Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Control System Engineering LM-25

Development of a BCI-Driven Drone Control System using ROSneuro, Gazebo, and PX4: from simulation to real-world deployment

Supervisor
Prof. Ruggero Carli

Graduating
Amato Gregorio

Academic Year 2023/2024

Graduation Date 23/10/2024

Questa Tesi è per te, Nonna.
Una volta mi dissero che ogniuno di noi si porta con se una stella.

Il più grande dei ringraziamenti va alla mia Famiglia,
senza il loro amore non sarei Qui a battere queste "quattro" parole.

Don Giuseppe che è stato presente in ogni momento,
il Collegio Santa Giustina che è stata una seconda casa, ma sopratutto,
Camera 10 composta da Giovanni, Leonardo, Giulio e Michele.
Federico, Francesco, Roberto e Caterina per questi anni fantastici passati insieme,
siete stati una famiglia in quel di Padova.

Abstract

Brain-Computer Interface (BCI) technology offers a groundbreaking approach to controlling devices through brain signals, with applications ranging from medical rehabilitation to entertainment. This thesis explores the integration of BCI with drone technology, focusing on controlling a quadcopter using thought alone. By leveraging electroencephalography (EEG) to capture brain activity and translate these signals into commands for the drone, the goal was to achieve intuitive, hands-free control in a simulated environment. The implementation relies critically on the use of ROS (Robot Operating System), which enabled the integration of the ROSNeuro framework, PX4 autopilot system, and Gazebo simulator to establish the link between EEG signals and the drone control system. EEG data, captured using the Emotiv EPOC X headset, was processed through the EmotivPRO application and transmitted to ROSNeuro. Although the original objective was to control a physical quadcopter and compare brain signal noise in ground versus flight conditions, technical challenges limited the project to simulation. This thesis outlines the methods used, the critical integration of ROS with hardware and software systems, and the limitations encountered, providing a foundation for future developments in BCI-driven drone control.

La tecnologia Brain-Computer Interface (BCI) offre un approccio rivoluzionario per il controllo di dispositivi attraverso segnali cerebrali, con applicazioni che spaziano dalla riabilitazione medica all'intrattenimento. Questa tesi esplora l'integrazione della BCI con la tecnologia dei droni, concentrandosi sul controllo di un quadricottero tramite il solo pensiero. Utilizzando l'elettroencefalografia (EEG) per rilevare l'attività cerebrale e tradurre questi segnali in comandi per il drone, l'obiettivo era ottenere un controllo intuitivo e senza mani in un ambiente simulato. L'implementazione si basa sull'uso fondamentale di ROS (Robot Operating System), che ha permesso l'integrazione del framework ROSNeuro, del sistema autopilota PX4 e del simulatore Gazebo per stabilire la connessione tra i segnali EEG e il sistema di controllo del drone. I dati EEG, acquisiti tramite il dispositivo Emotiv EPOC X, sono stati elaborati dall'applicazione EmotivPRO e trasmessi a ROSNeuro. Sebbene l'obiettivo iniziale fosse quello di controllare un quadricottero fisico e confrontare il rumore dei segnali cerebrali in condizioni a terra e in volo, sfide tecniche hanno limitato il progetto alla simulazione. Questa tesi descrive i metodi impiegati, l'integrazione critica di ROS con i sistemi hardware e software, e le limitazioni riscontrate, fornendo una base per futuri sviluppi nel controllo dei droni tramite BCI.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The concept of controlling a drone using only brain signals—without the need for physical controls such as joysticks or remote devices—represents a significant innovation at the intersection of neuroscience and unmanned aerial vehicle (UAV) technology. This vision is made possible through the integration of Brain-Computer Interface (BCI) technology with quadcopters, allowing for direct communication between the human brain and a computer system. In simple terms, a BCI is a system that enables the brain to send commands to a machine or device. When applied to quadcopters, BCI technology could enable a user to control and navigate a drone purely through thought. This technological breakthrough has the potential to revolutionize fields such as search and rescue, environmental monitoring, and entertainment. The key to this innovation lies in creating a seamless link between the brain's neural activity and the drone's control system, allowing for intuitive, hands-free operation. Typically, brain activity is recorded using electroencephalography (EEG) sensors, which capture specific brain signals associated with user intentions. These signals are then processed and translated into commands that control the movements of the drone. While still in its early stages, the potential applications of this technology are vast. However, significant challenges remain, including issues related to signal reliability, processing speed, and the noise inherent in brain signal data. This thesis focuses on controlling a quadcopter using a BCI in a simulated environment, employing the Gazebo simulator and the PX4 framework. While the ultimate goal of the research is to construct and fly a physical quadcopter and analyze brain signal noise in two scenarios—when the drone is on the ground and during flight—various technical challenges arose that limited the scope of the project to simulation. Before discussing the development of the project, it is important to provide an overview of the brain's structure, the process of acquiring EEG signals, and the technologies involved in capturing and interpreting neural activity.

1.1 The Nervous System: Coordination and Control of the Human Body

The human body's functions, from the complex interactions between organs to how we respond to our environment, are controlled and interpreted by the brain. The brain processes information through electrical signals, creating responses that it sends to various parts of the body via the spinal cord. This entire process is managed by the Central Nervous System (CNS), which includes the brain and spinal cord, and it works closely with the Peripheral Nervous System (PNS). The PNS is made up of a network of nerves and ganglia located outside the brain, which transmit signals between the CNS and the rest of the body.

- Central Nervous System (CNS): The CNS is the control center of the body. It processes sensory information, such as what we see, hear, and feel, and decides how the body should respond. The CNS is responsible for higher functions like thinking, memory, and emotion, as well as basic functions like breathing and heart rate.

- Brain: The brain is the most complex organ, responsible for processing all sensory data, controlling movements, and managing thoughts and emotions. It's divided into different regions, each with specialized functions, such as the cerebral cortex for thinking and the cerebellum for coordinating movement.

- Spinal Cord: The spinal cord is a long, thin bundle of nerves that runs from the brain down the spine. It acts as a highway for information traveling between the brain and the rest of the body. It also handles some reflexes, like pulling your hand away from something hot.

- Peripheral Nervous System (PNS): The PNS connects the CNS to the limbs and organs. It's responsible for transmitting signals from the brain and spinal cord to the rest of the body and vice versa. The PNS is divided into two main parts:

- Somatic Nervous System: This controls voluntary movements, like walking or picking up an object. It carries signals from the CNS to muscles and from sensory organs back to the CNS.

- Autonomic Nervous System (ANS): This controls involuntary functions, like heartbeat, digestion, and breathing. The ANS is further divided into: Sympathetic Nervous System: Prepares the body for stress-related activities, often referred to as the ”fight or flight” response. Parasympathetic Nervous System: Calms the body down after stress and helps with ”rest and digest” functions.

The brain is composed of gray matter, which contains the cell bodies of neurons, and white matter, which consists of neural axons [2]It is divided into two hemispheres: the right and the left. These hemispheres are connected by the corpus callosum, a thick band of nerve fibers. Despite their similar appearance on a macroscopic level, the two hemispheres have distinct structural differences at the microscopic level, leading to different functional specializations. The left hemisphere predominantly controls the right side of the body and is responsible for language abilities, logical reasoning, mathematical calculations, and analytical thinking. On the other hand, the right hemisphere governs the left side of the body and is involved in spatial recognition, object identification, imagination, and intuitive thinking. Essentially, the two hemispheres can be seen as functional opposites: the left is more pragmatic and analytical, while the right is more abstract and creative. Each hemisphere is further divided into four lobes 1.1: the frontal lobe, parietal lobe, occipital lobe, and temporal lobe. Each lobe specializes in particular functions, but they do not operate in isolation. Damage to any one of these lobes can disrupt the overall functioning of the brain.

- Frontal lobe (see figure 1.1): Located at the front of the brainis crucial for higher cognitive functions such as decision-making, problem-solving, planning, and social behavior. It also controls voluntary movement through the motor cortex.

- Parietal Lobe (see figure 1.1): Positioned at the top of the brain, the parietal lobe processes sensory information related to touch, temperature, and pain. It also plays a role in spatial orientation and body awareness.

- Temporal Lobe (see figure 1.1): Found on the sides of the brain, near the temples, the temporal lobe is involved in processing auditory information and is key for understanding language, as well as playing a role in memory formation.

- Occipital Lobe (see figure 1.1): Located at the back of the brain, the occipital lobe is primarily responsible for visual processing. It interprets information from the eyes and translates it into the images we see. This information is then sent to the parietal lobe and temporal lobe for further refinement of their decoding.

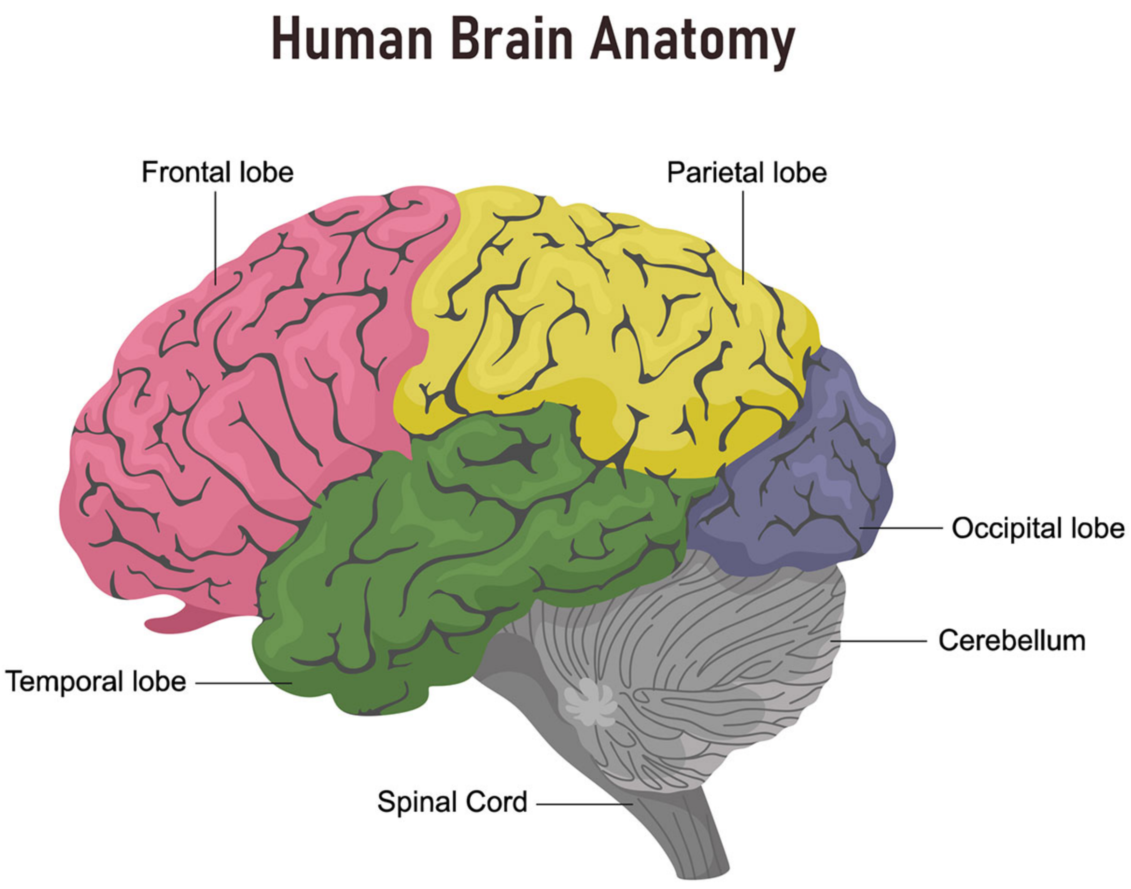


Figure 1.1: 3D view Human Brain Anatomy

In the context of Brain-Computer Interfaces (BCIs), motor imagery plays a significant role, particularly within the frontal and parietal lobes. Motor imagery refers to the mental simulation of movement without any actual physical execution. This process involves activating neural networks in regions such as the motor cortex, which is located in the frontal lobe, and the parietal lobe, which is involved in integrating sensory information and spatial orientation. When a person engages in motor imagery—such as imagining the movement of a hand or foot—specific areas in the brain light up as if the movement were actually occurring. This phenomenon is particularly relevant in BCI systems, where users can control external devices (such as prosthetic limbs, computer cursors, or even robotic systems) through imagined movements. Motor imagery is a critical component of BCI technology because it allows individuals, especially those with motor impairments, to interact with their environment in meaningful ways without needing to produce physical movement. For example, in patients who have suffered a stroke or have severe motor disabilities, the ability to imagine movement can help in both rehabilitation and communication. In BCIs, the signals generated during motor imagery are detected through methods such as electroencephalography (EEG) and then translated by the interface into commands that can operate external devices. The efficiency and accuracy of these systems often depend on the user's ability to engage in vivid motor imagery, as well as the system's ability to interpret these neural signals accurately. Thus, the integration of motor imagery within BCI frameworks not only highlights the sophisticated interplay between different brain regions but also opens up new possibilities for enhancing the quality of life for individuals with motor impairments. Returning to the content of the transmission of information nervous system, it is transmitted and processed through electrical impulses generated and propagated within specialized cells called neurons. These nerve cells are composed of three primary parts:

