more controllable. Their main advantage is that they are inherently stable since their transfer function presents all the poles at the origin of the Z-plane. Another crucial characteristic of these filters is that they have a linear phase response in the desired BW; this is of great interest to avoid significant distortions after filtering the signal. However, these filters present some drawbacks that must be considered. Firstly, the designer must be careful while choosing the order of the filter as the system will have less computational efficiency, requiring high memory. In addition, this will produce a considerable delay that may affect real-time applications such as the one implemented in this thesis. Therefore, before choosing the filter's order, one must weigh the benefits of having a highly selective filter and the disadvantages described. In the discrete-time domain, the output of a sample can be calculated by the convolution of the filter **h(k)**, as shown below:

$$\boldsymbol{y(n) = \sum_{k=0}^{M-1} h(k) \cdot x(n-k); \quad if\ causal\colon\ h(n) = 0, n < 0, n \geq M} \qquad \textbf{(Eq. 4.4)}$$

Where $\boldsymbol{y(n)}$ is the output in an instant **n**, **M** is the order of the filter, and $\boldsymbol{x(n)}$ is the input. The FIR filters have the following transfer function in the Z domain, where $\boldsymbol{b_k}$ are the filter's coefficients:

$$\boldsymbol{H(z) = \sum_{k=0}^{M} b_k \cdot z^{-k} = \frac{1}{z^M} \sum_{k=0}^{M} b_k \cdot z^{M-k}} \qquad \textbf{(Eq. 4.5)}$$

To calculate the group delay of a FIR filter, the subsequent formula can be used, where **M** is the order, and $\boldsymbol{f_s}$ the sampling frequency [88]:

$$\boldsymbol{Delay = \frac{M-1}{2 \cdot f_s}\ (seconds)} \qquad \textbf{(Eq. 4.6)}$$

On the flip side, the IIR filters are recursive and present feedback; therefore, they consider the present and all past samples of the input as well as the past outputs. These filters have better computational efficiency and require less memory storage compared to the FIR filters. Moreover, they do not have such a high delay time compared to the FIR filters. Another benefit of these filters is that they can straightforwardly be converted from digital to analog as they can be based on renowned analog filter configurations, such as the Butterworth, the Bessel, the Chebyshev, the Cauer, and many other types of filters. Nevertheless, some weaknesses can affect the performance of IIR filters. The most relevant one is that they can hardly achieve a linear phase response, which may produce a distorted output signal. In addition, these filters have poles that must be placed carefully to avoid instabilities; consequently, these filters are less flexible than FIR ones and harder to implement. In the discrete-time domain, the output of a sample can be calculated by the convolution of the filter **h(k),** as shown below:

$$\boldsymbol{y(n) = \sum_{k=0}^{\infty} h(k) \cdot x(n-k); \quad if\ causal\colon\ h(n) = 0, n < 0} \qquad \textbf{(Eq. 4.7)}$$

Where $\boldsymbol{y(n)}$ is the output in an instant **n**, and $\boldsymbol{x(n)}$ is the input. The IIR filters have the following transfer function in the Z domain, where $\boldsymbol{b_k}$ and $\boldsymbol{a_k}$ are the filter's coefficients:

$$\boldsymbol{H(z) = \frac{\sum_{k=0}^{M} b_k \cdot z^{-k}}{\sum_{k=0}^{N} a_k \cdot z^{-k}}} \qquad \textbf{(Eq. 4.8)}$$

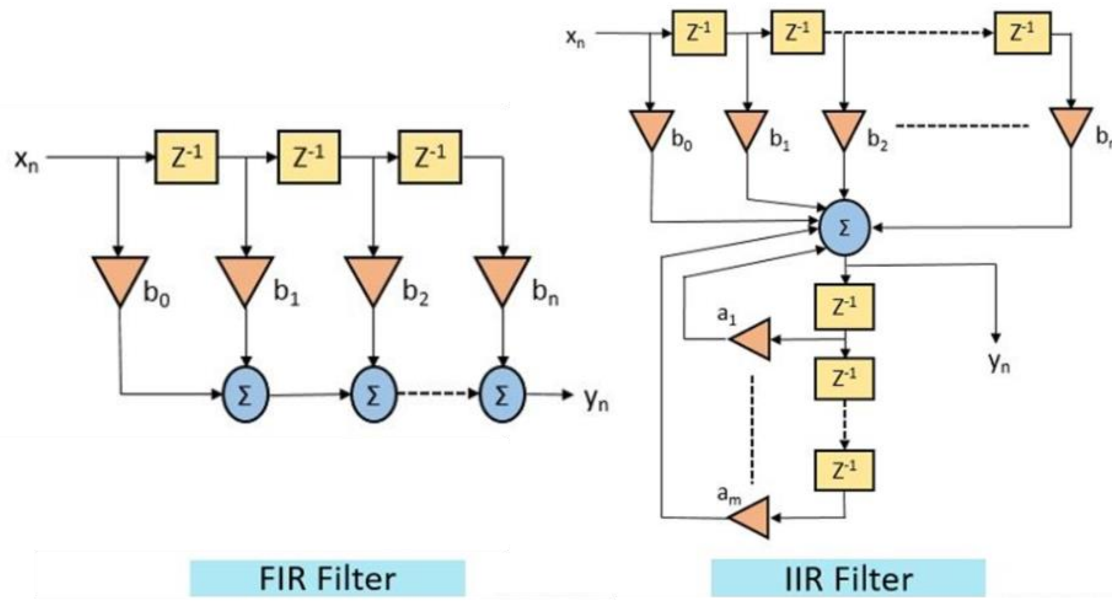


Figure 4.17.- Block diagrams of FIR and IIR filters. Figure extracted from [89].

Now that both types of filters are understood, it can be seen that the FIR filters have clear benefits that prevail over the IIR filters. Having a linear phase and avoiding instability scenarios is of great interest while implementing a suitable band-pass filter, especially in MI-BCI applications. Furthermore, if carefully designed, the computational efficiency and time delay reduces significantly. Thus, an FIR filter will be developed in this thesis.

Some considerations must be taken into account before designing the filter [90]:

1) **Filter type:** In this BCI application, the filter will be a bandpass, having an upper and a lower cut-off frequency. Additionally, it must be decided which passband ripple is accepted. For biomedical applications, it should be minimized. Moreover, one must consider which transition band is desired. Remember that the narrower the transition band, the more selective the filter is. Lastly, it is important to ensure that the stopband ripple does not considerably affect the output.

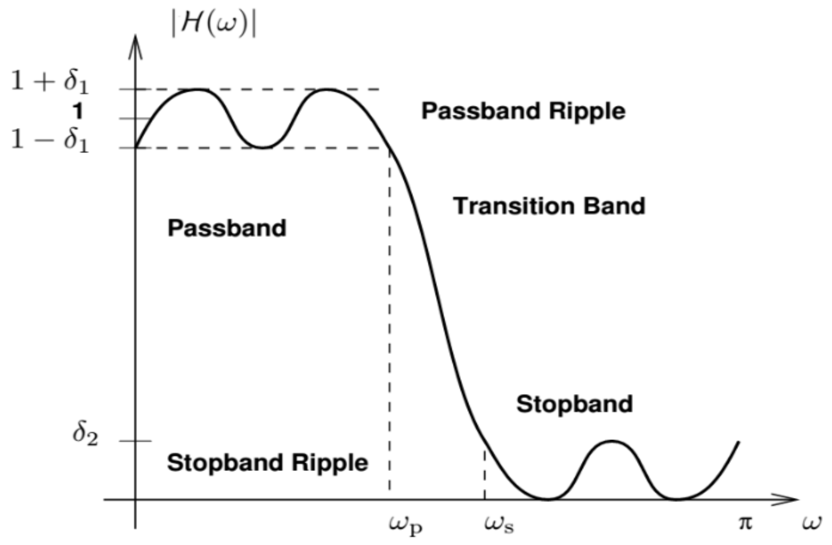


Figure 4.18.- Overview of the characteristics of a filter. Figure extracted from [90].

2) **Window type:** For avoiding abrupt changes in the tails of the filter's impulse response, it is preferable to operate with a window function. The Bode of an FIR filter presents a principal lobe (which is the most important) and many undesired sidelobes. By smoothing the extremes of the impulse response with a window (Hamming, Blackman, Hann, etc.), the sidelobes attenuate even more, which is optimal. The tradeoff is that the principal lobe width augments and filter selectivity can be affected. To solve this problem, one can increase the window's length (see next point).

The following graphics show the filter's impulse response with a rectangular (upper figure) and a Hamming window (lower figure). Notice how the extremes are softer in the Hamming window:

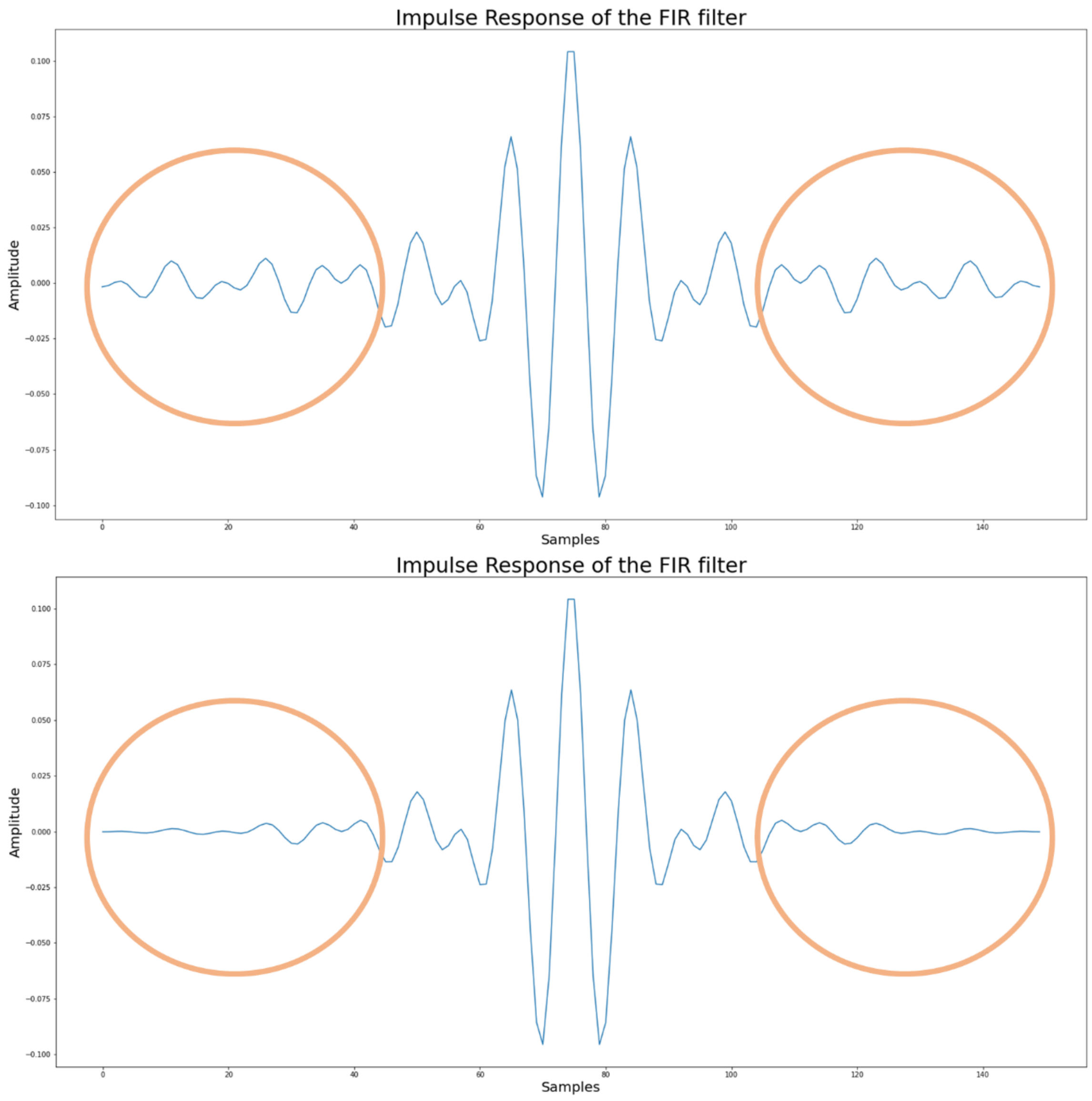


Figure 4.19.- Effect of the filter window in the impulse response of the filter. Rectangular window (upper figure) and Hamming window (lower figure). Figure created by the author.

The following Bode charts highlight the effect of using different window types with the same filter characteristics. The rectangular window (upper figure) has a narrower primary lobe in contrast with the Hamming window (lower figure), which seems to have doubled in width. Nevertheless, the Hamming windows have much more attenuated sidelobes in comparison to the rectangular window. Notice that the phase is linear, which is one of the main characteristics of a FIR filter:

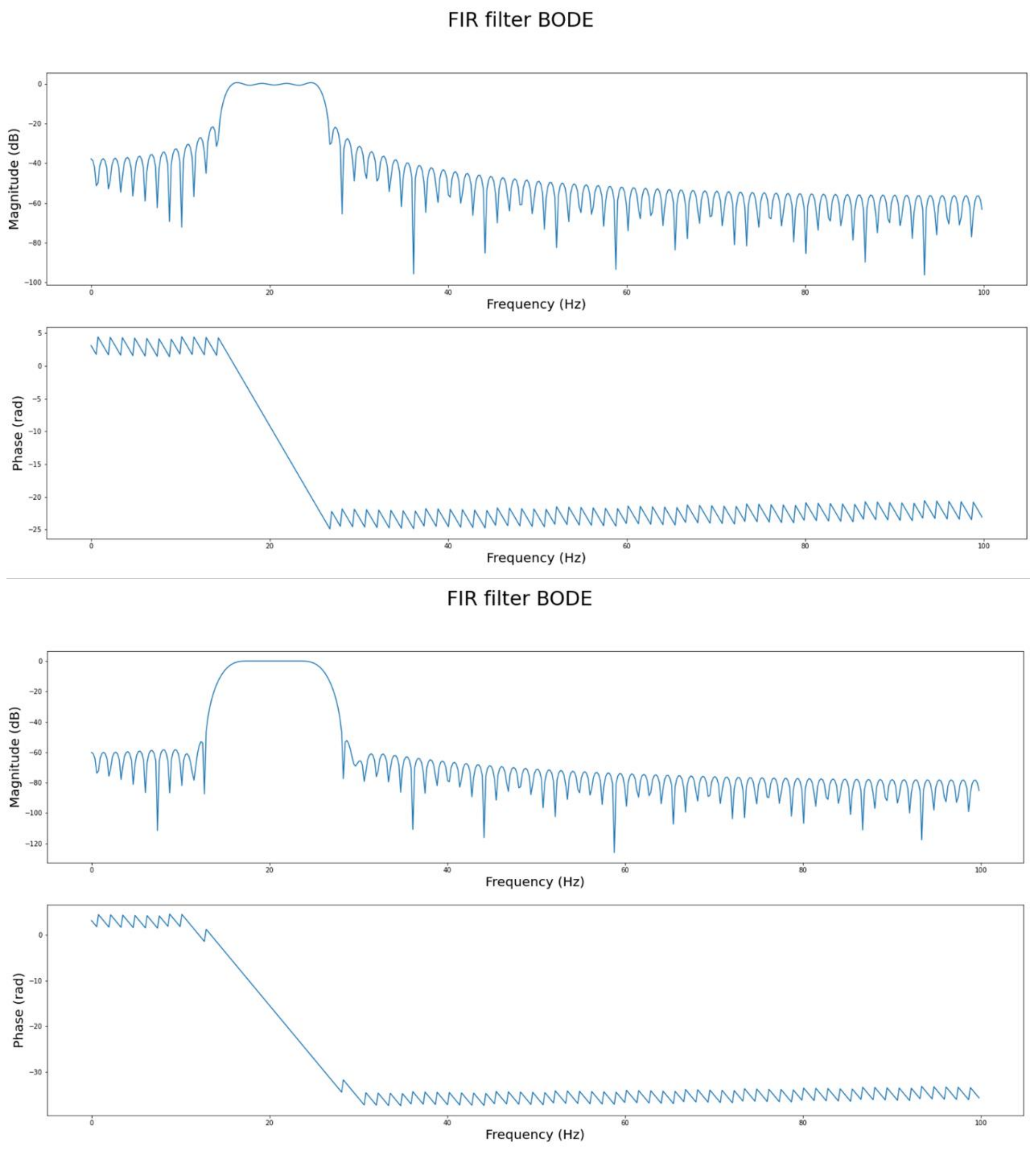


Figure 4.20.- Effect of the filter window in the magnitude and phase Bode charts of the filter. Rectangular window (upper figure) and Hamming window (lower figure). Figure created by the author.

3) **Window length:** The window length defines the number of lobes present in the Bode as well as the filter's order. Remember that the higher the order, the more selective the filter is. However, the filter's order must not be too large since the delay will increase. One way of

calculating this parameter is by looking at Table 4.4. and establish the width of the first lobe as the normalized passband.

Table 4.4.- Primary Lobe width and the peak of the secondary lobes of the different window types. Information extracted from [90].

Window Type	Approximate Width of Main Lobe	Peak Sidelobe Amplitude (dB)
Rectangular	$\frac{4\cdot\pi}{M+1}$	-13
Bartlett	$\frac{8\cdot\pi}{M}$	-25
Hanning	$\frac{8\cdot\pi}{M}$	-31
Hamming	$\frac{8\cdot\pi}{M}$	-41
Blackman	$\frac{12\cdot\pi}{M}$	-57

For instance, if the cutoff frequencies are 15 Hz and 26 Hz, and the sampling frequency is 200 Hz, the normalized passband can be calculated as:

$$\boldsymbol{PB_n = w_{2n} - w_{1n} = 2\cdot\pi\cdot\frac{f_2 - f_1}{f_s} = 2\cdot\pi\cdot\frac{26-15}{200} = 0.11\cdot\pi\, rad/s} \quad \textbf{(Eq. 4.9)}$$

If a Hamming window is used, it is possible to calculate the minimum length of the filter from Table 4.4.:

$$\boldsymbol{PB_n = \frac{8\cdot\pi}{M} \rightarrow M = \frac{8\cdot\pi}{PB_n} = \frac{8\cdot\pi}{0.11\cdot\pi} \approx 73\, samples} \quad \textbf{(Eq. 4.10)}$$

Thus, the length must be at least 73 samples, for example, 150 samples. The delay is then calculated:

$$\boldsymbol{Delay = \frac{M-1}{2\cdot f_s} = \frac{150-1}{2\cdot 200} = 372.5\, ms\, (\approx 75\, samples)} \quad \textbf{(Eq. 4.11)}$$

Which is more than acceptable for a MI-BCI. This can be easily checked by plotting the effect of an all-pass FIR filter of order 150:

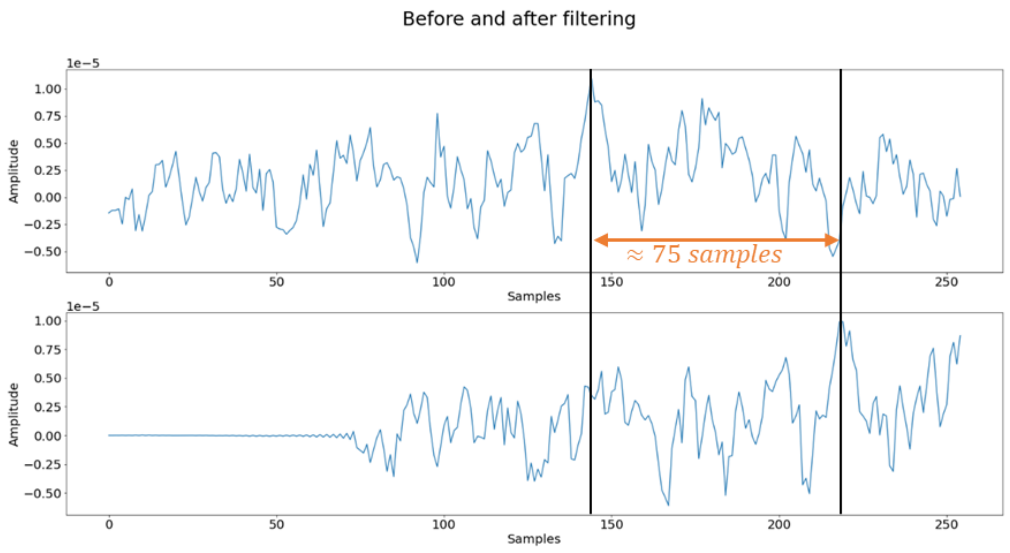


Figure 4.21.- Illustrative example to show the delay of a 150 order FIR filter. The filter is an all-pass so that it is easier to see the delay effect. Figure created by the author.

The filters that will be used in the project will have a window length of **150 samples** and will employ a **Hamming window**, with a **delay of 372.5 ms (75 samples)**. Notice that these filters can be applied for filtering signals in real-time. As an example, the following figures show the PSD (compare it to Figure 4.10.) and the EEG session of a filter with a bandwidth between 15 Hz and 26 Hz on Subject C:

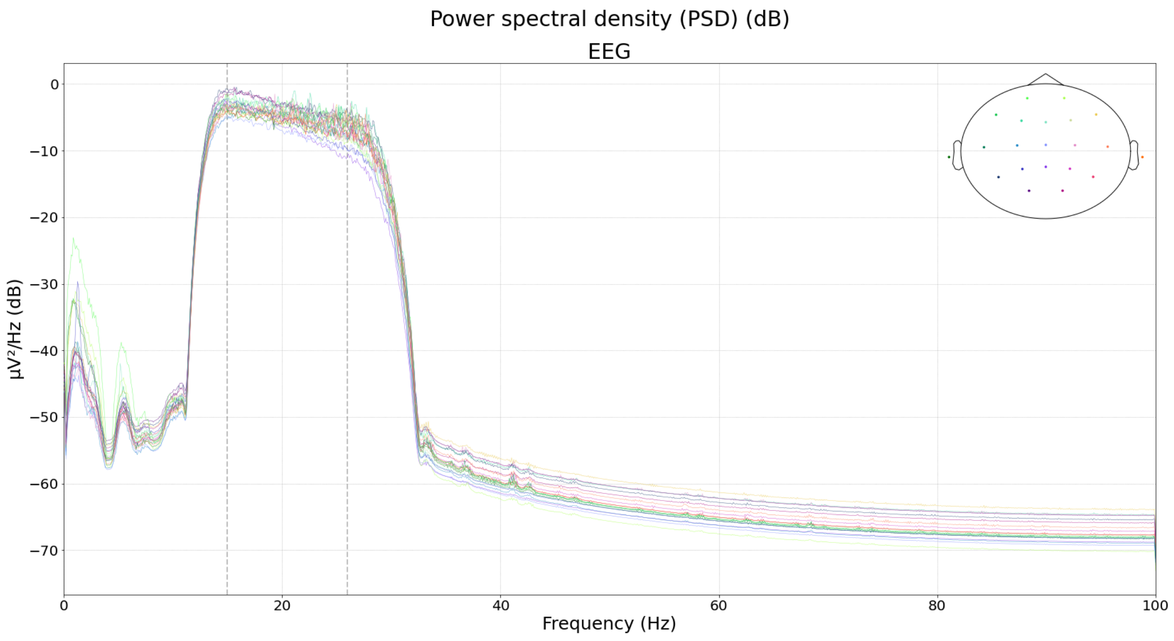


Figure 4.22.- PSD of the signals of each EEG channel used in dB after filtering the data. Each color represents a channel. X-axis shows the frequency in Hz (20 Hz per division) and the Y-axis shows the µV^2/Hz in dB (10 dB per division). The dashed lines represent the cutoff frequencies. Figure created by the author.

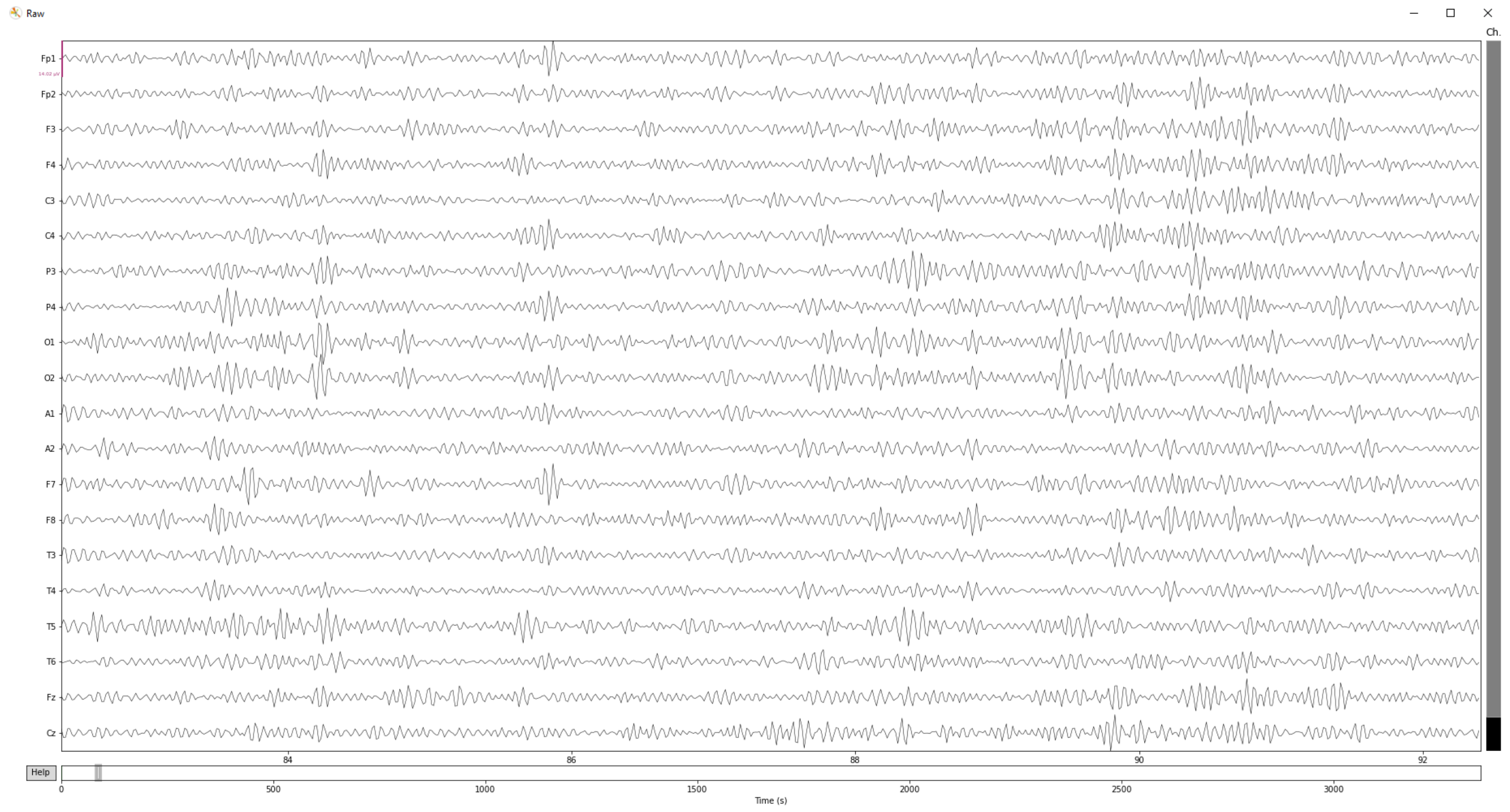


Figure 4.23.- EEG visualization of the filtered signals of each electrode of the EEG. The scale of the signals if (14.02 µV). The X-axis shows the time in seconds. Figure created by the author.

4.6. Preprocessing

In the previous subsections, two different preprocessing techniques have been carried out. The first one is the **common average reference**, which filters the data spatially to reduce the undesired smearing effect as it localizes with better precision the cortical sources responsible for the mental activity. The second one is the **band-pass FIR filter**, which maintains the specified frequency band and wipes out problematic regions of the spectrum.

Besides applying the two previous methods, other steps should be taken in order to clean the signals from artifacts and prepare the data to improve the overall quality of the extracted features. Therefore, two more helpful preprocessing procedures are going to be discussed in this subsection: the **independent component analysis (ICA**) and the **data segmentation and annotation**.

4.6.1. Independent component analysis (ICA)

As its name indicates, the independent component analysis (ICA) is a preprocessing tool capable of detecting and separating a defined number of statistically independent elements that composes a signal without having prior knowledge of the signal's content (blind source separation) [91]. As the EEG contains multiple electrodes that are distributed strategically around the scalp, the sources of these components can be localized. This technique is suitable for detecting and eliminating artifacts, such as the eye-blinks or the electromyographic activity of facial muscles, without altering the valuable signal [4].

The ICA seeks a linear transformation to reduce as much as possible the statistical dependence of the signal's components [92][93][94]. Consider that the EEG acquires signals of **n** samples from **M** electrodes; these signals can be arranged as a matrix $\mathbf{X = [x_1[n], x_2[n], \ldots, x_M[n]]^T}$, where **T** denotes the transpose operator. This matrix can be obtained from a linear combination of **M'** components of the signals, arranged as a matrix $\mathbf{S = [s_1[n], s_2[n], \ldots, s_{M'}[n]]^T}$. The mixing matrix **A**, which is the one linking together both matrixes, has a size of **(M,M')**. Therefore, any EEG signals signal can be obtained as follows:

$$X_{(M,n)} = A_{(M,M')} \cdot S_{(M',n)} \qquad \textbf{(Eq. 4.12)}$$

The objective of the analysis is to find the S matrix so that the components are identified; hence, matrix S must be isolated:

$$S = A^{-1} \cdot A \cdot S = A^{-1} \cdot X \qquad \textbf{(Eq. 4.13)}$$

However, the inverse matrix of A (**A**$^{-1}$) is unknown, and its value can be approximated with different mathematical and statistical methods. Therefore, the ICA estimates the most fitting matrix **W** (≈A^{-1}) that maximizes the non-Gaussianity and the independence of the components. Once this matrix is calculated, the components can be computed:

$$\boldsymbol{S'} = \boldsymbol{W} \cdot \boldsymbol{A} \cdot \boldsymbol{S} = \boldsymbol{W} \cdot \boldsymbol{X} \approx \boldsymbol{S} \qquad \textbf{(Eq. 4.14)}$$

Now that the approximated components are determined, it is possible to reconstruct the signal by zeroing the rows that correspond to undesired artifacts:

$$\boldsymbol{X_{without\_artifacts}} = \boldsymbol{W^{-1}} \cdot \boldsymbol{S'_{without\_artifacts}} \qquad \textbf{(Eq. 4.15)}$$

To exemplify it, a process to remove eye-blinking from Subject C will be described below to demonstrate the utility of this algorithm when it comes to eliminating artifacts that could deteriorate the classifier’s performance.

The first step is to choose the number of components to estimate, which cannot surpass the number of EEG electrodes, for instance, ten elements. The MNE library allows performing the ICA visually so that it is possible to localize the sources of its components on a topographical map, as can be seen in Figure 4.24.

From all the components, ICA000 and ICA002 seem to have a strong activity in the frontal area, which could signify a blinking. In order to confirm this, the component signals are plotted in Figure 4.25., where it can be seen that ICA000 and ICA002 indeed corresponded to a blinking. Nevertheless, only ICA000 will be eliminated to reconstruct the signal since ICA002 has higher frequency signals attached to it that could come from a mental source.

Figure 4.26. and Figure 4.27. show the before and after effects of employing the ICA, respectively. The red marks indicate the presence of a blink, which is manifested with abnormal high amplitude, especially in electrodes Fp1 and Fp2. The green ovals indicate the previous positions of the blinks, which disappeared from the signal thanks to the ICA. Furthermore, notice that the rest of the recording remains unchanged.

This same analysis could have been performed to remove other artifacts, such as muscular activity.

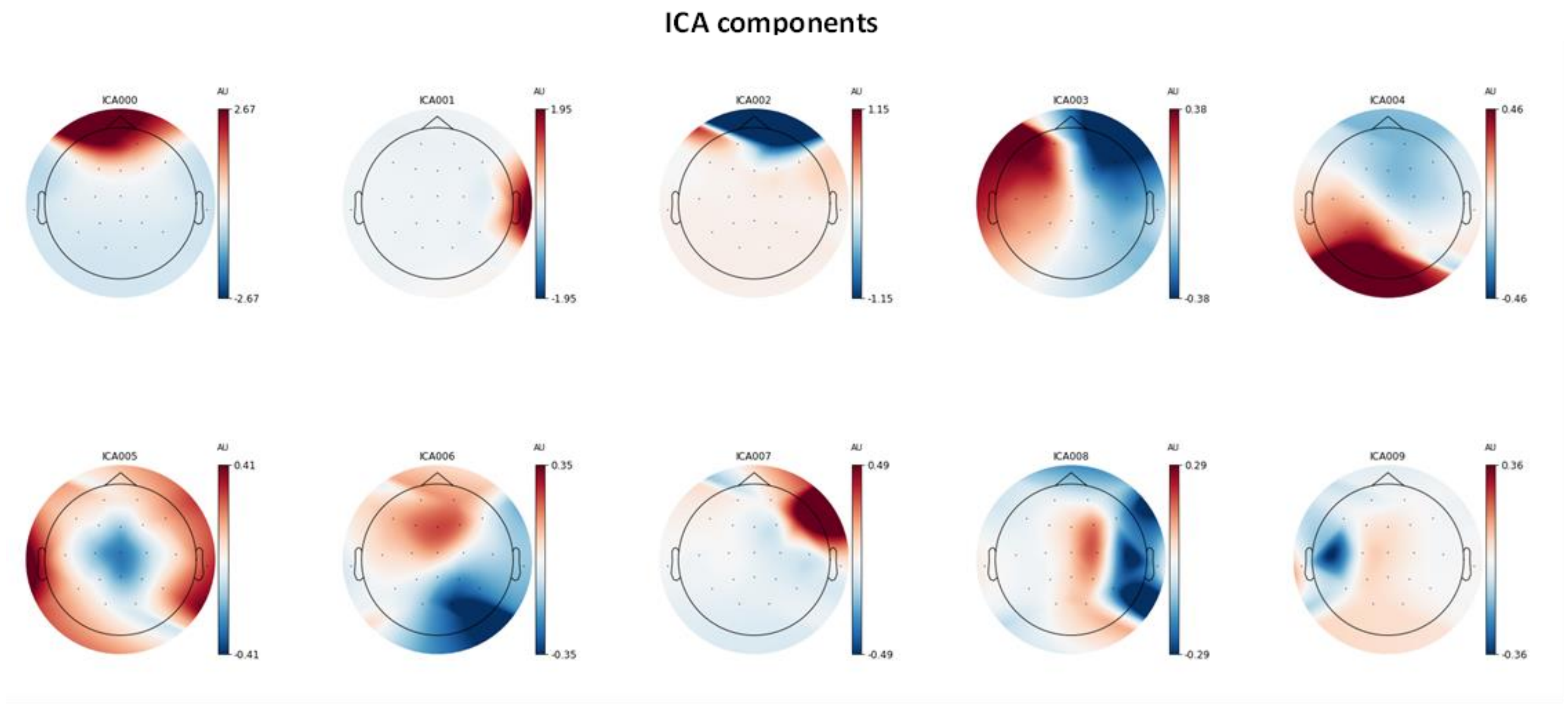


Figure 4.24.- Subject C. Topographical maps displaying the sources of 10 independent components computed with the ICA algorithm. Figure created by the author.

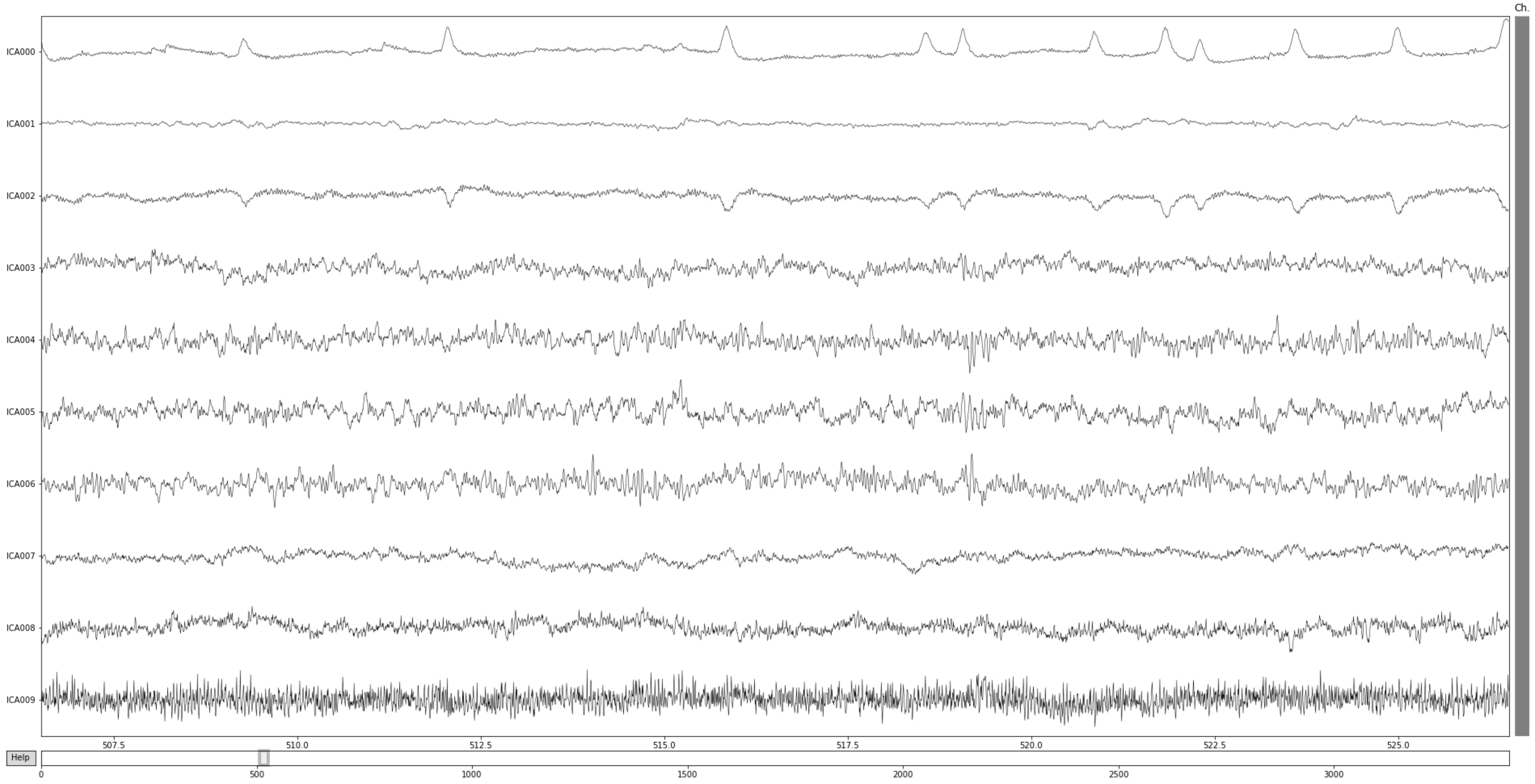


Figure 4.25.- Subject C. The signals of the 10 independent components computed with the ICA algorithm. Figure created by the author.