- The Soma: Typically spherical in shape, it contains the nucleus and all the organelles necessary for the neuron's survival and function.
- The Axon: This is a long projection through which the electrical impulse, known as the action potential, travels. The action potential is regenerated along the axon at specific intervals called the nodes of Ranvier—gaps [3] where the myelin sheath (a protective and insulating layer) is absent. This process ensures that the signal efficiently travels the length of the axon and reaches the synaptic terminals. At these terminals, neurotransmitter-containing vesicles release the signal across the synapse to the dendrites of the adjacent neuron, enabling communication between neurons.
- The Dendrites: These are branching structures that receive signals from the axons of other neurons, allowing for the integration of information from various sources.

Neurons vary in shape, structure, and function depending on their location in the body [4]:

- Sensory Neurons: These neurons transmit signals from external sources (somatic sensory neurons) or internal organs (visceral sensory neurons) to the Central Nervous System (CNS).
- Motor Neurons: it carry signals from the CNS to peripheral targets. Somatic motor neurons relay instructions to skeletal muscles, facilitating voluntary movements, while visceral motor neurons (or visceral effector neurons) transmit signals to smooth muscles and glands, controlling involuntary actions.
- Interneurons (Associative Neurons): These neurons are located within the CNS and play a crucial role in processing and interpreting sensory inputs. They coordinate and relay the appropriate responses by linking sensory and motor pathways.

Depending on their function and location, neurons are specifically adapted to efficiently manage their roles within the nervous system, ensuring effective communication between the brain, spinal cord, and the rest of the body.

1.2 Comparing Invasive and Non-Invasive Techniques for Brain Assessment: Accuracy, Safety, and Data Quality

To gather insights into how the brain functions and its overall health, we rely on various specialized techniques designed to detect specific patterns within the brain. These methods can be broadly categorized into invasive and non-invasive approaches, each with its own set of advantages and trade-offs concerning accuracy, safety, and data quality.

- Invasive techniques involve tools that physically interact with the body's internal tissues. Because they access areas within the body, these methods often provide highly detailed and accurate data. However, this precision comes at a cost: invasive procedures carry the risk of tissue damage, infection, and other complications, making them less desirable for routine use.

- non-invasive techniques operate externally, typically by placing sensors on the surface of the skin or just outside the body. While these methods offer a much safer option for patients, they often produce data that is less precise due to interference or "noise" from other sources. Despite this, their safety and ease of use make non-invasive techniques highly popular in both clinical and research settings.

One of the most common and widely used non-invasive techniques for monitoring brain activity is Electroencephalography (EEG). EEG involves placing electrodes on the scalp to measure electrical activity in the brain. This method is particularly valued for its safety and ease of use, as it doesn't require any invasive procedures. However, because the electrodes are on the scalp rather than inside the brain, the signals can be somewhat weaker and more prone to interference from other sources of electrical activity. Despite this, EEG is incredibly useful, especially in clinical settings and research, because it allows for real-time monitoring of brain activity with minimal risk to the patient. It's commonly used to diagnose conditions like epilepsy, sleep disorders, and other neurological issues. Additionally, EEG is a key tool in the development of Brain-Computer Interfaces (BCIs), where it helps translate brain activity into commands that can control external devices. Because the non-invasive technique of EEG was the focus of this thesis project, the following will report on its inception and use.

1.2.1 Electroencephalography(EEG)

Hans Berger[5], known for discovering the human EEG in the late 1920s, developed the technique to offer a "window into the brain." He observed signals that fluctuated rhythmically when a person's eyes were closed, but these signals became less rhythmic and had a lower amplitude when the eyes were open. Initially, the scientific community was skeptical of Berger's findings and his EEG method. Some critics argued that the scalp EEG reflected heart or muscle activity, while others believed brain activity shouldn't decrease in rhythm or amplitude when the eyes opened (a phenomenon now recognized as 'alpha blocking'). Additionally, many thought that the rhythmic fluctuations Berger detected were too slow to represent real neural activity, which was then assumed to only occur through action potentials. It wasn't until British physiologists Edgar Adrian and Bryan Matthews confirmed Berger's findings in 1934 that EEG was accepted as a valid, non-invasive method for measuring the brain's electric fields. Importantly, Adrian and Matthews gave full credit to Berger for discovering the scalp EEG. Berger's research was interrupted by the Nazi regime in Germany, and tragically, he took his own life in 1941. Despite this, Berger's lasting legacy is the introduction of a technique that quickly gained widespread use—not only in clinical settings but also in research fields ranging from neurophysiology to computer science. EEG provides a direct, real-time measurement of the brain's neural activity, making it possible to assess the functionality of specific neurophysiological pathways, states of consciousness and sleep, as well as the precise timing of brain activity or dysfunction. When this precise timing (temporal resolution) is combined with estimates of the brain regions responsible for the activity, EEG can also help map brain networks, analyze their connectivity, and determine whether certain processes occur sequentially or in parallel during a given task. However, because EEG records electrical potentials — essentially, the voltage that represents the energy required to move electrical charges between two points — it is crucial that users of EEG (and electrophysiological methods in general) fully understand the implications of measuring between an active site and a reference point. Misinterpretation of these measurements can affect the accuracy and analysis of the data. When analyzing EEG data, we can break it down into a series of sine waves, generating a frequency spectrum that reveals the brain's electrical activity. Each wave can be described by its amplitude (or power, in the case of a rectified signal) and its phase, which indicates where the sine wave is in its cycle. These properties are key in understanding brain dynamics during specific tasks, such as motor imagery, a process central to many BCI systems. For motor imagery and BCI applications, the EEG is often divided into distinct frequency bands that each reflect different neural processes. These commonly include:

- Delta (δ: 0.2–3.5 Hz): Typically linked to deep sleep and unconscious states.
- Theta (θ: 4–7.5 Hz): Associated with drowsiness and light sleep, but also memory and navigation processes.
- Alpha (α: 8–13 Hz): Crucial for BCI and motor imagery, especially over sensorimotor areas. When subjects imagine movement, the alpha rhythms (also known as mu rhythms) over the motor cortex often decrease in power—a phenomenon known as event-related

desynchronization (ERD).

- Beta (β: 14–30 Hz): Linked to active motor processing. During motor imagery or movement preparation, beta rhythms over the motor cortex often exhibit desynchronization.
- Gamma (γ: 30–90 Hz): Reflects higher cognitive functions and could also be involved in motor tasks, although less frequently used in BCIs for motor imagery.

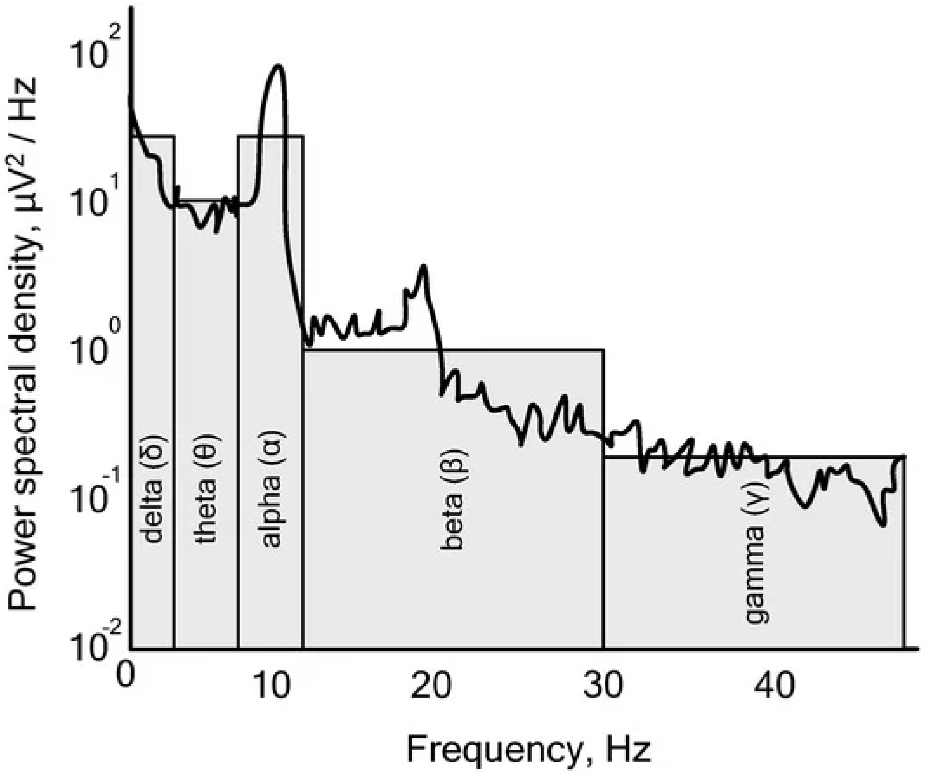


Figure 1.2: Example of the frequency spectrum of a human brain electroencephalogram

While these subdivisions have empirical grounding, a common misconception is that each frequency band corresponds to a specific brain function in a simple, one-to-one relationship. In reality, oscillatory activity is nested within itself, showing complex amplitude-phase interactions across different frequencies. For example, in motor imagery tasks, changes in mu and beta rhythms can occur simultaneously and interact dynamically, reflecting both preparatory states and imagined movement. One of the strengths of EEG in BCI research is its ability to track real-time changes in these oscillatory patterns. For instance, when analyzing motor imagery, we can detect desynchronization of alpha and beta rhythms, helping us interpret the brain's intent to move without actual physical motion. This process allows the BCI system to translate these neural signatures into actionable commands for devices like prosthetics or computer interfaces. However, it's important to note that EEG measures electrical potentials — essentially voltage differences between an active site and a reference point. This makes it crucial for researchers to fully grasp the implications of how these measurements are obtained and analyzed, as misinterpretations can lead to skewed data. Furthermore, EEG decomposition assumes the brain's electrical activity is made up of sine waves, yet the reality includes non-sinusoidal and even arrhythmic signals, adding further complexity to the analysis. Despite these challenges, frequency decomposition remains one of the most effective methods for distinguishing between evoked and induced brain activity. In the context of motor imagery and BCIs, evoked activity is often time-locked and phase-locked to an event (such as a visual cue prompting imagined movement), while induced activity is neither time-locked nor phase-locked, representing more spontaneous neural processes. The full potential of EEG for analysis and its wide range of applications remain underutilized. For a period, EEG was the leading non-invasive method for measuring brain activity. However, with the development of other techniques such as magnetoencephalography 1.4 (MEG) in 1968, it measures the magnetic fields produced by neural activity in the brain. Like EEG, it offers excellent temporal resolution, but it has the added advantage of pinpointing the source of brain activity with greater spatial precision. This makes MEG particularly useful for studying the timing and localization of brain processes. Positron emission tomography (PET) in 1975 is a functional imaging technique that tracks metabolic processes in the brain. By using radioactive tracers, PET scans can show areas of the brain that are more active based on glucose metabolism, providing valuable insights into brain function and disorders like Alzheimer's disease. However, it lacks the real-time resolution that EEG offers. Transcranial magnetic stimulation (TMS) in 1985 is a non-invasive method that uses magnetic fields to stimulate neurons in the brain. It can both excite and inhibit brain areas, making it a valuable tool for studying brain function and potentially treating conditions such as depression. While not primarily a brain imaging technique, it can be combined with EEG to study brain activity in response to stimulation and functional magnetic resonance imaging (fMRI) in 1990 has become one of the most widely used tools for studying brain function. It measures changes in blood flow (the BOLD signal) to infer brain activity. Although fMRI provides excellent spatial resolution, its temporal resolution is much slower compared to EEG and MEG, making it less effective for capturing the rapid dynamics of neural activity. EEG lost some prominence. Recently, though, renewed interest in brain dynamics and real-world applications has led to a resurgence in the use of EEG1.3, re-establishing it as a top neuroscience tool.