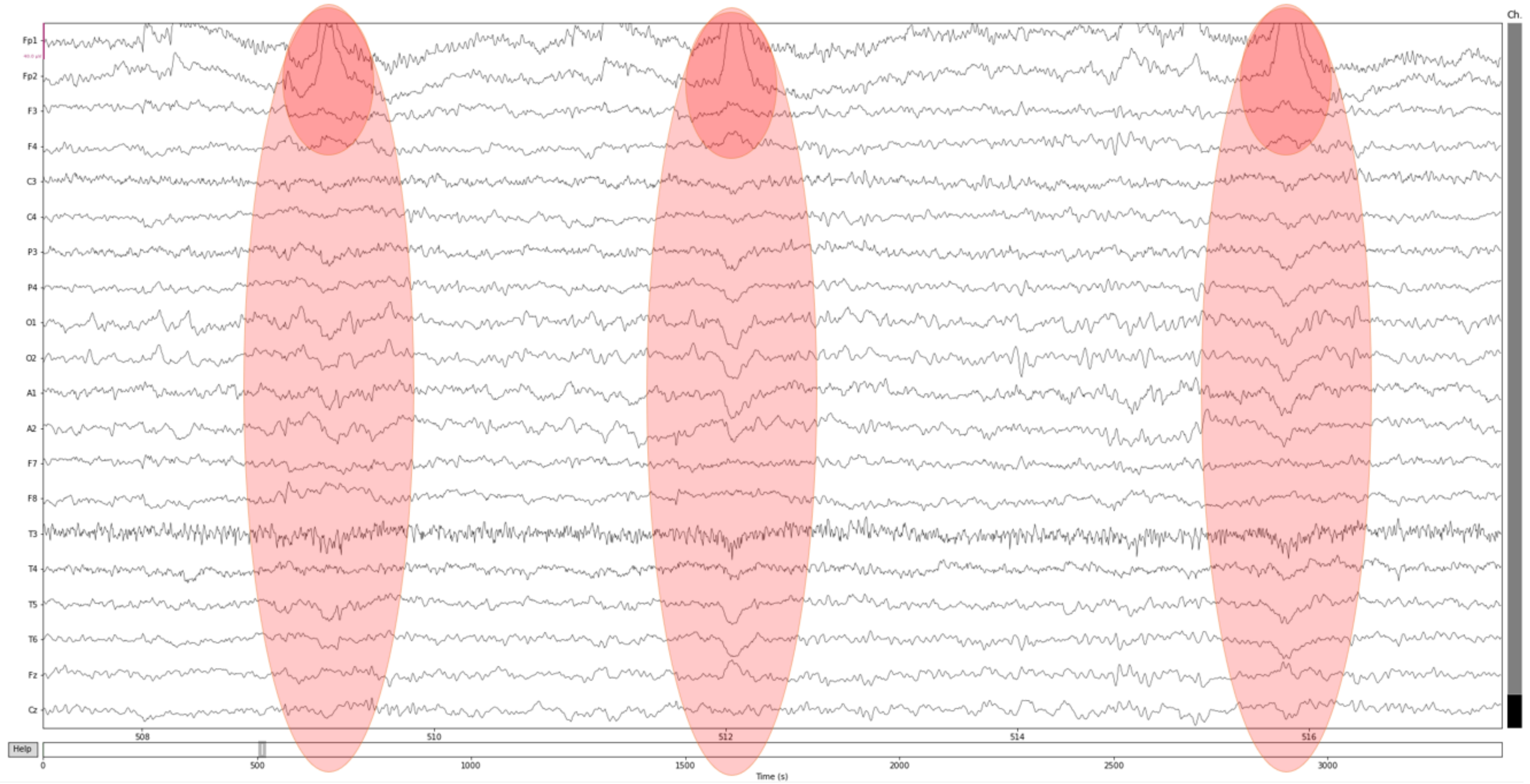


Figure 4.26.- Subject C. Raw data before applying the ICA. At channels Fp1 and Fp2 it is possible to see clear eye-blinking (red). Figure created by the author.

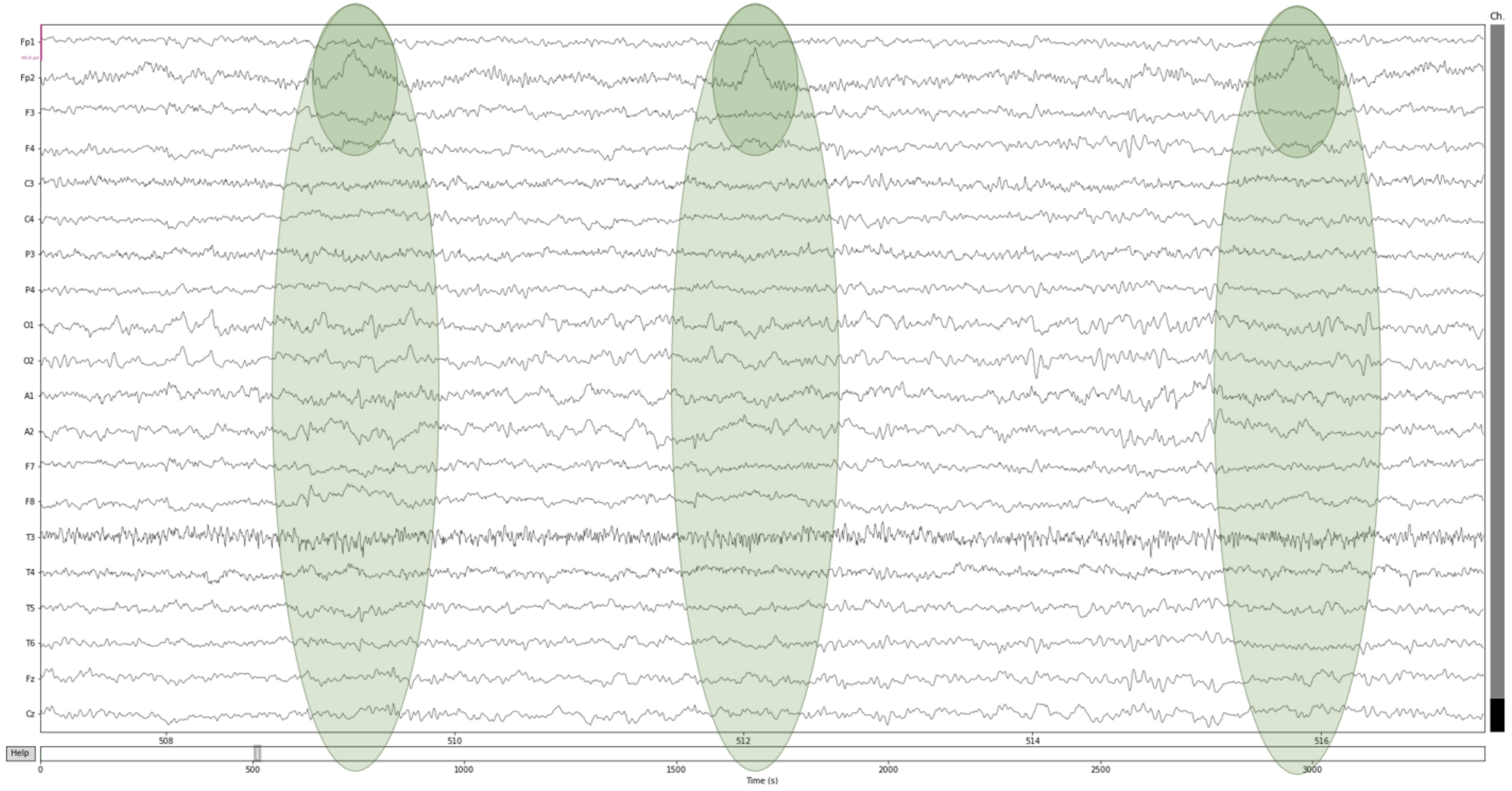


Figure 4.27.- Subject C. Raw data after applying the ICA (same signals as the previous figure). The eye-blinking disappeared from all channels (green). Figure created by the author.

4.6.2. Data segmentation and annotation

While the participants were carrying out the trials, they constantly followed the indications of the eGUI, which saved all the taken actions inside a marker vector (described in Table 4.2.). There are around six hundred left and right-hand imagery events in each EEG session used to train the AI models. In the following figure, it is possible to see some of the markers that appeared throughout the recording. The blue and the orange bands represent the indication to execute left and right-hand imagery, respectively.

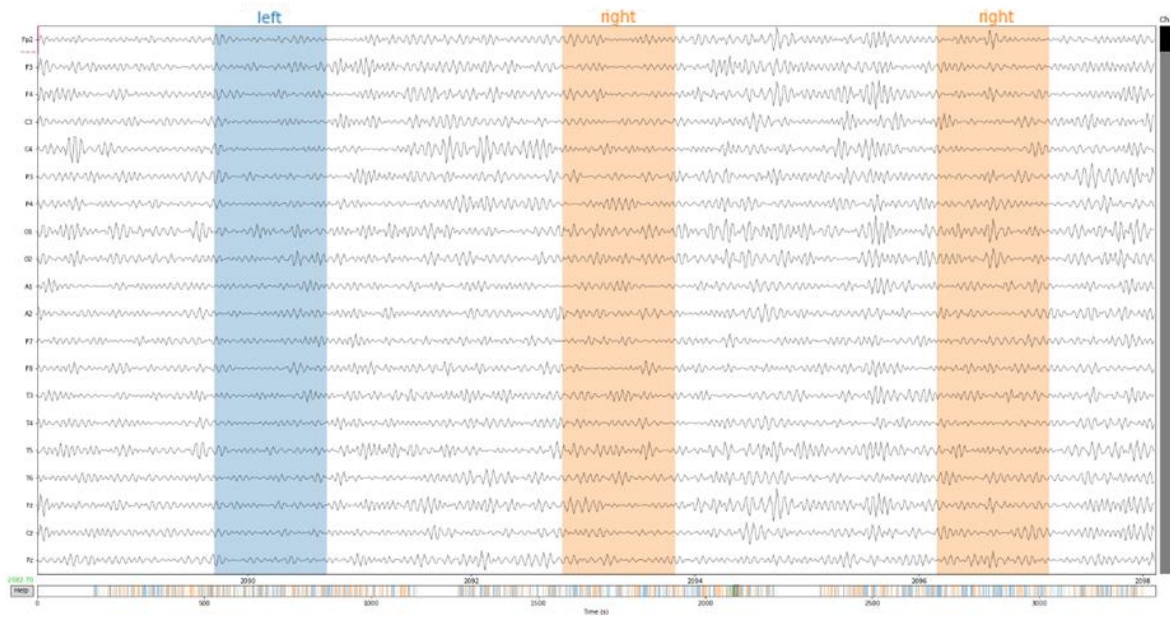


Figure 4.28.- EEG visualization of the filtered signals with the markers overlayed. Blue indicates a left-hand indication. Orange indicates a right-hand indication. The bar below shows all the markers that appeared during the session. Figure created by the author.

One might think that the mental activity starts right after the marker; however, that is not the case, as there is a reaction time from the subject that introduces a delay between the indication and the actual MI task. This retard can be estimated by looking at the ERS/ERD maps when the Beta band ERS begins. Furthermore, the length of each segment must be sufficient to grasp all the ERS (see the time window length column in Table 4.3.). The MNE library refers to each segment as an epoch.

After segmenting all the data, it is important to label each epoch. Without these annotations, the AI models proposed would not be able to distinguish the classes. The next figure shows a clustering of all the segmented data, ignoring other parts of the signal. Numbers 1 and 2 represent left and right-hand imagery epochs, respectively.

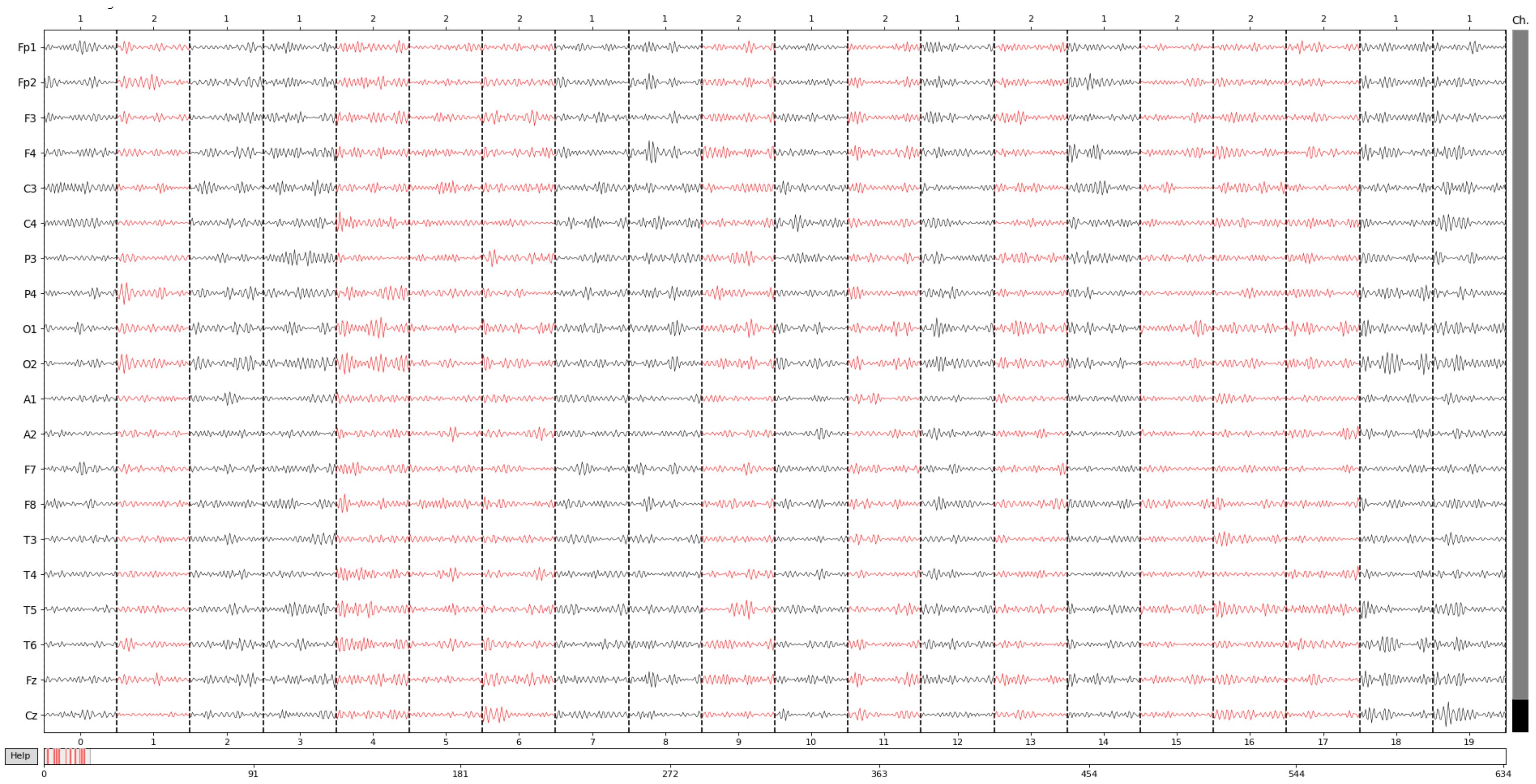


Figure 4.29.- Clustering of all the segmented and annotated data. Numbers 1 (black) and 2 (red) represent left and right-hand imagery epochs, respectively. The X-axis represents the number of epochs in the session. Figure created by the author.

4.7. Signal processing and feature extraction

The last step before training the AI models is to extract the most characteristic features that define the nature of the signals in the most distinctive way. By understanding the different techniques available to do so, it is possible to increase the overall performance of the MI-BCI considerably. The common spatial patterns (CSP) and the statistical features are used to train Machine Learning models, and the short-time Fourier transform to train the Deep Learning models. Notice that the two first methods also reduce the size of the input data (dimensionality reduction).

4.7.1. Common spatial patterns (CSP)

The common spatial patterns (CSP) technique is one of the most practical feature extraction methods for MI applications, enabling a more effective differentiation between two classes, such as the left and the right-hand imagery. With the CSP algorithm, it is possible to find a set of spatial filters that will highlight or de-emphasize the activity of various areas of the brain in order to maximize the variance of one class while minimizing the other one's variance. Consequently, both categories are more distinguishable [95].

This algorithm must be trained with labeled epochs (segmented and annotated trials) to find the optimal spatial filters weights with the highest inter-class variance. Therefore, it is a supervised technique that applies a transformation to the data to find the most suitable spatial features of the signals [96].

Fundamentally, the CSP can be obtained after the diagonalization of two covariance matrices, one for each hand imagery [97][98][99]. Being **X** a matrix that represents windowed EEG signals with **M** channels and **N** samples, the normalized covariance matrix is calculated as follows:

$$\boldsymbol{C} = \frac{\boldsymbol{X} \cdot \boldsymbol{X^T}}{\boldsymbol{trace}(\boldsymbol{X} \cdot \boldsymbol{X^T})} \qquad \textbf{(Eq. 4.16)}$$

Where **T** indicates the transposed matrix and **trace()** is an operation that sums up all the diagonal elements of the matrix. This calculation is done for all the left and right-hand epochs separately. Subsequently, the computed normalized covariance matrix is averaged for each class, and, by summing them, the compound covariance is obtained:

$$\boldsymbol{C_c} = \overline{\boldsymbol{C_L}} + \overline{\boldsymbol{C_R}} \qquad \textbf{(Eq. 4.17)}$$

Where $\overline{\boldsymbol{C_L}}$ and $\overline{\boldsymbol{C_R}}$ are the average covariance matrices for the left and right-hand imagery classes, respectively.

The next step for diagonalizing the matrices is to find the eigenvectors and eigenvalues of ***C**c*, which can be obtained with the following formulas [100]:

$$\boldsymbol{A \cdot \vec{v} = \lambda \cdot \vec{v}} \qquad \textbf{(Eq. 4.18)}$$

$$\boldsymbol{A \cdot \vec{v} - \lambda \cdot I \cdot \vec{v} = 0} \qquad \textbf{(Eq. 4.19)}$$

$$\boldsymbol{(A - \lambda \cdot I) \cdot \vec{v} = 0} \qquad \textbf{(Eq. 4.20)}$$

$$\boldsymbol{det(A - \lambda \cdot I) = 0} \qquad \textbf{(Eq. 4.21)}$$

Where ***A*** represents a matrix, $\boldsymbol{\vec{v}}$ the eigenvectors, $\boldsymbol{\lambda}$ the eigenvalues, ***I*** the identity matrix, and **det()** the determinant of a matrix. Therefore, $\boldsymbol{C_c}$ can be represented with its eigenvectors ($\boldsymbol{U_c}$) and eigenvalues ($\boldsymbol{\lambda_c}$) matrices:

$$\boldsymbol{C_c = U_c \cdot \lambda_c \cdot U_c^T} \qquad \textbf{(Eq. 4.22)}$$

For diagonalizing the matrix it is necessary to calculate the whitening matrix ***P***:

$$\boldsymbol{P = \frac{U_c^T}{\sqrt{\lambda_c}}} \qquad \textbf{(Eq. 4.23)}$$

Subsequently, the spatial coefficient matrices $\boldsymbol{S_L}$ and $\boldsymbol{S_R}$ can be obtained for each class:

$$\boldsymbol{S_L = P \cdot \overline{C_L} \cdot P^T} \qquad \textbf{(Eq. 4.24)}$$

$$\boldsymbol{S_R = P \cdot \overline{C_R} \cdot P^T} \qquad \textbf{(Eq. 4.25)}$$

In this way, the spatial coefficient matrices have common eigenvectors ***B***:

$$\boldsymbol{S_L = B \cdot \lambda_L \cdot B^T} \qquad \textbf{(Eq. 4.26)}$$

$$\boldsymbol{S_R = B \cdot \lambda_R \cdot B^T} \qquad \textbf{(Eq. 4.27)}$$

$$\boldsymbol{\lambda_L + \lambda_R = I} \qquad \textbf{(Eq. 4.28)}$$

Notice that the sum of the eigenvalues matrices gives the identity matrix; hence, the result of adding up two matching eigenvalues will be one. This interesting property will ensure that the larger an eigenvalue is in one class, the smaller it is for the other. Therefore, the eigenvectors are weighed up contrariwise for each group. By finding the highest (or lowest) eigenvalues $\boldsymbol{\lambda_R}$ and $\boldsymbol{\lambda_L}$, it is possible to determine which are the most discriminative eigenvectors that will ensure a better classification.

For transforming the input matrix ***X*** into the output matrix ***Z***, a projection matrix ***W*** must be computed, as shown in the following equations:

$$\boldsymbol{W} = \left(\boldsymbol{B}^{\boldsymbol{T}} \cdot \boldsymbol{P}\right)^{\boldsymbol{T}} \qquad \textbf{(Eq. 4.29)}$$

$$\boldsymbol{Z} = \boldsymbol{W} \cdot \boldsymbol{X} \qquad \textbf{(Eq. 4.30)}$$

Note that the columns of the inverse matrix of ***W*** (***W***$^{\boldsymbol{-1}}$) correspond common spatial patterns, which are time-invariant vectors.

The subsequent illustration shows the effect of applying the most discriminative CSP. In the beginning, the left and right-hand imagery classes have similar variances. After calculating the CSP and filtering the signal with the patterns corresponding to the highest eigenvalues of each imagery (in this case, CSP1 and CSP2), the inter-class variance is maximized:

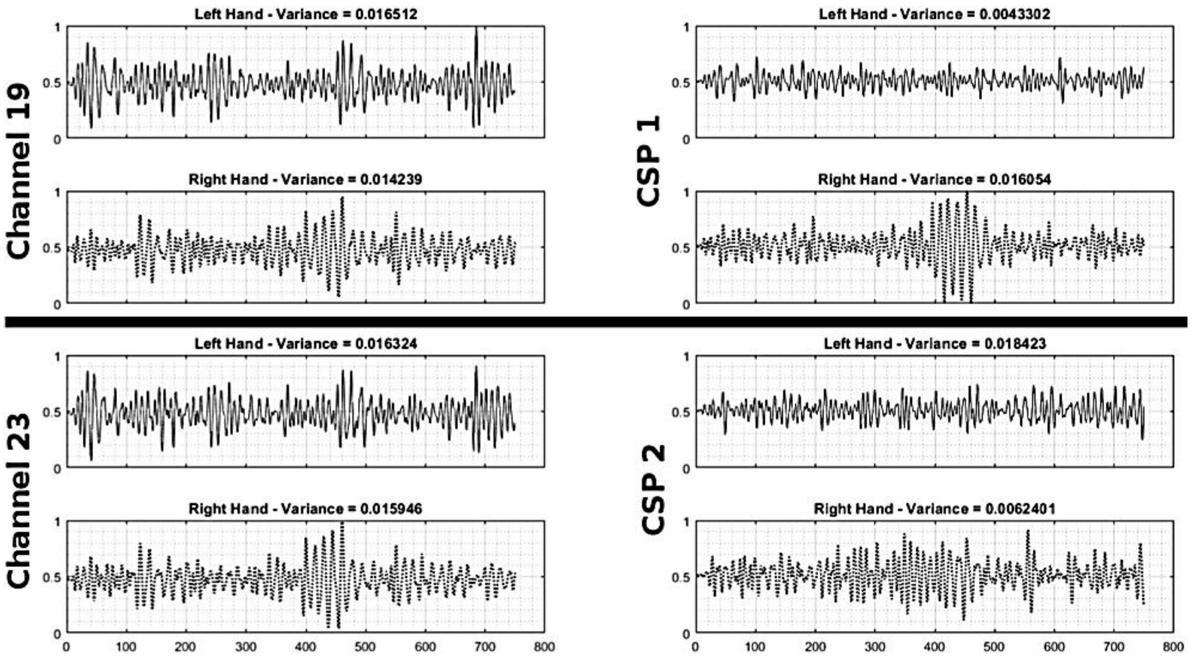


Figure 4.30.- Effect of applying the most discriminative CSP. The signals are more differentiable after using CSP1 and CSP2. Figure extracted from [95].

Finally, for extracting the features from the CSP and reducing the size of the data, the next formula is calculated:

$$f_p = log_{10}\left(\frac{var(Z_p)}{\sum_{i=1}^{2m} var(Z_i)}\right), where\ p = 1, 2, \ldots, 2m \qquad \textbf{(Eq. 4.31)}$$

Where f_p are the feature vectors, ***var()*** correspond to the variance, and **2*m*** depends on the number of filters selected. The subindexes p and i represent the rows of the output matrix ***Z***.

After applying all the previous calculations with the aid of the MNE library, the CSP were obtained. Figure 4.31. illustrates the CSP gotten from Subject C, showing how each behaves on a topographic map. Notice that CSP0 and CSP3 present emphasized patterns in electrodes C4 and C3, respectively; this is a clear indicator that the CSP algorithm can detect left and right-hand imageries correctly. After several trials, it was concluded that having 10 CSP provided better classification results.

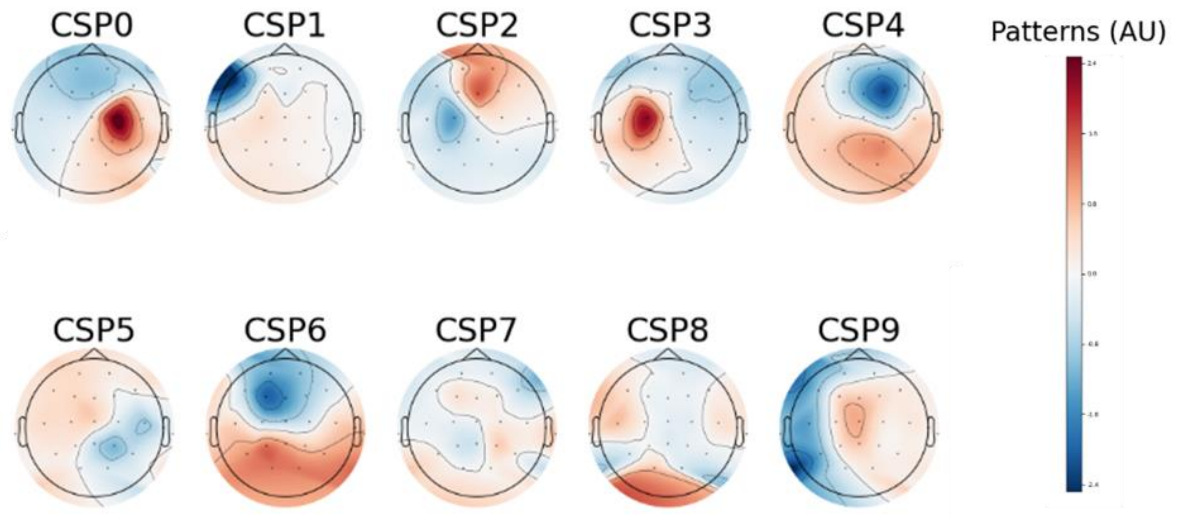


Figure 4.31.- Ten CSP for Subject C. The color bar indicates the weights of the patterns while applying the filters. Figure created by the author.

4.7.2. Short-time Fourier transform (STFT)

One of the AI models proposed in future sections involves using a convolutional neural network, which is usually trained with images; therefore, the EEG signals must undergo a process to describe their contents in a two-dimensional manner. For doing this, the spectrogram of the signals can be computed, a processing tool that enables a time-frequency analysis of the acquired data.

The short-time Fourier transform (STFT) is employed to compute the spectrograph. This mathematical technique divides the signal into multiple time segments with the help of a shifting time window, which is usually a Hamming to avoid spectral leakage and preserve continuousness. Furthermore, an ensuing window does not begin from the final position of the current window as there is an overlap to not lose the spectral information due to the Hamming window. After having all the segmented signals, a discrete Fourier transform (DFT) is calculated for each of them; as a result, the frequential information is obtained over time [101][102].

There is a tradeoff that must be addressed regarding the resolution of the spectrum. The larger the window length used in the STFT is, the better the frequential resolution and the more flawed the temporal resolution is. The contrary effect happens when the window gets shorter. Consequently, one must choose wisely the length of the window and the amount of overlap allowed [103].

The following equation describes how to carry out a STFT [101]:

$$\boldsymbol{S(m,k) = \sum_{n=0}^{N-1} s(n + m \cdot N') \cdot w(n) \cdot e^{-j \cdot \frac{2 \cdot \pi}{N} \cdot n \cdot k}} \qquad \textbf{(Eq. 4.32)}$$

Where $\boldsymbol{k}$ is a number going from 0 to N-1, $\boldsymbol{S(m,k)}$ specifies the time-frequency spectrograms for an index m, $\boldsymbol{N}$ is the window length, $\boldsymbol{N'}$ is the window's shifting step, and $\boldsymbol{w(n)}$ involves the window function used. As the coefficients obtained from the previous equation are complex numbers, the magnitude of $S(m,k)$ must be calculated to obtain the spectrogram.

However, performing this operation for all the electrodes can be computationally expensive and could not allow a real-time analysis. Hence, only the central EEG electrodes (**C3**, **C4**, and **Cz**) will be chosen, as they are the most relevant for capturing the MI. For instance, the following spectrograms correspond to a right-hand imagery epoch, where clear activity can be seen in C3 as expected.

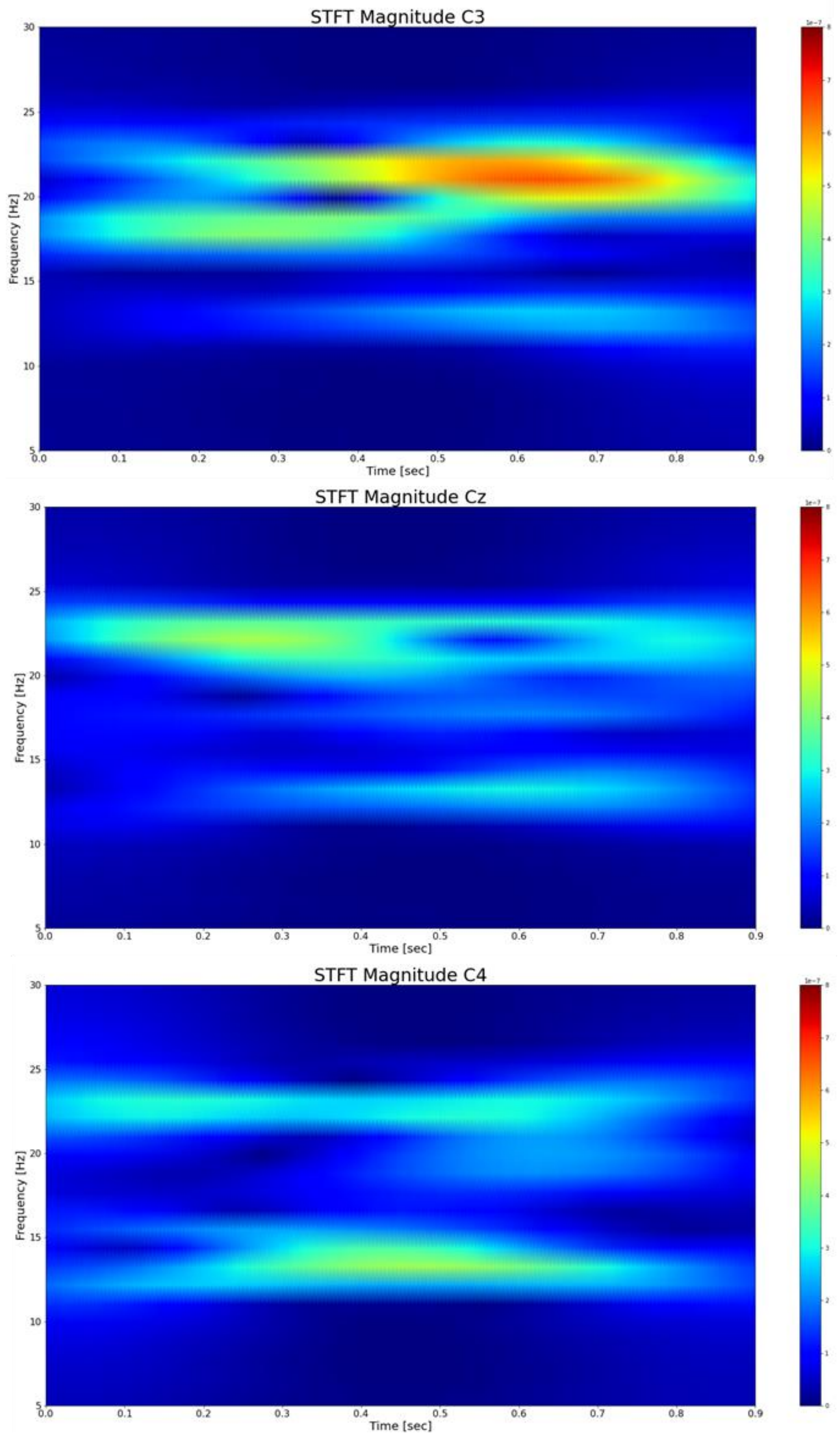


Figure 4.32.- Spectrograms created from the STFT. The effect of a right-hand imagery in time and frequency for the electrodes C3, C4, and Cz. X-axis indicates the time (0.1 seconds per division), Y-axis indicates the frequency (5 Hz per division) and the color bar indicates the magnitude. Figure created by the author.

4.7.3. Statistical Features

There are statistical parameters of the signal that can be useful for detecting the imageries or the passive state. Mainly they will be applied only to the central EEG electrodes (C3, C4, and Cz) as they are the ones of more importance in hand imagery tasks. In this subsection, they will be listed and concisely explained.

4.7.3.1. Probability density function (PDF)

The probability density function (PDF) determines the likelihood that a particular sample has of becoming an outcome. In other words, the Y-axis corresponds to the density of probability, showing the odds of obtaining values found on the X-axis. In the case of EEG signals, this probabilistic distribution can be estimated by looking at the chances of having a specific voltage value. The following image shows the PDF estimation for a certain epoch at the channels C3, C4, and Cz:

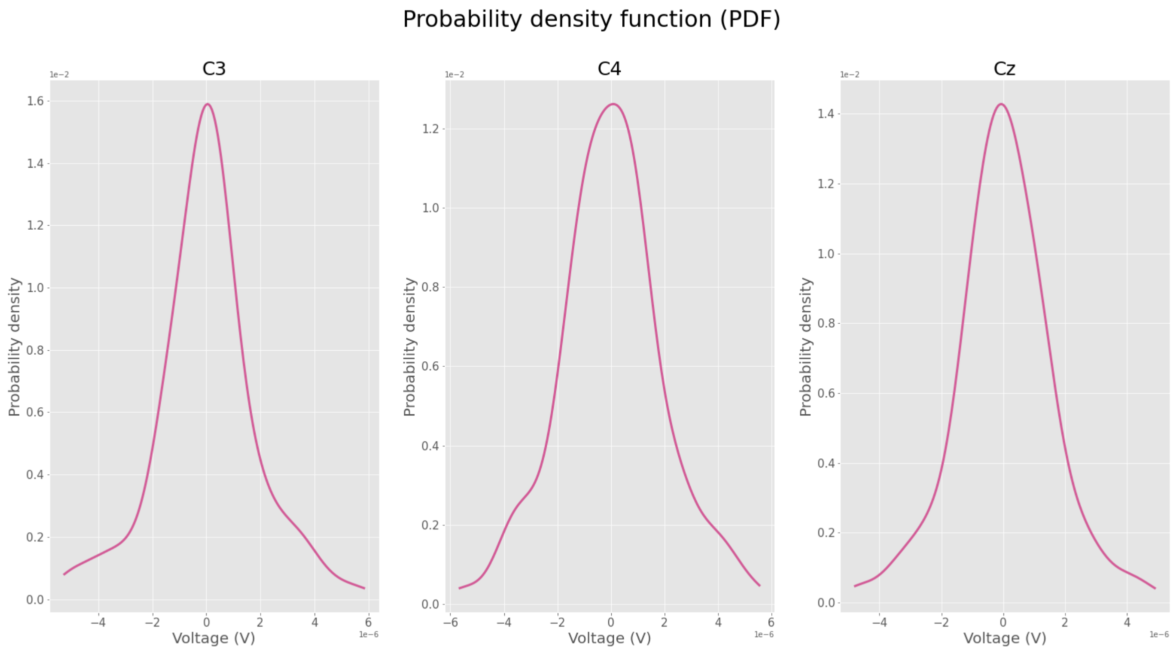


Figure 4.33.- Probabilistic density function for a certain epoch of Subject C. Channels C3, C4, and Cz. X-axis indicates the voltage ($2{\cdot}10^{-6}$ V per division). Y-axis indicates the probabilistic density. Figure created by the author.