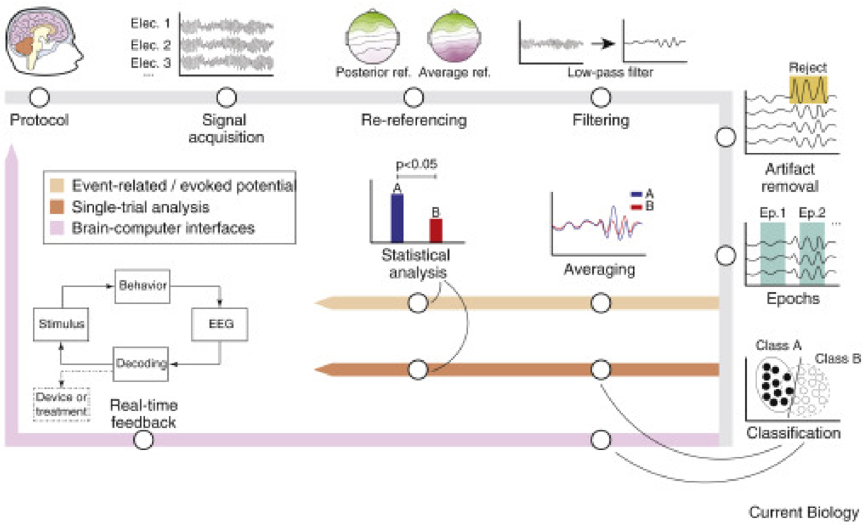


Figure 1.3: We outline three main steps in EEG processing and analysis. The first step, shown in gray in the diagram, is called ”pre-processing.” During this phase, we acquire the EEG signals and adjust each channel’s recording to a common reference voltage. Next, we filter the signals to remove any non-biological noise. We then check the signals for artifacts, such as those from muscle activity, and select the relevant data segments for further analysis. In the next step, we group and average different event-related potentials (ERPs) to create time series data, which are then compared statistically to find significant differences. For single-trial analysis, instead of averaging the data, we train a classifier. This algorithm learns to identify the condition that produced the signal based on specific features of the signal and measures how accurately it performs.

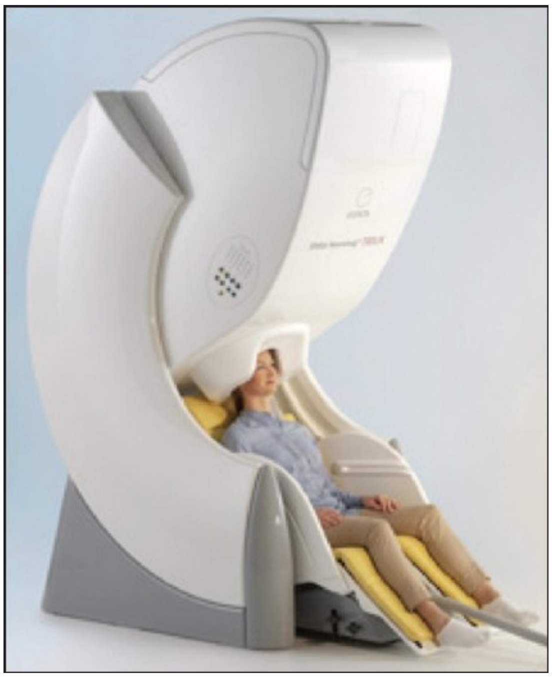


Figure 1.4: Elekta Neuromag Triux 306 sensors MEG system

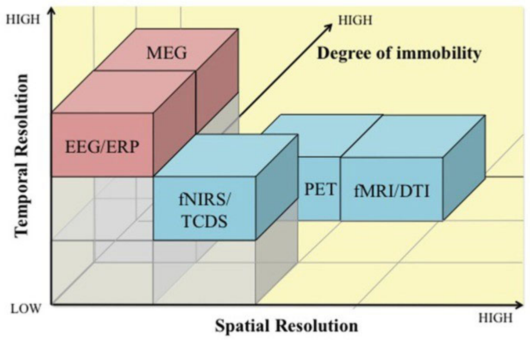


Figure 1.5: Temporal and spatial resolution and degree of immobility for the different neuroimaging methods. MEG presents excellent temporal resolution and very good spatial resolution compared to other techniques

1.2.2 Invasive Techniques for Brain Assessment: Overview, Differences, Challenges, and BCI Applications

In contrast to non-invasive methods, invasive techniques for brain assessment involve direct interaction with the brain's internal structures. These methods are characterized by their high precision and ability to provide detailed insights into brain activity, but they also come with significant risks and complexities. The primary invasive techniques for brain assessment include Electrocorticography (ECoG), intracranial electrodes, microelectrode arrays (MEAs), and deep brain stimulation (DBS).

Types of Invasive Techniques

- **Electrocorticography (ECoG):** ECoG involves placing electrodes directly on the surface of the brain, usually during neurosurgical procedures. This technique offers high spatial and temporal resolution, allowing for detailed monitoring of cortical activity. ECoG is particularly useful for surgical planning in epilepsy treatment and for research on cortical functions. Despite its high accuracy, ECoG carries risks such as infection and damage to brain tissue.

- **Intracranial Electrodes (Depth Electrodes):** These electrodes are implanted deep within the brain to record electrical activity from specific brain regions. Depth electrodes provide highly localized data and are often used to pinpoint the origin of epileptic seizures. The main risks associated with this technique include tissue damage, bleeding, and infection.

- **Microelectrode Arrays (MEAs):** MEAs consist of a grid of tiny electrodes implanted in the brain to record the activity of individual neurons or small groups of neurons. This method offers exceptional spatial resolution and is used in research to study neural circuits and brain-machine interfaces. The challenges of MEAs include potential long-term tissue damage and the complexity of maintaining electrode functionality over extended periods.

- **Deep Brain Stimulation (DBS):** DBS involves implanting electrodes into specific brain regions to deliver electrical stimulation. While primarily used as a therapeutic approach for disorders such as Parkinson's disease, DBS can also provide valuable data about brain function and connectivity. Risks include infection, bleeding, and potential adverse effects on brain function.

Differences Between Invasive Techniques

The invasive methods differ significantly in terms of their application, resolution, and risk profile:

- **Resolution and Specificity:** ECoG and intracranial electrodes provide high spatial and temporal resolution compared to non-invasive methods. MEAs offer even finer resolution by recording from individual neurons, which is valuable for studying neural circuits in detail.

- **Invasiveness and Risks:** All invasive techniques involve risks related to the surgical procedure and long-term effects. ECoG and depth electrodes are highly invasive, requiring surgery to access the brain. MEAs, while also invasive, offer high-resolution data but can lead to long-term tissue damage. DBS is less invasive compared to direct recording techniques but still involves surgical implantation.

- **Clinical and Research Applications:** ECoG and intracranial electrodes are used both in clinical settings for surgical planning and in research for studying brain function. MEAs are primarily used for research purposes due to their complexity and risk. DBS is mainly used for therapeutic purposes but also contributes to research on brain stimulation and connectivity.

Applications in Brain-Computer Interfaces (BCIs)

Invasive techniques have significantly advanced the development of BCIs by providing high-resolution brain activity data that can be used to control external devices with greater precision:

- **Neuroprosthetics:** Invasive methods like ECoG and depth electrodes enable users to control prosthetic limbs or other devices through direct brain signals. This enhances the precision and functionality of neuroprosthetic devices.

- **Communication Devices:** BCIs developed using invasive techniques can assist individuals with severe motor impairments in communicating. By providing more accurate and reliable control, these devices improve the quality of life for individuals with disabilities.

- **Neural Decoding and Research:** Invasive techniques contribute to the development of advanced neural decoding algorithms, which enhance the accuracy and effectiveness of BCIs. Research using methods like MEAs helps in understanding complex neural patterns and improving BCI performance.

In summary, while invasive techniques for brain assessment offer unparalleled accuracy and detailed insights, they come with significant risks and complexities. Their applications in BCIs demonstrate their potential to revolutionize the interaction between humans and technology, offering new possibilities for communication, control, and rehabilitation.

1.3 Brain Computer Interface (BCI)

After introducing various invasive and non-invasive techniques, this section will explore what Brain-Computer Interfaces (BCIs) are, along with the latest trends, challenges, and potential risks associated with them. It's important to note that the terms Brain-Computer Interface (BCI) and Brain-Machine Interface (BMI) are used interchangeably. Both refer to systems that create a direct communication link between the brain and external devices, bypassing the need for the body's normal nerve and muscle pathways. Humans typically rely on their peripheral nerves and muscles to interact with the physical world and complete tasks. However, for individuals with severe neurological conditions—such as amyotrophic lateral sclerosis (ALS) or those who have suffered brainstem strokes—this essential mechanism breaks down. These individuals lose the ability to control external devices and often require assistance from others, which may not always be readily available. To address this limitation, researchers developed BCIs, also known as BMIs. BCIs interpret brain signals and translate them into actions, bypassing the need for the body's motor systems. By doing so, BCIs provide a non-muscular communication channel, enabling individuals to control external devices such as computers or robotic limbs directly with their thoughts. This technology facilitates the acquisition, manipulation, analysis, and translation of brain signals, allowing people to interact with the environment in ways previously thought impossible. Since BCI technology was first introduced by Jacques Vidal in 1973 [6], the field has grown rapidly, offering vast and exciting possibilities. Today, BCIs [7] have a wide range of applications that can be grouped by industry: gaming and entertainment, healthcare, security and authentication, education, neuromarketing (commercial marketing based on neuroscience), and neuroergonomics (the application of neuroscience to improve workplace ergonomics). However, despite its promising advancements, BCI research still faces significant challenges. These include ensuring signal accuracy, addressing ethical concerns, and protecting user privacy from potential misuse. Nonetheless, BCI technology continues to evolve, and researchers are actively working to address these issues as they explore the future potential of direct brain-device communication.

1.3.1 Component of BCIs

The BCI system consists of three core components [7], each fulfilling a distinct function: signal acquisition, signal processing, and application . These elements are integrated and collaborate to enable the transmission of brain signals to the intended BCI application (e.g., a robotic arm). In certain cases, control signals from the BCI application can be fed back to the brain to stimulate typical human functions, like vision and hearing.