Plentiful statistical parameters can be found by analyzing the PDF, for instance, the mean, the variance, the skewness, or the Kurtosis.

4.7.3.2. Kurtosis

Kurtosis is a statistical parameter that defines the peakedness and the tailedness of a distribution relative to the normal distribution [104]. This coefficient can be calculated using the following formula:

$$S_{kr} = \frac{1}{n-1} \cdot \frac{\sum_{i=1}^{n}(X_i - \overline{X})^4}{S^4} \qquad \textbf{(Eq. 4.33)}$$

Where $\overline{X}$ is the mean of the samples, n is the total number of samples, and S is the sample standard deviation. If the result is greater than 3, the distribution is Leptokurtic; if it is smaller than 3, it is Platykurtic; and if it is equal to 3, it is Mesokurtic or normal. Normally, three units are subtracted to speak relative to zero. Therefore, a positive Kurtosis would indicate that the distribution is denser around the mean compared to the normal distribution; the opposite happens if the Kurtosis is negative [105].

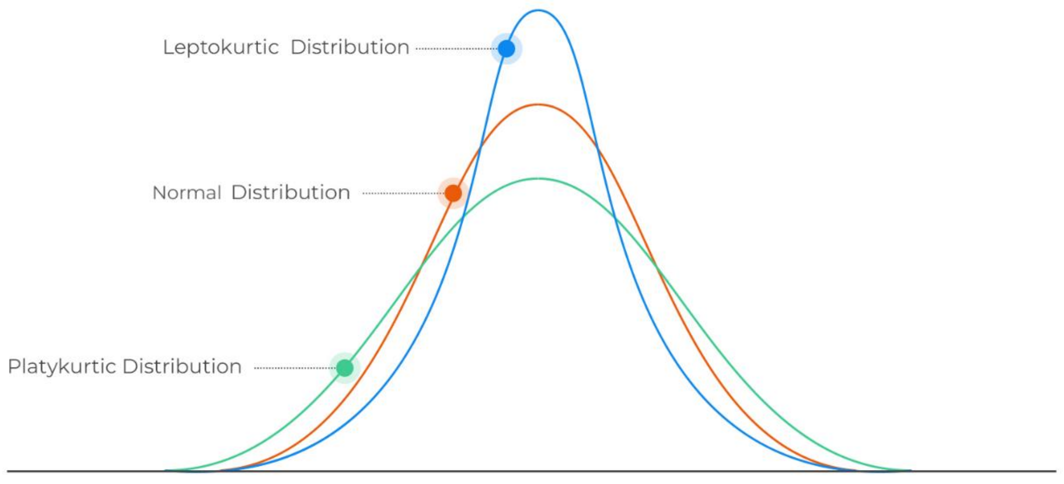


Figure 4.34.- A representation of Leptokurtic, Platykurtic, and normal distributions. Figure extracted from [106].

4.7.3.3. Skewness

The skewness defines the symmetry of the distribution and can quantify any distortion or asymmetry by comparing the distribution relative to the normal one [104]. It is calculated as follows:

$$S_{sk} = \frac{1}{n-1} \cdot \frac{\sum_{i=1}^{n}(X_i - \overline{X})^3}{S^3} \qquad \textbf{(Eq. 4.34)}$$

Where $\overline{\boldsymbol{X}}$ is the mean of the samples, $\boldsymbol{n}$ is the total number of samples, and $\boldsymbol{S}$ is the sample standard deviation. The skewness can be positive, negative, or symmetrical.

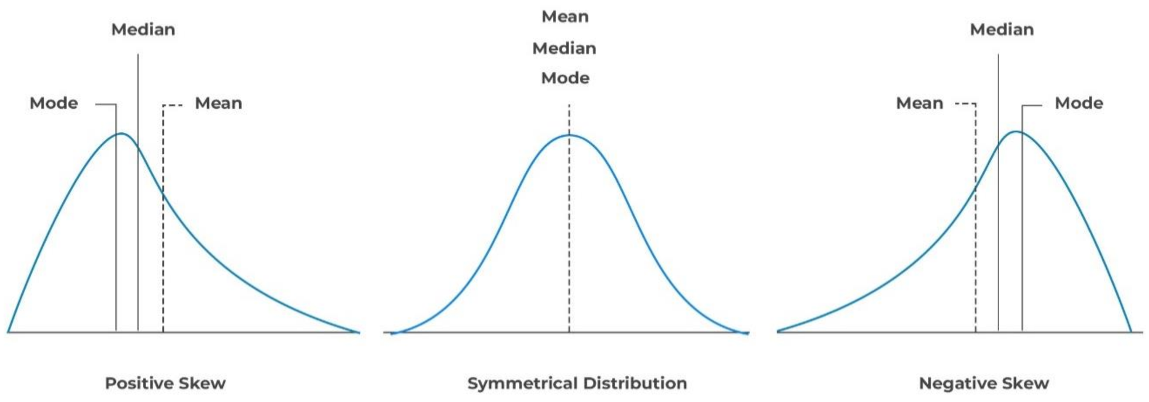


Figure 4.35.- A representation of the effect of skewness. Figure extracted from [106].

4.7.3.4. Energy

The energy of a discrete signal is defined by the sum of the squared samples. Hence, the energy of a signal $x(n)$ can be calculated with the subsequent formula [106]:

$$\boldsymbol{E} = \sum_{\boldsymbol{n}=-\infty}^{\infty} |\boldsymbol{x}(\boldsymbol{n})|^2 \qquad \textbf{(Eq. 4.35)}$$

4.7.3.5. Wavelet-based Entropy

The entropy measures the randomness of a signal, and it is one of the most characteristic features that can be obtained from EEG data. Besides the general definition of entropy, two more variations will be computed: the **Shannon entropy** and the **Log Energy-entropy**. All of them will be quantified after applying a **Morlet wavelet transform**, as indicated in [107] and [108].

A wavelet transform is based on a decomposition of the temporal and frequential contents of a signal by stretching, compressing, and shifting a waveform known as a wavelet. The coefficients that describe the segmented data can be obtained thanks to the PyWavelets library.

The formulas that define the entropies are presented below:

$$H_{En} = -\sum_{i=1}^{M} p_i \cdot log_2(p_i) \qquad \text{(Eq. 4.36)}$$

$$H_{ShannonEn} = -\sum_{i=1}^{M} p_i^2 \cdot \left(log_2(p_i)\right)^2 \qquad \text{(Eq. 4.37)}$$

$$H_{logEn} = -\sum_{i=1}^{M} \left(log_2(p_i)\right)^2 \qquad \text{(Eq. 4.38)}$$

Where p_i can be obtained as:

$$p_i = \frac{|W(a_i, t)|}{\sum_{j=1}^{M} W(a_j, t)} \qquad \text{(Eq. 4.39)}$$

Assuming that $W(a_i, t)$ (i =1, 2, …, M) is a set of wavelet coefficients defined as a matrix of size **(M,N)**, being **M** the scales of the transform and **N** the time points.

$$W_{MxN} = \begin{bmatrix} w_{a_1,1} & \cdots & w_{a_1,N} \\ \vdots & \ddots & \vdots \\ w_{a_M,1} & \cdots & w_{a_M,N} \end{bmatrix} \qquad \text{(Eq. 4.40)}$$

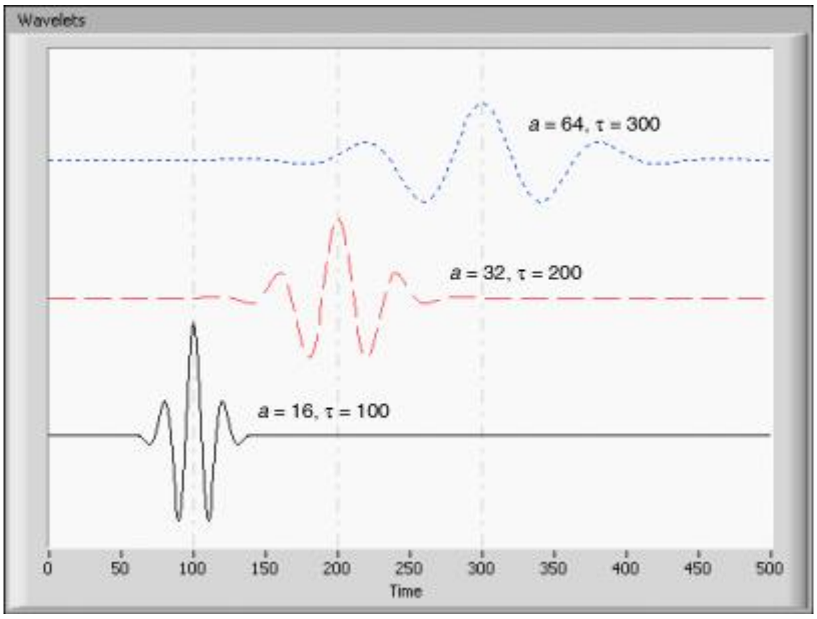


Figure 4.36.- Example of a Morlet wavelet being elongated and shifted. Figure extracted from [109].

4.8. Artificial Intelligence models

The last step for implementing the MI-BCI is to train AI models that will classify the three classes (left, right, and rest) accurately and precisely. The end goal is to find two different AI architectures, one for distinguishing between left and right-hand imageries and the other for differentiating imageries from the resting state (see Figure 4.2.). For doing so, manifold Machine and Deep Learning methods will be tested to see which are the most suitable. In this subsection, the most relevant models will be presented and described.

4.8.1. Train and test data distribution

From all the segmented data, **80 % will be used to train** the models and the remaining **20 % to test** them; this ratio will ensure a correct evaluation of the models. It is important to clarify that the data will be shuffled randomly. For the Machine Learning architectures, a **K-fold Cross-validation** will be used to have a more accurate estimate of the model's prediction performance. This resampling method divides the data into K groups, training the model with (K-1) groups and testing it with the remaining fold. This process is repeated for all K possible combinations, and the performance is then averaged from each iteration's result.

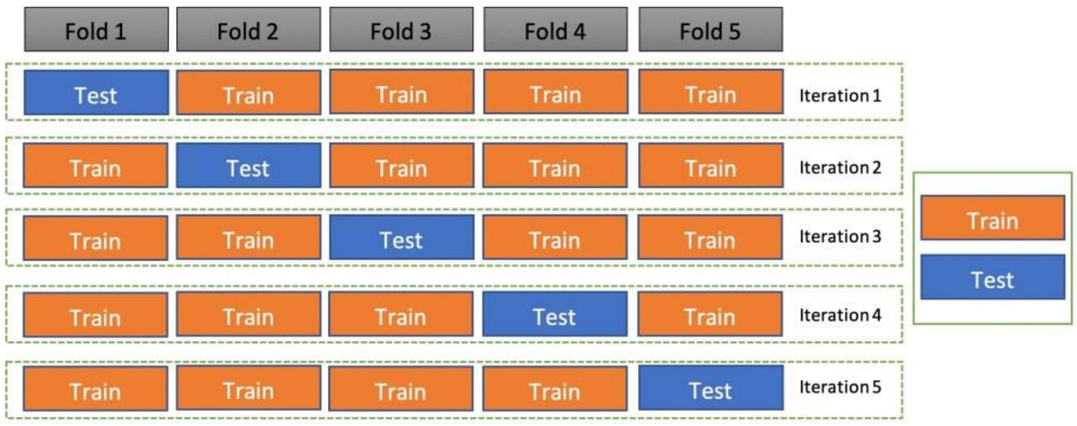


Figure 4.37.- K-fold cross-validation. Figure extracted from [110].

Furthermore, remember that one session from each participant will be used for implementing a real-time application; therefore, that session will not be employed for training the offline models.

4.8.2. Right vs Left imagery

The left and right-hand imagery classification will be tested mainly with Machine Learning methods. Among all the implemented architectures, the best results have been obtained with the following ones:

- Linear Discriminant Analysis (LDA).
- Support Vector Machines (SVM).
- Random Forest (RF).

Each of the previous classifiers will be trained with the Common Spatial Patterns (CSP).

4.8.3. Imagery vs Resting state

The imagery and resting-state classification will be checked out with the previously mentioned Machine Learning models (Right vs Left imagery), but instead of using CSP they will employ the statistical features. Bear in mind that, before inserting the features inside the Machine Learning models, the statistical characteristics must be scaled, for instance, with a Standard Scaler. Furthermore, a Deep learning technique based on Convolutional Neural Networks (CNN) will also be analyzed; this classifier will use the features extracted from the Short-Time Fourier Transform (STFT).

4.8.4. Model tuning

The model's hyperparameters were selected with the help of appropriate tuning techniques in order to enhance the classifier's performance. This optimization is achieved with tools available in the AI libraries of Python (Scikit-learn and TensorFlow), such as the **Random Grid Search** and the **Bayesian Optimization**; with them, one can assess the effects of choosing different hyperparameters.

4.8.5. Performance metrics

Knowing how to evaluate the AI models is crucial in order to see if they perform correctly while implemented. One might think that the **accuracy**, which indicates the correct prediction over the total predictions, is the only metric that matters, but this depends on the application. Accuracy is relevant for gaining an insight into the overall behavior of the model; however, this metric doesn't take into account how well each class is predicted. For instance, in the case of this project, it is particularly important to minimize false MI detections, and it is more acceptable not detecting a MI when it is actually happening; this cannot be measured with accuracy.

Most evaluation metrics can be obtained from the **confusion matrix**, which contrasts the ground truth labels with the predicted classes of the model. Hence, it is easier to visualize the **True Positives** (TP), the **True Negatives** (TN), the **False Positives** (FP), and the **False Negatives** (TN) of the model.

Table 4.5.- The structure of a confusion matrix. Table created by the author.

Confusion Matrix		Predicted	
		No hand imagery	Hand imagery
Ground truth	No hand imagery	TN	FP
	Hand imagery	FN	TP

Besides accuracy, the most relevant metrics for classification models are:

- **Precision:** It indicates the ratio of true positives and total positives predicted. It is a measure of the exactness of the model.

$$precision = \frac{TP}{TP + FP} \qquad \textbf{(Eq. 4.41)}$$

- **Recall/Sensitivity:** It specifies how well the model predicts true positives over actual positive cases. It is a measure of the completeness of the model.

$$recall = \frac{TP}{TP + FN} \qquad \textbf{(Eq. 4.42)}$$

- **F1 Score:** As there is a tradeoff while choosing precision and recall, the F1 Score combines them by applying a harmonic mean.

$$F1\ score = 2 \cdot \frac{(precision \cdot recall)}{(precision + recall)} \qquad \textbf{(Eq. 4.43)}$$

- **Specificity:** It stipulates how well the model predicts true negatives over actual negative cases.

$$specificity = \frac{TN}{TN + FP} \qquad \textbf{(Eq. 4.44)}$$

4.8.6. Linear Discriminant Analysis (LDA)

Linear discriminant analysis (LDA) is a supervised classification technique widely used in Machine Learning that focuses on maximizing the separability between classes. This objective is done by applying a dimensionality reduction and finding a suitable linear combination of the features. Therefore, the categories are projected into a new axis where they will appear more distinguishable [111]. For doing so, two criteria must be taken into account:

1) Maximize the mean between the categories.
2) Minimize the variation, also known as scatter, of each category.

To simultaneously reflect the previous points, the LDA algorithm optimizes them by working with the following ratio (considering two categories):

$$\boldsymbol{R} = \frac{(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^2}{\boldsymbol{\sigma_1^2} + \boldsymbol{\sigma_2^2}} \qquad \textbf{(Eq. 4.45)}$$

Where $\boldsymbol{\mu}$ is the mean, $\boldsymbol{\sigma^2}$ is the scatter, and the subindices indicate the class. The subsequent illustration show how this method functions:

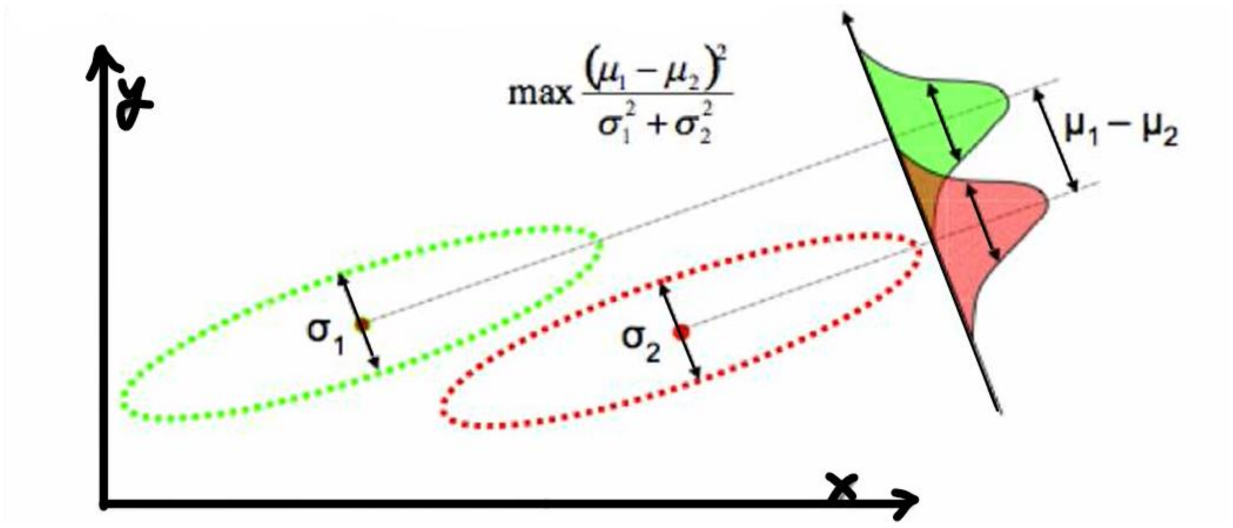


Figure 4.38.- LDA. Projection of the red and green class to another axis where they can be better separated. Figure extracted from [112].

4.8.7. Support Vector Machines (SVM)

The aim of support vector machines (SVM) is to separate the data using hyperplanes, which act as thresholds for classifying the data. Therefore, the classes can be differentiated depending on where the data point falls on the hyperplane. Note that the hyperplane dimensionality depends on the number of features. For instance, two characteristics requires a one-dimensional hyperplane (line), and three features requires a plane [113].

Moreover, it is crucial to have a large **margin**, which indicates the shortest distance between the observations and the hyperplane. The hyperplane should be positioned with an equal distance between the classes to encounter the highest margin. However, if there are outliers, the margins will be reduced substantially, leading to a faulty classifier. For that reason, with the help of iterative methods such as Cross Validation, one can find the best margins by allowing outliers (soft margins). Hence, it is possible to find new data points for each category, known as **support vectors**, that will be considered for finding a better margin. SVM algorithms require a step which enables them to move the data to higher dimensions with tools known as kernels. By doing so, the model's performance increases considerably [114].

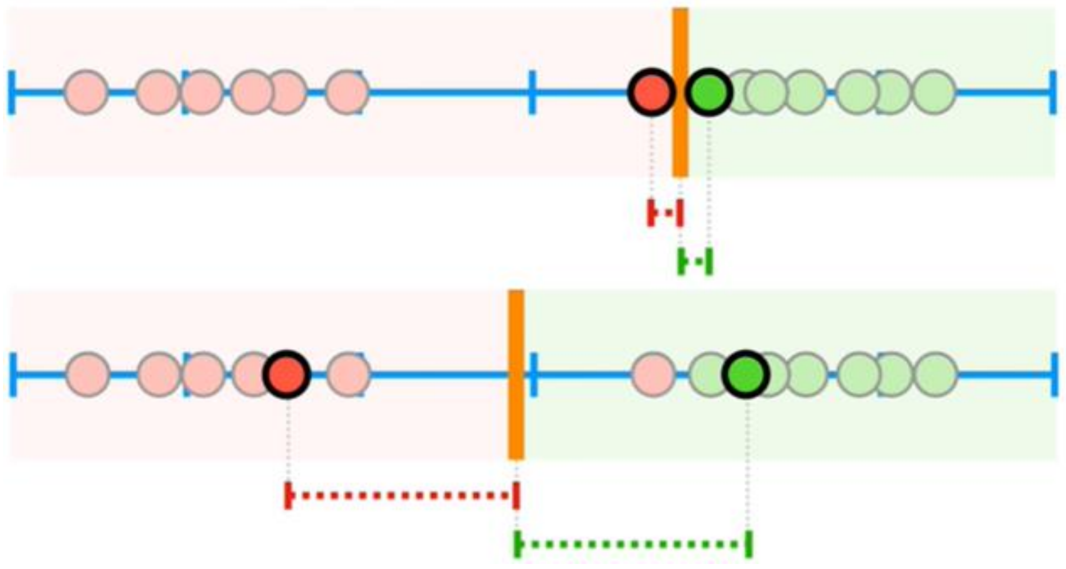


Figure 4.39.- SVM. Margins with and without outliers. The top case shows how small the margins are. The bottom case enlarges the margins by finding the support vectors that gave the best margin, found with cross validation. Figure extracted from [114].

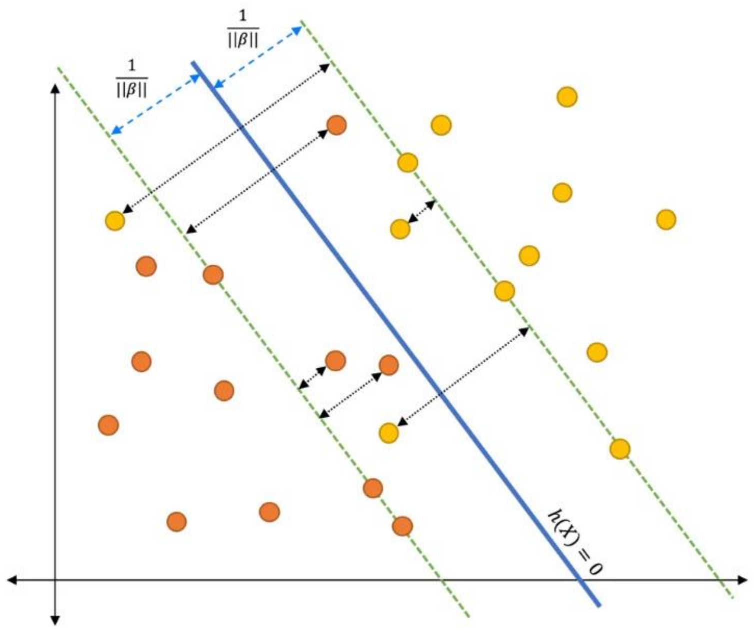
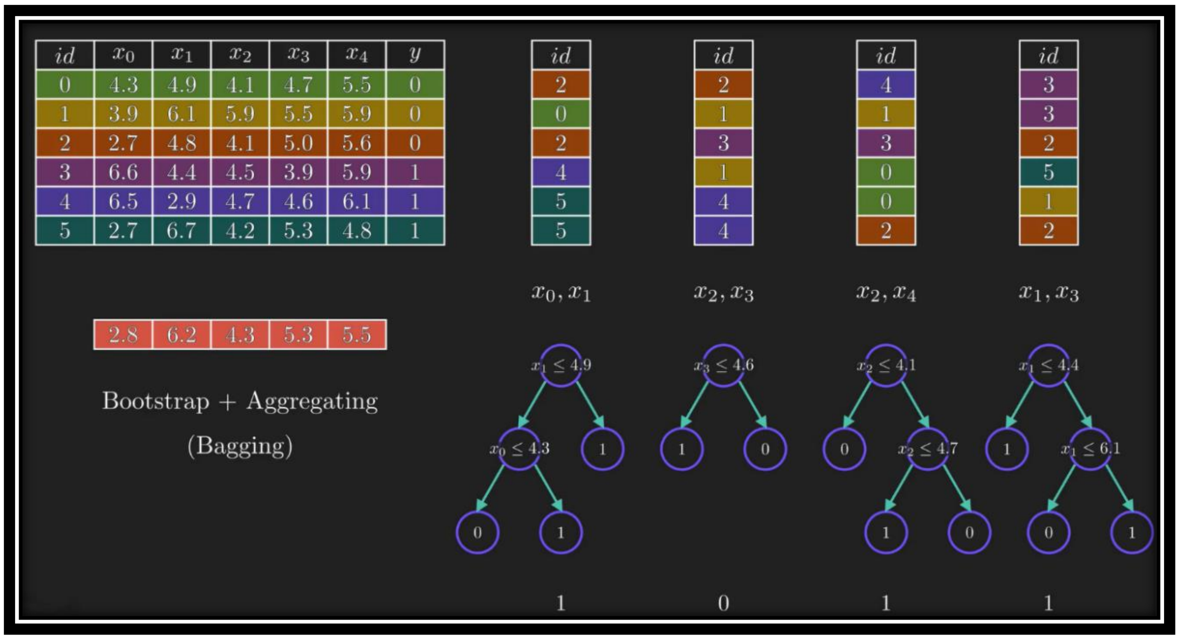


Figure 4.40.- SVM. Hyperplane (blue) and margins (green) drawn. Notice that there are outliers and that the margins are equally distanced. Figure extracted from [115].

4.8.8. Decision Trees and Random Forest (RF)

As its name suggests, a decision tree classifier is a Machine Learning algorithm with a tree-like branching structure that allows data classification. It performs a recursive splitting of the dataset using decision nodes, which contain certain conditions. If they are satisfied, it activates the left path; otherwise, it triggers the right one. This process occurs until a leaf node is reached, which will contain a class label that will indicate the prediction of the decision tree. The best splits can be found after maximizing the entropy gain while training the model.

The drawback of this algorithm is that they are highly sensitive to the training data, resulting in higher variance. Hence, the random forest (RF) technique is used to tackle this problem, which combines multiple decisions trees simultaneously. The first step for training these algorithms is to create various random datasets from the original one; this process is called **bootstrapping**, employed to ensure that the trees are trained with different data. Note that this technique creates datasets with the same amount of rows as the original one, allowing the repetitions of samples. The next step is to arbitrarily pick out a subset of features to train each bootstrapped dataset independently; this process is named **random feature selection**. The last step is to build a decision tree for each dataset. Finally, an **aggregation** algorithm is applied to combine all the predictions from each decision tree and decide on a single output. If bootstrap and aggregation are combined, the procedure is called **bagging**. Everything explained is summarized in the following image:



id	x_0	x_1	x_2	x_3	x_4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

2.8	6.2	4.3	5.3	5.5

id	id	id	id
2	2	4	3
0	1	1	3
2	3	3	2
4	1	0	5
5	4	0	1
5	4	2	2

Figure 4.41.- RF. The dataset is divided into four datasets by using bootstrap and two random features. For each dataset, a decision tree is created. The output of the RF is established by aggregating the results of each decision tree. Figure extracted from [116].

4.8.9. Artificial Neural Networks (ANN)

Inspired by the biological concepts of the nervous system, artificial neural networks (ANN) are a subbranch of Machine Learning and the nucleus of Deep Learning algorithms that consists of the interconnection of nodes, known as **neurons**, that receive and transmit information. The end goal is to train the ANN to activate or inhibit the neurons for finding complex patterns used for carrying out prediction tasks [117].

They are composed of multiple layers with multiple neurons each; the first and last layers are referred to as the input and output, respectively. The remaining ones are the hidden layers; their behavior is so complex that it is nearly impossible for a human to interpret what the model is actually doing; hence, neural networks are known to be black-box systems. If there are manifold hidden layers, the ANN is a deep neural network.

Basically, the inputs of one neuron come from the **weighted sum** of the signals coming from all the neurons of the previous layer, which are real numbers parametrized in each neuron. Then, the neuron processes the result by adding a **bias** and applying an activation function, such as the Sigmoid or the ReLU. It is strongly recommended to introduce nonlinearity to the system with the activation function for the correct functioning of the ANN for solving nonlinear problems. The outcome of this operation defines the value of the neuron. Finally, this calculated parameter will be multiplied by an associated **weight** and introduced to all the neurons of the following layer. The values of the neurons are initialized with the data in the input layer, and the model's results depend on the neurons activated in the outcome layer.

The end goal of ANNs is to find the most appropriate bias and weights that can **minimize a loss/cost** function, which measures how well the algorithm predicts an expected value. This can be achieved by training the AI model, thanks to a process known as **back-propagation**. Hence, during the training session, the data flows forward and backward for a pre-established number of cycles.

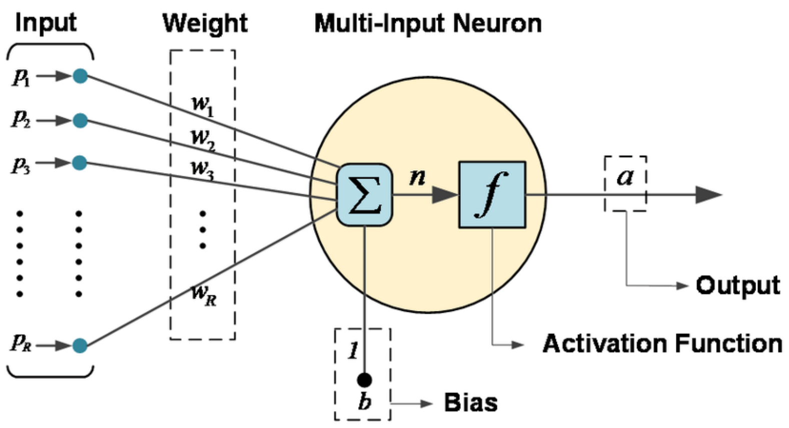


Figure 4.42.- A neuron of an ANN. All the operation needed to have an output. Figure extracted from [118].

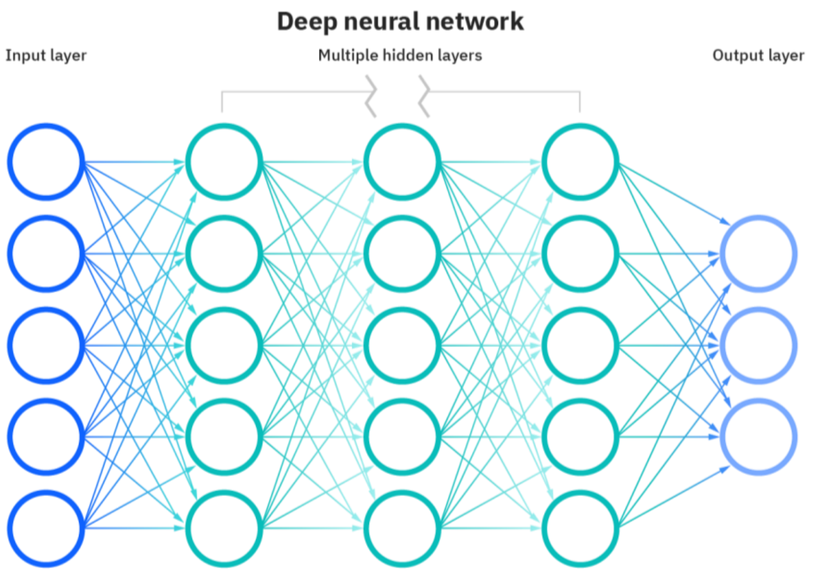


Figure 4.43.- The layers of a deep neural network. Figure extracted from [117].

The main difficulty occurring while working with ANN, especially when they are deep, is the undesired effect of **overfitting**, where the model performs excellently in the training dataset and poorly while testing it. Hence, the models memorize patterns from the training data and are less capable of generalizing the predictions. Many **regularization techniques** can help to reduce the effect of overfitting.

One method is establishing a **dropout** ratio to randomly drop several connections of the ANN in each iteration of the training session; this makes the model more likely to generalize a solution. Another one is to apply an **early stopping**, which enables the user to select an optimal number of iterations required (forward and back-propagation) for training the model before it overfits.

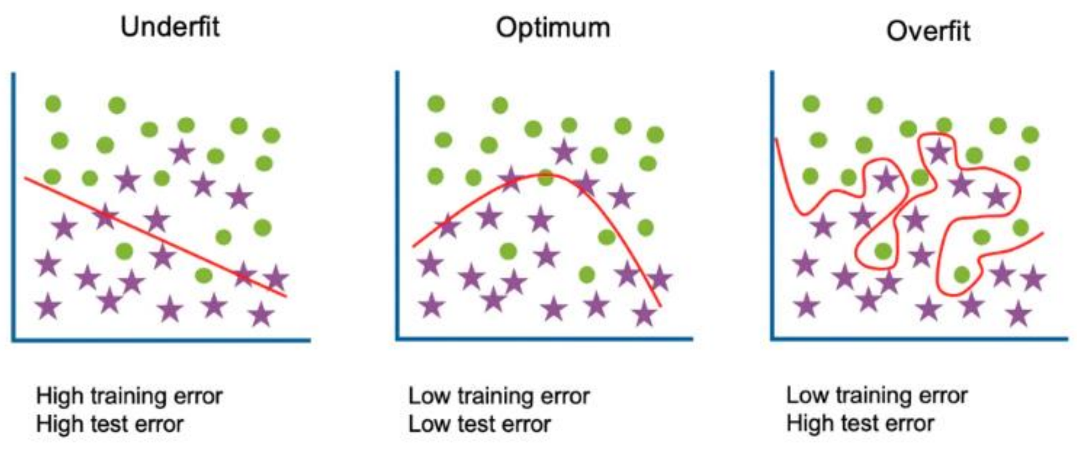


Figure 4.44.- A comparison between an underfitted, an optimum, and an overfitted model. Figure extracted from [119].

4.8.10. Convolutional Neural Networks (CNN)

A convolutional neural network (CNN) is a type of Deep Learning algorithm that is widely used for classifying, segmenting, or detecting objects from an image, which can be described as a matrix [120]. The process starts when multiple filters, known as **kernels** or **masks**, are applied to the matrix. These kernels are smaller than the picture (usually of size 3x3) and run over all the image by executing a **convolution**, which is a mathematical operation that enables the image transformation with these steps [121]:

1) Flip the kernel horizontally and vertically. This step is done only once.
2) Put the kernel over a section of the image.
3) Calculate the dot product (multiply the values of the mask with their corresponding image's values; then sum the results to obtain the output value in that position).
4) Shift the kernels to the next position of the image. This parameter can be defined with the stride, which indicates how many positions the mask shifts in each iteration.
5) Repeat the process until all the positions are covered.

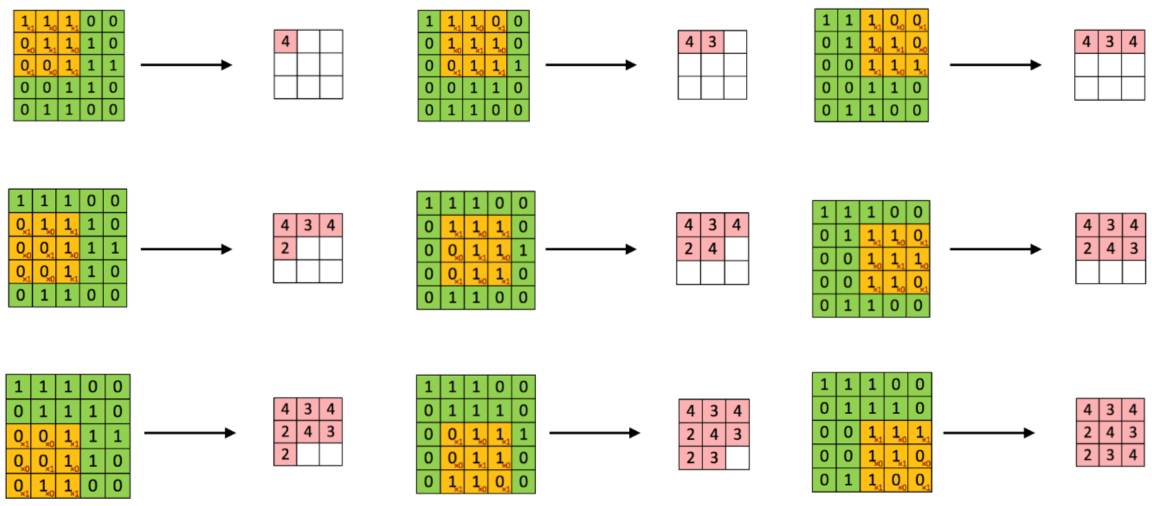


Figure 4.45.- Application of convolution to an image (green with black numbers) with a kernel (yellow with red numbers). The result of the convolution is shown in the pink matrix. Figure extracted from [122].

Notice that due to the mask size the edges of the image are eliminated from the output result. This issue can be solved by **zero-padding** the images. In this way, the output can have the same size as the input image.

After the convolution layer, an **activation function** (ReLU, etc.) is applied to introduce non-linearity to the system. The next step is to carry out **pooling** operations, such as the maximum or the average pooling, to reduce the sizes of the matrices and the computational cost. Furthermore, it decreases the risk of overfitting the system and makes the model more tolerant in front of slight distortions and variations.

The convolutional and pooling layers can be computed sequentially multiple times to extract features from the images. Once all the characteristics are obtained, the matrices are converted into a single dimension in the **flatten layer**. Subsequently, a fully connected ANN is used for classifying the image.