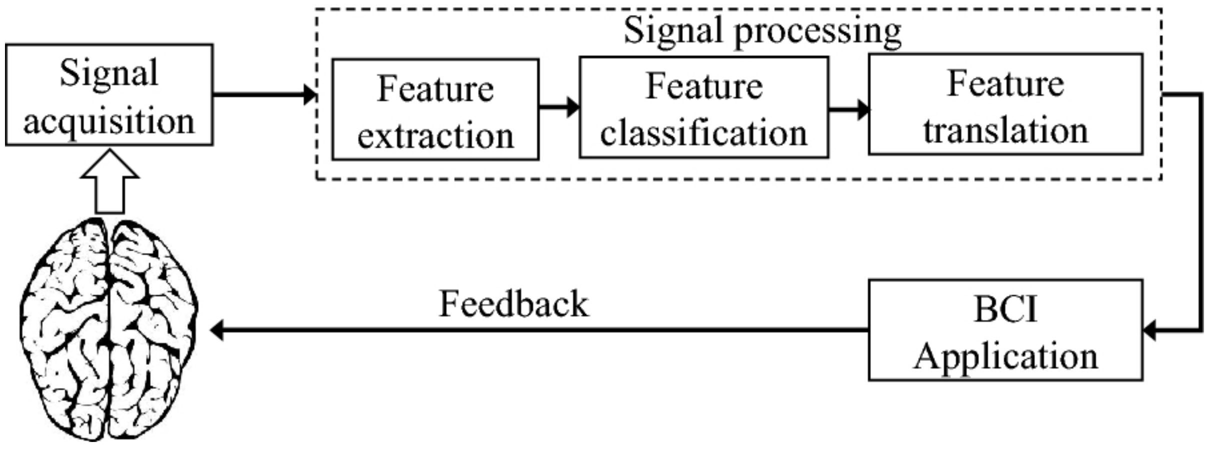


Figure 1.6: Main components of the brain–computer interface (BCI) system

Looking at the diagram in the figure1.6 above, we start to describe starting from the left to right:

1.3.2 Signal Acquisition

This component consists of an electronic device equipped with electrodes designed to capture brain signals, which are oscillating electrical voltages generated by the brain's biological activity that indicate its neurophysiological states. Signal acquisition involves recording electrophysiological signals that correspond to specific brain activities. Most BCI systems, including commercial ones, utilize the electrophysiological signals that have been reported in the following section 1.2 that are : electroencephalography (EEG), which records the brain's electrical activity via electrodes placed on the scalp; electrocorticography (ECoG), which measures brain signals directly through electrodes placed on the surgically exposed cerebral cortex; local field potentials (LFPs), which capture electric potentials in the neuron's extracellular space; and neuronal action potentials, which are rapid, short-term changes in the membrane potential of neurons. Before moving to the next stage of the BCI process, the recorded brain signals are filtered, amplified, and digitized.

1.3.3 Feature Extraction

In this phase, the BCI system identifies key electrophysiological characteristics from the captured signals to represent brain activities, thereby encoding the user's intent [21]. As with the previous stage, it is crucial that feature extraction is performed accurately to ensure that the extracted features strongly correlate with the user's intentions, improving the system's efficiency and overall performance. Standard BCI systems utilize features in either the time or frequency domain, which can include various attributes such as the amplitude or latency of event-related potentials, frequency power spectra, or neuronal firing rates. Therefore, before developing the BCI system, it is essential to determine the domain transformation and feature characteristics. Additionally, any interfering artifacts within the extracted features, which could adversely affect the later stages of the BCI system, should be removed.

1.3.4 Feature Classification

The extracted features represent the brain's activity related to specific intended actions. The classification process is then used to identify patterns within these features that correspond to the intended movements or tasks. For example, if a user intends to move a robotic arm to the right, the BCI system will analyze the brain signals associated with that action, extracting relevant features such as increased activity in motor cortex regions. These features are then classified into recognizable patterns that signal the desired movement. Similarly, if the user wants to select a letter on a screen using only their thoughts, the system can detect and classify specific neural patterns related to visual attention or focus, allowing the selection to be made. By accurately classifying these patterns, the BCI system can reliably interpret the user's intent and translate it into real-world actions, such as controlling a wheelchair, operating a computer, or even interacting with smart home devices or as in the case of the thesis being drafted on flying a drone

1.3.5 Feature Translation

In this stage of signal processing, the classified features are converted into specific commands that control an external device, such as the BCI application. For instance, in the context of a BCI system designed to pilot a drone, the brain signals related to directional thoughts—such as thinking about moving forward, backward, left, or right—are processed. These signals are translated into corresponding commands, allowing the drone to move in the desired direction. For example, when the user focuses on a particular mental pattern associated with moving forward, the system classifies this as the intent to fly the drone in that direction. This classification is then transformed into an actual command, sending signals to the drone's control system to propel it forward. Similarly, when the user thinks about ascending or descending, the BCI processes those brain signals and converts them into commands that control the drone's altitude. This seamless conversion of mental intention into real-world actions enables the user to navigate and control the drone purely through thought, bypassing traditional physical control mechanisms.

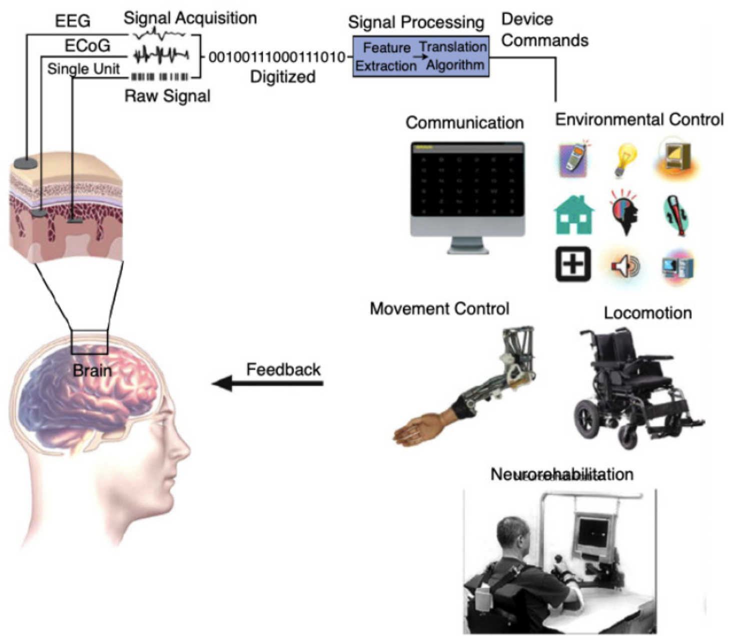


Figure 1.7: Components of a typical BCI system and its communication methods—simplified scheme.

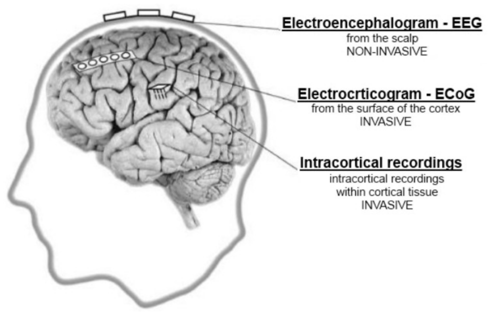


Figure 1.8: Three different methods for electrical activity of brain recordings

1.3.6 Stimulation Paradigms in Brain-Machine Interfaces: Exogenous and Endogenous

In the last few years, different interactions between BMIs and robotic devices have been explored based on the nature of the mental task performed by the user and the neural processes involved [8]. Brain-machine interfaces (BMIs) are primarily based on three key paradigms for controlling external devices: exogenous stimulation, which leverages brain signals evoked by external stimuli; endogenous stimulation, which focuses on the voluntary modulation of brain activity without external cues; and error-related potentials (ErrPs), which detect the user's perception of errors during system operation. In BMI systems based on exogenous stimulation, neural responses are triggered by external stimuli such as visual or auditory signals. A common example is the use of steady-state visual evoked potentials (SSVEP), which exploit the brain's response to visual stimuli flickering at specific frequencies (between 3.5 and 60 Hz). When a user fixates on a flickering target, the visual cortex generates a neural response that is correlated with the flicker frequency, allowing the BMI system to decode the user's intent. These systems are widely used to control telepresence robots, powered wheelchairs, and other assistive devices, enabling users to issue low-level commands (e.g., "turn left" or "move forward") or to select a final destination. Another exogenous paradigm is based on P300 signals [9], [10], which are brain potentials evoked approximately 300 milliseconds after the user perceives a rare stimulus among frequent ones. This paradigm is commonly applied in communication systems such as the Farwell–Donchin speller, but it can also be used to control robotic devices. External stimuli elicit the P300 response, which is then decoded and translated into operational commands for the device. In contrast, BMI systems based on endogenous stimulation do not rely on external cues but instead depend on the user's ability to voluntarily modulate brain activity. A well-known example is motor imagery, where the user imagines the movement of a specific body part, such as the right or left hand, causing changes in brain rhythms known as event-related synchronizations (ERS) or desynchronizations (ERD). These modulations are typically observed in the μ (8–12 Hz) and β (16–30 Hz) frequency bands in the motor and premotor cortices. The specific location of brain activity depends on the imagined task: imagining movement of the right or left hand activates contralateral brain regions (near electrodes C1–3 and C2–4), while imagining foot movement activates medial regions near electrode Cz. This type of interaction allows the user to continuously control an external device, such as a robot, with the help of visual feedback, which displays in real time the effects of their brain signal modulation. This mutual learning process between the user and the system helps refine the user's ability to modulate their brain signals, improving the accuracy and control of the robotic device. A third important element in BMIs is the detection of error-related potentials (ErrPs). These signals are generated when the user detects a mismatch between their intended action and the system's response. ErrPs are usually observed in the anterior cingulate cortex and occur approximately 200 milliseconds after the user perceives an error. In real-time BMI systems, ErrPs can be used to automatically detect and correct errors, increasing the reliability of the interface. For example, when controlling a robot or wheelchair, if the system executes an incorrect movement, the ErrP signal can trigger a corrective action or request further user input. This error-detection mechanism significantly enhances the overall robustness of BMI systems. Complementarity of the Paradigms Exogenous and endogenous stimulation, along with error-related potentials, offer distinct yet complementary approaches to BMI control. Exogenous paradigms capitalize on the brain's response to external stimuli, providing quick and precise interaction in well-defined contexts. In contrast, endogenous paradigms like motor imagery enable more continuous and flexible control, relying on the user's self-regulation of brainwaves. Error-related potentials add an extra layer of robustness by allowing real-time detection and correction of system errors, ensuring more reliable control. Together, these paradigms are essential for developing BMI systems that can adapt to various applications, from robotic control to communication systems and assistive technologies.

1.4 Unmanned Aerial Vehicle (UAV)

UAV which stands for Unmanned Aerial Vehicle, is a pilotless aircraft commonly known as a drone. It is a flying device that can be remotely controlled by a human operator or programmed to fly autonomously following a predefined path or executing specific actions. If a UAV is manually controlled by a human operator, it typically occurs through the use of a GCS (Ground Control Station). There are other autonomous vehicles like:

1. UGV (Unmanned Ground Vehicles): Unmanned ground vehicles designed to operate on land surfaces. They can be used for military purposes such as reconnaissance and surveillance, as well as for civilian applications like delivery of goods, underwater exploration, and search and rescue.

2. UUV (Unmanned Underwater Vehicles): Unmanned underwater vehicles designed to operate in the depths of the ocean. They can be used for ocean exploration, inspection of underwater infrastructure, scientific research, and maritime security operations.

3. USV (Unmanned Surface Vehicles): Unmanned surface vehicles designed to operate on surface waters. They can be used to patrol and monitor coastal waters, conduct search and rescue operations, and support maritime security operations.

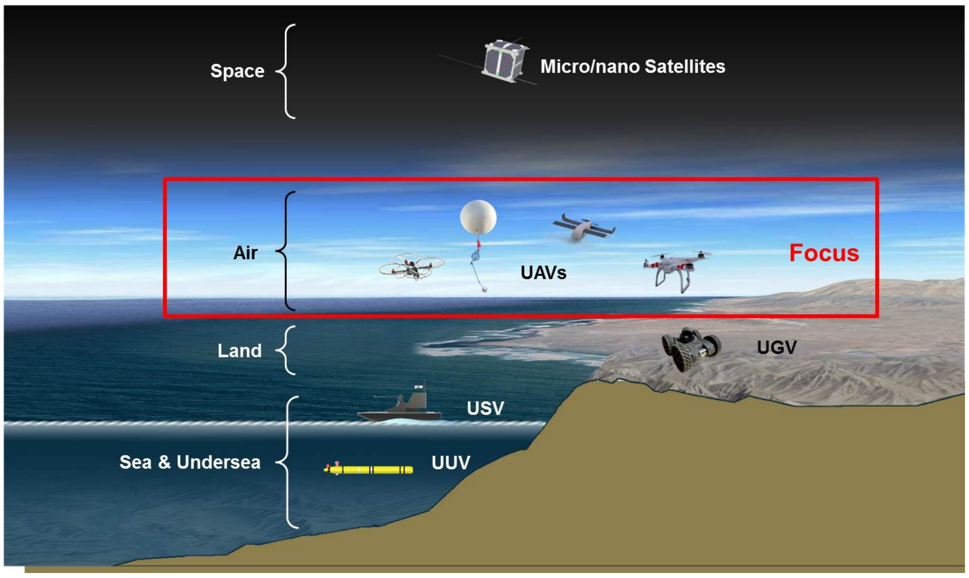


Figure 1.9: Small Autonomous System Domains

but in this case study use only UAV but there would be the possibility of expanding the field to those listed above. The use of unmanned flying devices has ancient roots. The first aircraft concepts date back to the 19th century. Towards the conclusion of World War I in 1918, American engineer Charles Franklin Kettering devised a radio-controlled torpedo, a precursor to contemporary cruise missiles. He named it the ”Kettering Bug”. During World War II, both Axis powers and the Allies developed and utilized unmanned aerial vehicles for military purposes such as reconnaissance and attack. A notable example was the German ”V-1 Flying Bomb”, a self-propelled piloted aerial missile. During the Cold War, the United States and the Soviet Union continued the development of UAVs for military purposes, including reconnaissance and surveillance of enemy areas. UAVs became a crucial element of United States military operations during the wars in Afghanistan and Iraq. They were utilized for surveillance, target detection, and armed attacks. Since the First World War, UAVs have always been fundamental in the military field, but in recent decades they have also begun to spread in the civil field. They have been used for commercial, agricultural, scientific research, environmental monitoring and disaster relief purposes. and even for recreational activities such as aerial photography. The term ”drone” shares a similar origin: it originates from the German word ”drohne”, which means ”male bee”, due to the distinctive buzzing sound it emits during takeoff, reminiscent of a swarm of bees.
There are two categories of UAV 1.10:

- Fixed Wing UAV
- Rotatory Wing UAV

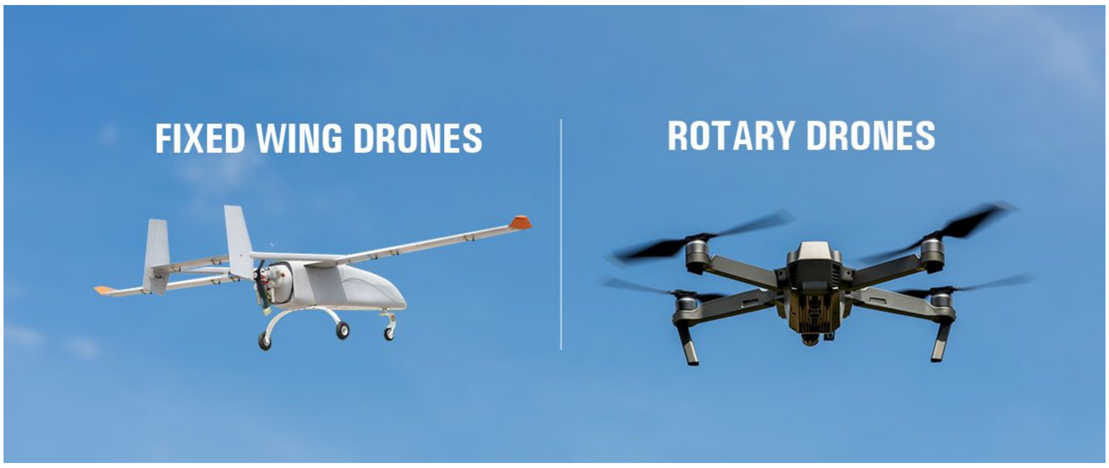


Figure 1.10: Comparison of UAV Types: Rotary-wing UAVs offer vertical and short take-off and landing, hovering, and omnidirectional flight with easy maneuverability. Fixed-wing UAVs, characterized by rigid wings, provide forward lift, have a simpler structure, longer flight range, and lower maintenance and repair costs.

The fixed-wing UAVs have become increasingly popular in military and defense settings due to their high speed and capacity for carrying heavy payloads. However, they are not ideal for stationary tasks or close inspections. Unlike rotary-wing UAVs, fixed-wing models rely on their wings for lift and use energy primarily to move in a specific direction, rather than hovering in one spot. They are designed for long-range missions and can cover extensive areas, staying airborne for up to 16 hours thanks to their gas engines, which offer greater endurance compared to electric engines. Rotary-wing UAVs, including quadcopters[13], hexacopters, tricopters, and helicopters, are specifically designed for aerial monitoring and surveillance. These UAVs are well-suited for tasks such as detecting and tracking border security, monitoring military equipment, and other remote surveillance applications. Unlike fixed-wing UAVs, rotary-wing models have limited speed and payload capacity but can remain stationary in the air, allowing for detailed close inspections. The choice between UAV types depends on their intended use. Over the past two decades, UAVs have become increasingly versatile, with advancements in computing, sensors, communication, and networking technologies. They are now widely used in both military and civilian sectors. For example, in scenarios where communication networks are compromised, UAVs can serve as communication relays, facilitating coordination between rescue teams and enhancing the efficiency of rescue operations.

Chapter 2

Methods and Experimental Setup

In this chapter, we present the pipeline adopted for the experiment, detailing the workflow implemented in both Windows and Ubuntu OSs. A pipeline is crucial in data processing as it enables the structured flow and orchestration of various stages of operations on a continuous data stream. The figure 2.1 illustrates the overall experimental pipeline used to control a drone through a Brain-Computer Interface (BCI), integrating several software components distributed across two systems: one running Windows 11 and the other Ubuntu 20.04.

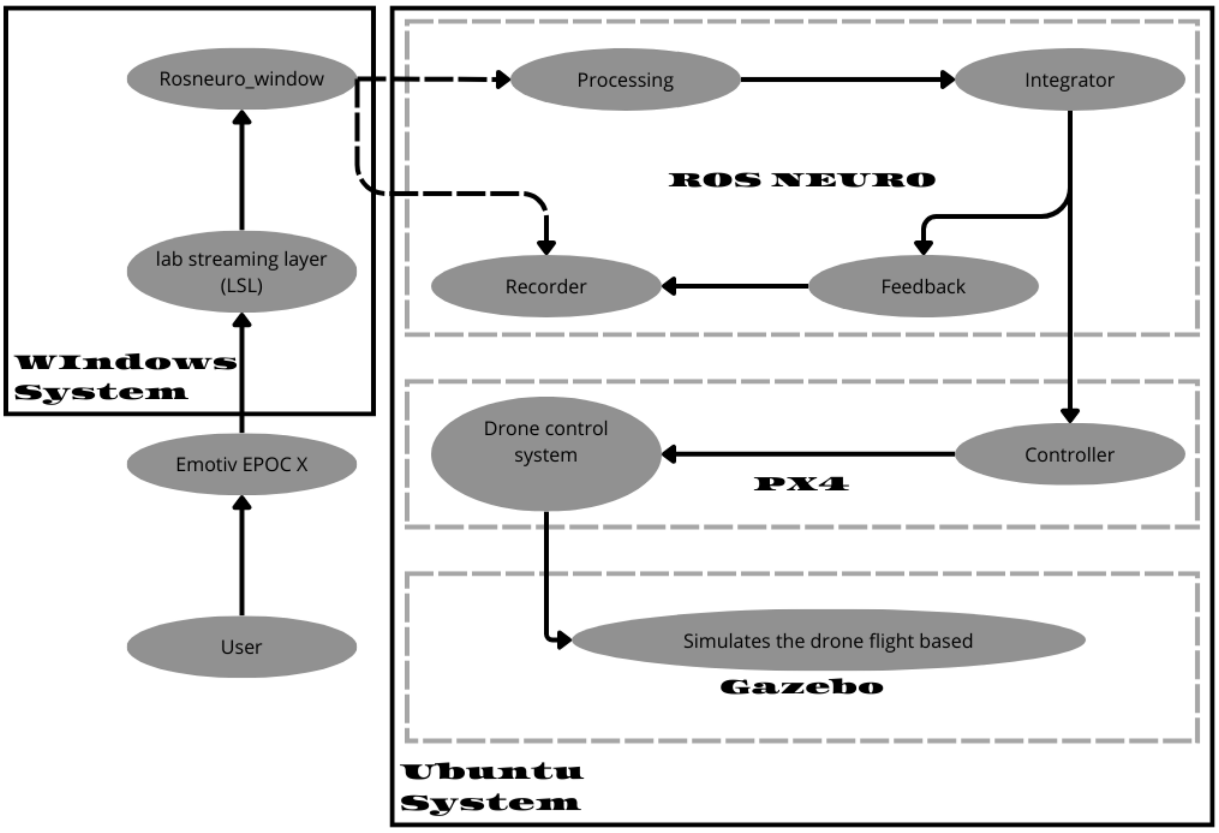


Figure 2.1: Complete experiment pipeline

On Windows System:

- Emotiv EPOC X: This is a wireless EEG headset used to capture the user's brain activity. The headset measures electrical signals from the brain and transmits them to the EmotivPRO software for preprocessing.
- Lab Streaming Layer (LSL): The LSL is a system that allows for the streaming of time series data, such as the EEG signals from Emotiv. In this setup, LSL is used to send the EEG data from the Emotiv EPOC X headset to other components of the system for further processing.
- Rosneuro_window : It is a piece of software designed to receive EEG data via the Lab Streaming Layer (LSL) and send it to the ROS master in the Ubuntu system, where the drone control is handled. It uses messages from the rosneuro_acquisition package, ensuring compatibility with the broader ROSNeuro framework. Additionally, rosneuro_window has been extended to support offline testing. This development allows the insertion of GDF files saved by rosneuro_recorder. By doing so, previously recorded EEG signals can be replayed, enabling tests and validation of the drone simulation without requiring real-time input. This offline capability is especially useful for verifying the pipeline's functionality under different conditions and scenarios.

On ubuntu system is installed The rosneuro suite that is composed by:

- Processing (rosneuro_processing): This module processes the brain signals received from the Windows system. Here, the signals undergo various transformations and filters to extract relevant control information from the user's brain activity. The processed data is prepared for integration with the drone control system.
- Integrator (rosneuro_integrator): The integrator combines the processed brain signals with the overall drone control strategy, ensuring that the user's brain activity is correctly translated into control actions for the drone.
- Recorder (rosneuro_recorder): This module records the processed brain signals and control actions for further analysis and evaluation of the experiment. Recording is crucial for assessing the performance and accuracy of the system over time.
- Feedback (rosneuro_feedback_wheel): Feedback is provided to the user based on the performance of the drone control, enabling adjustments and improvements in signal interpretation. It creates a feedback loop that helps refine the control strategy

PX4 (Ubuntu System)

- Controller: The controller module interfaces with PX4, an open-source flight control stack used to operate drones. It receives the control signals generated from the user's brain activity via the ROSNeuro system and translates these into actionable commands that the PX4 can understand.

- Drone Control System: This system, running within PX4, handles the lower-level control of the drone, ensuring that the commands sent from the BCI through the ROSNeuro pipeline are executed accurately in the simulation environment.

Gazebo (Ubuntu System)

- Simulation Environment: Gazebo is a powerful simulation tool that provides a realistic 3D environment where the drone flight can be simulated. This environment is crucial for testing the control system before applying it to a real drone. The processed control signals are used to simulate drone movements in real-time within this virtual environment.