While training the CNN model, the back-propagation allows finding suitable weights, bias, and kernels for enhancing the architecture's performance by reducing the loss function [123].

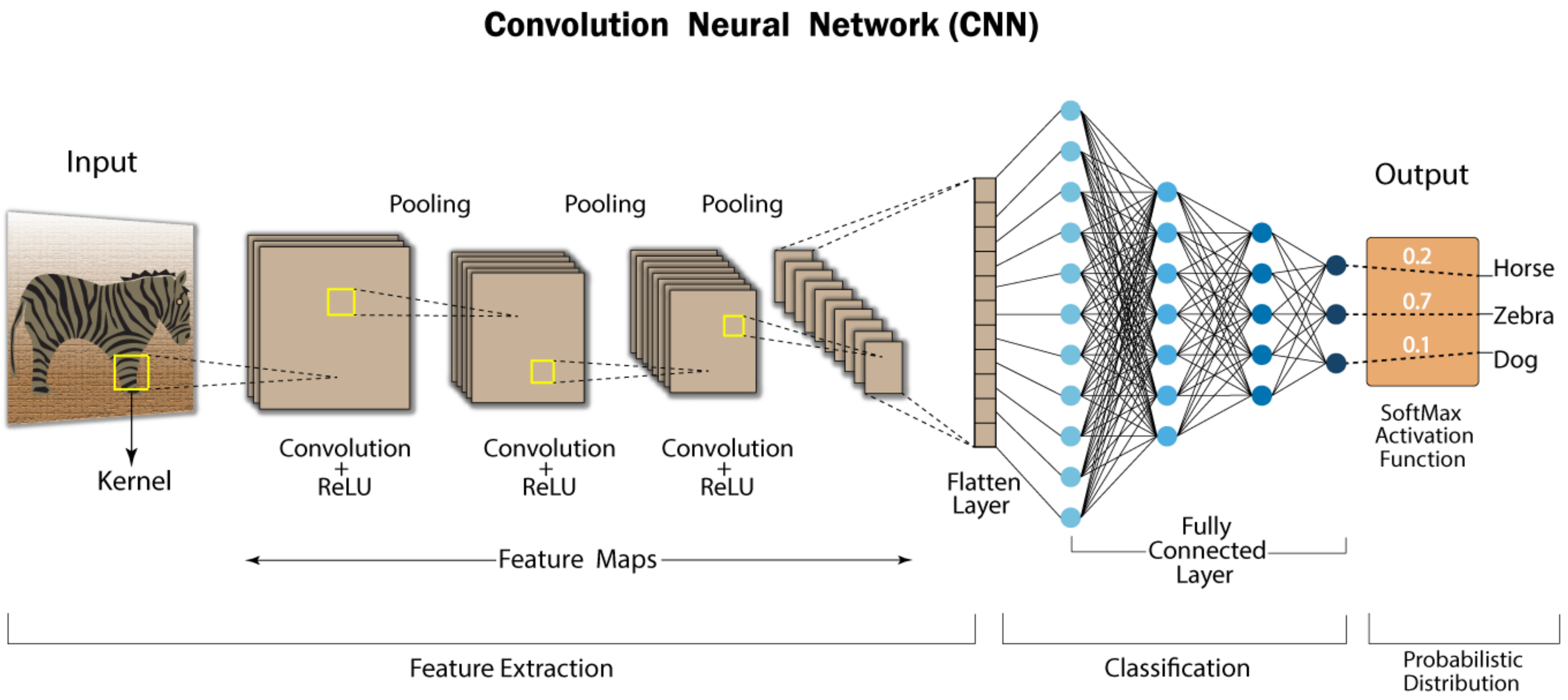


Figure 4.46.- Overview of the CNN architecture for classifying the image of a zebra. Figure extracted from [124].

5. Results

After training all the models mentioned in the previous section, the performance of the AI models must be evaluated to choose the most appropriate ones for real-time implementation. It is important to remark that each model was trained by using the data of Subjects C, E, and F. Bear in mind that the *right vs left imagery* models will be assessed in two separate ways:

- Mixing all the testing data of each subject for obtaining **general estimation models**.
- Treating the inter-subject testing data independently to see how the models performs **for each participant**.

In the case of the *imagery vs non-imagery* models, only the general estimation will be considered.

5.1. Chosen AI models

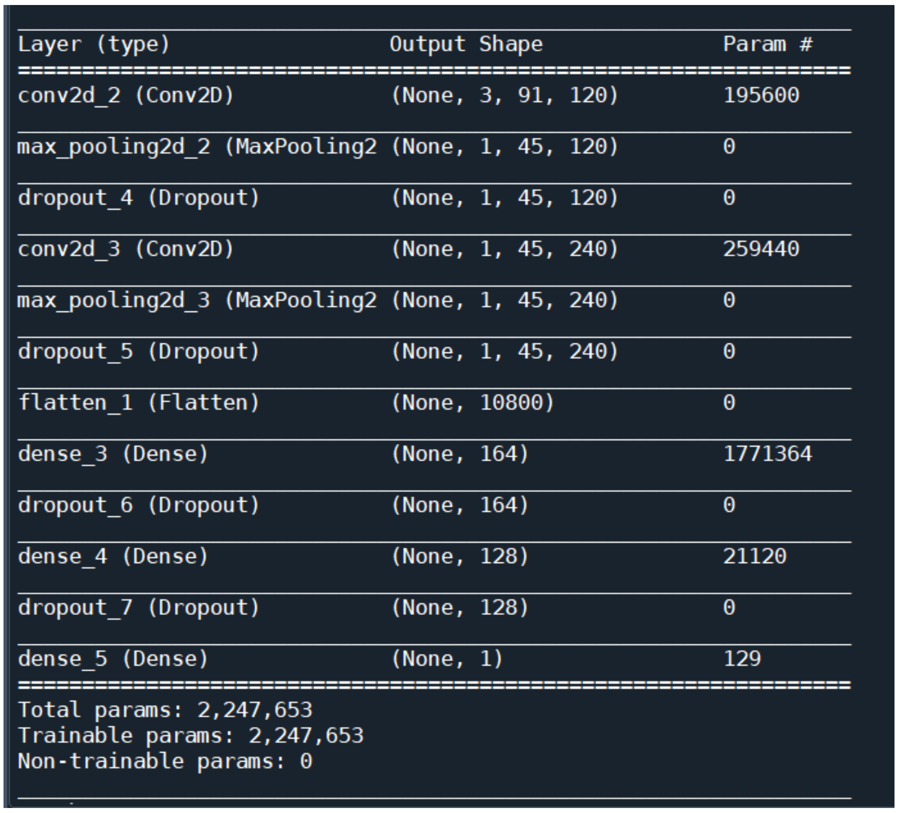
The hyperparameter used for the Machine Learning models are the following:

- **LDA:** A singular value decomposition (SVD) solver and no shrinkage.
- **SVM:** A radial basis function (RBF) kernel, a gamma of 0.00001, and a C parameter of 10000.
- **RF:** 100 number of estimators, a Gini criterion for measuring the quality of a split, a maximum depth of 11, a minimum number of samples to split an internal node of 5, a minimum number of samples to be considered as a leaf node of 4, and square root (sqrt) function to calculate the maximum number of features.

The structure of the CNN is shown in Figure 5.1. Firstly, there is a 2D convolutional layer with 120 filters, a 3x3 kernel, a ReLU activation function, and an input size of (3,91,181); this is followed by a MaxPooling layer of size two and a dropout of 20%. The second convolutional layer has 240 filters, a 3x3 kernel, and a ReLU activation function, followed by a MaxPooling of size one and a dropout of 20%.

Afterward, a flatten layer prepares the data for entering inside two dense layers with 164 and 128 neurons, respectively. Both have a dropout of 20% and a ReLU activation function. The model ends with a one neuron layer with a Sigmoid activation function.

Regarding the CNN's hyperparameters, there is a learning rate of 0.0001 with a decay rate of 0.0001/300 and a momentum of 0.8; these parameters are introduced inside a Stochastic Gradient Descent (SGD) optimizer. The loss will be evaluated with the binary cross-entropy.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 3, 91, 120)        195600
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 45, 120)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 1, 45, 120)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 45, 240)        259440
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 45, 240)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 1, 45, 240)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10800)             0
_________________________________________________________________
dense_3 (Dense)              (None, 164)               1771364
_________________________________________________________________
dropout_6 (Dropout)          (None, 164)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               21120
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,247,653
Trainable params: 2,247,653
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.1.- The structure of the CNN. Figure created by the author.

5.1.1. Right vs Left-hand imagery

As can be seen in Table 5.1, all three models present similar results, being the **RF architecture** slightly better than the others with an **overall accuracy of 83%, reaching 88% in Subject C**. Moreover, the precision and the recall reveal that all the models are more likely to detect left-hand imageries than right-hand ones. Therefore, the **RF will be the chosen model for distinguishing left and right-hand imageries.**

5.1.2. Imagery vs non-imagery

As has been previously remarked, it is extremely important to avoid false positives so that a MI is not wrongly detected. Hence, accuracy and precision should be assessed conjunctively to find an optimal solution.

By using the **statistical features, the RF has the best accuracy (77%) and precision (85%)** relative to LDA and SVM. It is possible to obtain the statistical features that are more capable of better explaining and defining the behavior of the model; this can be done with the Shapley values (Figure 5.2.). The most relevant features are the **energy**, the **entropy,** and the **log energy-entropy**.

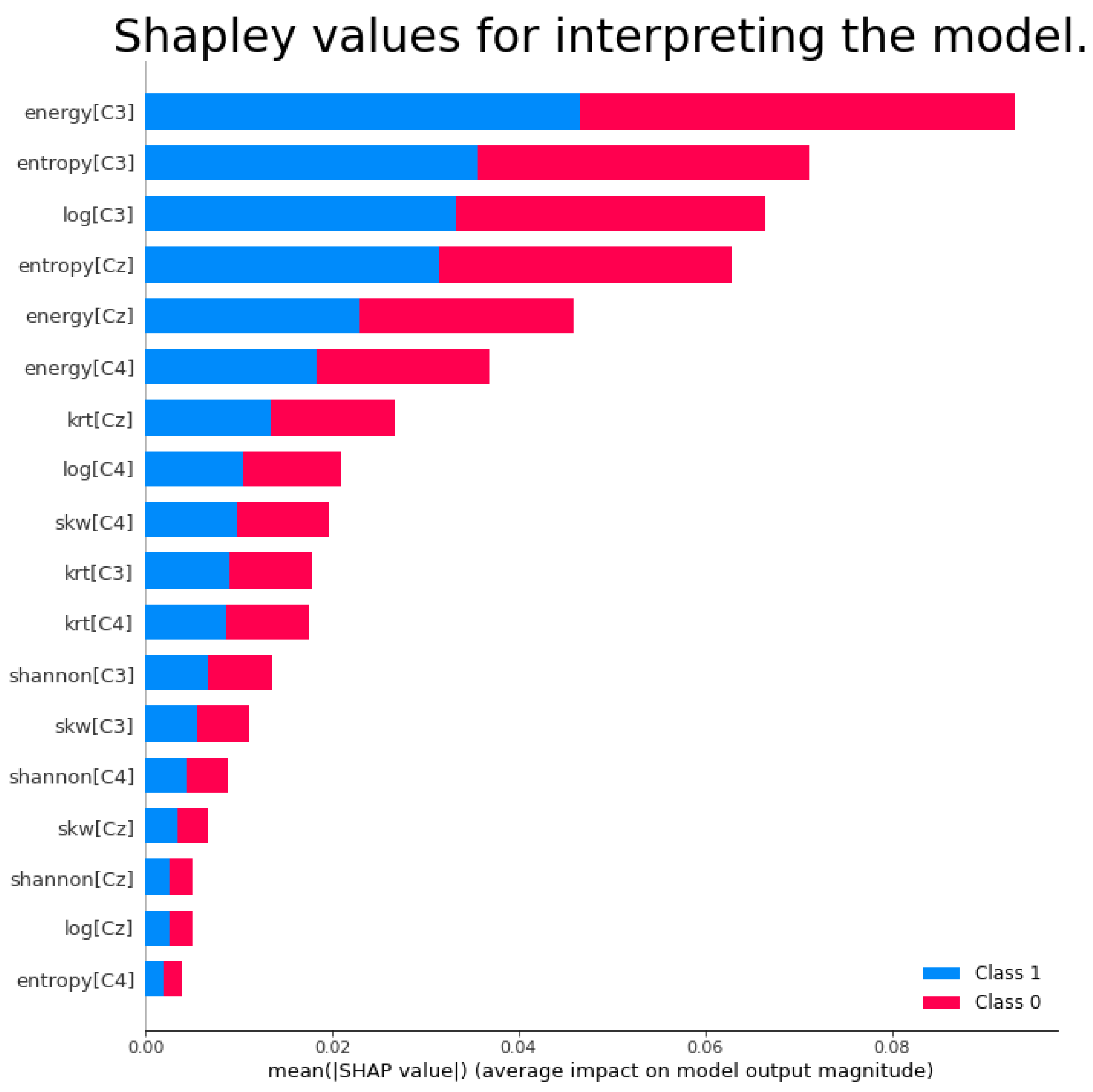


Figure 5.2.- Shapley values for interpreting the model's classes. Blue is for the resting state and red for the imagery class. The X-axis represents the average impact of the features on the model (0.01 SHAP values per division). Figure created by the author.

However, the **CNN** outperforms the RF architecture as it has a **78% accuracy and a precision of 90%**, which is reasonable for the BCI. Therefore, **the CNN will be the chosen model for distinguishing imageries and resting states**.

Figure 5.3. shows how the architecture performed in each iteration; visualizing this type of graph is crucial in order to ensure that there is no overfitting by finding the suitable number of epochs that minimizes the testing loss.

Table 5.1.- A summary of the AI model results. Table created by the author.

AI models results			General (All Subjects)				Subject C				Subject E				Subject F			
			LDA	SVM	RF	CNN	LDA	SVM	RF	CNN	LDA	SVM	RF	CNN	LDA	SVM	RF	CNN
Right vs Left-hand imagery	CSP	Accuracy	82%	82%	**83%**		87%	88%	**88%**		78%	79%	**79%**		80%	79%	**81%**	
		Precision	87%	88%	**87%**		92%	93%	**94%**		87%	87%	**84%**		84%	83%	**83%**	
		Recall	75%	75%	**77%**		82%	82%	**80%**		67%	68%	**72%**		74%	75%	**79%**	
Imagery vs non-imagery	Statistical Features	Accuracy	74%	76%	77%													
		Precision	83%	83%	85%													
		Recall	59%	63%	63%													
	STFT	Accuracy				**78%**												
		Precision				**90%**												
		Recall				**72%**												

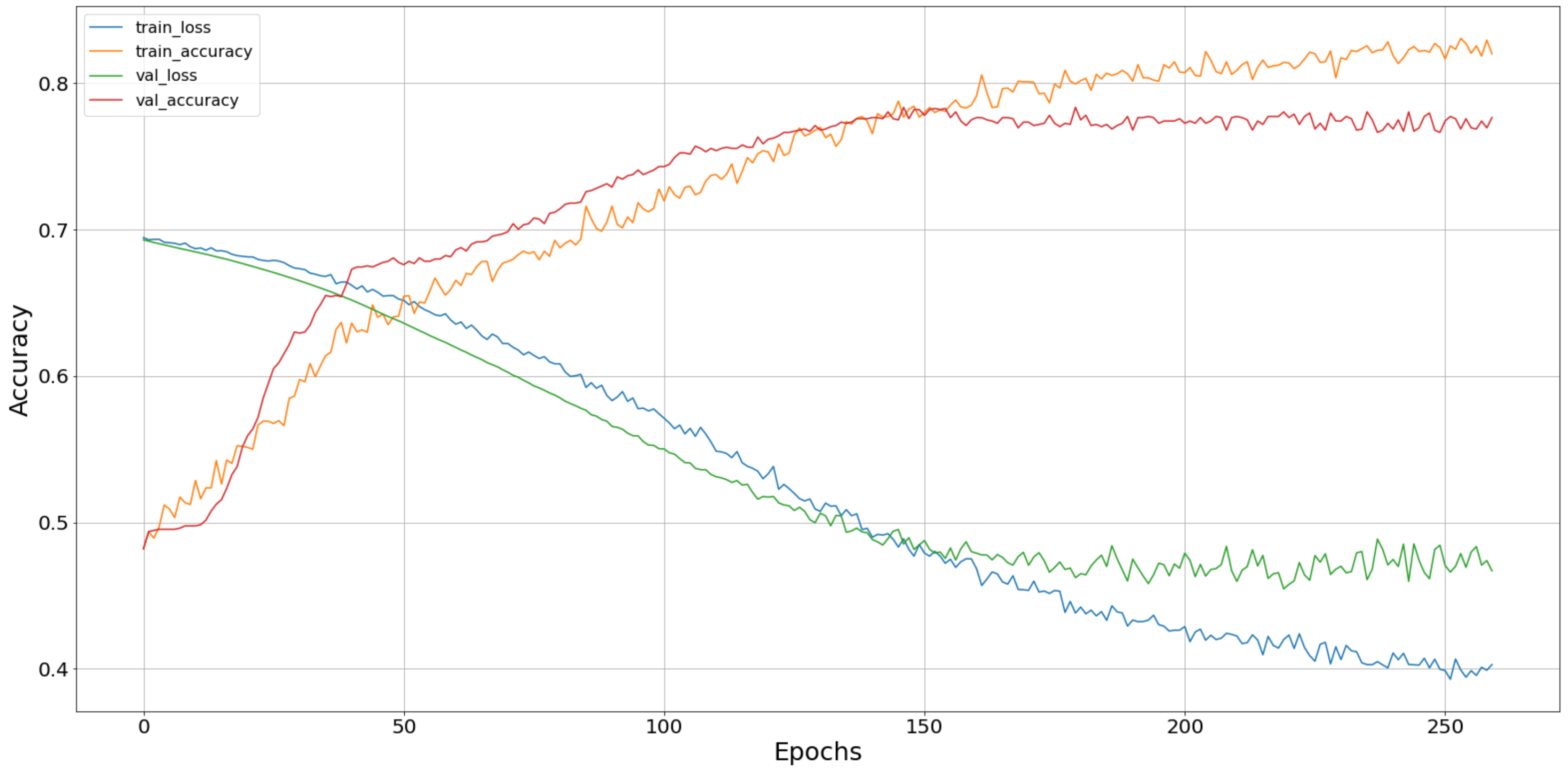


Figure 5.3.- The training and the testing of the CNN model. Train accuracy (orange), train loss (blue), test accuracy (red), test loss (green). The X-axis represents the epochs (50 epochs per division) and the Y-axis represents the accuracy and the loss (0.1 units per division). Figure created by the author.

5.2. Real-time implementation

Once the best models are chosen, they will be used to classify an entire EEG session that was unseen while training them. Therefore, the models will simulate an online categorization of the data, mimicking the behavior that would be obtained from a live collection of EEG data.

To carry it out, the EEG session will be introduced sequentially inside a window with a size of 180 samples (0.9 seconds since the sampling rate is 200 Hz). The data inside the window is filtered with the designed FIR filter; bear in mind that there will be a delay of 75 samples (justified in the Digital filtering section). Subsequently, the STFT features are extracted, and the CNN model will classify whether the segment is an imagery or a resting state. If an imagery is detected, the CSP are obtained, and the RF model will distinguish between left and right-hand imageries. If a resting condition is detected, the BCI will ignore the segment (see the diagram drawn in Figure 4.2.). Afterward, the data is shifted 20 samples (0.1 seconds) into the window, and the cycle starts again (Figure 5.4.).

In order to avoid noisy classifications, the model must predict four consecutive right or left-hand imageries to efficiently decide if a movement is really performed or not. For instance, if the model has been predicting a resting state for 10 seconds and at 10.1 seconds detects a right-hand movement continued by a passive state, this would be considered a noisy classification.

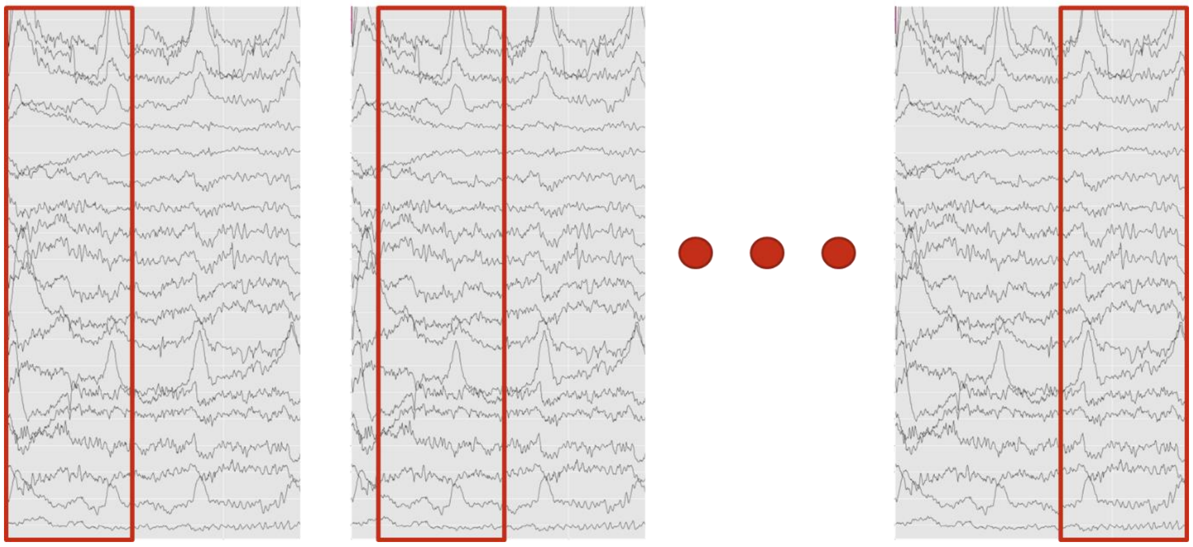


Figure 5.4.- A visual explanation of how the data slides inside the window (red). Figure created by the author.

In the following pages, a set of images will highlight some of the predictions from an unseen session of Subject C, showing cases in which the models perform flawlessly and others where there are some faults (see the captions of each illustration to have a detailed explanation). Overall, the BCI detects the imageries precisely, outputting most of the time the correct hand movement.

On the one hand, the red and green color bands indicate the eGUI's instruction to imagine a right or left-hand movement, respectively. On the flip side, the orange and blue colors bands denote the right and left-hand imageries predictions, correspondingly. Notice that there will always be a short delay between the eGUI's indication and the prediction; this is caused due to the FIR filter and the subject's response time after seeing the stimuli.

The last step is to move a *MeArm* robotic manipulator left or right, depending on the prediction; this is achieved by establishing serial communication between Python and Arduino. In other words, a servomotor is activated based on the model's outputs. The servomotor is connected to Pin 9 of the Arduino Mega 2560. The set up is shown in the following illustration:

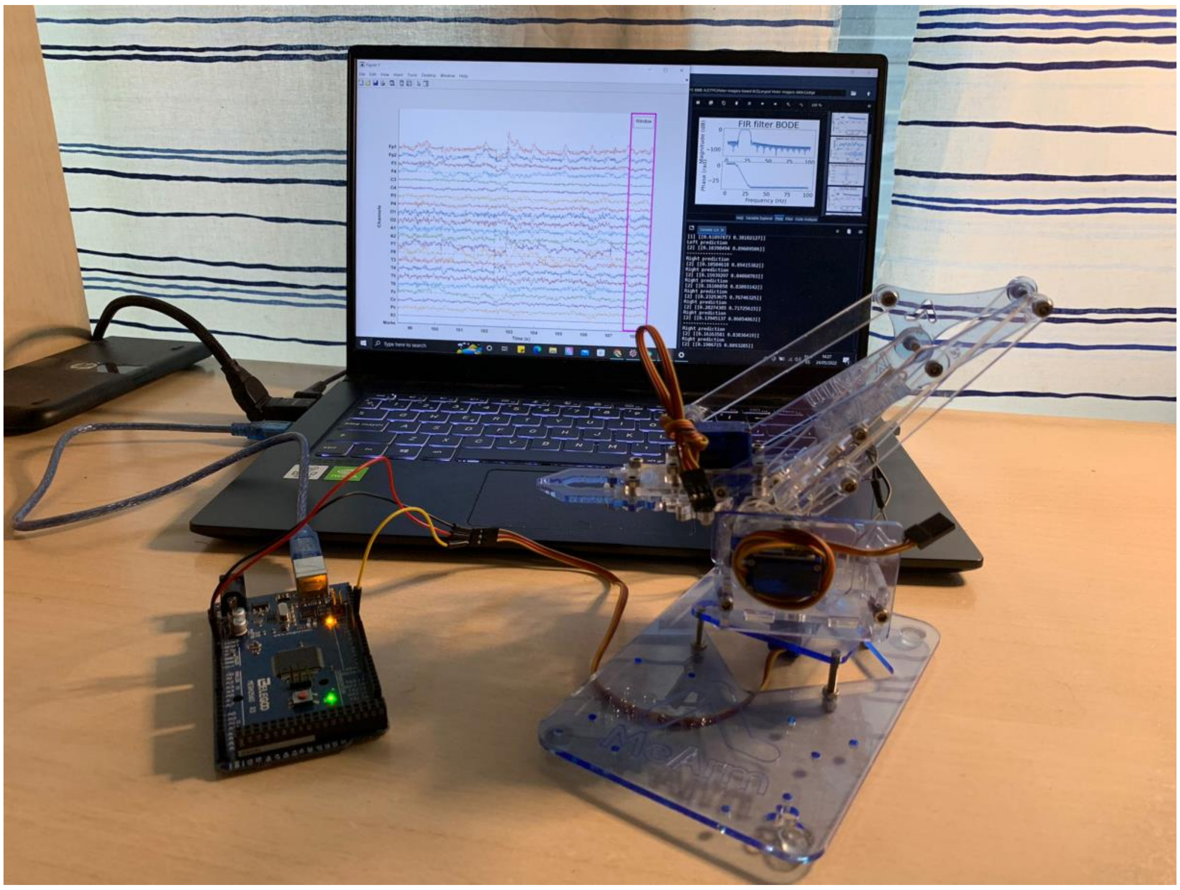


Figure 5.5.- Setting up the MeArm robotic manipulator. Figure created by the author.

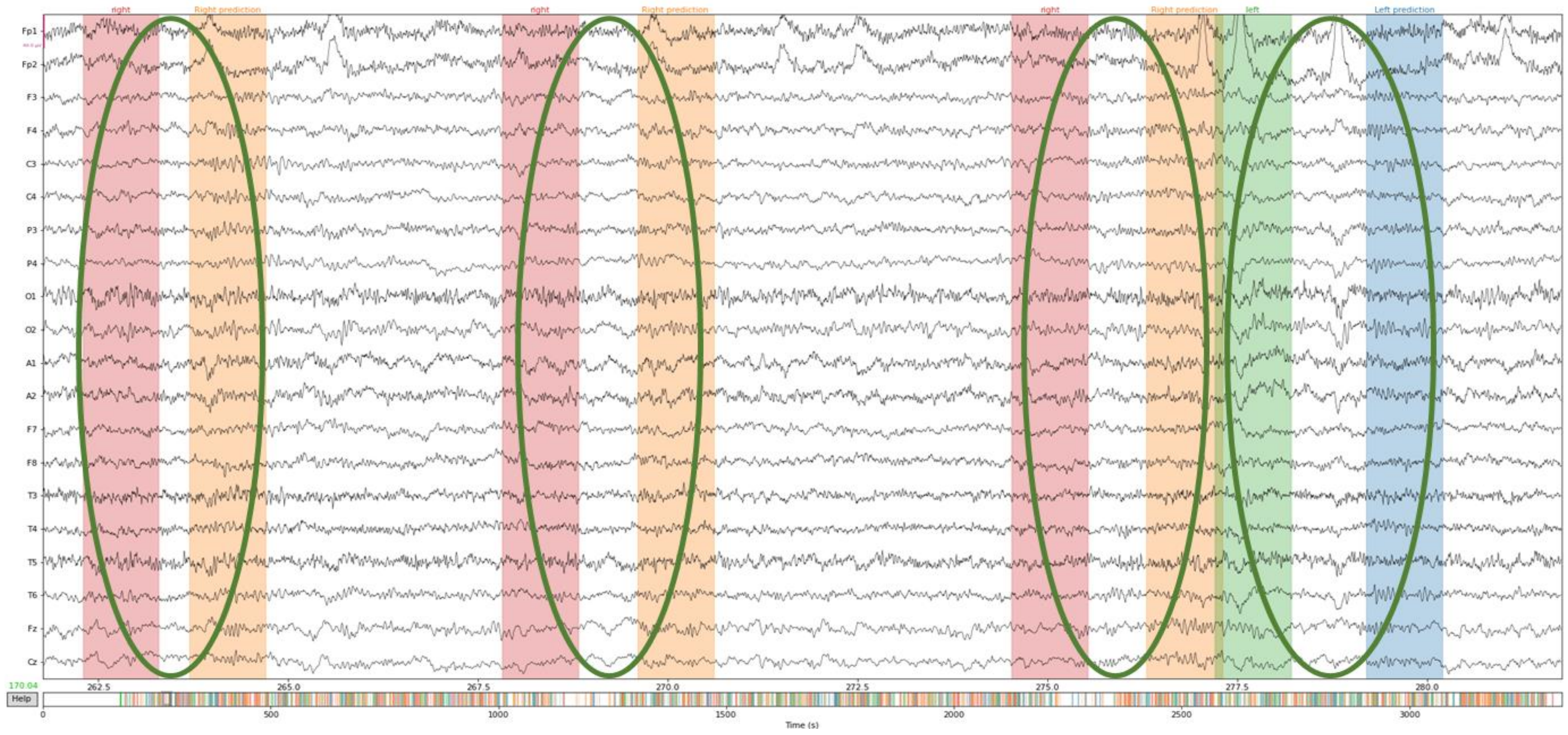


Figure 5.6.- Predictions of an unseen session (Subject C). All the classes were correctly classified (green circles), and the resting state was successfully detected. Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

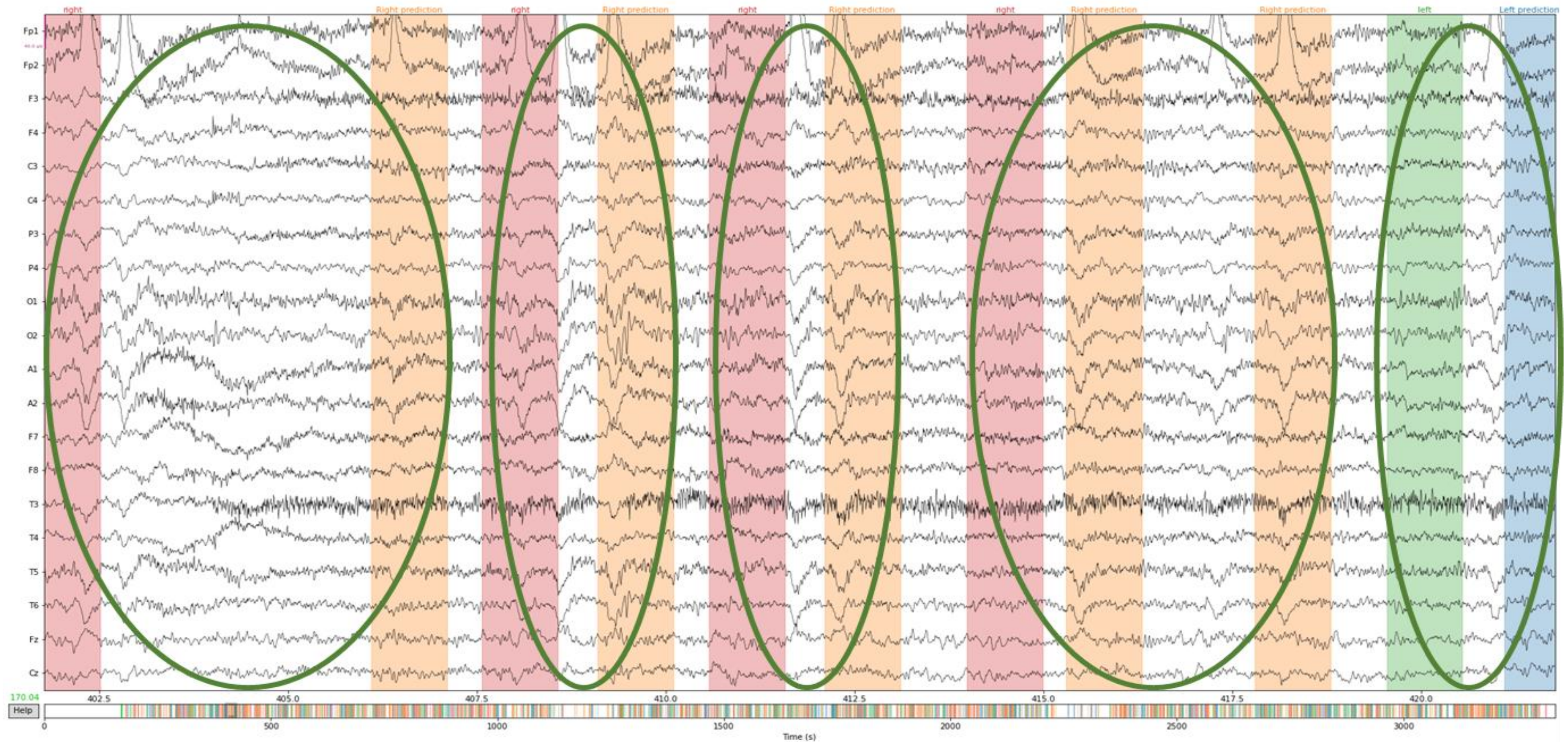


Figure 5.7.- Predictions of an unseen session (Subject C). All the classes were correctly classified (green circles). Notice that sometimes the prediction is delayed a little bit more, which is normal. Moreover, a prediction can be repeated consecutively, indicating that the mental activity is maintained. Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

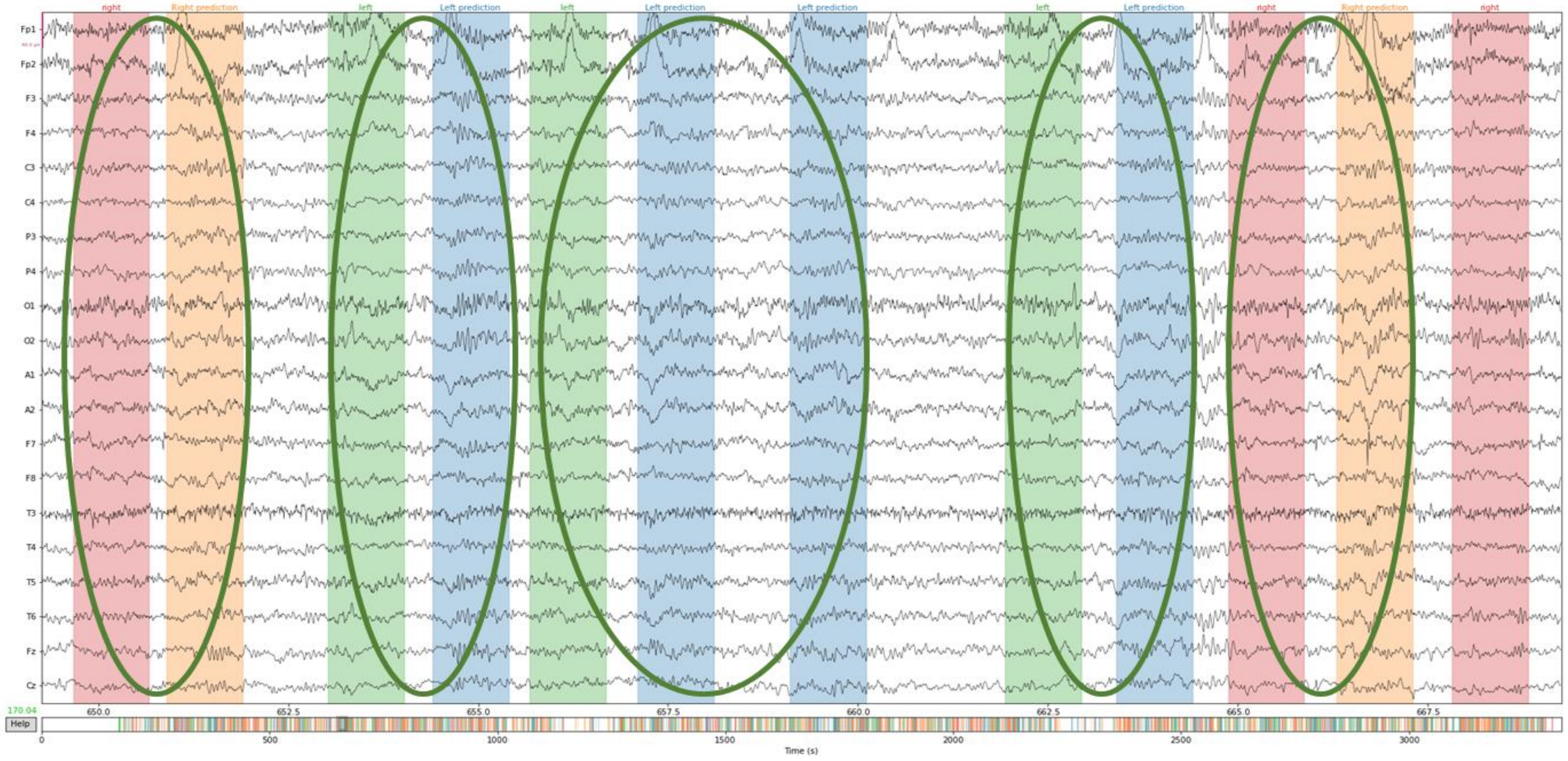


Figure 5.8.- Predictions of an unseen session (Subject C). All the classes were correctly classified (green circles), and the resting state was successfully detected. Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