2.1 The Role of ROS in the Experimental Setup

A crucial component of the experimental setup is the Robot Operating System (ROS), an open-source framework for robot software development that provides the backbone for communication between various systems and applications across both Windows and Ubuntu environments. ROS is required to be installed on both laptops, as it offers the infrastructure needed to enable seamless communication between the two machines through the ROS master node [14]. The ROS master plays a pivotal role in coordinating the interactions between different software nodes and components. It manages the publish/subscribe messaging system, ensuring that data flows smoothly between the applications running on Windows and Ubuntu. By acting as the central framework, ROS ensures that all these components are synchronized and can operate in real time. Each application, whether it be for brain signal processing, control commands, or simulation, relies on the ROS ecosystem to exchange data and commands efficiently. The necessity of installing ROS on both systems is particularly important because the two laptops must communicate through ROS to ensure that the entire pipeline—from signal acquisition with Emotiv EPOC X on Windows, to the final simulation in Gazebo on Ubuntu—works without interruption. The ROS master allows nodes on both machines to interact as if they were part of a single unified system, despite running on different operating systems. In this way, ROS serves as the foundation for the entire experimental pipeline, allowing the real-time control of the drone through brain signals while ensuring compatibility and communication between the various applications and frameworks that are essential to the system's operation.

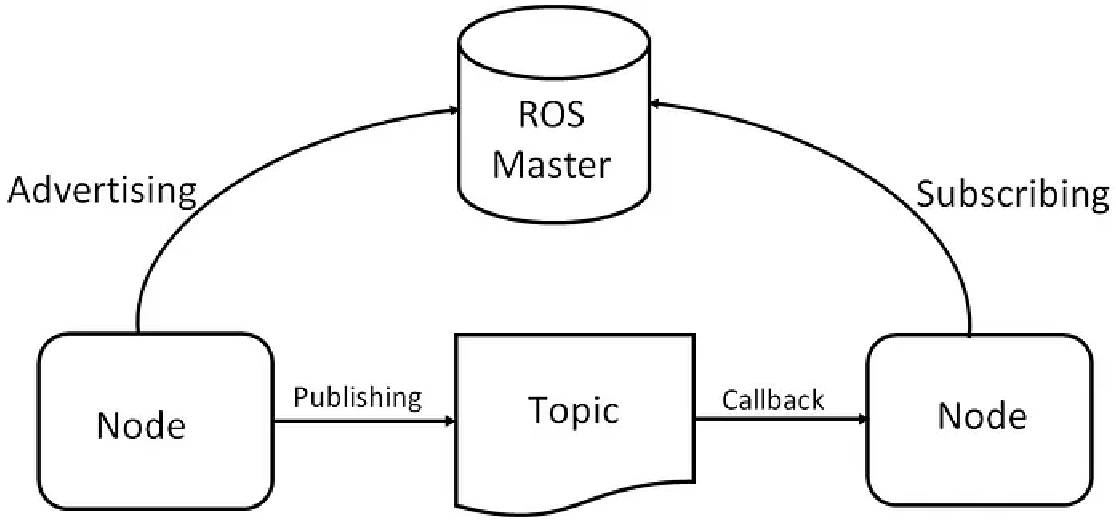


Figure 2.2: ROS Topic Communication Flow: Nodes publish and subscribe to topics via the ROS Master, enabling inter-node communication.

2.2 Hardware and Software Setup

The experiment was conducted using two laptops, each running different operating systems and software environments. The first laptop, an HP with AMD Ryzen 5 5500U processor, integrated graphics, and 8GB of RAM, was running Ubuntu 20.04.6 LTS 64-bit. This machine was responsible for running the ROSNeuro framework, the PX4 autopilot, and the simulation environment Gazebo. The second laptop, running Windows, was used to interface with the Emotiv EPOC X headset via the EmotivPRO application. This application captures and processes the brain signals, which are then forwarded to the ROSNeuro Windows framework, ensuring the continuous transmission of data between the user and the drone control system

2.3 User

The experiment was conducted on a single subject (26 years old, male) who had no prior experience in Brain-Computer Interface (BCI) technologies and exhibited no neurological pathologies. The subject was referred to as ”g1” for data acquisition and recording purposes. A total of 9 recording sessions were carried out, with each test session consisting of 3 consecutive data acquisitions and recordings. During each session, the subject was instructed to perform mental tasks that could influence the EEG signals.

2.4 Experimental Environment

The experiments were conducted in two different environments—noisy and quiet—to assess the robustness of the pipeline and the system's ability to handle external disturbances that could affect the brain signal acquisition. These conditions are critical in testing the reliability of the Emotiv EPOC X in real-world applications, where external noise and environmental factors may interfere with signal accuracy

2.5 Emotiv EPOC X

This experiment utilized a non-invasive EEG approach by using the Emotiv EPOC X headset, which collects neural data in real-time (as discussed in Section 1.2.1). The EEG headset consists of 14 electrodes positioned according to the 10-20 system. The following table presents the key technical specifications of the device: The headset operates with a sampling frequency of

Number of Channels	14 (plus CMS/DRL references at P3/P4 locations M1/M2)
10-20 positions	AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4
Sampling Method	Sequential sampling. Single ADC
Sampling Rate	128 SPS / 256 SPS (2048 Hz internal)
Bandwidth	0.2 - 45Hz, digital notch filters at 50Hz and 60Hz
Filtering	Built-in digital 5th order Sinc filter
EEG Resolution	14 bits 1 LSB = 0.51μV
Connectivity	Proprietary 2.4GHz wireless, BLE and USB (Extender only)

Table 2.1: Technical specifications

256 Hz, providing sufficient temporal resolution for capturing neural signals in real-time. To enhance the signal quality, an electrolyte solution was periodically applied to the electrodes to maintain low electrode-skin impedance, which is critical for ensuring a high signal-to-noise ratio (SNR) during acquisition. The digital filters embedded in the device help eliminate power line interference (50Hz and 60Hz) and suppress other low-frequency noise components. The Emotiv EPOC X communicates wirelessly through a proprietary 2.4 GHz protocol with the receiver dongle, which connects to the host computer. The raw EEG data is streamed via the Lab Streaming Layer (LSL), which enables real-time synchronization and data transmission to other systems, such as the ROSNeuro framework in Ubuntu.

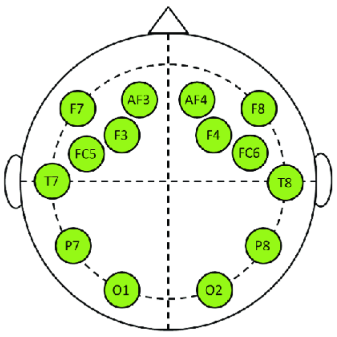


Figure 2.3: Emotiv EPOC X headset with 10-20 system

2.6 EMOTIV

EMOTIV is a leading company in the neurotechnology sector, specializing in the development of electroencephalography (EEG) headsets and brain-monitoring software. The EPOC X is one of EMOTIV's flagship EEG devices, designed for both research and consumer markets. It provides high-fidelity EEG data collection with advanced features suitable for various applications, including brain-computer interfaces (BCIs). Using the EPOC X for BCI Applications. Utilizing the EPOC X in BCI applications follows a structured workflow:

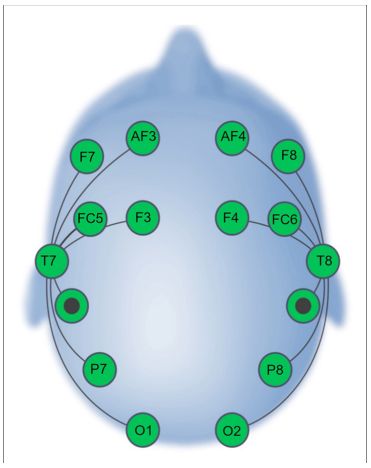
- Preparation: Ensure that the headset is properly fitted to the user's head for optimal signal acquisition. It's essential to check electrode conductivity and install the necessary drivers and tools provided by EMOTIV.

- Data collection: The EPOC X, along with its associated software, is used to capture EEG signals from the user while they perform tasks or mental activities relevant to the BCI application. This stage may involve recording baseline EEG data, as well as capturing event-related potentials (ERPs) or other neural responses.

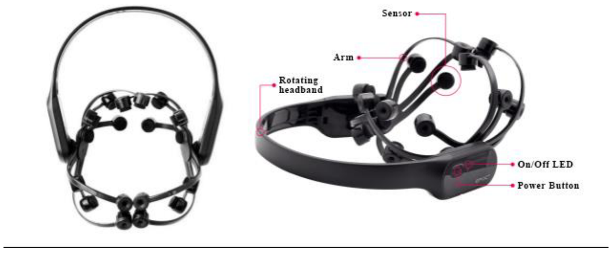
- Signal processing: The raw EEG data is then processed using various signal processing techniques. This typically includes filtering to reduce noise, artifact removal (e.g., blink or muscle noise), feature extraction, and classification. These steps aim to isolate relevant features from the EEG signals that can indicate the user's cognitive state or intentions.

- BCI implementation: After processing the EEG signals, a BCI system is developed or configured to interpret these signals and convert them into control commands for external devices. This might involve controlling virtual objects, performing tasks such as typing, or interacting with user interfaces using brain signals.

- Evaluation and refinement: The performance of the BCI system is evaluated through user testing and validation experiments. Based on the results, the system is refined iteratively to improve accuracy, reliability, and overall user experience.



(a) Position of electrodes on Emotiv EPOC X



(b) Emotiv EPOC X

Figure 2.4: Emotiv EPOC X and electrode placement

2.7 Lab Streaming Layer (LSL)

Lab Streaming Layer (LSL) is a protocol designed to facilitate the collection of time-series data across multiple machines and applications in a synchronized manner. It plays a crucial role in neuroscience research, where sub-millisecond timing precision is required, especially for real-time applications like BCI. In this pipeline, LSL enables efficient, bi-directional communication between EmotivPRO and other third-party software. In our BCI pipeline, LSL is responsible for streaming EEG data captured by the EPOC X to subsequent processing layers in real time, enabling synchronization between brain signals and external events. The steps to configure LSL via the EMOTIV Launcher are as follows:

- Stream name: Set the stream name for transmission. Typically, this starts with the prefix "EmotivDataStream-" followed by the specific data type such as EEG, Motion, Performance Metrics, or Band Power.

- Data stream: Select the type of data to transmit, which can include EEG signals, performance metrics, or motion data. Each type of data generates a separate LSL stream.

- Data format: Currently, two data formats are supported: 'cf_float32' and 'cf_double64'. However, when using 'cf_float32', the timestamp accuracy might be affected due to the value range constraints, as noted in the LSL documentation.

- Transmission type: You can select between 'Sample' or 'Chunk' transmission modes. In 'Chunk' mode, you can further specify the chunk size, ranging from 4 to 256.

Once markers are sent via LSL, they become visible in the Inlet page, and after clicking on ”connect,” markers can be seen being added to the data stream.

2.8 Windows System

While ROSNeuro and other key frameworks are natively supported in Ubuntu, the use of Windows was essential due to the limitations of the Emotiv SDK, which does not provide Linux support for streaming raw EEG data. The Emotiv application for Windows, which includes Lab Streaming Layer (LSL) support, was used to transmit the EEG data to Ubuntu. This decision avoided the complexity of using alternative methods, such as reverse-engineering the USB dongle protocol. Initially, an attempt was made to utilize the Emokit library, which interfaces with the USB dongle and decrypts the proprietary EEG data stream. This required significant effort to adapt the code to Python 3, and a custom 128-byte key was generated from the dongle’s serial number to decrypt the data packets. The Emotiv EEG uses a USB Human Interface Device (HID) protocol, sending 32-byte reports at 128 Hz. However, these solutions presented challenges in terms of stability and performance, leading to the final decision to use the Windows system for EEG acquisition and streaming. The Windows system thus acted as an intermediary, collecting EEG data from the Emotiv device and transmitting it via LSL to the Ubuntu environment, where the ROSNeuro framework would process and interpret the signals. The use of LSL ensured seamless data transfer across platforms, enabling time-stamped and synchronized data streams.

2.9 Ubuntu System

Once the raw EEG data was transmitted to Ubuntu, it was processed using the ROSNeuro framework. As illustrated in Figure 2.1, the pipeline in the ”Ubuntu System” section includes multiple stages. The first stage involves the ”Processing” component of ROSNeuro, which filters and processes the incoming EEG signals. Unlike the standard ROSNeuro workflow, which typically begins with data acquisition, this experiment bypassed that stage due to the unique acquisition process in Windows. Several modifications were made to ROSNeuro to ensure compatibility with the Emotiv headset. Signal processing techniques such as band-pass filtering, artifact removal (e.g., from eye blinks), and feature extraction were applied to enhance the relevant neural patterns. Following processing, the signals were classified and translated into commands for the drone control system in the PX4 framework. PX4 is an open-source flight control stack widely used for Unmanned Aerial Vehicles (UAVs). The processed EEG signals were mapped to specific control commands, such as thrust, pitch, and yaw adjustments. Gazebo

Classic, a 3D simulation environment, was used to simulate the drone's flight dynamics based on the control commands generated from the EEG data. Gazebo provides a realistic physics engine, allowing for the evaluation of the system's performance in scenarios involving object avoidance, stability control, and navigation. This simulation framework is especially useful for testing the robustness and accuracy of the BCI-driven control system before applying it to real-world UAVs.

2.10 ROS Neuro

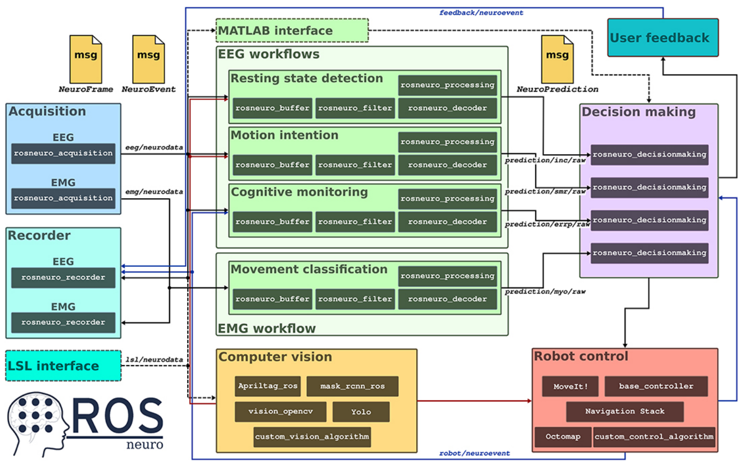


Figure 2.5: ROSNeuro Pipeline

ROSNeuro[15] is a framework that facilitates the integration between neural interfaces and robotic systems. It extends the ROS (Robot Operating System), which is widely used in the robotics community for modular and flexible control system design. ROSNeuro adds specific modules for processing neural signals and translating them into control commands for robots or other systems. The pre-processing pipeline (Figure 2.7b) implemented in ROSNeuro is composed of two Butterworth filters and a Common Average Reference (CAR) filter. These filters are used to remove high- and low-frequency noise, as well as common noise, from the EEG data. The Butterworth filter is a type of signal processing filter characterized by a maximally flat frequency response in the passband. This property makes it ideal for preserving the integrity of the signal while removing unwanted noise. It is particularly useful in EEG data analysis where minimal distortion is critical. The Common Average Reference (CAR) technique is a method used to mitigate noise common across all EEG channels. By subtracting the average signal from all electrodes at each time point, CAR reduces the impact of environmental noise, resulting in a cleaner signal. The steps involved are:

- Calculation of the Common Average: The average EEG signal across all channels is computed at each time point, representing common noise and interference.
- Subtraction from individual electrodes: This common average is subtracted from each electrode's signal, leaving behind brain-specific activity and filtering out the shared noise.

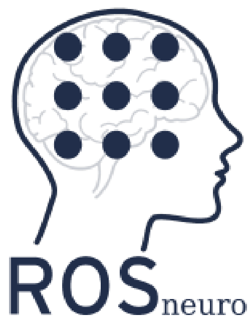
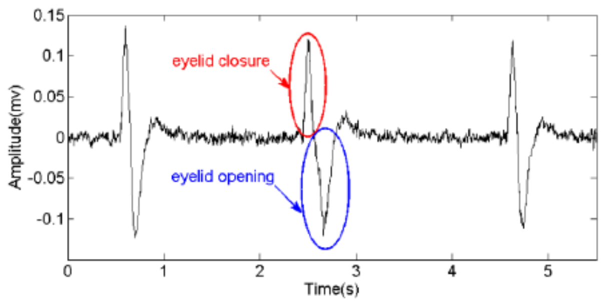
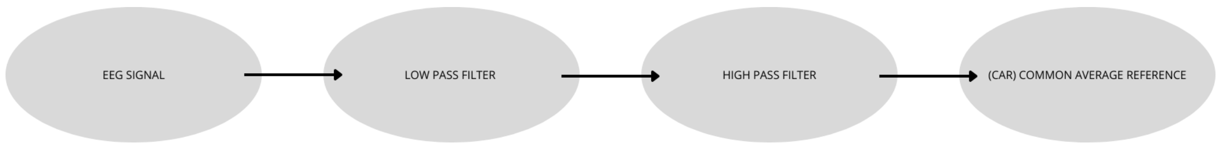


Figure 2.6: ROSNeuro Framework

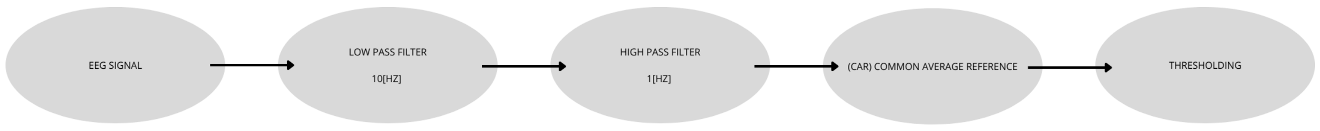
Eye blinks are a common artifact in EEG recordings2.7a, typically appearing as large spikes in the signal, especially in the frontal electrodes. One straightforward approach to handling blink artifacts is thresholding2.7c, where the system detects and removes any data points exceeding a predefined amplitude.



(a) Eye Blinking



(b) Pre-processing pipeline: two Butterworth filters (Low Pass Filter and High Pass Filter) and a Common Average Reference (CAR)



(c) Pre-processing pipeline with eye blinking removal: two Butterworth filters, CAR, and a threshold

Figure 2.7: Comparison of pre-processing pipelines with and without eye blinking removal

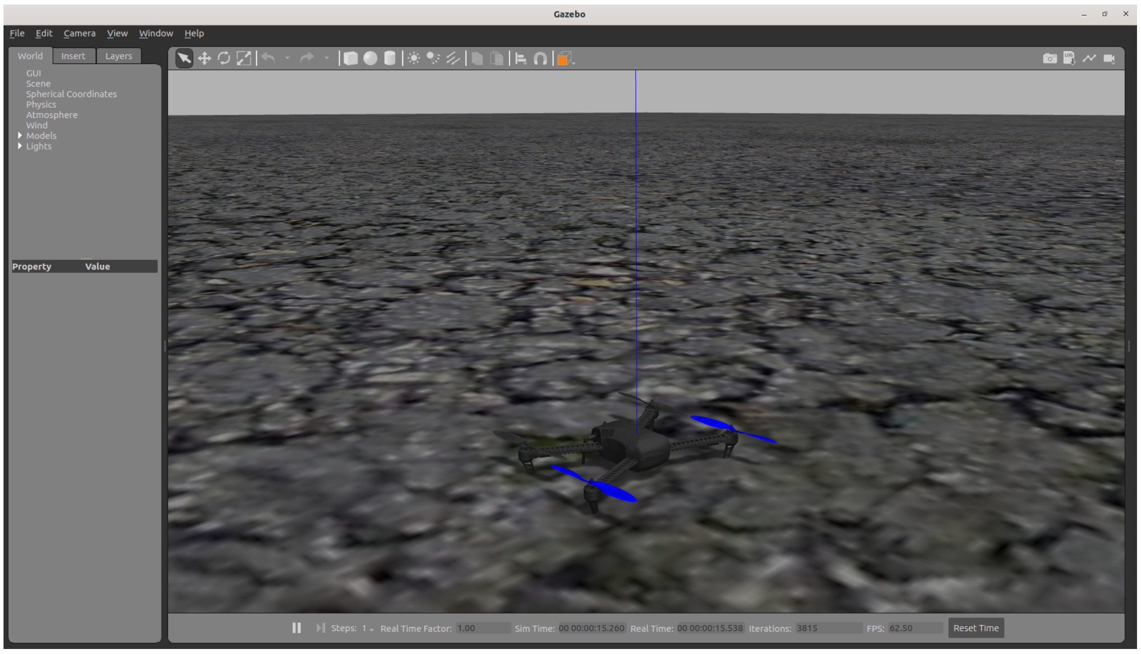
2.11 PX4

PX4 Autopilot is an open-source flight control software designed for drones and other unmanned aerial vehicles (UAVs). It offers a highly customizable platform for managing all aspects of UAV flight, including stabilization, navigation, and mission planning. One of the major advantages of PX4 is its flexibility, making it well-suited for integration with external control systems such as BCIs. In this case study, the drone's autonomous takeoff is initiated independently of the BCI, with a predefined altitude of 1 meters and a fixed starting position. After takeoff, the BCI is used to send commands to the drone, allowing the user to control lateral movements such as turning left or right based on their brain signals.

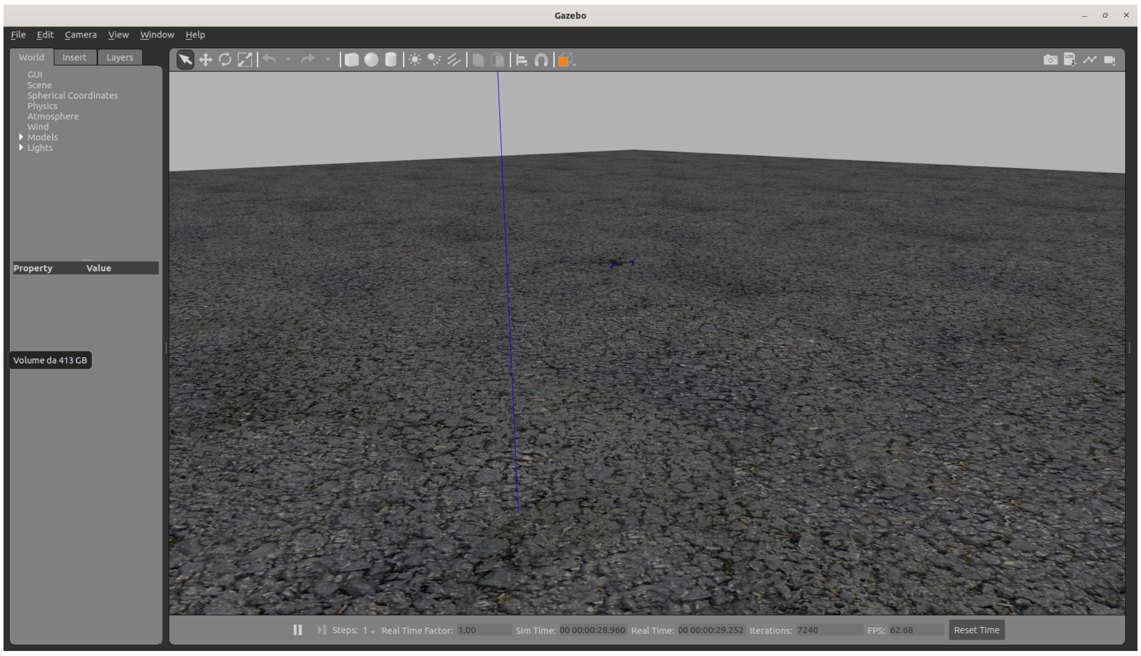
Figure 2.8: PX4 Autopilot Software

2.12 Gazebo

Gazebo is an advanced robotics simulator that provides a realistic environment for testing and experimenting with robots and autonomous vehicles, including drones. Integrated with ROS (Robot Operating System), Gazebo offers tools for simulating complex scenarios, managing physical dynamics, and visualizing robot behavior in real-time. In conjunction with PX4 for drone simulation enables the testing and development of flight algorithms and strategies without the need for physical hardware, saving time and resources.