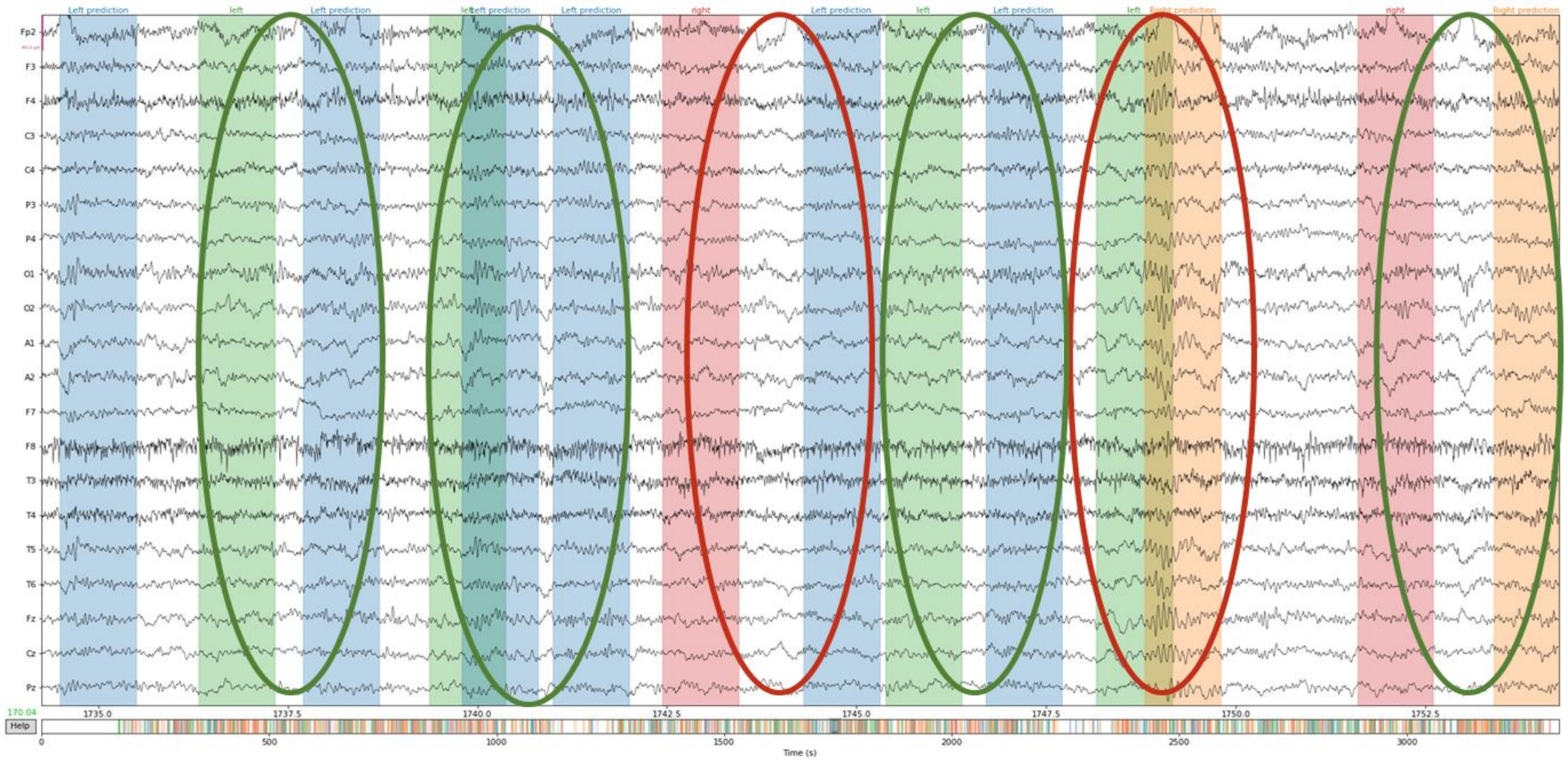


Figure 5.9.- Predictions of an unseen session (Subject C). Almost all the classes were correctly classified (green circles), and the resting state was successfully detected. However, there is one right imagery detected as left and one left detected as right (red circles). Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

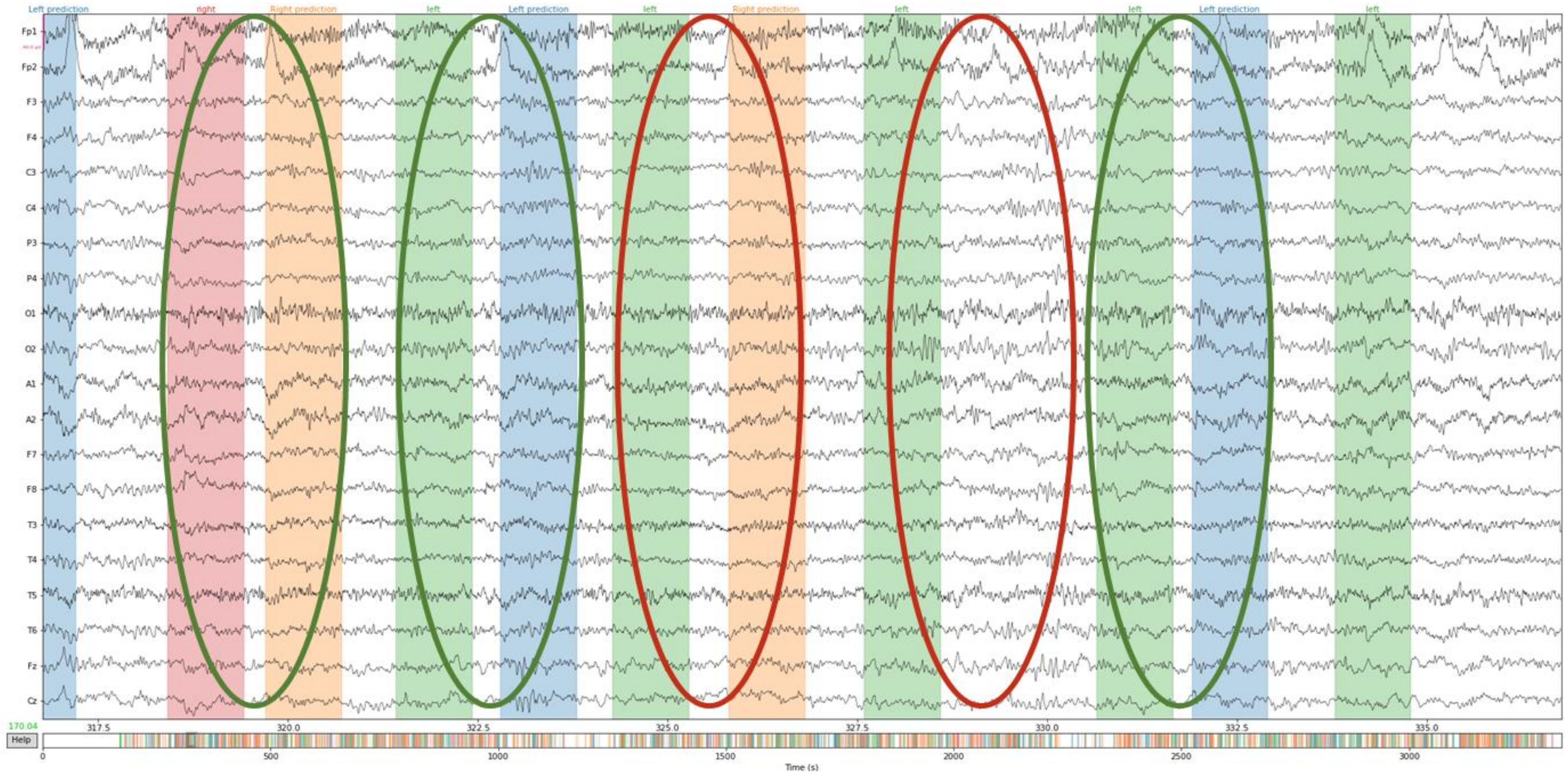


Figure 5.10.- Predictions of an unseen session (Subject C). Almost all the classes were correctly classified (green circles). However, there is one left imagery detected as right and another left imagery that was not detected at all (red circles). Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

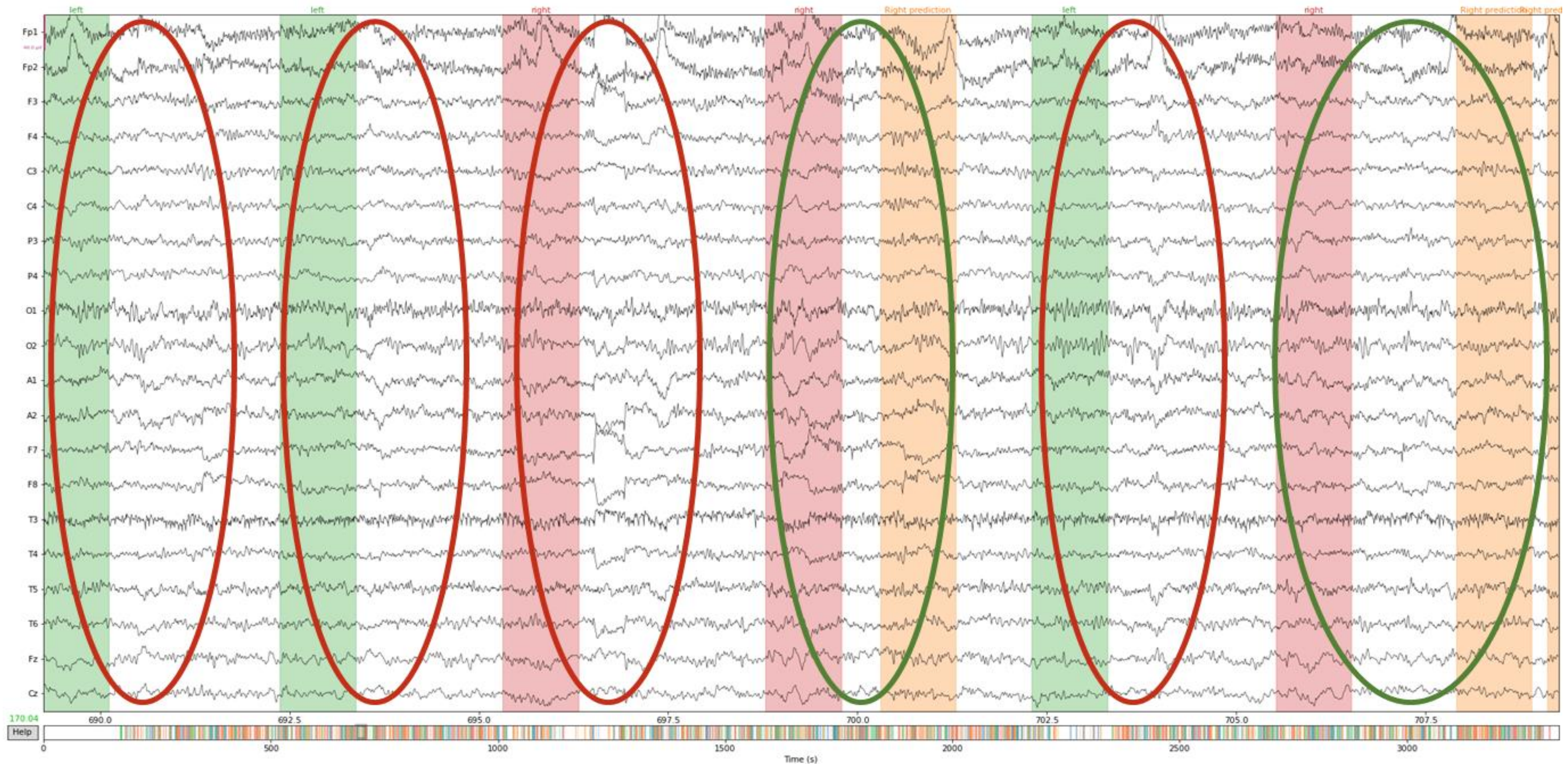


Figure 5.11.- Predictions of an unseen session (Subject C). Almost all the classes were not detected at all, the model considered them as a resting state (red circles); only two were correctly categorized (green circles). Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

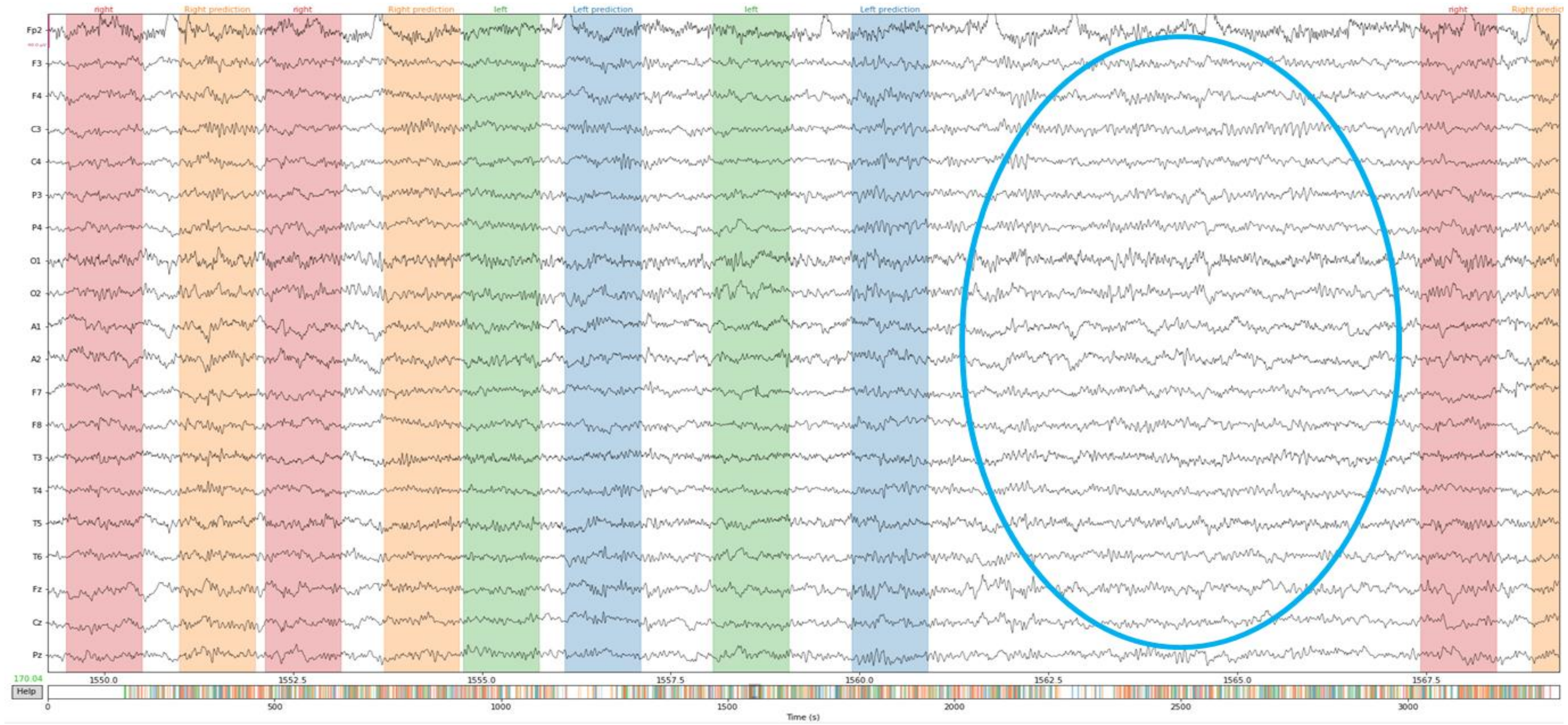


Figure 5.12.- Predictions of an unseen session (Subject C). This case shows how well the model predicts a resting state, being passive when there is no indication to perform left or right-hand imageries (blue circle). Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

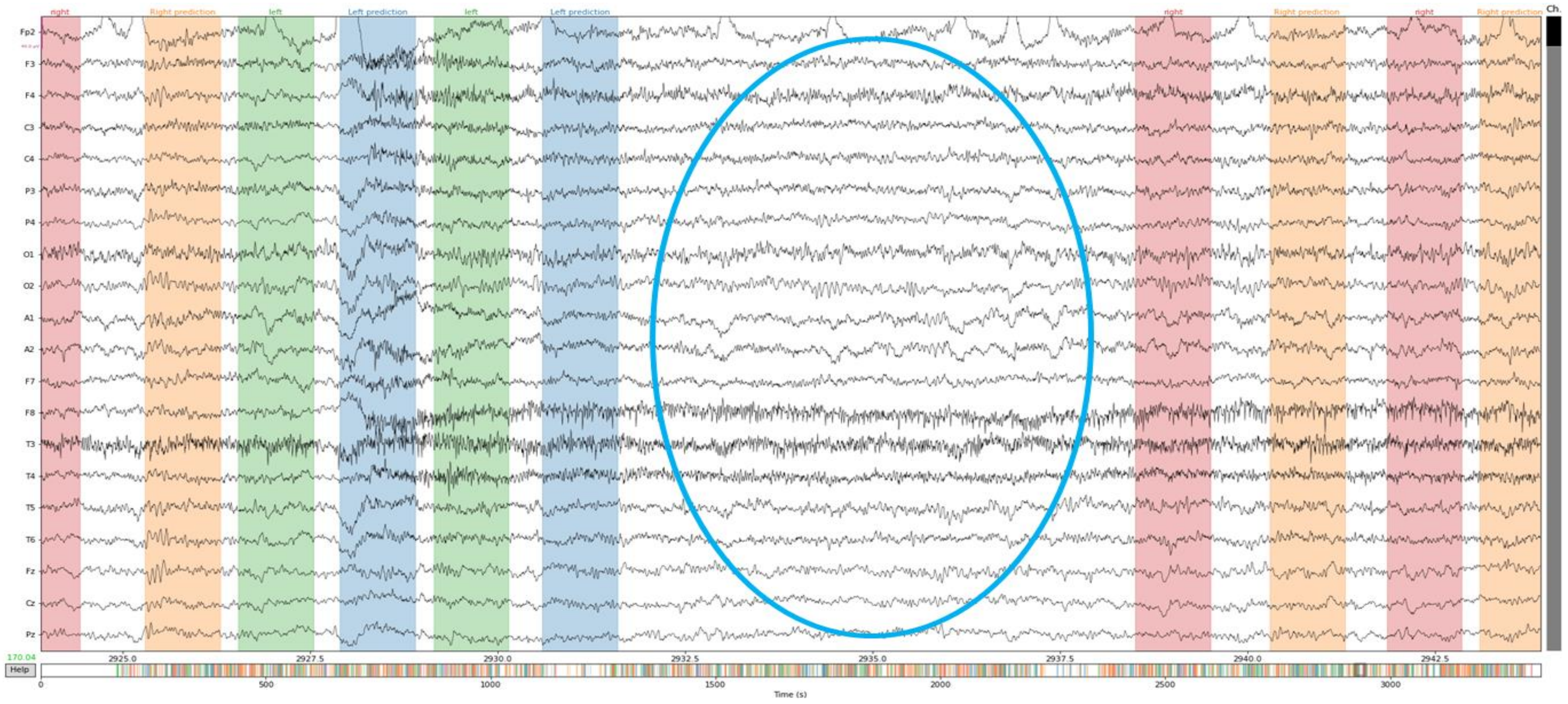


Figure 5.13.- Predictions of an unseen session (Subject C). This case shows how well the model predicts a resting state, being passive when there is no indication to perform left or right-hand imageries (blue circle). Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

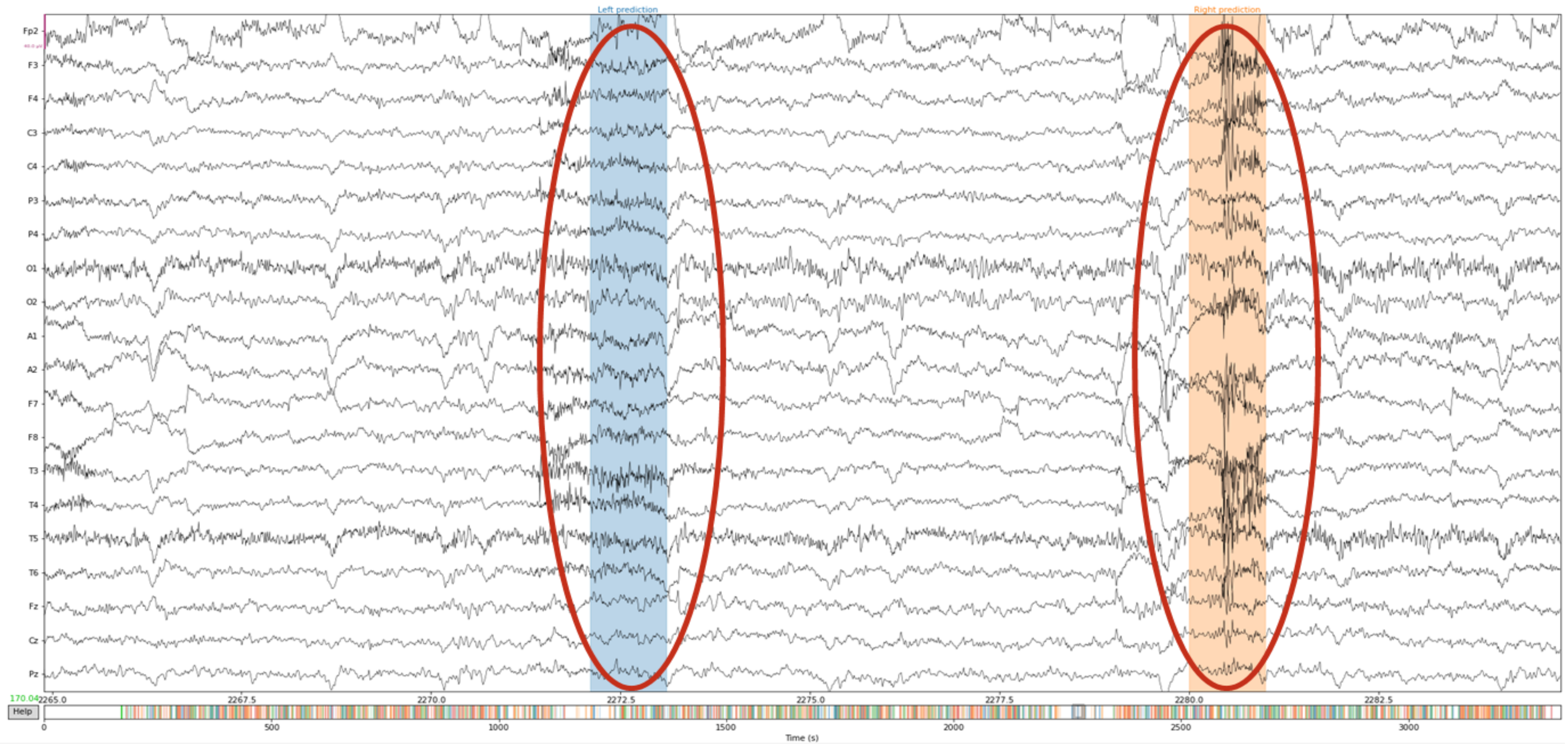


Figure 5.14.- Predictions of an unseen session (Subject C). This case shows that sometimes an imagery state is detected even though the patient is in a resting state (red circles). These cases are not abundant since the precision of the *imagery vs non-imagery* model is high. Red band: Indication to perform a right imagery. Green band: Indication to perform a left imagery. Orange band: right prediction. Blue band: left prediction. Figure created by the author.

6. Environmental analysis

Climate change is a dangerous threat that can devastate the environment and the organisms living in it. For that reason, one must be constantly aware of the environmental impact that any project will cause, knowing all the consequences of the actions taken.

All the hardware employed in this project, including the Arduino and the EEG that the researchers in [9] used for creating the dataset (JE-921A EEG), has been chosen carefully to respect the environment. Each of the selected components is certified with quality stamps, more precisely complying with the EU directive 2011/65/EU (RoHS compliant) [125][126]. With the RoHS, the use of toxic and harmful substances for the environment found on the device's electronics is limited (chromium, mercury, cadmium, etc.).

Furthermore, one must not overlook the equivalent carbon emissions due to the GPU's power consumption while extracting features and optimizing, tuning, and training the AI models. Approximately, in this thesis, the computer executed the previous operations for 120 hours. The GPU used is an NVIDIA GeForce GTX 1080, which has a power consumption of 180 W [127]; hence, the energy consumption can be calculated as follows:

$$\boldsymbol{GPU\ Consumption = 180\ W \cdot 120\ h = 21.6\ kWh} \qquad \textbf{(Eq. 6.1)}$$

According to the Catalan Office of Climate Change [128], the estimated carbon produced based on the local power grid (Catalonia) is 0.321 kg eq. CO_2/kWh; therefore, the equivalent CO_2 generated due to the Machine and Deep Learning models is **6.9 Kg of CO_2**:

$$\boldsymbol{Equivalent\ CO_2 = 21.6\ kWh \cdot 0.321 \frac{Kg\ eq.CO_2}{kWh} = 6.9\ Kg\ of\ CO_2} \qquad \textbf{(Eq. 6.2)}$$

7. Project planning

To successfully carry out all the required tasks of this project, the author designed a Gantt chart (Figure 7.1.) to work efficiently and in an organized way. Each period of this diagram represent a week.

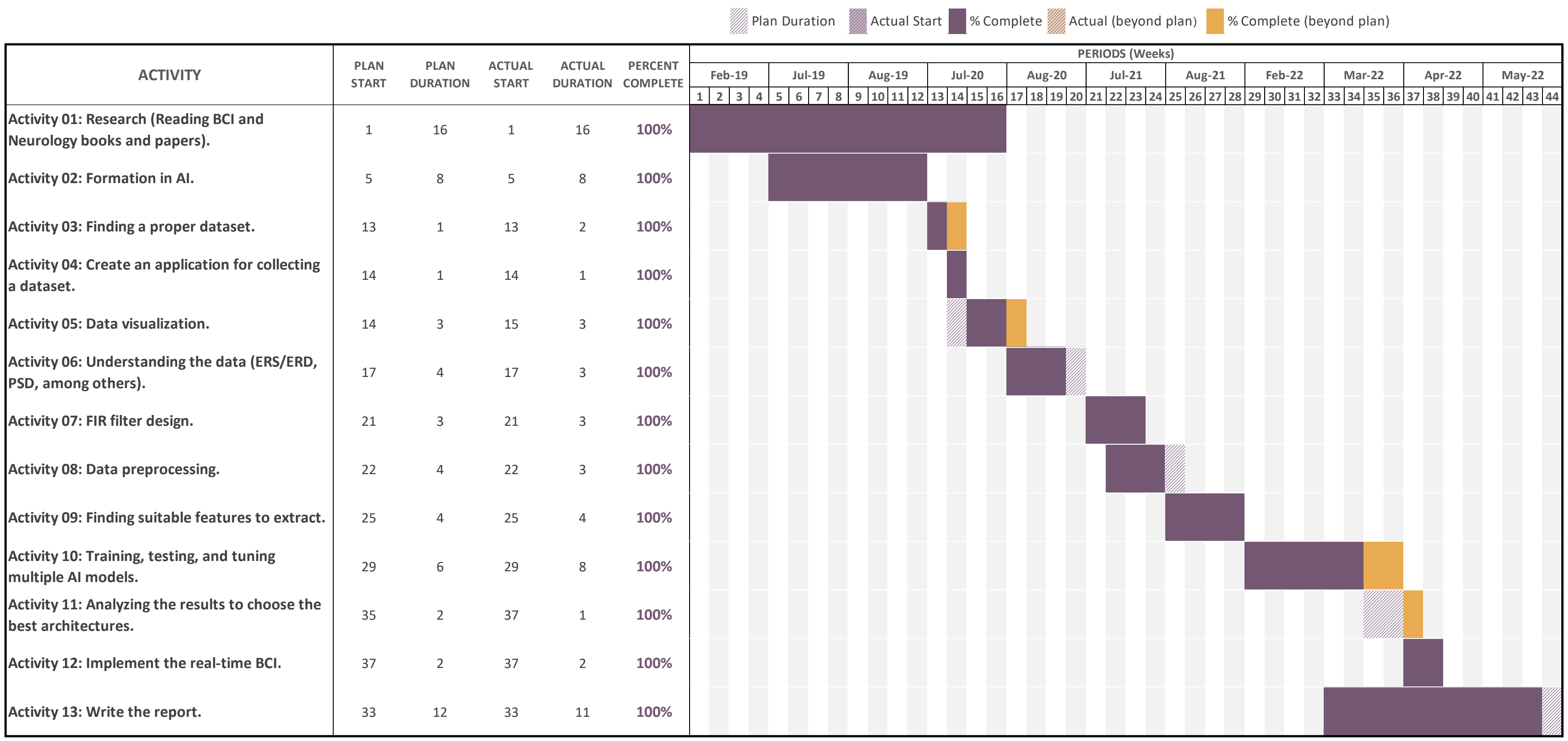
As it can be seen, the author worked on the thesis intermittently, starting his research in February 2019 and his formation in the AI world in July and August of 2019. He developed the BCI system during the summers of the succeeding years (2020 and 2021) and the last quarter of his double degree (from February 2022 to May 2022).

Most of the activities were completed on time; only activities 3, 5, 10, and 11 were completed beyond the proposed dates, which was not problematic since the project was finished one week before expected. Notice that activity 10 took two weeks more to complete because many more models not mentioned in the thesis were evaluated; only the best performing architectures were cited in the report. The following table will document the hours worked on each task:

Table 7.1.- Hours worked in each activity. Table created by the author.

Task	Hours
Activity 01: Research (Reading BCI and Neurology books and papers).	300
Activity 02: Formation in AI.	180
Activity 03: Finding a proper dataset.	50
Activity 04: Create an application for collecting a dataset.	10
Activity 05: Data visualization.	75
Activity 06: Understanding the data (ERS/ERD, PSD, among others).	100
Activity 07: FIR filter design.	70
Activity 08: Data preprocessing.	30
Activity 09: Finding suitable features to extract.	90
Activity 10: Training, testing, and tuning multiple AI models.	180
Activity 11: Analyzing the results to choose the best architectures.	25
Activity 12: Implement the real-time BCI.	50
Activity 13: Write the report.	275
TOTAL	**1435**

Thus, the total time dedicated to this project is **1435 hours**, which is sufficient to cover the workload of a 48 ECTS thesis.



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Activity 01: Research (Reading BCI and Neurology books and papers).	1	16	1	16	100%
Activity 02: Formation in AI.	5	8	5	8	100%
Activity 03: Finding a proper dataset.	13	1	13	2	100%
Activity 04: Create an application for collecting a dataset.	14	1	14	1	100%
Activity 05: Data visualization.	14	3	15	3	100%
Activity 06: Understanding the data (ERS/ERD, PSD, among others).	17	4	17	3	100%
Activity 07: FIR filter design.	21	3	21	3	100%
Activity 08: Data preprocessing.	22	4	22	3	100%
Activity 09: Finding suitable features to extract.	25	4	25	4	100%
Activity 10: Training, testing, and tuning multiple AI models.	29	6	29	8	100%
Activity 11: Analyzing the results to choose the best architectures.	35	2	37	1	100%
Activity 12: Implement the real-time BCI.	37	2	37	2	100%
Activity 13: Write the report.	33	12	33	11	100%

Figure 7.1.- Gantt chart to plan and track all the activities. Figure created by the author.

Conclusions

In this work, the author successfully implemented a BCI system that can translate EEG signals containing imaginations of hand movements coming from the motor cortex activity into commands for controlling a robotic manipulator. The project highlights that BCIs will revolutionize the lifestyle of individuals with movement disabilities, enabling them to be more independent and integrated into society.

The first step taken was to understand the data to decide on further actions to consider. The power spectral density revealed that electrodes Fp1 and Fp2 constantly captured eye-blinking signals as they contained higher amplitudes at lower frequencies relative to other electrodes; additionally, it was confirmed that the higher the frequency, the lower the magnitude of the signals. Furthermore, to reduce the spatial smearing effect, the common average referencing has shown to localize the sources of the EEG signals.

The ERD/ERS maps were extremely useful for comprehending the frequential behavior of each imagery over time. From the six subjects analyzed, only Subject C had a map that mostly reassembled the theoretical one, having clear ERD followed by an ERS in the Beta band and an ERS in the Mu band. To the author's surprise, the rest of the subjects lacked any ERD. Moreover, most participants only had ERS activity in the Beta and not the Mu band. For the previously stated reasons, only ERS from the Beta band was considered for the study. Only three participants (Subjects C, E, and F) proved to have good-quality recordings; hence, the other subjects were discarded for training the artificial intelligence models.

Before training the model, the data was preprocessed to enhance the overall performance. The ICA is the perfect solution to remove artifacts, such as eye-blinking, muscular or cardiac activity, without altering the crucial information of the signals. Furthermore, the data must be segmented correctly based on the duration of the imagery activity displayed in the ERD/ERS maps.

Moreover, a finite impulse response filter was carefully designed to maintain the Beta band's ERS for all the selected subjects (15 Hz – 26 Hz). The filter's parameters were justified so that there are proper transition bands and no ripples in the passband. In addition, a Hamming window effectively increased the attenuation of the undesired secondary lobes that appear on the Bode. Also, as the filter is intended for a real-time application, the order chosen was 150, so the delay is not noticeable (372.5 ms).

The next step was to train two models for classifying three classes: left imagery, right imagery, and a resting state. The hyperparameters of each architecture were chosen with the help of tuning and optimizing methods, which significantly enhanced the accuracy and precision.

On the one hand, model LR, which classified left and right-hand imageries, was trained with various Machine Learning models with the CSP as features. The random forest algorithm outperformed the other models with an overall accuracy of 83%, reaching 88% in Subject C; these great results demonstrate that the model generalizes the data and decently handles the inter and intra-individual variability. The CSP undoubtedly does an excellent job discriminating between the left and right-hand classes.

On the other hand, model IR, which distinguishes between an imagery event and the resting state of the subject, has shown to perform better with the STFT features in a convolutional neural network than with statistical features in other Machine Learning models; the obtained accuracy was 78%. In addition, the precision was 90%, which is a good indicator that the model hardly has false positives; this is important so that the robotic manipulator does not get triggered without any valid imagery from the user.

Both models worked cooperatively (see the diagram drawn in Figure 4.2.) to do a real-time BCI. For testing this live analysis, an unseen session from Subject C was scrolled inside a 0.9 second time window, constantly filtering and classifying the signals overlapping with the window. The results were outstanding and much more robust than expected: the overall system can detect the three classes with few erroneous classifications.

Future work

Unfortunately, the author could not gain access to an EEG device; however, based on the excellent results achieved, this system can be fully operable for distinguishing the three categories by following the methodology described. Hence, after acquiring the device, the whole MI-BCI system will be implemented and tested (hardware and software).

Furthermore, a new line of research into different algorithms and features that could enable a multiclass classification will be started, for instance, detecting left/right hand and feet imaginations or categorizing the imagery of finger movements. To do so, the users must employ EEGs with much more electrodes in order to improve spatial resolution.

Indeed, all these processes can be applied in an ECoG instead of an EEG to improve the signals' quality, reduce the smearing effect, increase the spatial resolution, and gain information at higher frequencies hence enabling the classification of many more classes. Despite being an implantable device, they are the most viable way for the paralytic community to use a BCI. As a matter of fact, this thesis is just a first approach to the BCI world to the author, as he aims to end up working with ECoG in his succeeding career.

Budget

In this section, the author will hypothesize the costs that would be needed to professionally develop this project. Thanks to this economic analysis, the reader can see the expenses usually spent on BCI projects, considering the personnel and material costs.

Engineering cost

First and foremost, this work requires a Biomedical Engineer, an Electronics Engineer, or a Data Scientist, which is the most valuable asset of the project. Supposing that the employee is a junior Electronics Engineer and that the average salaries of these engineers in Spain are typically around 30 thousand euros a year (considering 217 working days in a year and 8 hours per day) without including the 33% regarding the social security, the hourly cost of the engineer would be of:

$$\boldsymbol{Hourly\ cost} = \frac{\mathbf{30000\ € + 30000\ € \cdot 0.33}}{\mathbf{217}\ \boldsymbol{days} \cdot \mathbf{8}\,\frac{\boldsymbol{hours}}{\boldsymbol{day}}} = \mathbf{23}\,\frac{\mathbf{€}}{\mathbf{hour}} \qquad \textbf{(Eq. 0.1)}$$

Since the total duration of the project is 1435 hours (justified in Table 7.1.), including research and implementation, the personnel cost is **33 005 €**.

Material cost

The following table lists the costs of all the software and hardware used in the project, the total cost of the materials and licenses is **7 270 €**.

Table 0.1.- Material and licenses costs. Table created by the author.

Material/License	Cost
Asus ROG Strix GL553V (Laptop)	1050 €
Arduino MEGA 2560	45 €
EEG-1200 JE-921A	5890 €
Books	250 €
Microsoft Office License	35 €
TOTAL	**7270 €**

Total budget

The total expenses are calculated by summing the engineering and materials costs (taking into account the 21% IVA), which gives a **total cost of 48 732 €** (Table 0.2):

Table 0.2.- The total expenses of the project. Table created by the author.

Materials/License	Cost
Materials	7270 €
Engineering	33005 €
Total	40275 €
TOTAL (+21% IVA)	**48732 €**

Bibliography

[1] "Spinal Cord Injury Personal Experience Interviews." https://facingdisability.com/video-interviews (accessed Feb. 25, 2022).

[2] M. Csobonyeiova, S. Polak, R. Zamborsky, and L. Danisovic, "Recent progress in the regeneration of spinal cord injuries by induced pluripotent stem cells," *Int. J. Mol. Sci.*, vol. 20, no. 15, Aug. 2019, doi: 10.3390/IJMS20153838.

[3] "Spinal cord injury." https://www.who.int/news-room/fact-sheets/detail/spinal-cord-injury (accessed Feb. 28, 2022).

[4] C. S. Nam, A. Nijholt, and F. Lotte, *Brain-computer interfaces handbook: technological and theoretical advances*. 2018.

[5] R. Aler, I. M. Galván, and J. M. Valls, "Applying evolution strategies to preprocessing EEG signals for brain-computer interfaces," *Inf. Sci. (Ny).*, vol. 215, pp. 53–66, 2012, doi: 10.1016/j.ins.2012.05.012.

[6] "The first fully-implanted 1000+ channel brain-machine interface - Neuralink." https://neuralink.com/blog/monkey-mindpong/ (accessed Mar. 01, 2022).

[7] J. Xu, S. Mitra, C. Van Hoof, R. F. Yazicioglu, and K. A. A. Makinwa, "Active Electrodes for Wearable EEG Acquisition: Review and Electronics Design Methodology," *IEEE Rev. Biomed. Eng.*, vol. 10, pp. 187–198, 2017, doi: 10.1109/RBME.2017.2656388.

[8] J. Vorwerk, Ü. Aydin, C. H. Wolters, and C. R. Butson, "Influence of head tissue conductivity uncertainties on EEG dipole reconstruction," *Front. Neurosci.*, vol. 13, no. JUN, p. 531, 2019, doi: 10.3389/FNINS.2019.00531/BIBTEX.

[9] M. Kaya, M. K. Binli, E. Ozbay, H. Yanar, and Y. Mishchenko, "Data descriptor: A large electroencephalographic motor imagery dataset for electroencephalographic brain computer interfaces," *Sci. Data*, vol. 5, no. August, pp. 1–16, 2018, doi: 10.1038/sdata.2018.211.

[10] "Introduction to the Nervous System | SEER Training." https://training.seer.cancer.gov/anatomy/nervous/ (accessed Mar. 08, 2022).

[11] F. F. Evans-Martin, *The nervous system*. 2012.

[12] G. J. Kress and S. Mennerick, "Action potential initiation and propagation: upstream influences on neurotransmission," *Neuroscience*, vol. 158, no. 1, p. 211, Jan. 2009, doi: 10.1016/J.NEUROSCIENCE.2008.03.021.

[13] D. Sadava, D. M. Hillis, H. H. Craig, and M. R. Berenbaum, *Life: The Science of Biology, 9th Edition*. 2011.

[14] "How do impulses travel across a synapse? ." https://socratic.org/questions/how-do-impulses-travel-across-a-synapse (accessed Mar. 09, 2022).

[15] "Neurons | Biology for Majors II." https://courses.lumenlearning.com/wm-biology2/chapter/neurons/ (accessed Mar. 09, 2022).

[16] B. L. Harty *et al.*, "Myelinating Schwann cells ensheath multiple axons in the absence of E3 ligase component Fbxw7," *Nat. Commun. 2019 101*, vol. 10, no. 1, pp. 1–12, Jul. 2019, doi: 10.1038/s41467-019-10881-y.

[17] F. F. Offner, "Ion flow through membranes and the resting potential of cells," *J. Membr. Biol.*, vol. 123, no. 2, pp. 171–182, Aug. 1991, doi: 10.1007/BF01998087.

[18] C. Leterrier, "The Axon Initial Segment: An Updated Viewpoint," *J. Neurosci.*, vol. 38, no. 9, pp. 2135–2145, Feb. 2018, doi: 10.1523/JNEUROSCI.1922-17.2018.

[19] M. H. Grider, R. Jessu, and R. Kabir, "Physiology, Action Potential," *StatPearls*, May 2021, Accessed: Mar. 12, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK538143/.

[20] "The Action Potential - YouTube." https://www.youtube.com/watch?v=HYLyhXRp298&ab_channel=BozemanScience (accessed Mar. 12, 2022).

[21] "Brain anatomy, Anatomy of the human brain | Mayfield Brain & Spine Cincinnati, Ohio." https://mayfieldclinic.com/pe-anatbrain.htm (accessed Mar. 14, 2022).

[22] "Brain Anatomy - Atlanta Brain and Spine Care." https://www.atlantabrainandspine.com/brain-anatomy/ (accessed Mar. 15, 2022).

[23] S. Sciacca, J. Lynch, I. Davagnanam, and R. Barker, "Midbrain, pons, and medulla: Anatomy and syndromes," *Radiographics*, vol. 39, no. 4, pp. 1110–1125, Jul. 2019, doi: 10.1148/RG.2019180126/ASSET/IMAGES/LARGE/RG.2019180126.FIG31A.JPEG.

[24] S. Jimsheleishvili and M. Dididze, "Neuroanatomy, Cerebellum," *StatPearls*, Jul. 2021, Accessed: Mar. 15, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK538167/.

[25] T. J. Torrico and S. Munakomi, "Neuroanatomy, Thalamus," *StatPearls*, Jul. 2021, Accessed: Mar. 14, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK542184/.

[26] "Hypothalamus: MedlinePlus Medical Encyclopedia." https://medlineplus.gov/ency/article/002380.htm (accessed Mar. 15, 2022).

[27] "Grey Matter vs White Matter in the Brain." https://www.spinalcord.com/blog/gray-matter-vs-white-matter-in-the-brain (accessed Mar. 15, 2022).

[28] "Motor Cortex (Section 3, Chapter 3) Neuroscience Online: An Electronic Textbook for the Neurosciences | Department of Neurobiology and Anatomy - The University of Texas Medical School at Houston." https://nba.uth.tmc.edu/neuroscience/m/s3/chapter03.html (accessed Mar. 15, 2022).

[29] A. Solodkin, P. Hlustik, E. E. Chen, and S. L. Small, "Fine modulation in network activation

during motor execution and motor imagery," *Cereb. Cortex*, vol. 14, no. 11, pp. 1246–1255, Nov. 2004, doi: 10.1093/CERCOR/BHH086.

[30] J. Munzert, B. Lorey, and K. Zentgraf, "Cognitive motor processes: The role of motor imagery in the study of motor representations," *Brain Res. Rev.*, vol. 60, no. 2, pp. 306–326, May 2009, doi: 10.1016/J.BRAINRESREV.2008.12.024.

[31] J. An *et al.*, "Cortical activation pattern for grasping during observation, imagery, execution, FES, and observation-FES integrated BCI: An fNIRS pilot study," *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, pp. 6345–6348, 2013, doi: 10.1109/EMBC.2013.6611005.

[32] N. Sharma, P. S. Jones, T. A. Carpenter, and J. C. Baron, "Mapping the involvement of BA 4a and 4p during Motor Imagery," *Neuroimage*, vol. 41, no. 1, pp. 92–99, May 2008, doi: 10.1016/J.NEUROIMAGE.2008.02.009.

[33] "Homunculus Sensory and Motor Cortex." https://www.ebmconsult.com/articles/homunculus-sensory-motor-cortex (accessed Mar. 15, 2022).

[34] "Motor Cortex Function and Location | Simply Psychology." https://www.simplypsychology.org/motor-cortex.html (accessed Mar. 15, 2022).

[35] J. M. Bekkers, "Pyramidal neurons," 2011, doi: 10.1016/j.cub.2011.10.037.

[36] F. Walter, F. Röhrbein, and A. Knoll, "Computation by Time," *Neural Process. Lett.*, vol. 44, no. 1, pp. 103–124, Aug. 2016, doi: 10.1007/S11063-015-9478-6.

[37] A. F. Jackson and D. J. Bolger, "The neurophysiological bases of EEG and EEG measurement: A review for the rest of us," *Psychophysiology*, vol. 51, no. 11, pp. 1061–1071, 2014, doi: 10.1111/psyp.12283.

[38] S. Næss *et al.*, "Biophysically detailed forward modeling of the neural origin of EEG and MEG signals," *Neuroimage*, vol. 225, p. 117467, Jan. 2021, doi: 10.1016/J.NEUROIMAGE.2020.117467.

[39] S. B. Rutkove, "Introduction to Volume Conduction."

[40] B. Burle, L. Spieser, C. Roger, L. Casini, T. Hasbroucq, and F. Vidal, "Spatial and temporal resolutions of EEG: Is it really black and white? A scalp current density view," *Int. J. Psychophysiol.*, vol. 97, no. 3, pp. 210–220, Sep. 2015, doi: 10.1016/J.IJPSYCHO.2015.05.004.

[41] S. P. Ahlfors *et al.*, "Cancellation of EEG and MEG signals generated by extended and distributed sources," *Hum. Brain Mapp.*, vol. 31, no. 1, p. 140, Jan. 2010, doi: 10.1002/HBM.20851.

[42] W. J. Freeman, M. D. Holmes, and B. C. Burke, "Spatial spectra of scalp EEG and EMG from awake humans," *Clin. Neurophysiol*, vol. 114, no. 6, pp. 1055–5060, 2003.

[43] F. Sauter-Starace *et al.*, "Long-Term Sheep Implantation of WIMAGINE®, a Wireless 64-Channel Electrocorticogram Recorder," *Front. Neurosci.*, vol. 13, p. 847, Aug. 2019, doi:

10.3389/FNINS.2019.00847/BIBTEX.

[44] W. M. Haddad, Q. Hui, and J. M. Bailey, "Human Brain Networks: Spiking Neuron Models, Multistability, Synchronization, Thermodynamics, Maximum Entropy Production, and Anesthetic Cascade Mechanisms," *Entropy 2014, Vol. 16, Pages 3939-4003*, vol. 16, no. 7, pp. 3939–4003, Jul. 2014, doi: 10.3390/E16073939.

[45] "Main features of the EEG amplifier explained | Bitbrain." https://www.bitbrain.com/blog/eeg-amplifier# (accessed Mar. 19, 2022).

[46] "Aliasing Effect - Wikimedia Commons." https://commons.wikimedia.org/wiki/File:AliasingSines.svg (accessed Mar. 19, 2022).

[47] J. Rosell, J. Colominas, P. Riu, R. Pallas-Areny, and J. G. Webster, "Skin Impedance From 1 Hz to 1 MHz," *IEEE Trans. Biomed. Eng.*, vol. 35, no. 8, pp. 649–652, 1988, doi: 10.1109/10.4599.

[48] P. Simmons and D. Harmon, "Common Mode Rejection Ratio TI Precision Labs-Current Sense Amplifiers."

[49] Analog devices, "Op Amp Common-Mode Rejection Ratio (CMRR)," 2009.

[50] L. V Marcuse, M. FIELDS, and J. YOO, *Rowan's primer of EEG*. 2016.

[51] L. Troy M., G. Joseph T., and F. Daniel P., "How Many Electrodes Are Really Needed for EEG-Based Mobile Brain Imaging?," *J. Behav. Brain Sci.*, vol. 2012, no. 03, pp. 387–393, Aug. 2012, doi: 10.4236/JBBS.2012.23044.

[52] V. Jurcak, D. Tsuzuki, and I. Dan, "10/20, 10/10, and 10/5 systems revisited: their validity as relative head-surface-based positioning systems," *Neuroimage*, vol. 34, no. 4, pp. 1600–1611, Feb. 2007, doi: 10.1016/J.NEUROIMAGE.2006.09.024.

[53] "Neoprene Headcap Component | Neuroelectrics." https://www.neuroelectrics.com/solution/spareparts-consumables/cap (accessed Mar. 22, 2022).

[54] "What are Brainwaves? - Improve Brain Health with Neurofeedback." https://www.sinhaclinic.com/what-are-brainwaves/ (accessed Mar. 23, 2022).

[55] "What are Brainwaves ? Types of Brain waves | EEG sensor and brain wave ." https://brainworksneurotherapy.com/what-are-brainwaves (accessed Mar. 23, 2022).

[56] "The Forgotten History of Alpha Brain Waves | Bitbrain." https://www.bitbrain.com/blog/alpha-brain-waves (accessed Mar. 22, 2022).

[57] J. J. Shih, D. J. Krusienski, and J. R. Wolpaw, "Brain-Computer Interfaces in Medicine," *Mayo Clin. Proc.*, vol. 87, no. 3, p. 268, 2012, doi: 10.1016/J.MAYOCP.2011.12.008.

[58] M. Clerc, L. Bougrain, and F. Lotte, *Brain-Computer Interfaces 1: Foundations and Methods*. 2016.

[59] V. Kaiser, A. Kreilinger, G. R. Müller-Putz, and C. Neuper, "First steps toward a motor imagery based stroke BCI: New strategy to set up a classifier," *Front. Neurosci.*, no. JUL, 2011, doi: 10.3389/FNINS.2011.00086/PDF.

[60] "Brain-Computer Interface - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/engineering/brain-computer-interface (accessed Mar. 24, 2022).

[61] K. J. Miller, G. Schalk, E. E. Fetz, M. Den Nijs, J. G. Ojemann, and R. P. N. Rao, "Cortical activity during motor execution, motor imagery, and imagery-based online feedback," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 107, no. 9, p. 4430, Mar. 2010, doi: 10.1073/PNAS.0913697107.

[62] N. Yamawaki, C. Wilke, Z. Liu, and B. He, "An enhanced time-frequency-spatial approach for motor imagery classification," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 14, no. 2, p. 250, Jun. 2006, doi: 10.1109/TNSRE.2006.875567.

[63] H. Yu, S. Ba, Y. Guo, L. Guo, and G. Xu, "Effects of Motor Imagery Tasks on Brain Functional Networks Based on EEG Mu/Beta Rhythm," *Brain Sci. 2022, Vol. 12, Page 194*, vol. 12, no. 2, p. 194, Jan. 2022, doi: 10.3390/BRAINSCI12020194.

[64] G. Pfurtscheller, "Functional brain imaging based on ERD/ERS," *Vision Res.*, vol. 41, no. 10–11, pp. 1257–1260, 2001, doi: 10.1016/S0042-6989(00)00235-2.

[65] Y. Jeon, C. S. Nam, Y. J. Kim, and M. C. Whang, "Event-related (De)synchronization (ERD/ERS) during motor imagery tasks: Implications for brain-computer interfaces," *Int. J. Ind. Ergon.*, vol. 41, no. 5, pp. 428–436, Sep. 2011, doi: 10.1016/J.ERGON.2011.03.005.

[66] S. C. Wriessnegger, G. R. Müller-Putz, C. Brunner, and A. I. Sburlea, "Inter- and Intra-individual Variability in Brain Oscillations During Sports Motor Imagery," *Front. Hum. Neurosci.*, vol. 14, p. 448, Oct. 2020, doi: 10.3389/FNHUM.2020.576241/BIBTEX.

[67] F. Škola, S. Tinková, and F. Liarokapis, "Progressive Training for Motor Imagery Brain-Computer Interfaces Using Gamification and Virtual Reality Embodiment," *Front. Hum. Neurosci.*, vol. 13, p. 329, Sep. 2019, doi: 10.3389/FNHUM.2019.00329/BIBTEX.

[68] "Motor Imagery Training - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/psychology/motor-imagery-training (accessed Mar. 25, 2022).

[69] J. E. Gehringer, D. J. Arpin, E. Heinrichs-Graham, T. W. Wilson, and M. J. Kurz, "Practice modulates motor-related beta oscillations differently in adolescents and adults," *J. Physiol.*, vol. 597, no. 12, pp. 3203–3216, Jun. 2019, doi: 10.1113/JP277326.

[70] "RehaBCI – complete BCI research bundle for rehabilitation | g.tec medical engineering." https://gtecmedical.wordpress.com/2011/05/04/new-rehabci-complete-bci-research-bundle-for-rehabilitation/ (accessed Mar. 26, 2022).

[71] P. Walsh, N. Kane, S. Butler, and N. Kane, "THE CLINICAL ROLE OF EVOKED POTENTIALS," *J Neurol Neurosurg Psychiatry*, vol. 76, pp. 16–22, 2005, doi: 10.1136/jnnp.2005.068130.

[72] A. Materka and P. Poryzała, “A Robust Asynchronous SSVEP Brain- Computer Interface Based on Cluster Analysis of Canonical Correlation Coefficients,” *Adv. Intell. Syst. Comput.*, vol. 300, pp. 3–14, 2014, doi: 10.1007/978-3-319-08491-6_1.

[73] I. Rejer and Ł. Cieszyński, “Independent component analysis for a low-channel SSVEP-BCI,” *Pattern Anal. Appl.*, vol. 22, no. 1, pp. 47–62, Feb. 2019, doi: 10.1007/S10044-018-0758-4/FIGURES/9.

[74] “EEG SSVEP Dataset II - Experimental Protocol - YouTube.” https://www.youtube.com/watch?v=xnjfuqRtAgE&ab_channel=MAMEMProject (accessed Mar. 27, 2022).

[75] “Neurofax EEG systems: the perfect brain insight.” http://content.page.mn/uploads/users/36/files/citmn/Datasheet_Neurofax_03.pdf (accessed Mar. 30, 2022).

[76] “Re-referencing - EEGLAB Wiki.” https://eeglab.org/tutorials/ConceptsGuide/rereferencing_background.html (accessed Apr. 03, 2022).

[77] K. A. Ludwig, R. M. Miriani, N. B. Langhals, M. D. Joseph, D. J. Anderson, and D. R. Kipke, “Using a Common Average Reference to Improve Cortical Neuron Recordings From Microelectrode Arrays,” *J. Neurophysiol.*, vol. 101, no. 3, p. 1679, Mar. 2009, doi: 10.1152/JN.90989.2008.

[78] X. Yu, P. Chum, and K. B. Sim, “Analysis the effect of PCA for feature reduction in non-stationary EEG based motor imagery of BCI system,” *Optik (Stuttg).*, vol. 125, no. 3, pp. 1498–1502, Feb. 2014, doi: 10.1016/J.IJLEO.2013.09.013.

[79] “Power Spectral Density - an overview | ScienceDirect Topics.” https://www.sciencedirect.com/topics/computer-science/power-spectral-density (accessed Apr. 04, 2022).

[80] “Estimate the Power Spectrum in MATLAB - MATLAB & Simulink.” https://www.mathworks.com/help/dsp/ug/estimate-the-power-spectrum-in-matlab.html (accessed Apr. 04, 2022).

[81] “mne.io.Raw — MNE 1.0.0 documentation.” https://mne.tools/stable/generated/mne.io.Raw.html#mne.io.Raw.plot_psd (accessed Apr. 04, 2022).

[82] G. Pfurtscheller and F. H. Lopes Da Silva, “Event-related EEG/MEG synchronization and desynchronization: basic principles,” *Clin. Neurophysiol.*, vol. 110, no. 11, pp. 1842–1857, Nov. 1999, doi: 10.1016/S1388-2457(99)00141-8.

[83] B. Graimann, J. E. Huggins, S. P. Levine, and G. Pfurtscheller, “Visualization of significant ERD/ERS patterns in multichannel EEG and ECoG data,” *Clin. Neurophysiol.*, vol. 113, no. 1, pp. 43–47, Jan. 2002, doi: 10.1016/S1388-2457(01)00697-6.

[84] “Compute and visualize ERDS maps — MNE documentation.”

https://mne.tools/dev/auto_examples/time_frequency/time_frequency_erds.html (accessed Apr. 05, 2022).

[85] "FIR Filters - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/engineering/fir-filters (accessed Apr. 09, 2022).

[86] "FIR and IIR Transfer Functions." https://www.astro.rug.nl/~vdhulst/SignalProcessing/Hoorcolleges/college07.pdf (accessed Apr. 09, 2022).

[87] "Digital filter design - Introduction." https://www.gaussianwaves.com/2020/02/introduction-to-digital-filter-design/ (accessed Apr. 09, 2022).

[88] "FIR Filter Properties." https://dspguru.com/dsp/faqs/fir/properties/ (accessed Apr. 09, 2022).

[89] "Difference Between FIR Filter and IIR Filter." https://circuitglobe.com/difference-between-fir-filter-and-iir-filter.html (accessed Apr. 09, 2022).

[90] "Design Examples of FIR Filters Using the Window Method - Technical Articles." https://www.allaboutcircuits.com/technical-articles/design-examples-of-fir-filters-using-window-method/ (accessed Apr. 09, 2022).

[91] L. Sun, Y. Liu, and P. J. Beadle, "Independent component analysis of EEG signals," *Proc. 2005 IEEE Int. Work. VLSI Des. Video Technol. IWVDVT 2005*, pp. 293–296, 2005, doi: 10.1109/IWVDVT.2005.1504590.

[92] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," Accessed: Apr. 11, 2022. [Online]. Available: www.elsevier.com/locate/neunet.

[93] J. Dammers and M. Schiek, "Detection of Artifacts and Brain Responses Using Instantaneous Phase Statistics in Independent Components," *Magnetoencephalography*, Nov. 2011, doi: 10.5772/27523.

[94] J. Sebek, R. Bortel, and P. Sovka, "Suppression of overlearning in independent component analysis used for removal of muscular artifacts from electroencephalographic records," 2018, doi: 10.1371/journal.pone.0201900.

[95] C. D. Virgilio Gonzalez, J. H. Sossa Azuela, E. Rubio Espino, and V. H. Ponce Ponce, *Classification of motor imagery EEG signals with CSP filtering through neural networks models*, vol. 11288 LNAI, no. October 2018. Springer International Publishing, 2018.

[96] J. Khan, M. H. Bhatti, U. G. Khan, and R. Iqbal, "Multiclass EEG motor-imagery classification with sub-band common spatial patterns," *Eurasip J. Wirel. Commun. Netw.*, vol. 2019, no. 1, 2019, doi: 10.1186/s13638-019-1497-y.

[97] A. Barachant, S. Bonnet, M. Congedo, C. Jutten, and S. Bonnet, "Common Spatial Pattern revisited by Riemannian Geometry," p. 472, 2010, doi: 10.1109/MMSP.2010.5662067ï.

[98] H. Ramoser, J. Müller-Gerking, and G. Pfurtscheller, "Optimal spatial filtering of single trial EEG

during imagined hand movement," *IEEE Trans. Rehabil. Eng.*, vol. 8, no. 4, pp. 441–446, 2000, doi: 10.1109/86.895946.

[99] B. Yang, M. He, Y. Liu, and Z. Han, "Multi-class feature extraction based on common spatial patterns of multi-band cross filter in BCIs," *Commun. Comput. Inf. Sci.*, vol. 326 CCIS, no. PART 1, pp. 399–408, 2012, doi: 10.1007/978-3-642-34381-0_46.

[100] "Eigenvectors and eigenvalues | Essence of linear algebra - YouTube." https://www.youtube.com/watch?v=PFDu9oVAE-g&ab_channel=3Blue1Brown (accessed Apr. 15, 2022).

[101] A. Zabidi, W. Mansor, Y. K. Lee, and C. W. N. F. Che Wan Fadzal, "Short-time Fourier Transform analysis of EEG signal generated during imagined writing," *Proc. 2012 Int. Conf. Syst. Eng. Technol. ICSET 2012*, 2012, doi: 10.1109/ICSENGT.2012.6339284.

[102] T. H. Shovon, Z. Al Nazi, S. Dash, and M. F. Hossain, "Classification of motor imagery EEG signals with multi-input convolutional neural network by augmenting STFT," *2019 5th Int. Conf. Adv. Electr. Eng. ICAEE 2019*, pp. 398–403, Sep. 2019, doi: 10.1109/ICAEE48663.2019.8975578.

[103] "Short-Time Fourier Transform - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/engineering/short-time-fourier-transform (accessed Apr. 17, 2022).

[104] D. Zwillinger, S. Kokoska, B. Raton, L. New, and Y. Washington, "Standard probability and Statistics tables and formulae CRC," 2000, Accessed: Apr. 18, 2022. [Online]. Available: www.crcpress.com.

[105] "Kurtosis - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/earth-and-planetary-sciences/kurtosis (accessed Apr. 18, 2022).

[106] "Kurtosis: Leptokurtic, Mesokurtic, and Platykurtic distributions." https://analystprep.com/cfa-level-1-exam/quantitative-methods/kurtosis/ (accessed Apr. 18, 2022).

[107] S. AydIn, H. M. Saraoğlu, and S. Kara, "Log energy entropy-Based EEG classification with multilayer neural networks in seizure," *Ann. Biomed. Eng.*, vol. 37, no. 12, pp. 2626–2630, Dec. 2009, doi: 10.1007/S10439-009-9795-X.

[108] B. F. Yan, A. Miyamoto, and E. Bru¨hwilerbru¨hwiler, "Wavelet transform-based modal parameter identification considering uncertainty," *J. Sound Vib.*, vol. 291, pp. 285–301, 2006, doi: 10.1016/j.jsv.2005.06.005.

[109] "WA Analytic Wavelet Transform VI - LabVIEW 2010 Advanced Signal Processing Toolkit Help - National Instruments." https://zone.ni.com/reference/en-XX/help/371419D-01/lvwavelettk/wa_analytic_wavelet_transform/ (accessed Apr. 18, 2022).

[110] "Introduction to k-fold Cross-Validation in Python - SQLRelease." https://sqlrelease.com/introduction-to-k-fold-cross-validation-in-python (accessed Apr. 26, 2022).

[111] “Linear Discriminant Analysis, Explained | by YANG Xiaozhou | Towards Data Science.” https://towardsdatascience.com/linear-discriminant-analysis-explained-f88be6c1e00b (accessed Apr. 21, 2022).

[112] “PCA 11: linear discriminant analysis - YouTube.” https://www.youtube.com/watch?v=6Ht-nIf_NKc&ab_channel=VictorLavrenko (accessed Apr. 21, 2022).

[113] “Support Vector Machine — Introduction to Machine Learning Algorithms | by Rohith Gandhi | Towards Data Science.” https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (accessed Apr. 21, 2022).

[114] “Support Vector Machines Part 1 - YouTube.” https://www.youtube.com/watch?v=efR1C6CvhmE&t=424s (accessed Apr. 21, 2022).

[115] “Support Vector Machines - Linear SVM.” http://www.adeveloperdiary.com/data-science/machine-learning/support-vector-machines-for-beginners-linear-svm/ (accessed Apr. 21, 2022).

[116] “Random Forest Algorithm Clearly Explained - YouTube.” https://www.youtube.com/watch?v=v6VJ2RO66Ag&ab_channel=NormalizedNerd (accessed Apr. 21, 2022).

[117] “What are Neural Networks? | IBM.” https://www.ibm.com/cloud/learn/neural-networks (accessed Apr. 23, 2022).

[118] C. Lv *et al.*, “Levenberg-marquardt backpropagation training of multilayer neural networks for state estimation of a safety-critical cyber-physical system,” *IEEE Trans. Ind. Informatics*, vol. 14, no. 8, pp. 3436–3446, Aug. 2018, doi: 10.1109/TII.2017.2777460.

[119] “What is Overfitting? | IBM.” https://www.ibm.com/cloud/learn/overfitting (accessed Apr. 26, 2022).

[120] “Best Practice Guide - Deep Learning.” https://www.researchgate.net/publication/332190148_Best_Practice_Guide_-_Deep_Learning (accessed Apr. 25, 2022).

[121] “2D Convolution in Image Processing - Technical Articles.” https://www.allaboutcircuits.com/technical-articles/two-dimensional-convolution-in-image-processing/ (accessed Apr. 25, 2022).

[122] “Convolutional Neural Network | Introduction to Data Science.” https://scientistcafe.com/ids/convolutional-neural-network.html (accessed Apr. 25, 2022).

[123] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *J. Big Data 2021 81*, vol. 8, no. 1, pp. 1–74, Mar. 2021, doi: 10.1186/S40537-021-00444-8.

[124] “What do you mean by Convolutional Neural Network? - Data Science, AI and ML.” https://discuss.boardinfinity.com/t/what-do-you-mean-by-convolutional-neural-network/8533 (accessed Apr. 25, 2022).

[125] "Certifications | Arduino Documentation | Arduino Documentation." https://docs.arduino.cc/certifications/ (accessed May 16, 2022).

[126] "Recycling Passport | Nihon Kohden Global Site." https://www.nihonkohden.com/recycle/index2.html (accessed May 16, 2022).

[127] "GeForce GTX 1080 | Specifications | GeForce." https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1080/specifications/ (accessed May 16, 2022).

[128] "GUIA PRÀCTICA PER AL CÀLCUL D'EMISSIONS DE GASOS AMB EFECTE D'HIVERNACLE (GEH)." https://canviclimatic.gencat.cat/web/.content/04_ACTUA/Com_calcular_emissions_GEH/guia_de_calcul_demissions_de_co2/190301_Guia-practica-calcul-emissions_CA.pdf (accessed May 16, 2022).

Annexes

These annexes will recollect the most important codes used in the thesis; the scripts are commented and organized (in classes and functions) so that the reader can understand them more easily.

A1. Annex 1: Left vs right-hand imagery (Organized in classes) [Left_vs_Right.py]

```python
# -*- coding: utf-8 -*-
"""
LEFT VS RIGHT-HAND IMAGERY.
In this script, numerous classes used for preprocessing, processing, and classifying left vs right-hand imageries
will be coded.


@author: Ali Abdul Ameer Abbas
"""

import numpy as np
import matplotlib.pyplot as plt
import mne
import pymatreader
import scipy.stats
import pandas as pd
from mne.decoding import CSP

from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis # LDA
from sklearn.model_selection import ShuffleSplit, cross_val_score
from sklearn.svm import SVC                              # SVM
from sklearn.neighbors import KNeighborsClassifier  # KNN
from sklearn.naive_bayes import GaussianNB               # Naive Bayesian
from sklearn.ensemble import RandomForestClassifier # Random Forest
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis #QDA                          #
QDA
from sklearn.model_selection import learning_curve
from sklearn.metrics import plot_confusion_matrix, confusion_matrix, precision_score, recall_score, auc,
roc_curve
from sklearn.model_selection import RandomizedSearchCV


from scipy.stats import norm
import pywt



#%%
class PreprocessDataset:
    """
    This class will preprocess a file containing EEG recordings.
    It assigns a common reference, apply an ICA, filter the data. It also segments and annotates the
data.

    """

    def __init__(self, path, montage = 'standard_1020', show_plots = False, bad_chan = []):

        """
        Initialize the class by reading and reconstructing the signals.
        INPUTS:
            montage --> The reference selected.
            path --> The path of the file.
            bad_channels --> Bad channels to eliminate.
            show_plots --> To show the plots of the EEG recording.
        """
        # Convert the matlab files into a python compatible file.
        self.struct = pymatreader.read_mat(path, ignore_fields=['previous'], variable_names = 'o')
        self.struct = self.struct['o']


        self.chann = self.struct['chnames'] # Save the channels names.
        self.chann = self.chann[:-1]        # Eliminate the X3 channel, which is only used for the data adquisition.

        #Set important properties for the dataset.
        self.info  = mne.create_info(ch_names = self.chann,
                               sfreq = float(self.struct['sampFreq']),ch_types='eeg',
                               montage= montage, verbose=None)

        #Create the data variable.
        self.data   = np.array(self.struct['data'])
        self.data   = self.data[:,:-1]
        self.data_V = self.data*1e-6        # mne reads units are in Volts, convert it to uV.

        # Create the raw instance for working with the data in the MNE library.
        self.raw   = mne.io.RawArray(self.data_V.transpose(),self.info, copy = "both")
        self.raw.info['bads'] = bad_chan # exclude bad channels

        # Show the raw data if show_plot is true.
        if show_plots:
            self.raw.plot(title = 'Raw signal')

    def channel_reference(self, ref = 'average' ):

        """
        Set a reference to the data.
        INPUTS:
            ref --> Reference.

        OUTPUTS:
            raw --> Referenced data.
        """

        self.raw = self.raw.set_eeg_reference(ref, projection= False)

        return(self.raw)

    def MNE_bandpass_filter(self, HP = 15, LP = 26, filt_type = 'firwin', show_plots = False, skip_by_annotation='edge'):

        """
        Apply a BP FIR filter.

        INPUTS:
            LP,HP --> Low Pass filter, High Pass filter.
            filt_type --> filter type.
            show_plots --> To show the plots of the EEG recording.
        OUTPUTS:
            raw --> Filtered data.
        """
        self.raw.filter(HP, LP, fir_design = filt_type, skip_by_annotation = 'edge')

        if show_plots:
            self.raw.plot(title = 'Filtered Raw')
            self.raw.plot_psd()


        return(self.raw)


    def apply_ICA(self, n_components = 10, ICA_exclude = [0], show_plots = False):

        """
        Apply the ICA for removing eye blinking, EMG signals, EKG signals, among other artifacts.

        INPUTS:
            n_components --> Number of components of the ICA.
            ICA_exclude --> Array used to eliminate components from teh signals.
            show_plots --> To show the plots of the EEG recording.
        OUTPUTS:
            raw_ica --> Data after ICA.
        """
        # Apply ICA for blink removal
        self.ica = mne.preprocessing.ICA(n_components = n_components, random_state = 0, 
method='fastica')

        # The MNE documentation recommends to HP filter the ICA at 1 Hz.
        filt_raw = self.raw.copy()                        # Create a copy of teh data
        filt_raw.filter(l_freq=1., h_freq=None)     # HP filter at 1 Hz
        self.ica.fit(filt_raw)                            # Find the ICA parameters.

        # Plot the ICA components to detect artifacts.
        if show_plots:
            self.ica.plot_components(outlines = 'skirt', colorbar=True, contours = 0)
            self.ica.plot_sources(self.raw)

        # Once the bad components are detected, we procede to remove them.
        self.ica.exclude = ICA_exclude

        self.raw_ica = self.ica.apply(self.raw.copy(), exclude = self.ica.exclude)

```

```python
        # Plot the results.
        if show_plots:
            self.raw.plot()
            self.raw_ica.plot()

        return self.raw_ica

    def type_pos(self, sfreq = 200.0):
        """
        Gets the marker position.
        Just left and right positions, NO PASS marker
        Only takes into account the right and left class, not the passive class.

        INPUTS:
            sfreq --> Sampling frequency.
        OUTPUTS:
            mark --> markers.
            pos --> position of the marker.
            time --> time of the marker.
        """
        self.markers =  np.array(self.struct['marker']).transpose() # Include the markers

        # Intitialize the output arrays.
        mark = []
        pos  = []
        time = []

        desc = ['left', 'right'] # for assigning the movements--> left = 1, right = 2, pass = 3

        #Evaluate the markers in order to find the position and time of each marker, as well as its type.
        for i in range(len(self.markers)-1):
            if self.markers[i]==0 and self.markers[i+1] != 0 and (self.markers[i+1] in [1,2]):

                mark.append(desc[self.markers[i+1]-1])
                pos.append((i+2))
                time.append((i+2)/sfreq)
            else:
                continue

        return mark, pos, time

    def add_annotations(self, duration = 1):
        """
        This function uses the tye_pos function to annotate the markers.
        INPUTS:
            duration -->Its the duration of the annotation.
        """

        [self.mark, self.pos, self.time] = self.type_pos()                 # Use the typo pos function.

        self.annotations = mne.Annotations(self.time, duration, self.mark) # Annotate the markers.
        self.raw.set_annotations(self.annotations)                         # Set the annotation.

    def create_epochs(self, raw_ica, tmin= 2.1, tmax= 3, show_plots = False):
        """'
        Segment and label all the data.
        INPUTS:
            raw_ica --> Preprocessed data.
            tmin, tmax --> times for segmenting the data.
            show_plots --> To show the plots of the EEG recording.
        OUTPUTS:
            epochs.get_data() --> get the epochs.
            labels --> get the labels.
        """
        self.events = mne.events_from_annotations(self.raw) # extract the events from the annotations.

        self.picks = mne.pick_types(self.raw.info, meg=False, eeg=True, stim=False, eog=False,
                                    exclude='bads')  # Exclude the bad events.

        # Save the epochs inside a variable.
        self.epochs = mne.Epochs(self.raw_ica, self.events[0], event_id = self.events[1],
                                 preload = True, tmin=tmin, tmax=tmax, baseline=None, picks = self.picks)
        # Save the labels inside a variable.
        self.labels = self.epochs.events[:, -1]

        # Plot the segmented data.
        if show_plots:
            self.epochs.plot()

        return(self.epochs.get_data(), self.labels)

    def plot_electrodes_and_psd(self):
        """
        Plot the position of the electrodes and the psd.
        """

        fig = self.raw.plot_psd(dB=False, xscale='linear', estimate='power')  # show the power spectrum density
        fig.suptitle('Power spectral density (PSD)',  fontsize = 30)
        plt.show()

        fig = self.raw.plot_psd(dB=True, xscale='linear', average = False )  # show the power spectrum density
        fig.suptitle('Power spectral density (PSD) (dB)', fontsize = 30)
        plt.show()


```

```python


        fig = plt.figure()
        ax2d = fig.add_subplot(121)
        ax3d = fig.add_subplot(122, projection='3d')
        self.raw.plot_sensors(show_names = True, axes = ax2d)
        self.raw.plot_sensors(show_names = False, axes = ax3d, kind = '3d')
        # Make a sphere to project the 3d electrodes
        u = np.linspace(0, 2 * np.pi, 200)
        v = np.linspace(0, np.pi, 200)
        x = 0.1 * np.outer(np.cos(u), np.sin(v))
        y = 0.1 * np.outer(np.sin(u), np.sin(v))
        z = 0.15 * np.outer(np.ones(np.size(u)), np.cos(v))
        ax3d.plot_surface(x, y, z,alpha = 0.8, color='navajowhite')
        plt.show()






#%%

# Now a class for processing the data will be created. Mainly, this class will perform the CSP, the statistical features, and the STFT.

class ProcessDataset():
    """
    This class will process a file containing EEG recordings.
    It is a subclass of the PreprocessDataset class.
    Mainly, this class will perform the CSP.

    """

    def __init__(self, class_pre, epochs, labels):

        """
        Initialize the class by getting the labels and epochs.
        INPUTS:
            class_pre --> Introduce any class of PreprocessDataset so that important information (info, chnames, ...) are defined.
            epochs --> Epochs obtained from the PreprocessingDataset class. This epochs should combine multiple subjects.
            labels --> Labels obtained from the PreprocessingDataset class. This epochs should combine multiple subjects.
        """
        # Define important parameters for the class:

        self.struct = class_pre.struct
        self.chann  = class_pre.chann
        self.info   = class_pre.info
        self.data   = class_pre.data
        self.data_V = class_pre.data_V
        self.raw    = class_pre.raw
        self.raw_ica= class_pre.raw_ica
        self.ica    = class_pre.ica
        self.markers= class_pre.markers
        self.mark   = class_pre.mark
        self.time   = class_pre.time
        self.pos    = class_pre.pos

        self.epochs = epochs
        self.labels = labels
        #Only done to plot the CSP.
        self.epochs_subj = class_pre.epochs.get_data()
        self.labels_subj = class_pre.labels


    # This method is for the CSP.
    def create_CSP(self, reg = 'oas', n_components = 10, show_plot = False):

        """
        Create the CSP to obtain spatial featuers.
        INPUTS:
            reg --> reg function. reg may be 'auto', 'empirical', 'diagonal_fixed',
                    'ledoit_wolf', 'oas', 'shrunk', 'pca', 'factor_analysis', 'shrinkage'
            n_components --> Number of components.
            show_plot --> Compute the CSP for the Subject selected and display the CSP.
                          Notice that the CSP plotted are only from the selected subject
        OUTPUTS:
            csp --> The CSP.
        """
        # Create the CSP structure.
        self.csp = CSP(n_components= n_components, reg = reg, rank = 'info') # Very important to set rank to info, otherwise a rank problem may occur

        # Plot the CSP.
        if show_plot:
            #IMPORTANT, set_baseline, must be 0
            csp_subj = CSP(n_components= n_components, reg = reg, rank = 'info') # Very important to set rank to info, otherwise a rank problem may occur.
            csp_subj.fit_transform(self.epochs_subj, self.labels_subj)

            csp_subj.plot_patterns(self.info, ch_type='eeg', units='Patterns (AU)', size=1.5)

        return(self.csp)


```

```


#%%

class Model_LR:
    """
    Model Left/Right. Class to organize the AI models with the best reults used to distinguis between Left and Right-hand imageries.
    """

    def ML_classification(self, csp, epochs_train, labels_train, epochs_test, labels_test, classif = 'RF'):
        """
        Uses ML classifiers in conjunction with CSP to train and test the motor imageries.

        INPUTS:
            csp --> Introduce the CSP.
            epochs_train, labels_train --> Introduce the epochs and labels for training the models.
            epochs_test, labels_test --> Introduce the epochs and labels for testing the models.
            classif --> The classifier used. (LDA, RF, SVM, KNN, NB, QDA)
        OUTPUTS:
            clf --> return the model.
        """

        # Initialize a score array.
        self.scores = []

        self.labels = labels_train
        self.epochs_data_train = epochs_train

        self.epochs_test = epochs_test
        self.labels_test = labels_test

```

```python
        self.csp = csp
        # set the validation and test data
        self.cv = ShuffleSplit(10, test_size=0.2, random_state=None)
        self.cv_split = self.cv.split(self.epochs_data_train)

        # Assemble a classifier

        self.lda = LinearDiscriminantAnalysis()
        self.svm = SVC(kernel ='rbf', gamma = 0.00001, C = 10000)
        self.knn = KNeighborsClassifier(n_neighbors = 25, weights = 'uniform')
        self.NB  = GaussianNB()
        self.RF  = RandomForestClassifier(n_estimators = 100,
                                     min_samples_split = 16,
                                     min_samples_leaf = 4,
                                     max_features = 'sqrt',
                                     max_depth = 8,
                                     bootstrap = True)
        self.qda = QuadraticDiscriminantAnalysis()




        # Use scikit-learn Pipeline with cross_val_score function
        if classif == 'LDA':
            self.clf = Pipeline([('CSP', self.csp), ('LDA', self.lda)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'QDA':
            self.clf = Pipeline([('CSP', self.csp), ('QDA', self.qda)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'SVM':
            self.clf = Pipeline([('CSP', self.csp), ('SVM', self.svm)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'KNN':
            self.clf = Pipeline([('CSP', self.csp), ('KNN', self.knn)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'NB':
            self.clf = Pipeline([('CSP', self.csp), ('NB', self.NB)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        else:
            self.clf = Pipeline([('CSP', self.csp), ('RF', self.RF)])

        self.scores = cross_val_score(self.clf, self.epochs_data_train, self.labels, cv=self.cv, n_jobs=1)

        self.clf.fit(self.epochs_data_train[:], self.labels[:])  # We fit the model for making it usable


        self.testing_score = self.clf.score(self.epochs_test, self.labels_test) #Testing Data (Never seen before)
        print('\nAccuracy: ', self.testing_score)
        print('Precision: ', precision_score(self.labels_test[:], self.clf.predict(self.epochs_test[:],))) # It gives more importance that the pass class is detected correctly. TruePositive/(TruePositives+FalsePositives)
        print('Recall: ', recall_score(self.labels_test[:], self.clf.predict(self.epochs_test,)))

        # Printing the results
        self.class_balance = np.mean(self.labels == self.labels[0])
        self.class_balance = max(self.class_balance, 1. - self.class_balance)
        print("Classification accuracy: %f +- %f / Chance level: %f" % (np.mean(self.scores), np.std(self.scores),
                                                                          self.class_balance))
        plt.rc('xtick', labelsize=20)
        plt.rc('ytick', labelsize=20)
        plt.rcParams.update({'font.size': 16})
        plt.rc('axes', titlesize=30, labelsize=25)
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```

        plot_confusion_matrix(self.clf, self.epochs_test, self.labels_test)
        plt.show()

        return self.clf

    def test_model(self, epochs_test, labels_test, show_plot = False):
        """
        The only purpose of this function is to test the models for each subject.
        It assess accuracy, precision, and recall.
        Also plots the confusion matrix.
        Always run this method after the ML_classification function.
        INPUTS:
            epochs_test --> Epochs to test.
            labels_test --> Labels to have the ground truth.
            show_plot --> To show the confunsion matrix.
        """

        self.testing_score = self.clf.score(epochs_test, labels_test) #Testing Data (Never seen before)
        print('Accuracy: ', self.testing_score)
        print('Precision: ', precision_score(labels_test[:], self.clf.predict(epochs_test[:],))) # It gives more importance that the pass class is detected correctly. TruePositive/(TruePositives+FalsePositives)
        print('Recall: ', recall_score(labels_test[:], self.clf.predict(epochs_test,)))

        if show_plot:
            plt.rc('xtick', labelsize=20)
            plt.rc('ytick', labelsize=20)
            plt.rcParams.update({'font.size': 16})
            plt.rc('axes', titlesize=30, labelsize=25)

            plot_confusion_matrix(self.clf, epochs_test, labels_test)
            plt.show()
```

A2. Annex 2: Left vs right-hand imagery (Executable) [Main_Left_vs_Right.py]

```python
# -*- coding: utf-8 -*-
"""
Main script for training and testing the AI models for the Left vs Right-hand Imagery.

@author: Ali Abdul Ameer Abbas
"""

# Import basic libraries
import numpy as np
import matplotlib.pyplot as plt
import pickle

# Import the modules that were created in Left_vs_Right.py
from Left_vs_Right import PreprocessDataset, ProcessDataset, Model_LR

#%% Plot configuration.
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.rc('axes', titlesize=30, labelsize=25)

#%%
# Firstly, we must define the paths of dataset' files.

path1_C = '../Data/CLA/CLASubjectC1512233StLRHand.mat'
path2_C = '../Data/CLA/CLASubjectC1512163StLRHand.mat'
path3_C = '../Data/CLA/CLASubjectC1511263StLRHand.mat'


```

```
path1_B = '../Data/CLA/CLASubjectB1510193StLRHand.mat'
path2_B = '../Data/CLA/CLASubjectB1510203StLRHand.mat'
path3_B = '../Data/CLA/CLASubjectB1512153StLRHand.mat'


path1_E = '../Data/CLA/CLASubjectE1512253StLRHand.mat'
path2_E = '../Data/CLA/CLASubjectE1601193StLRHand.mat'
path3_E = '../Data/CLA/CLASubjectE1601223StLRHand.mat'


path1_F = '../Data/CLA/CLASubjectF1509163StLRHand.mat'
path2_F = '../Data/CLA/CLASubjectF1509173StLRHand.mat'
path3_F = '../Data/CLA/CLASubjectF1509283StLRHand.mat'

fs= 200 # Define the sampling frequency.

#%%
# Apply the preprocessing class to the data.

#Initialize the classes for each subject.

SubjC_1 = PreprocessDataset(path = path1_C, show_plots=True)
SubjC_2 = PreprocessDataset(path = path2_C)
SubjC_3 = PreprocessDataset(path = path3_C)

SubjE_1 = PreprocessDataset(path = path1_E)
SubjE_2 = PreprocessDataset(path = path2_E)
SubjE_3 = PreprocessDataset(path = path3_E)

SubjF_1 = PreprocessDataset(path = path1_F)
SubjF_2 = PreprocessDataset(path = path2_F)
SubjF_3 = PreprocessDataset(path = path3_F)

```

```
# Set the reference.
SubjC_1.channel_reference()
SubjC_2.channel_reference()
SubjC_3.channel_reference()

SubjE_1.channel_reference()
SubjE_2.channel_reference()
SubjE_3.channel_reference()

SubjF_1.channel_reference()
SubjF_2.channel_reference()
SubjF_3.channel_reference()

# Filter the data.

SubjC_1.MNE_bandpass_filter(HP = 15, LP = 26, show_plots=True)
SubjC_2.MNE_bandpass_filter(HP = 15, LP = 26)
SubjC_3.MNE_bandpass_filter(HP = 15, LP = 26)

SubjE_1.MNE_bandpass_filter(HP = 15, LP = 26)
SubjE_2.MNE_bandpass_filter(HP = 15, LP = 26)
SubjE_3.MNE_bandpass_filter(HP = 15, LP = 26)

SubjF_1.MNE_bandpass_filter(HP = 15, LP = 26)
SubjF_2.MNE_bandpass_filter(HP = 15, LP = 26)
SubjF_3.MNE_bandpass_filter(HP = 15, LP = 26)

# Add annotations.

SubjC_1.add_annotations()
SubjC_2.add_annotations()
SubjC_3.add_annotations()

```

```
SubjE_1.add_annotations()
SubjE_2.add_annotations()
SubjE_3.add_annotations()

SubjF_1.add_annotations()
SubjF_2.add_annotations()
SubjF_3.add_annotations()

# Apply ICA.

ica_C_1 = SubjC_1.apply_ICA(ICA_exclude = [], show_plots=True)
ica_C_2 = SubjC_2.apply_ICA(ICA_exclude = [])
ica_C_3 = SubjC_3.apply_ICA(ICA_exclude = [])

ica_E_1 = SubjE_1.apply_ICA(ICA_exclude = [])
ica_E_2 = SubjE_2.apply_ICA(ICA_exclude = [])
ica_E_3 = SubjE_3.apply_ICA(ICA_exclude = [])

ica_F_1 = SubjF_1.apply_ICA(ICA_exclude = [])
ica_F_2 = SubjF_2.apply_ICA(ICA_exclude = [])
ica_F_3 = SubjF_3.apply_ICA(ICA_exclude = [])

# Create Epochs.

[epoch1_C, label1_C] = SubjC_1.create_epochs(ica_C_1, tmin = 2.1, tmax = 3, show_plots=True)
[epoch2_C, label2_C] = SubjC_2.create_epochs(ica_C_2, tmin = 2.1, tmax = 3)
[epoch3_C, label3_C] = SubjC_3.create_epochs(ica_C_3, tmin = 2.1, tmax = 3)

[epoch1_E, label1_E] = SubjE_1.create_epochs(ica_E_1, tmin = 2.2, tmax = 3.1)
[epoch2_E, label2_E] = SubjE_2.create_epochs(ica_E_2, tmin = 2, tmax = 2.9)
[epoch3_E, label3_E] = SubjE_3.create_epochs(ica_E_3, tmin = 2.2, tmax = 3.1)

[epoch1_F, label1_F] = SubjF_1.create_epochs(ica_F_1, tmin = 1, tmax = 1.9)
```

```
[epoch2_F, label2_F] = SubjF_2.create_epochs(ica_F_2, tmin = 1.3, tmax = 2.2)
[epoch3_F, label3_F] = SubjF_3.create_epochs(ica_F_3, tmin = 1.4, tmax = 2.3)

# Plot the elctrode positions and the PSD in dB and not dB.
SubjC_1.plot_electrodes_and_psd()

# Now, the data of all the subjects will be concatenated, and separated in training and testing.

epochs_train = np.concatenate((epoch1_C, epoch2_C, epoch1_E,epoch2_E, epoch1_F,epoch2_F))  # Epochs for training.
labels = np.concatenate((label1_C, label2_C, label1_E, label2_E, label1_F,label2_F))       # Labels for training.
epochs_test = np.concatenate((epoch3_C, epoch3_E, epoch3_F))                               # Epochs for testing.
label_test = np.concatenate((label3_C, label3_E, label3_F))                                # Labels for testing.



#%% Obtain features.

Train_feats = ProcessDataset(SubjC_1, epochs_train, labels)  # Create a class to process the data.
Test_feats = ProcessDataset(SubjC_1, epochs_test, label_test)  # Create a class to process the data.

# Obteain the CSP object.
csp = Train_feats.create_CSP(show_plot = True)# Create the CSP




#%% Train and test the models

Subj_All = Model_LR()
```

```

# Test and train all the subject at once.
clf = Subj_All.ML_classification(csp, epochs_train, labels, epochs_test, label_test, classif = 'RF')

print('\nAll Subjects:')
# Test Subject C.
Subj_All.test_model(epochs_test, label_test, show_plot = True)
print('\nSubject C:')
# Test Subject C.
Subj_All.test_model(epoch3_C, label3_C )
print('\nSubject E:')
# Test Subject E.
Subj_All.test_model(epoch3_E, label3_E )
print('\nSubject F:')
# Test Subject F.
Subj_All.test_model(epoch3_F, label3_F )

# Save the model.

with open('LeftRight_Classification_More_Data_all.pkl','wb') as f:
    pickle.dump(clf,f)

```

A3. Annex 3: Imagery vs Resting state (Organized in classes) [Imagery_vs_Resting.py]

```python
# -*- coding: utf-8 -*-
"""
IMAGERY VS NON-IMAGERY.
In this script, numerous classes used for preprocessing, processing, and classifying imageries vs resting states
will be coded.


@author: Ali Abdul Ameer Abbas
"""

import numpy as np
import matplotlib.pyplot as plt
import mne
import pymatreader
import scipy.stats
import pandas as pd
from mne.decoding import CSP

from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis # LDA
from sklearn.model_selection import ShuffleSplit, cross_val_score
from sklearn.svm import SVC                                  # SVM
from sklearn.neighbors import KNeighborsClassifier  # KNN
from sklearn.naive_bayes import GaussianNB          # Naive Bayesian
from sklearn.ensemble import RandomForestClassifier # Random Forest
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis #QDA                         # QDA
from sklearn.model_selection import learning_curve
from sklearn.metrics import plot_confusion_matrix, confusion_matrix, precision_score, recall_score, auc, roc_curve
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing     import StandardScaler, MinMaxScaler

import tensorflow as tf
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D, Dropout, Activation,
TimeDistributed, LSTM, concatenate
from tensorflow.keras.layers import Conv1D, Conv2D, MaxPooling2D, MaxPooling1D, Embedding, Reshape,
Flatten
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from keras import regularizers
from keras.layers.normalization import BatchNormalization

from scipy.signal import stft
from scipy.stats import norm
import pywt

#%%

class PreprocessDatasetPass:
    """
    This class will preprocess a file containing EEG recordings.
    It assigns a common reference, apply an ICA, filter the data. It also segments and annotates the
data.
    It is used for Imagery vs Non-imagery.

    """

    def __init__(self, path, montage = 'standard_1020', show_plots = False, bad_chan = []):

        """
        Intialize the class by reading and reconstructing the signals.
        INPUTS:
            montage --> The reference selected.
            path --> The path of the file.
            bad_channels --> Bad channels to eliminate.
            show_plots --> To show the plots of the EEG recording.
        """
        # Convert the MATLAB files into a python compatible file.
        self.struct = pymatreader.read_mat(path, ignore_fields=['previous'], variable_names = 'o')
        self.struct = self.struct['o']


        self.chann = self.struct['chnames'] # Save the channels names.
        self.chann = self.chann[:-1]        # Eliminate the X3 channel, which is only used for the data acquisition.

        #Set important properties for the dataset.
        self.info  = mne.create_info(ch_names = self.chann,
                               sfreq = float(self.struct['sampFreq']),ch_types='eeg',
                               montage= montage, verbose=None)

        #Create the data variable.
        self.data   = np.array(self.struct['data'])
        self.data   = self.data[:,:-1]
        self.data_V = self.data*1e-6        # mne reads units are in Volts, convert it to uV.

        # Create the raw instance for working with the data in the MNE library.
        self.raw   = mne.io.RawArray(self.data_V.transpose(),self.info, copy = "both")
        self.raw.info['bads'] = bad_chan # exclude bad channels

        # Show the raw data if show_plot is true.
        if show_plots:
            self.raw.plot(title = 'Raw signal')

```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```
    def channel_reference(self, ref = 'average' ):

        """
        Set a reference to the data.
        INPUTS:
            ref --> Reference.

        OUTPUTS:
            raw --> Referenced data.
          """

          self.raw = self.raw.set_eeg_reference(ref, projection= False)

          return(self.raw)

      def MNE_bandpass_filter(self, HP = 15, LP = 26, filt_type = 'firwin', show_plots = False,
skip_by_annotation='edge'):

          """
          Apply a BP FIR filter.

          INPUTS:
              LP,HP --> Low Pass filter, High Pass filter.
              filt_type --> filter type.
              show_plots --> To show the plots of the EEG recording.
          OUTPUTS:
              raw --> Filtered data.
          """
          self.raw.filter(HP, LP, fir_design = filt_type, skip_by_annotation = 'edge')

          if show_plots:
              self.raw.plot(title = 'Filtered Raw')
              self.raw.plot_psd()
```

```python


        return(self.raw)


    def apply_ICA(self, n_components = 10, ICA_exclude = [0], show_plots = False):

        """
        Apply the ICA for removing eye blinking, EMG signals, EKG signals, among other artifacts.

        INPUTS:
            n_components --> Number of components of the ICA.
            ICA_exclude --> Array used to eliminate components from teh signals.
            show_plots --> To show the plots of the EEG recording.
        OUTPUTS:
            raw_ica --> Data after ICA.
        """
        # Apply ICA for blink removal
        self.ica = mne.preprocessing.ICA(n_components = n_components, random_state = 0, method='fastica')

        # The MNE documentation recommends to HP filter the ICA at 1 Hz.
        filt_raw = self.raw.copy()                 # Create a copy of teh data
        filt_raw.filter(l_freq=1., h_freq=None)    # HP filter at 1 Hz
        self.ica.fit(filt_raw)                     # Find the ICA parameters.

        # Plot the ICA components to detect artifacts.
        if show_plots:
            self.ica.plot_components(outlines = 'skirt', colorbar=True, contours = 0)
            self.ica.plot_sources(self.raw)

        # Once the bad components are detected, we procede to remove them.
        self.ica.exclude = ICA_exclude

        self.raw_ica = self.ica.apply(self.raw.copy(), exclude = self.ica.exclude)

        # Plot the results.
        if show_plots:
            self.raw.plot()
            self.raw_ica.plot()

        return self.raw_ica

    def type_pos_pas(self, sfreq = 200.0):
        """
        Gets the marker position.
        Just left and right positions, NO PASS marker
        I takes into account the right, left, and pass class.

        INPUTS:
            sfreq --> Sampling frequency.
        OUTPUTS:
            mark --> markers.
            pos --> position of the marker.
            time --> time of the marker.
        """
        self.markers =  np.array(self.struct['marker']).transpose() # Include the markers

        # Initialize the output arrays.
        mark = []
        pos  = []
        time = []

        desc = ['left', 'right', 'pass'] # for assigning the movements--> left = 1, right = 2, pass = 3

```

```python
        #Evaluate the markers in order to find the position and time of each marker, as well as its type.
        for i in range(len(self.markers)-1):
            if self.markers[i]==0 and self.markers[i+1] != 0 and (self.markers[i+1] in [1,2,3]):

                mark.append(desc[self.markers[i+1]-1])
                pos.append((i+2))
                time.append((i+2)/sfreq)
            else:
                continue

        return mark, pos, time

    def multi_annotations(self, marks, time, window_time = 1.0):
        """
        This method is used in order to obtain all the possible pairs of annotation possible
        for a future distinctions between Left/Right classes relative to Pass classes.
        INPUTS:
            marks, time --> The EEG markers and the times thereof.
            window_time --> Length of the annotation window.
        OUTPUTS:
            annotations_left_right, annotations_left_pass, annotations_right_pass --> Annotations for each pair.
        """
        # Create annotations for Left and Right.
        mark_left_right = []
        time_left_right = []
        for mar in range(len(marks)):
           if marks[mar] != 'pass':
               mark_left_right.append(marks[mar])
               time_left_right.append(time[mar])

        # Create annotations for Left and Pass.
        mark_left_pass = []
        time_left_pass = []
        for mar in range(len(marks)):
           if marks[mar] != 'right':
               mark_left_pass.append(marks[mar])
               time_left_pass.append(time[mar])

        # Create annotations for Right and Pass.
        mark_right_pass = []
        time_right_pass = []
        for mar in range(len(marks)):
           if marks[mar] != 'left':
               mark_right_pass.append(marks[mar])
               time_right_pass.append(time[mar])

        # Save the annotations in a variable.
        self.annotations_left_right = mne.Annotations(time_left_right, window_time, mark_left_right)
        self.annotations_left_pass  = mne.Annotations(time_left_pass, window_time,  mark_left_pass)
        self.annotations_right_pass = mne.Annotations(time_right_pass, window_time, mark_right_pass)

        return self.annotations_left_right, self.annotations_left_pass, self.annotations_right_pass

    def add_annotations_and_epochs(self, tmin= 2.1, tmax= 3,  duration = 1):
        """
        This function uses the tye_pos_pas function to annotate the markers obtained from all the combinations
        of the multi_annotations function. In other words, it stacks together Left and Right imageries, and
        differentiate them from the Pass imagery. Furthermore, it creates epochs and balances the data.

        INPUTS:
            tmin, tmax --> Times for segmenting the data.
            duration -->Its the duration of the annotation.
```

```python
        OUTPUTS:
            All_epochs --> Get the epochs.
            labels --> Get the labels.
        """

        # Use the typo_pos_pas function to obbtain markers.
        [self.mark, self.pos, self.time] = self.type_pos_pas()
        # Use the multiannotation function.
        [self.annotations_l_r, self.annotations_l_p, self.annotations_r_p] = self.multi_annotations(self.mark, self.time, window_time = 1.0)

        # The following lines will join Left and Right movements.
        self.raw.set_annotations(self.annotations_l_r)                         # Set the Left/Right annotation.
        self.events = mne.events_from_annotations(self.raw)
        self.picks = mne.pick_channels(self.raw.info["ch_names"], ["C3", "Cz", "C4"]) # Only channels C3, C4 and Cz will be considered.

        # Concatenate all Left and right epochs
        self.epochs = mne.Epochs(self.raw, self.events[0], self.events[1], tmin, tmax,
                            picks=self.picks, baseline=None, preload=True)
        # Save the imagery data in a variable.
        self.Left_Right_epochs_data = self.epochs.get_data()


        # To balance the data, new Pass classes will be created by finding moments in which there are no imageries.
        self.raw.set_annotations(self.annotations_l_p) # Find the Pass classes, for instance, in Left vs Pass.
        self.events = mne.events_from_annotations(self.raw)
        self.picks = mne.pick_channels(self.raw.info["ch_names"], ["C3", "Cz", "C4"]) # Only channels C3, C4 and Cz will be considered.

```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```
        epochs1 = mne.Epochs(self.raw, self.events[0], self.events[1], tmin, tmax,
                            picks=self.picks, baseline=None, preload=True)

        epochs2 = mne.Epochs(self.raw, self.events[0], self.events[1], tmin-tmin, tmax-tmin, # This is done to create new Pass classes.
                            picks=self.picks, baseline=None, preload=True)

        e_p1 = epochs1['pass'].get_data()
        e_p2 = epochs2['pass'].get_data()
        # Save the pass data in a variable.
        self.Pass_epochs_data = np.concatenate((e_p1,e_p2))

        # concatenate all the data.
        self.All_epochs = np.concatenate((self.Left_Right_epochs_data,self.Pass_epochs_data))

        # Set labels (Left/Right --> 1; Pass --> 2)
        labels1 = np.array([1 for i in range(len(self.Left_Right_epochs_data))])
        labels2 = np.array([2 for i in range(len(self.Pass_epochs_data))])
        self.labels = np.concatenate((labels1,labels2))

        return self.All_epochs, self.labels


#%%
class ProcessDatasetPass():
    """
    This class will process a file containing EEG recordings.
    It is a subclass of the PreprocessDataset class.
    Mainly, this class will perform the statistical features and the STFT.


    """

    def __init__(self, class_pre, epochs, labels):

        """
        Initialize the class by getting the labels and epochs.
        INPUTS:
            class_pre --> Introduce any class of PreprocessDataset so that important information (info, chnames, ...) are defined.
            epochs --> Epochs obtained from the PreprocessingDataset class. This epochs should combine multiple subjects.
            labels --> Labels obtained from the PreprocessingDataset class. This epochs should combine multiple subjects.
        """
        # Define important parameters for the class:

        self.struct = class_pre.struct
        self.chann  = class_pre.chann
        self.info   = class_pre.info
        self.data   = class_pre.data
        self.data_V = class_pre.data_V
        self.raw    = class_pre.raw
        self.markers= class_pre.markers
        self.mark   = class_pre.mark
        self.time   = class_pre.time
        self.pos    = class_pre.pos

        self.epochs = epochs
        self.labels = labels
        #Only done to plot the CSP.
        self.epochs_subj = class_pre.epochs.get_data()
        self.labels_subj = class_pre.labels

    # The following methods are used for extracting the statistical features.
    def energy(self, signal):
        """
        Used for calculating the Energy of the signal.
        INPUTS:
            signal --> Signal used to obtain the energy.
        OUTPUTS:
            energy_value --> Energy of the signal.

        """

        energy_value = 0.0

        for x in signal:
            energy_value += (x)**2  # Apply the Energy formula.
        return energy_value

    def all_entropy(self, pdf):
        """
        Calculate the Entropies of the signals.
        INPUTS:
            pdf  ---> The Probability density function.

        RETURNS:
            entropy, shannon_entropy, log_energy_entropy
        """
        self.shannon_entropy = 0.0
        self.entropy = 0.0
        self.log_energy_entropy = 0.0
        self.pdf = pdf

        for freq in self.pdf:
            self.entropy += freq * np.log2(freq)
            self.shannon_entropy += (freq)**2 * (np.log2(freq)**2)
            self.log_energy_entropy += (np.log2(freq))**2

        self.entropy = -self.entropy
        self.shannon_entropy = -self.shannon_entropy
        self.log_energy_entropy = -self.log_energy_entropy

        return self.entropy,self.shannon_entropy, self.log_energy_entropy

    def feature_extracter(self):
        """
        Extract the Features of channels C3, Cz, and C4.
        OUTPUTS:
            skw_epoch_dic, krt_epoch_dic, energy_epoch_dic,
            entropy_epoch_dic, shannon_entropy_epoch_dic, log_energy_entropy_epoch_dic --> dictionaries of all the features.

        """
        ch = ["C3", "Cz", "C4"] # Define the channels.

        # Initialize the dictianaries for each feature.
        self.skw_epoch_dic = {ch[0]: [], ch[1]: [], ch[2]: []}
        self.krt_epoch_dic = {ch[0]: [], ch[1]: [], ch[2]: []}
        self.energy_epoch_dic = {ch[0]: [], ch[1]: [], ch[2]: []}
        self.entropy_epoch_dic = {ch[0]: [], ch[1]: [], ch[2]: []}
        self.shannon_entropy_epoch_dic = {ch[0]: [], ch[1]: [], ch[2]: []}
        self.log_energy_entropy_epoch_dic = {ch[0]: [], ch[1]: [], ch[2]: []}

        # Calculate the features for each epoch.
        for epoch in self.epochs:

            data = epoch

            self.pdf_all = [] # Initialize a list for saving the PDF of each epoch.
            for i in range((data.shape[0])):
```

```
                # A channel is selected from an epoch ['C3' --> 0, 'C4' --> 1, 'Cz' --> 2]
                gau = scipy.stats.gaussian_kde(data[i])

                # Values in which the gausian will be evaluated
                dist_space = np.linspace( min(data[i]), max(data[i]), 100 )
                gau_total = np.sum(gau(dist_space))# Normalize, this is the integral under the curve.
                pdf = gau(dist_space)/gau_total
                self.pdf_all.append(pdf)


            self.skw_epoch = [] # Initialize a list for saving the Skewness of each epoch.
            for i in range((data.shape[0])):
                self.skw_epoch.append(scipy.stats.skew(data[i], axis=0, bias=True)) # Calculate skw.


            self.krt_epoch = [] # Initialize a list for saving the Kurtosis of each epoch.
            for i in range((data.shape[0])):
                self.krt_epoch.append(scipy.stats.kurtosis(data[i], axis=0, fisher=True, bias=True,
nan_policy='propagate')) # Calculate krt.


            self.energy_epoch = [] # Initialize a list for saving the Energy of each epoch.
            for i in range((data.shape[0])):
                energy_value = self.energy(data[i]) # Call the Energy method.
                self.energy_epoch.append(energy_value)



            self.Pi_all = [] # here we save the values for each channel

            # Wavelets are needed in order to perform the Shannon Energy, for instance.
            for i in range((data.shape[0])):

```

```
            coef,_= pywt.cwt(data[i],  np.arange(7,20,0.5), 'morl') # Apply the Wavelet transform

            # Now, the following formulas are used:
                #'https://dsp.stackexchange.com/questions/13055/how-to-calculate-cwt-shannon-entropy'
            [M,N] = coef.shape; # M --> scale number, N --> time segments
            Ej = []
            for j in range(M):
                Ej.append(sum(abs(coef[j,:])));

            Etot=sum(Ej);

            pi = []
            for i in Ej:
                pi.append(i/Etot)
            self.Pi_all.append(pi)
        self.Pi_all = np.array(self.Pi_all)

        # Save the entropies in lists.
        self.entropy_epoch = []
        self.shannon_entropy_epoch = []
        self.log_energy_entropy_epoch = []
        for i in range((data.shape[0])):
            # S = -sum(pk * log(pk)) --> pk is the Probability Density Function
            entr, shan, log_en = self.all_entropy(self.pdf_all[i]) # Call the all_entropy method.

            self.entropy_epoch.append(entr)
            self.shannon_entropy_epoch.append(shan)
            self.log_energy_entropy_epoch.append(log_en)

        # Save all the calues in dictionaries.
        self.skw_epoch_dic[ch[0]].append(self.skw_epoch[0]), self.skw_epoch_dic[ch[1]].append(self.skw_epoch[1]), self.skw_epoch_dic[ch[2]].append(self.skw_epoch[2])
```

```
            self.krt_epoch_dic[ch[0]].append(self.krt_epoch[0]), self.krt_epoch_dic[ch[1]].append(self.krt_epoch[1]), self.krt_epoch_dic[ch[2]].append(self.krt_epoch[2])
            self.energy_epoch_dic[ch[0]].append(self.energy_epoch[0]), self.energy_epoch_dic[ch[1]].append(self.energy_epoch[1]), self.energy_epoch_dic[ch[2]].append(self.energy_epoch[2])
            self.shannon_entropy_epoch_dic[ch[0]].append(self.shannon_entropy_epoch[0]), self.shannon_entropy_epoch_dic[ch[1]].append(self.shannon_entropy_epoch[1]), self.shannon_entropy_epoch_dic[ch[2]].append(self.shannon_entropy_epoch[2])
            self.log_energy_entropy_epoch_dic[ch[0]].append(self.log_energy_entropy_epoch[0]), self.log_energy_entropy_epoch_dic[ch[1]].append(self.log_energy_entropy_epoch[1]), self.log_energy_entropy_epoch_dic[ch[2]].append(self.log_energy_entropy_epoch[2])
            self.entropy_epoch_dic[ch[0]].append(self.entropy_epoch[0]), self.entropy_epoch_dic[ch[1]].append(self.entropy_epoch[1]), self.entropy_epoch_dic[ch[2]].append(self.entropy_epoch[2])

        return self.skw_epoch_dic, self.krt_epoch_dic, self.energy_epoch_dic, self.entropy_epoch_dic, self.shannon_entropy_epoch_dic, self.log_energy_entropy_epoch_dic

    def feature_ex(self, skw, krt, energy, entropy, shannon, log):
        """
        Method for organizing the features of all the epochs.
        INPUTS:
            skw, krt, energy, entropy, shannon, log --> dictionaries of each feature.
        OUTPUTS:
            big_feat --> Organized features.
        """

        big_feat = []  # Initialize the big_feat list.
        ch = ['C3','C4', 'Cz'] # Define the channels.
        # Organize the Features.
        for j in range(len(skw['C3'])):
            small_feat = []
            for i in range(len(ch)):

                new_small_feat = [skw[ch[i]][j], krt[ch[i]][j], energy[ch[i]][j], entropy[ch[i]][j], shannon[ch[i]][j], log[ch[i]][j]]
                small_feat.extend(new_small_feat)
            big_feat.append(small_feat)
        return big_feat

    # Define the STFT features.
    def ShortTimeFourierTransform(self, fs=200.0, window='hann',nperseg=181, noverlap=180):
        """
        Obtain the  STFT features.
        INPUTS:
            fs --> Sampling frequency.
            window --> Window type.
            nperseg --> Samples per segment.
            noverlap --> Overlap allowance.
        OUTPUT:
            coef --> STFT Coeficients
        """
        # Compute the STFT.
        frec, tim, self.coef = stft(self.epochs, fs = fs, window = window, nperseg = nperseg, noverlap = noverlap)
        self.coef= np.abs(self.coef) # Get the absolute value.

        return self.coef



#%%
class Model_IR:
    """
    Model Imagery/Resting. Class to organize the AI models with the best results used to distinguish between an Imagery
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```python
        and an Imagery.
        """

    def ML_classification(self, epochs_train, labels_train, epochs_test, labels_test, classif = 'RF'):
        """
        Uses ML classifiers in conjunction with the statistical_features to train and test the motor imageries.

        INPUTS:
            epochs_train, labels_train --> Introduce the statistical features and labels for training the models.
            epochs_test, labels_test --> Introduce the statistical features  and labels for testing the models.
            classif --> The classifier used. (LDA, RF, SVM, KNN, NB, QDA)
        OUTPUTS:
            clf --> return the model.
        """

        # Initialize a score array.
        self.scores = []

        self.labels = labels_train
        self.epochs_data_train = epochs_train

        self.epochs_test = epochs_test
        self.labels_test = labels_test

        # set the validation and test data
        self.cv = ShuffleSplit(10, test_size=0.2, random_state=None)
        self.cv_split = self.cv.split(self.epochs_data_train)

        # Assemble a classifier

        self.lda = LinearDiscriminantAnalysis()
        self.svm = SVC(kernel ='rbf', gamma = 0.00001, C = 10000)
        self.knn = KNeighborsClassifier(n_neighbors = 25, weights = 'uniform')
        self.NB  = GaussianNB()
        self.qda = QuadraticDiscriminantAnalysis()
        self.RF  = RandomForestClassifier(n_estimators = 100,
                             min_samples_split = 16,
                             min_samples_leaf = 4,
                             max_features = 'sqrt',
                             max_depth = 8,
                             bootstrap = True)


        # Define a normalizer.

        self.scale = StandardScaler()


        # Use scikit-learn Pipeline with cross_val_score function
        if classif == 'LDA':
            self.clf = Pipeline([('scale', self.scale), ('LDA', self.lda)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'QDA':
            self.clf = Pipeline([('scale', self.scale), ('QDA', self.qda)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'SVM':
            self.clf = Pipeline([('scale', self.scale), ('SVM', self.svm)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'KNN':
            self.clf = Pipeline([('scale', self.scale), ('KNN', self.knn)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        elif classif == 'NB':
            self.clf = Pipeline([('scale', self.scale), ('NB', self.NB)]) # Pipeline firstly extracts features from CSP and then does the LDA classification
        else:
            self.clf = Pipeline([('scale', self.scale), ('RF', self.RF)])


        self.scores = cross_val_score(self.clf, self.epochs_data_train, self.labels, cv=self.cv, n_jobs=1)

        self.clf.fit(self.epochs_data_train[:], self.labels[:])  # We fit the model to make it usable


        self.testing_score = self.clf.score(self.epochs_test, self.labels_test) #Testing Data (Never seen before)
        print('Accuracy: ', self.testing_score)
        print('Precision: ', precision_score(self.labels_test[:], self.clf.predict(self.epochs_test[:],))) # It gives more importance that the pass class is detected correctly. TruePositive/(TruePositives+FalsePositives)
        print('Recall: ', recall_score(self.labels_test[:], self.clf.predict(self.epochs_test,)))

        # Printing the results
        self.class_balance = np.mean(self.labels == self.labels[0])
        self.class_balance = max(self.class_balance, 1. - self.class_balance)
        print("Classification accuracy: %f +- %f / Chance level: %f" % (np.mean(self.scores), np.std(self.scores),
                                                              self.class_balance))
        plt.rc('xtick', labelsize=20)
        plt.rc('ytick', labelsize=20)
        plt.rcParams.update({'font.size': 16})
        plt.rc('axes', titlesize=30, labelsize=25)

        plot_confusion_matrix(self.clf, self.epochs_test, self.labels_test)
        plt.show()

        return self.clf


    def CNN_classification(self, Coef_train, labels, Coef_test, label_test, input_shape = (3,91,181),
                           filter1 = 120, filter2 = 240, dense1 = 164, dense2 = 128, kern_size = (3,3),
                           padding = 'same', activation1 = 'relu', activation2 = 'sigmoid', drop = 0.2,
                           pool_size1 = 2, pool_size2 = 1, lr = 0.0001, dec = 300, mom = 0.8,
                           bch_size = 40, epo = 260):
        """

        Uses CNN classifiers in conjunction with STFT to train and test imageries vs the resting state.
        This CNN has 2 CNN layers and 3 dense layers.

        INPUTS:
            Coef_train, labels --> Introduce the STFT Coeficients and labels for training the models.
            Coef_test, label_test --> Introduce the STFT Coeficients and labels for testing the models.
            input_shape --> Shape inserted on the first layer of teh CNN.
            filter1, filter2 --> Filter size of layers 1 and 2 of the CNN.
            dense1, dense2 --> Size of Dense layers 1 and 2 of the CNN.
            ker_size --> Kernel size.
            padding --> Padding type.
            activation1 --> Activation function of all the layers except the last one.
            activation2 --> Activation function of the last layer.
            drop --> Dropout ratio.
            pool_size1, pool_size2 --> Pools size of layers 1 and to of the CNN.
            lr --> Learning rate.
            dec --> Denominator for the decay rate DECAY = lr/dec.
            mom --> Momentum.
            bch_size --> Batch Size.
            epo --> Epochs of the CNN (Iterations).
        OUTPUTS:
            model --> Return the model.
            scale -> Returns the scaler.

        """
        # Activate the GPU.
        tf.config.list_physical_devices('GPU')

        # Initialize the model.
        self.model = Sequential()

        # First layer of the CNN.
        self.model.add(Conv2D(filters = filter1, kernel_size = kern_size, padding = padding, activation
= activation1,input_shape = input_shape)) #INPUTS_SIZE =(NUMBER OF SAMPLES IN EACH EPOCH, NUMBER OF
CHANNELS
        self.model.add(MaxPooling2D(pool_size = pool_size1))
        self.model.add(Dropout(drop))

        # Second layer of the CNN.
        self. model.add(Conv2D(filters = filter2, kernel_size = kern_size, padding = padding,
activation = activation1)) #INPUTS_SIZE =(NUMBER OF SAMPLES IN EACH EPOCH, NUMBER OF CHANNELS
        self.model.add(MaxPooling2D(pool_size = pool_size2))
        self.model.add(Dropout(drop))

        # Flatten layer.
        self.model.add(Flatten())

        # First Dense layer.
        self.model.add(Dense(dense1, activation = activation1))
        self.model.add(Dropout(drop))

        # Second Dense layer.
        self.model.add(Dense(dense2, activation = activation1))
        self.model.add(Dropout(drop))
```

```

        # Last Dense layer.
        self.model.add(Dense(1, activation = activation2))

        # Define some parameters (learning rate, decay rate, and momentum).
        learning_rate = lr
        decay_rate = learning_rate/dec
        momentum = mom

        # Create the optimizer.
        self.opt = SGD(learning_rate=learning_rate, momentum=momentum, decay=decay_rate,
nesterov=False)
        #self.pt = RMSprop(learning_rate=learning_rate, momentum=0.2, decay=decay_rate)
        #self.opt = Adam(learning_rate=learning_rate, decay=decay_rate)

        #Compile the model.
        self.model.compile(loss='binary_crossentropy', optimizer= self.opt, metrics=['accuracy'])

        self.model.summary() # Show a summary.

        # Separate the data into test and train.
        X_train = Coef_train
        y_train = labels-1
        X_test = Coef_test
        y_test = label_test-1


        # It's important to normalize the data and also reshape it in order to correctly fit the model.
        # Reshape it according to --> (NUMBER OP EPOCHS, NUMBER OF SAMPLES IN EACH EPOCH, NUMBER OF
CHANNELS)
        X_train_re = X_train.reshape(X_train.shape[0],
X_train.shape[1]*X_train.shape[2]*X_train.shape[3])
        self.scaler = MinMaxScaler(feature_range=(0,1))
```

```
          self.X_train_norm = self.scaler.fit_transform(X_train_re)
          self.X_train_norm = 
self.X_train_norm.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],X_train.shape[3])

          X_test_re = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2]*X_test.shape[3])
          self.X_test_norm = self.scaler.transform(X_test_re)
          self.X_test_norm = self.X_test_norm.reshape(X_test.shape[0], 
X_test.shape[1],X_test.shape[2],X_test.shape[3])

          # Fit the model.
          history = self.model.fit(self.X_train_norm, y_train, batch_size = bch_size, epochs = epo, 
validation_data=(self.X_test_norm, y_test))

          # Plot the historical behavior of the model along the iterations.
          history_df = pd.DataFrame(self.model.history.history).rename(columns={"loss":"train_loss", 
"accuracy":"train_accuracy"})
          history_df.plot(figsize=(8,8))
          plt.grid(True)
          plt.xlabel('Epochs')
          plt.ylabel('Accuracy')
          plt.show()

          # Test teh model.
          self.predictions = (self.model.predict(self.X_test_norm) > 0.5).astype("int32")
          self.testing_score = self.model.evaluate(self.X_test_norm, y_test) #Testing Data (Never seen 
before)
          print('Accuracy: ', self.testing_score)
          print(recall_score(y_test, self.predictions)) # It gives more importance that the pass class is 
detected correctly. TruePositive/(TruePositives+FalsePositives)
          print(precision_score(y_test, self.predictions))

          return self.model, self.scaler

```

A4. Annex 4: Imagery vs Resting state (Executable) [Main_Imagery_vs_Resting.py]

```python
# -*- coding: utf-8 -*-
"""
Main script for training and testing the AI models for the Imagery vs Resting state.

@author: Ali Abdul Ameer Abbas
"""

# Import basic libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import stft
import pickle

# Import the modules that were created in Imagery_vs_Resting.py
from Imagery_vs_Resting import PreprocessDatasetPass, ProcessDatasetPass, Model_IR

#%% Plot configuration.
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.rc('axes', titlesize=30, labelsize=25)

#%%
# Firstly, we must define the paths of dataset' files.

path1_C = '../Data/CLA/CLASubjectC1512233StLRHand.mat'
path2_C = '../Data/CLA/CLASubjectC1512163StLRHand.mat'
path3_C = '../Data/CLA/CLASubjectC1511263StLRHand.mat'


path1_B = '../Data/CLA/CLASubjectB1510193StLRHand.mat'
path2_B = '../Data/CLA/CLASubjectB1510203StLRHand.mat'
```

```python
path3_B = '../Data/CLA/CLASubjectB1512153StLRHand.mat'


path1_E = '../Data/CLA/CLASubjectE1512253StLRHand.mat'
path2_E = '../Data/CLA/CLASubjectE1601193StLRHand.mat'
path3_E = '../Data/CLA/CLASubjectE1601223StLRHand.mat'


path1_F = '../Data/CLA/CLASubjectF1509163StLRHand.mat'
path2_F = '../Data/CLA/CLASubjectF1509173StLRHand.mat'
path3_F = '../Data/CLA/CLASubjectF1509283StLRHand.mat'

fs= 200 # Define the sampling frequency.

#%%

#Initialize the classes for each subject.

SubjC_1 = PreprocessDatasetPass(path = path1_C, show_plots=True)
SubjC_2 = PreprocessDatasetPass(path = path2_C)
SubjC_3 = PreprocessDatasetPass(path = path3_C)

SubjE_1 = PreprocessDatasetPass(path = path1_E)
SubjE_2 = PreprocessDatasetPass(path = path2_E)
SubjE_3 = PreprocessDatasetPass(path = path3_E)

SubjF_1 = PreprocessDatasetPass(path = path1_F)
SubjF_2 = PreprocessDatasetPass(path = path2_F)
SubjF_3 = PreprocessDatasetPass(path = path3_F)

# Set the reference.
SubjC_1.channel_reference()
SubjC_2.channel_reference()
```

```
SubjC_3.channel_reference()

SubjE_1.channel_reference()
SubjE_2.channel_reference()
SubjE_3.channel_reference()

SubjF_1.channel_reference()
SubjF_2.channel_reference()
SubjF_3.channel_reference()

# Filter the data.

SubjC_1.MNE_bandpass_filter(HP = 15, LP = 26, show_plots=True)
SubjC_2.MNE_bandpass_filter(HP = 15, LP = 26)
SubjC_3.MNE_bandpass_filter(HP = 15, LP = 26)

SubjE_1.MNE_bandpass_filter(HP = 15, LP = 26)
SubjE_2.MNE_bandpass_filter(HP = 15, LP = 26)
SubjE_3.MNE_bandpass_filter(HP = 15, LP = 26)

SubjF_1.MNE_bandpass_filter(HP = 15, LP = 26)
SubjF_2.MNE_bandpass_filter(HP = 15, LP = 26)
SubjF_3.MNE_bandpass_filter(HP = 15, LP = 26)

# Add annotations and create epochs.

# Create Epochs.

[epoch1_C, label1_C] = SubjC_1.add_annotations_and_epochs(tmin = 2.1, tmax = 3)
[epoch2_C, label2_C] = SubjC_2.add_annotations_and_epochs(tmin = 2.1, tmax = 3)
[epoch3_C, label3_C] = SubjC_3.add_annotations_and_epochs(tmin = 2.1, tmax = 3)

[epoch1_E, label1_E] = SubjE_1.add_annotations_and_epochs(tmin = 2.2, tmax = 3.1)
```

```
[epoch2_E, label2_E] = SubjE_2.add_annotations_and_epochs(tmin = 2, tmax = 2.9)
[epoch3_E, label3_E] = SubjE_3.add_annotations_and_epochs(tmin = 2.2, tmax = 3.1)

[epoch1_F, label1_F] = SubjF_1.add_annotations_and_epochs(tmin = 1, tmax = 1.9)
[epoch2_F, label2_F] = SubjF_2.add_annotations_and_epochs(tmin = 1.3, tmax = 2.2)
[epoch3_F, label3_F] = SubjF_3.add_annotations_and_epochs(tmin = 1.4, tmax = 2.3)


# Now, the data of all the subjects will be concatenated, and separated in training and testing.

# All_epochs = np.concatenate((epoch1_C, epoch2_C)) # Epochs for training.
# labels = np.concatenate((label1_C, label2_C))     # Labels for training.
# All_epochs_test = epoch3_C                        # Epochs for testing.
# label_test = label3_C                             # Labels for testing.


All_epochs = np.concatenate((epoch1_C, epoch2_C, epoch1_E,epoch2_E, epoch1_F, epoch2_F))  # Epochs for training.
labels = np.concatenate((label1_C, label2_C, label1_E, label2_E, label1_F, label2_F))     # Labels for training.
All_epochs_test = np.concatenate((epoch3_C, epoch3_E, epoch3_F))                         # Epochs for testing.
label_test = np.concatenate((label3_C, label3_E, label3_F))                              # Labels for testing.

#%% Obtain features.

Train_feats = ProcessDatasetPass(SubjC_1, All_epochs, labels)  # Create a class to process the data.
Test_feats = ProcessDatasetPass(SubjC_1, All_epochs_test, label_test)  # Create a class to process the data.


# Obtain the Statistical features.
C_skw, C_krt, C_energy, C_entropy, C_shannon, C_log = Train_feats.feature_extracter() # All epochs train.
```

```
D_skw, D_krt, D_energy, D_entropy, D_shannon, D_log= Test_feats.feature_extracter()   # All epochs 
test.

FEAT = Train_feats.feature_ex(C_skw, C_krt, C_energy, C_entropy, C_shannon, C_log)
FEAT = np.array(FEAT) # Training features.

FEAT_test = Test_feats.feature_ex(D_skw, D_krt, D_energy, D_entropy, D_shannon, D_log)
FEAT_test = np.array(FEAT_test) # Testing features.



# Obtain the STFT coefficients.
# Train coefs.
Coef_train = Train_feats.ShortTimeFourierTransform(fs=200.0, window='hann',nperseg=181, noverlap=180)

# Test coefs.
Coef_test = Test_feats.ShortTimeFourierTransform(fs=200.0, window='hann',nperseg=181, noverlap=180)

# Plot a random STFT spectrogram
for i in range(3):
    frec, tim, Zx =stft(All_epochs[-1,i], fs=200.0, window='hann',nperseg=181, noverlap=180)

    plt.figure()
    plt.pcolormesh(tim, frec, np.abs(Zx), shading='gouraud', cmap='jet',vmin=-0,vmax=0.8e-6)
    plt.title('STFT Magnitude {}'.format(["C3", "Cz", "C4"][i]), fontsize=30)
    plt.ylim([5,30])
    plt.ylabel('Frequency [Hz]', fontsize=20)
    plt.xlabel('Time [sec]', fontsize=20)
    plt.xticks(fontsize=16)
    plt.yticks(fontsize=16)
    plt.colorbar()
    plt.show()

```

```python
#%%
Subj_train = Model_IR()
## Train the ML model with Statistical Features.
clf = Subj_train.ML_classification(FEAT, labels, FEAT_test, label_test, classif = 'RF')

# Save the ML model:

with open('RightLeft_vs_Pass_Classification_all.pkl','wb') as f:
    pickle.dump(clf,f)

#%% Train the CNN model with STFT.
model, scaler = Subj_train.CNN_classification(Coef_train, labels, Coef_test, label_test)

# Save the CNN model:

model.save('Pass_vs_Left_Right_with_DL_all.h5')

# #MinMaxScaler
with open('Pass_vs_Left_Right_with_DL_Scaler_all.pkl','wb') as f:
    pickle.dump(scaler,f)

```

A5. Annex 5: Sliding Window and serial communication (Executable) [Main_Sliding_Window.py]

```python
# -*- coding: utf-8 -*-
"""
In this script, the trained AI models will be tested on one entire session of Subject C, by sliding
the session inside a Window. Both AI models will work cooperatively to classify each signal the same
way as it would do on a real-time implementation.


@author: Ali Abdul Ameer Abbas
"""

# Import important libraries.
import pickle
import pywt
import numpy as np
import matplotlib.pyplot as plt
import mne
import pymatreader
import scipy.stats
import tensorflow as tf
from scipy.signal import firwin, lfilter
from scipy.signal import stft
import serial
import time

# Import the modules that were created in Left_vs_Right.py and Imagery_vs_Resting.py
from Left_vs_Right import PreprocessDataset, ProcessDataset
from Imagery_vs_Resting import PreprocessDatasetPass, ProcessDatasetPass

#%% Plot configuration.
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.rc('axes', titlesize=30, labelsize=25)

#%% load and preprocess data

path = '../Data/CLA/CLASubjectC1511263StLRHand.mat' # Only an unseen session of Subject C will be used.

#Initialize the class

Subject_C = PreprocessDataset(path)

# Set the reference.
Subject_C.channel_reference()

#%% Proceed on loading the models created in the other scripts:

# Load the Left vs Right imagery chosen Model.

with open('LeftRight_Classification_More_Data_all.pkl', 'rb') as f:
    clf4 = pickle.load(f)

# Load the Imagery vs Resting state chosen Model.
DL_model_Pass_detection = tf.keras.models.load_model('Pass_vs_Left_Right_with_DL_all.h5')
# Load CNN's Scaler.
with open('Pass_vs_Left_Right_with_DL_Scaler_all.pkl', 'rb') as f:
    Pass_scaler = pickle.load(f)


#%% Get the data from the signals.
# Get the data for all channels.
AciscoAll = Subject_C.raw.get_data()

# Get the data for the C3, Cz, and C4 channels. Useful for the statistical features and STFT analysis.
picks = mne.pick_channels(Subject_C.raw.info["ch_names"], ["C3", "Cz", "C4"])
```

```python
Acisco = Subject_C.raw.get_data(picks=picks)


#%% Define and design the FIR filter used in the real-time simulation.
#Article in how to design a FIR filter:
#https://www.allaboutcircuits.com/technical-articles/design-examples-of-fir-filters-using-window-method/

def bandpass_firwin(ntaps, lowcut, highcut, fs = 200, window = 'hann'):
    '''
    This function enables the user to define the parameters for BP FIR filter.

    INPUTS:
        ntaps --> Length of the filter.
        lowcut --> Low frequency.
        highcut --> High frequency.
        fs --> Sampling frequency,
        window --> Window type. (Hanning, Hamming, etc.)
    OUTPUTS:
        taps --> Filter coefficients.
    '''
    nyq = 0.5 * fs
    taps = firwin(ntaps, [lowcut, highcut], nyq=nyq, pass_zero=False,
                  window=window, scale=False)
    return taps

# Design the filter.
filt_length = 150 # It says the order of the filter.
lowcut = 15       # Low frequency.
highcut = 26      # High frequency.

delay = np.ceil((filt_length-1)/2) # the delay of the filter (in samples) is calculated as following.

FIR_Coeficients = bandpass_firwin(filt_length,lowcut,highcut, fs = 200, window='hamming')
```

```

# Compute the frequency response of the digital filter.
    w, mag = scipy.signal.freqz(FIR_Coeficients, 1, fs=200) # The numerator are the coefficients and the denominator is 1 (no poles).

    # Plot the filter's impulse response.

    plt.figure()
    plt.title('Impulse Response of the FIR filter', fontsize="30")
    plt.xlabel('Samples', fontsize=20)
    plt.ylabel('Amplitude', fontsize=20)
    plt.plot(FIR_Coeficients)
    plt.show()

    # Plot the Bode.

    fig, axs = plt.subplots(2)
    fig.suptitle('FIR filter BODE', fontsize=30)
    axs[0].plot(w, 20*np.log10(np.abs(mag)))   # Bode magnitude plot
    axs[0].set_xlabel('Frequency (Hz)', fontsize=20)
    axs[0].set_ylabel('Magnitude (dB)', fontsize=20)
    #axs[0].grid(True)
    axs[1].plot(w, np.unwrap(np.angle(mag,deg=False)))   # Bode phase plot
    axs[1].set_xlabel('Frequency (Hz)', fontsize=20)
    axs[1].set_ylabel('Phase (rad)', fontsize=20)
    #axs[1].grid(True)
    plt.show()

    # Plot an example of the filter's effect.


    fig, axs = plt.subplots(2)
    fig.suptitle('Before and after filtering', fontsize=30)
```

```python

ACA = [Acisco[:, int(2500*200):int(2500*200+180+delay)]]
axs[0].plot(ACA[0][1])
axs[0].set_xlabel('Samples', fontsize=20)
axs[0].set_ylabel('Amplitude', fontsize=20)
axs[1].set_xlabel('Samples', fontsize=20)
axs[1].set_ylabel('Amplitude', fontsize=20)
ACA = lfilter(FIR_Coeficients, 1.0, ACA)
axs[1].plot(ACA[0][1])

#%% Create a sliding window for inserting the datapoints and classifying each window
# by using the saved RF model with CSP (Left vs Right --> clf4)
# and the CNN with STFT (Imagery vs Resting --> DL_model_Pass_detection, Pass_scaler).

# Define the serial communication with Arduino.

# Ensure that the Arduino port is initially closed. Otherwise, errors may occur.
try:
    arduino.close()
except:
    pass

# Open the Serial Communication with Arduino.
try:
    arduino = serial.Serial('COM3', timeout=None, baudrate=9600)
    time.sleep(2)
except:
    print('Arduino not connected')

# Initialize parameters.

times = [0]
prediction_markers = [] # Save the predictions.
```

```python
prediction_times = []   # Save the times of the prediction,

# Define the times of the session to analyze.
start_time = 170 # Seconds.
stop_time = 500  # Seconds.
step = 0.1       # Seconds. Step of the window at each iteration (Always 0.1).

# Start the analysis and save the data in the vectors.

for i in np.arange(start_time, stop_time, step):
    vala = i
    # Encapsulate the data in the time window.
    # Remember that the sampling frequency is 200. Since the window has size 0.9, there are 180 samples (0.9*200).
    ACA_All = [AciscoAll[:, int(vala*200):int(vala*200+181+delay)]]   # Keep the data that is insede the time window.
    ACA_All = lfilter(FIR_Coeficients, 1.0, ACA_All)                  # Filter the data
    ACA_All = np.array(ACA_All)[:,:,int(delay):]                      # Remove the delay for inserting the data to the models.
    # Do the same but just for channels C3, Cz, and C4.
    ACA = [Acisco[:, int(vala*200):int(vala*200+181+delay)]]
    ACA = lfilter(FIR_Coeficients, 1.0, ACA)
    ACA = np.array(ACA)[:,:,int(delay):]

    # Extract the STFT features.
    frec, tim, Zx =stft(ACA, fs=200.0, window='hann',nperseg=181, noverlap=180)
    FEAT = np.abs(Zx)

    # Predict if the window is in an Imagery or Resting state.
    pred = [int((DL_model_Pass_detection.predict(Pass_scaler.transform(np.array(FEAT).reshape(1,3*91*181)).reshape(1,3,91,181))> 0.2).astype("int32"))+1]

```

```python
        # If the prediction is an imagery, apply the Left vs Right prediction (pred2)
        if pred == [1]:
            if i > (times[0]):
                # Print the prediction and the probability of the prediction
                pred2 = list(clf4.predict(np.array(ACA_All)))
                pred2_prob = clf4.predict_proba(np.array(ACA_All))
                print(clf4.predict(np.array(ACA_All)), clf4.predict_proba(np.array(ACA_All)))

                # Send the pred2 variable to Arduino in order to move the Robotic Manipulator (Left or Right).
                try:
                    arduino.write((str(pred2[0])+'\n').encode())
                    #print('Data sent.')
                except:
                    pass


                # The following lines are used for later plotting the results in a graph.
                if times[0]+0.5 < i:   #If there is a difference of 0.5 sec between predictions, it is considered another imagery.
                    print('-----------------')
                    # Send a reset indication to Arduino in order to reset the Left and Right Counters (see Arduino code).
                    try:
                        arduino.write(('Reset'+'\n').encode())

                    except:
                        pass

                if pred2 == [1]: # If Left is detected.
                    print(round(i, 1), 'Left prediction')
                    prediction_markers.append('Left prediction') # Add the prediction to the list.
                    prediction_times.append(i)                   # Add the prediction time to the list.
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```

            if pred2 == [2]: # If Right is detected.
                print(round(i, 1), 'Right prediction')
                prediction_markers.append('Right prediction') # Add the prediction to the list.
                prediction_times.append(i)                    # Add the prediction time to the list.

            times[0] = i # Update the time.

# Close the Serial communication.
try:
    arduino.close()
    print('Data sent.')
except:
    pass

# Convert the lists to numpy arrays.
prediction_markers = np.array(prediction_markers)
prediction_times = np.array(prediction_times)

# Now, create a loop in which each imagery be separated their time and length.
# Initialize the needed variables.
new_prediction_markers = []
new_prediction_times = []
segment = []                # It will separate different imageries events.
times = prediction_times[0]

# Create a loop for organizing the lists.
for i, timer in enumerate(prediction_times):
    # If there is a difference of 0.5 sec between predictions, it is considered another imagery.
    if timer > times + 0.5:
        # To be more robust, consider a correct imagery when the model has predicted a class 4 times consecutively.
        if len(segment) >= 4:
            segment_label = prediction_markers[segment]
            segment_time = prediction_times[segment]

            left_count = np.char.count(segment_label, 'Left prediction').sum()
            right_count = np.char.count(
                segment_label, 'Right prediction').sum()
            # If Left prediction are more abundant thanright ones, consider Left.
            if left_count > right_count and right_count <= 4:

                new_prediction_markers.append('Left prediction')
                new_prediction_times.append(list(segment_time)[0])

            # If Right prediction are more abundant than left ones, consider Right.
            elif left_count < right_count and left_count <= 4:

                new_prediction_markers.append('Right prediction')
                new_prediction_times.append(list(segment_time)[0])

            else:
                pass

            segment = []
        else:
            segment = []
    else:
        segment.append(i)
    times = timer

# Now plot the results.

# Put the ground truth markers as well as the newly generated prediction markers.

[mark, pos, timer] = Subject_C.type_pos() # Obtain the ground truth markers.
```

```

mark.extend(new_prediction_markers) # Add the new markers.
timer.extend(new_prediction_times)   # Add the new markers' times.

# Set annotations.

annotations = mne.Annotations(timer, 1.0, mark)
Subject_C.raw.set_annotations(annotations)
Subject_C.raw.plot(duration=1000.0,start=500.0)
```

A6. Annex 6: Arduino receiving inputs for moving the *MeArm* (Executable) [Arduino_serial_com.ino]

```
/*
This code enables a Serial communication with Python in order to control
a MeArm robotic manipulator. When 4 consecutive predictions are the same,
the robotic manipulator moves.

@author: Ali Abdul Ameer Abbas
*/

#include <Servo.h> // Import the library for the Servo motors.
Servo baseMotor;   // Define the servo.
// Initialize counters.
int count_left = 0;
int count_right = 0;
// Define the input Byte from Python.
String InputBytes;

// Set up the code.
void setup() {
  Serial.begin(9600);
  baseMotor.attach(9); // Define the pin of the servomotor.
  baseMotor.write(60); // Move the MeArm to the neutral position (60 degrees).
}

// Initialize the Loop.
void loop() {
  // Check if the Serial port is available.
  if (Serial.available()>0){
    // Read the input byte.
    InputBytes = Serial.readStringUntil('\n');
    Serial.flush();
    // If the Byte is "1" and there are 4 consecutive "1".
```

```
      // Move the MeArm to the Left.
      if (InputBytes == "1"){
        count_left += 1; // Sum the Left counter for counting the consecutives.
        count_right = 0; // Reset the Right counter.
        if (count_left == 4){
          baseMotor.write(120); // Move the MeArm base to 120 degrees.
          delay(100);
        }
      }
      // If the Byte is "2" and there are 4 consecutive "2".
      // Move the MeArm to the Left.
      if (InputBytes == "2"){
        count_left = 0;   // Sum the Right counter for counting the consecutives.
        count_right += 1; // Reset the Left counter.
        if (count_right == 4){
          baseMotor.write(0);  // Move the MeArm base to 0 degrees.
          delay(100);
        }
      // If the Byte is "Reset", reset the counters.
      if (InputBytes == "Reset"){
        count_left = 0;
        count_right= 0;
        }
      }
    }
}
```

A7. Annex 7: ERDS maps generator (Executable) [ERDS_maps.py]

```
# -*- coding: utf-8 -*-
"""
In this code the ERDS maps will be visualized for determining a suitable frequency and time for segmenting
and filtering the data.

@author: Ali Abdul Ameer Abbas (with the help of the MNE documentation)
"""


import numpy as np
import matplotlib.pyplot as plt
import mne
from mne.datasets import eegbci
from mne.io import concatenate_raws, read_raw_edf
from mne.time_frequency import tfr_multitaper
from mne.stats import permutation_cluster_1samp_test as pcluster_test
from mne.viz.utils import center_cmap
import pymatreader

#%% load and preprocess data ####################################################
#path = '../Data/CLA/CLASubjectA1601083StLRHand.mat'
path = '../Data/CLA/CLASubjectC1512233StLRHand.mat'
#path = '../Data/CLA/CLASubjectC1512163StLRHand.mat'
#path = '../Data/CLA/CLASubjectC1511263StLRHand.mat'
#path = '../Data/CLA/CLASubjectE1512253StLRHand.mat'
#path = '../Data/CLA/CLASubjectF1509163StLRHand.mat'

struct = pymatreader.read_mat(path, ignore_fields=['previous'], variable_names = 'o')
struct = struct['o']

```

```

chann = struct['chnames']
chann = chann[:-1] # pop th X3 channel, which is only used for the data adquisition.
info = mne.create_info(ch_names = chann,
                               sfreq = float(struct['sampFreq']),ch_types='eeg', montage='standard_1020',
verbose=None)

data = np.array(struct['data'])
data = data[:,:-1]
data_V = data*1e-6 # mne reads units are in Volts

raw = mne.io.RawArray(data_V.transpose(),info, copy = "both")

raw = raw.set_eeg_reference('average', projection= False)

def type_pos(markers, sfreq = 200.0 ):
    """
        Gets the marker position.
        Just left and right positions, NO PASS marker
        Only takes into account the right and left class, not the passive class.

        INPUTS:
            markers --> Markers.
            sfreq --> Sampling frequency.
        OUTPUTS:
            mark --> markers.
            pos --> position of the marker.
            time --> time of the marker.
    """
    mark = []
    pos = []
    time = []
    desc = ['left', 'right'] # for assigning the movements--> left =1, right = 2, pass = 3
    for i in range(len(markers)-1):
        if markers[i]==0 and markers[i+1] != 0 and (markers[i+1] in [1,2]):

            mark.append(desc[markers[i+1]-1])
            pos.append((i+2))
            time.append((i+2)/sfreq)
        else:
           continue
    return mark, pos, time

markers =  np.array(struct['marker']).transpose()
[mark, pos, time] = type_pos(markers)

annotations = mne.Annotations(time, 3.0, mark) # this is the annotation of every class
raw.set_annotations(annotations)

events, _ = mne.events_from_annotations(raw)

picks = mne.pick_channels(raw.info["ch_names"], ["C3", "Cz", "C4"])

#%% Get the epoch data ################################################################
tmin, tmax = -1, 4  # define epochs around events (in s)
event_ids = dict(left=1, right=2)  # map event IDs to tasks

epochs = mne.Epochs(raw, events, event_ids, tmin - 0.5, tmax + 0.5,
                    picks=picks, baseline=None, preload=True)

#%% compute ERDS maps ################################################################
freqs = np.arange(2, 36, 1)  # frequencies from 2-35Hz
n_cycles = freqs  # use constant t/f resolution
vmin, vmax = -1, 1.5  # set min and max ERDS values in plot
baseline = [-1, 0]  # baseline interval (in s)
cmap = center_cmap(plt.cm.jet, vmin, vmax)  # zero maps to white
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

```
kwargs = dict(n_permutations=100, step_down_p=0.05, seed=1,
              buffer_size=None)  # for cluster test

# Run TF decomposition overall epochs
 tfr = tfr_multitaper(epochs, freqs=freqs, n_cycles=n_cycles,
                      use_fft=True, return_itc=False, average=False,
                      decim=2)

 tfr.crop(tmin, tmax)
 tfr.apply_baseline(baseline, mode="percent")

 for event in event_ids:
     # select desired epochs for visualization
     tfr_ev = tfr[event]
     fig, axes = plt.subplots(1, 4, figsize=(12, 4),
                              gridspec_kw={"width_ratios": [10, 10, 10, 1]})
     for ch, ax in enumerate(axes[:-1]):  # for each channel
         # positive clusters
         _, c1, p1, _ = pcluster_test(tfr_ev.data[:, ch, ...], tail=1, **kwargs)
         # negative clusters
         _, c2, p2, _ = pcluster_test(tfr_ev.data[:, ch, ...], tail=-1,
                                      **kwargs)

         # note that we keep clusters with p <= 0.05 from the combined clusters
         # of two independent tests; in this example, we do not correct for
         # these two comparisons
         c = np.stack(c1 + c2, axis=2)  # combined clusters
         p = np.concatenate((p1, p2))  # combined p-values
         mask = c[..., p <= 0.05].any(axis=-1)

         # plot TFR (ERDS map with masking)
         tfr_ev.average().plot([ch], vmin=vmin, vmax=vmax, cmap=(cmap, False),
                               axes=ax, colorbar=False, show=False, mask=mask,
```

```python
                              mask_style="mask")

        ax.set_title(epochs.ch_names[ch], fontsize=10)
        ax.axvline(0, linewidth=1, color="black", linestyle=":")  # event

        if not ax.is_first_col():
            ax.set_ylabel("")
            ax.set_yticklabels("")

    fig.colorbar(axes[0].images[-1], cax=axes[-1])
    fig.suptitle("ERDS ({})".format(event))
    fig.show()

```

A8. Annex 8: Streamlit application for collecting the dataset (Executable) [dataset_collector_BCI.py]

```python
# -*- coding: utf-8 -*-
"""
In this script, an eGUI will be created to guide each participant in the motor imagery
movements that they must do in order to collect data for the dataset appropriately.
The application will be created thanks to the streamlit library.

@author: Ali Abdul Ameer Abbas
"""

# Import libraries.
from random import randint
import streamlit as st
from PIL import Image
import pandas as pd
import time

# Set the page configurations and layout.

st.set_page_config(
     page_title="EEG Imagery",
     page_icon="🧊",
     layout="wide",
     initial_sidebar_state="expanded",
     menu_items={
         'Get Help': 'https://www.extremelycoolapp.com/help',
         'Report a bug': "https://www.extremelycoolapp.com/bug",
         'About': "Motor Imagery"
     }
)

```

```
primaryColor="#F63366"
backgroundColor="#FFFFFF"
secondaryBackgroundColor="#F0F2F6"
textColor="#111111"
font="sans serif"

st.markdown(
         f"""
         <style>
         .stApp {{
             background: url("https://png.pngtree.com/thumb_back/fw800/background/20190830/pngtree-color-network-with-dots-on-white-background-image_310198.jpg");

         }}
         </style>
         """,
         unsafe_allow_html=True

     )

# Add a title, subheader, and other texts.

st.title("Motor Imagery BCI", anchor=None)
st.subheader("Ali Abdul Ameer Abbas")

patient = st.text_input('Participant Identification', 'Unkown')
trial = st.text_input('Trial', 'Unkown')

# Separate the page into three columns and add photos.
col1, col2, col3 = st.columns(3)
with col1:
    st.markdown("<h1 style='text-align: center; color: black;'>LEFT</h1>", unsafe_allow_html=True)
with col2:
    st.markdown("<h1 style='text-align: center; color: black;'>PASS</h1>", unsafe_allow_html=True)
with col3:
    st.markdown("<h1 style='text-align: center; color: black;'>RIGHT</h1>", unsafe_allow_html=True)

placeholder_image_1 = col1.empty()
placeholder_image_2 = col2.empty()
placeholder_image_3 = col3.empty()

with col1:
    img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_IZQUIERDA.png")
    placeholder_image_1.image(img3)
with col2:
    img3 = Image.open("FOTOS_ALAWI/NADA.jpg")
    placeholder_image_2.image(img3)
with col3:
    img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_DERECHA.png")
    placeholder_image_3.image(img3)

# Define some parameters, useful for saving the events in a csv file.
segundos = 3                 # Time of green display of an action.
fs = 200                     # Sampling frequency.
repetitions = segundos * fs # Samples due to the seconds waited.
storage_vector = []          # Vector for saving the markers in a csv file.

# Set the Start and Stop buttons, with the help of a flag.
flag = 0
if st.button('START',key="14"):
    flag = 1

if st.button("STOP",key="2"):
    flag = 0

# Randomly activate an activity (Left-hand imagery, Right-hand imagery, Pass)
```

```

while(flag == 1):
    for _ in range(10):
        number = randint(1, 3)

        with col1:
            if number == 1:

                img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_IZQUIERDA_VERDE.png")
                placeholder_image_1.image(img3)
                time.sleep(segundos)
                storage_vector.extend([1] * repetitions)
                img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_IZQUIERDA.png")
                placeholder_image_1.image(img3)
                time.sleep(segundos)
                storage_vector.extend([0] * repetitions)

            else:
                img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_IZQUIERDA.png")
                placeholder_image_1.image(img3)


        with col2:
            if number == 2:
                img3 = Image.open("FOTOS_ALAWI/NADA_VERDE.jpeg")
                placeholder_image_2.image(img3)
                time.sleep(segundos)
                storage_vector.extend([2] * repetitions)
                img3 = Image.open("FOTOS_ALAWI/NADA.jpg")
                placeholder_image_2.image(img3)
                time.sleep(segundos)
                storage_vector.extend([0] * repetitions)

            else:
                img3 = Image.open("FOTOS_ALAWI/NADA.jpg")
                placeholder_image_2.image(img3)


        with col3:
            if number == 3:
                img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_DERECHA_VERDE.png")
                placeholder_image_3.image(img3)
                time.sleep(segundos)
                storage_vector.extend([3] * repetitions)
                img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_DERECHA.png")
                placeholder_image_3.image(img3)
                time.sleep(segundos)
                storage_vector.extend([0] * repetitions)
            else:
                img3 = Image.open("FOTOS_ALAWI/MANO_VACIA_DERECHA.png")
                placeholder_image_3.image(img3)

        # Save the markers in a csv file.
        df = pd.DataFrame(storage_vector)
        print(df)
        # saving the dataframe
        df.to_csv(f'{patient}_{trial}_Record.csv'
```