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:15.260 Real Time: 00 00:00:15.538 Iterations: 3815 FPS: 62.50 Reset Time

(a) IRIS quadcopter in a stationary state on the ground.


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Volume da 413 GB
Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:28.960 Real Time: 00 00:00:29.252 Iterations: 7240 FPS: 62.68 Reset Time

(b) IRIS during takeoff and in an offboard mode, indicating control being transferred from manual input to an external system

(c) Logo of Gazebo Software, a popular robotics simulation tool.

Figure 2.9: The images represent different stages of simulating a quadcopter in the Gazebo environment

Chapter 3

Inside of ROSNeuro

The pipeline begins with the acquisition node3.2. This node was present inside the RosNeuro framework [15], during the project it was noticed that it could not be used because Emotiv company did not provide LabStreaming Layer (LSL) for the Ubuntu operating system, so i created a topic that had the rosneuro_msg messages that communicated with rosneuro_acquisition in such a way as to have the same structure in output but with the possibility of transmitting my raw_data from a Windows device to Ubuntu device, the code is written in pseudo-code3.1 and the ROS Node Configuration for rosneuro_windows3.3. Data transmission occurs via wifi connection, via lan there was no possibility as the laptop where Windows is installed does not have one, therefore there was no possibility of testing any network latency between the two methods. These signals, representing brain activity, are then recorded by the recorder node, which saves them for further analysis. Next, the recorded EEG data is processed in the smrbci node, which I will cover at the end of the Training_Wheel . Here, sophisticated Bayesian algorithms interpret brain signals, focusing on patterns associated with motor imagery. Bayesian approaches are particularly effective in this context because they integrate prior knowledge about the user's neural activity and continuously update the probability of different mental states based on incoming data. In motor imagery, the brain generates patterns similar to those produced during actual movement. It models capture these patterns by calculating the likelihood that specific EEG signals correspond to imagined movements, such as moving a hand or leg. This probabilistic framework allows the system to adapt and improve over time, offering robust and accurate interpretation of motor imagery, even in the presence of noise or variability in brain signals[16]. These processed neural predictions are then integrated in the integrator node. In the testing phase, the integrated neural predictions are fed into the controlwheel node, which drives the visual interface representing the wheel. Users are presented with a visual cue at the center of the wheel, indicating whether they should imagine the action associated with rightward or leftward movement. Based on the user's mental response to this cue, the Control_Wheel node translates the neural predictions into actual movement commands for the wheel. For instance, if the user imagines the volleyball bump, the wheel moves right; if they imagine standing up to block, the wheel moves left. Throughout both phases, leveraging the power of ROS and ROSMaster the pipeline ensures seamless communication between the user's brain signals and the movement of the control wheel, enabling intuitive and responsive interaction in real-time. Now we will define more specifically which commands will be used. The launch file defines three ROS nodes: "acquisition", "recorder", and "Training_Wheel". These nodes are configured to acquire data, record data, and provide visual feedback or other feedback based on certain thresholds. Now we will define them more specifically in order.

```
PROGRAM ROSNEURO_WINDOWS

DEFINE global variables:
neuroseq = 0
frameRate = 256
samplerate = 256
num_samples_per_frame = samplerate / frameRate

CLASS acquisition:
METHOD __init__:
Initialize data_buffer

FUNCTION newNeuroFrame(timestamp, data):
UPDATE min and max values
CONFIGURE NeuroFrame
RETURN NeuroFrame

FUNCTION get_acquisition_info(response):
IF data and timestamp available:
SET response with NeuroFrame
ELSE:
SET response to empty frame
RETURN response

FUNCTION main:
INITIALIZE ROS node
CREATE publisher
START service thread

RESOLVE EEG stream
CREATE StreamInlet

WHILE ROS is running:
    PULL sample
    APPEND to data_buffer

    IF enough samples:
    PUBLISH NeuroFrame
    CLEAR data_buffer
    SLEEP

    IF __name__ == '__main__':
    CALL main

```

Listing 3.1: Pseudo code of Rosneuro_Windows

3.1 Acquisition

```
<!-- acquisition arguments -->
      <!--<arg name="plugin" default='rosneuro::EGDDevice'/>-->
      <arg name="plugin" default='rosneuro::LSLDevice'/>
      <arg name="devarg" default='$(env HOME)/feedback/test16channels.
   gdf'/>
      <arg name="framerate"  default='16'/>
      <arg name="samplerate" default='512'/>
<!-- aquisition node -->
   <node name="acquisition" pkg="rosneuro_acquisition" type="acquisition
   " output="screen" >
      <param name="~plugin"     value="$(arg plugin)"/>
      <param name="~devarg"     value="$(arg devarg)"/>
      <param name="~samplerate" value="$(arg samplerate)"/>
      <param name="~framerate"  value="$(arg framerate)"/>
    </node>
```

Listing 3.2: Rosneuro_Acquisition

```
   <!-- aquisition node -->
   <node name="rosneuro_windows" pkg="rosneuro_windows" type="
   rosneuro_windows.py" output="screen" >
   </node>
```

Listing 3.3: Rosneuro_Windows

In the Rosneuro_acquisition node that is responsible for acquiring data from the specified device. The device is specified through the ”plugin” and ”devarg” arguments, which indicate the type of ROS plugin for data acquisition and the specific arguments for that plugin, respectively. The data sampling rate is set through the ”samplerate” argument and the frame rate through the ”framerate” argument. The package acquires data from external devices, in the case study we will use EMOTIV EPOC X, publishing it as a NeuroFrame message. Each message that’s published represents a set of data points organized as samples x channels. This sets a pace for how frequently the data is sent to other modules. For instance, if the device collects data at 512 Hz (sampling rate) and the system is configured to send updates at a rate of 16 Hz, each NeuroFrame will consist of 32 data points. Emotiv Epoc X has a samplerate and framerate of 256Hz, so from how it was called in the pseudo-code3.1 it will mean that i will have 1 data point

3.2 Recorder

```
<!-- recorder node -->
   <node name="recorder" pkg="rosneuro_recorder" type="recorder" >
      <param name="~autostart" value="$(arg autostart)"/>
      <param name="~filepath" value="$(arg filepath)"/>
   </node>
```

Listing 3.4: ROSNeuro_Recorder

The Rosneuro_Acquisition and Rosneuro_Windows are responsible for recording the acquired data to disk. The ”autostart” argument controls whether recording should start automatically upon node startup. The path to the recorded files is specified through the ”filepath” argument. The package offers a node to read the incoming NeuroFrame and NeuroEvent messages and save them in a GDF file. The package relies on the library libxdffleio [17] that is available in the NeuroDebian reposity. The node waits for the first NeuroFrame message to setup the data structure and to start the recording, if we want to launch RosNeuro_recorder use :

```
rosrun rosneuro_recorder recorder _filename:=[FILENAME] _filepath:=[
   FILEPATH]
```

Listing 3.5: Run only ROS_Recorder out ROSNeuro

the default filename is set in the format :
$SUBJECT.DATE.TIME.MODALITY.TASK.EXTRA.gdf$.
the path of the default where the file is saved : $\$HOME$

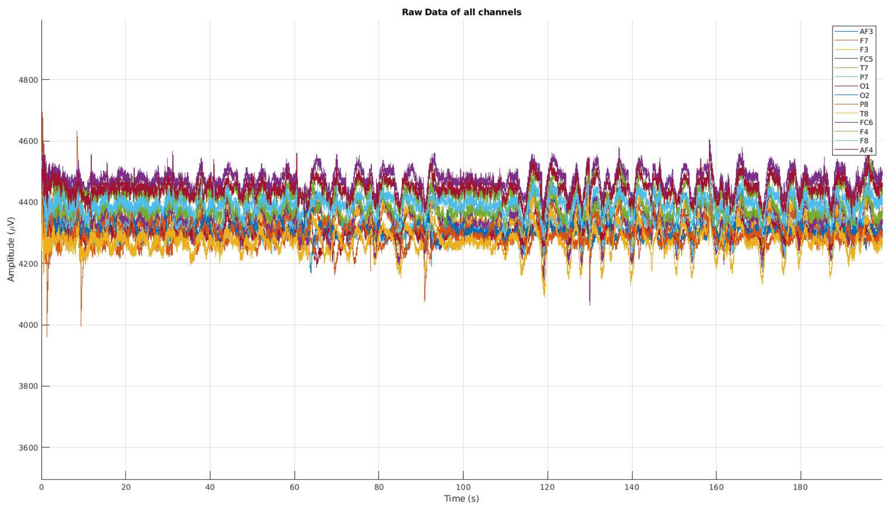


Figure 3.1: RAW DATA with DC offset

Remove the DC offset from EEG signals collected by the Emotiv Epoc X headset. It is an essential preprocessing step because this constant component (DC) in the signal can distort the true EEG data, which might interfere with accurate analysis, especially in Brain-Computer Interface (BCI) tasks

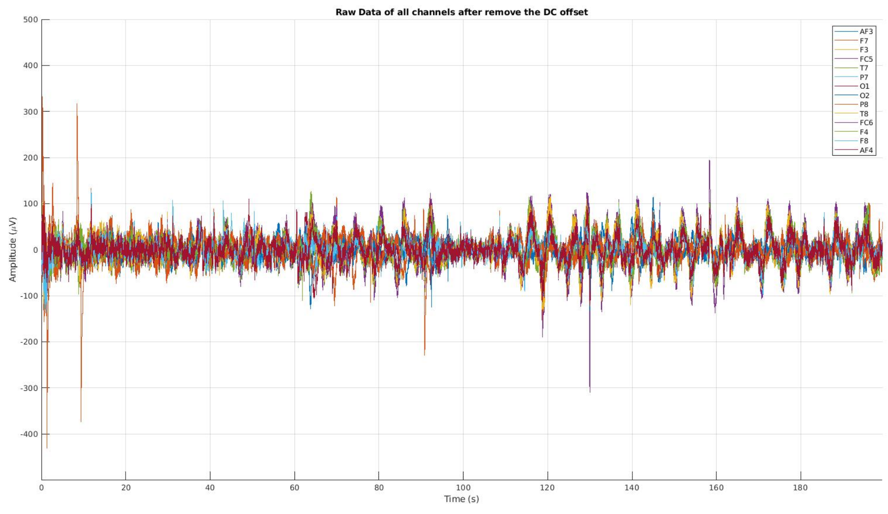


Figure 3.2: RAW DATA after remove the DC offset, using an IIR filter and subtracts it from the raw EEG data. An Infinite Impulse Response (IIR) filter is used here to dynamically track the baseline of the signal. It computes a running average of the signal, which is then subtracted to remove the slow, low-frequency components (like the DC offset)

3.3 Training Wheel

```
<!-- neurowheel node -->
  <node name="trainingwheel" pkg="rosneuro_feedback_wheel" type="
  trainingwheel"  output="screen">
     <rosparam param="modality" subst_value="True">$(arg modality)</
  rosparam>
     <rosparam param="thresholds" subst_value="True">$(arg thresholds)<
  /rosparam>
     <rosparam param="classes" subst_value="True">$(arg classes)</
  rosparam>
     <rosparam param="trials" subst_value="True">$(arg trials)</
  rosparam>
     <rosparam param="show_on_rest" subst_value="True">$(arg
  show_on_rest)</rosparam>
  </node>
```

Listing 3.6: Node of training wheel of calibration (offline) and evaluation(online) wheel

This node 3.6 is responsible for providing feedback based on the acquired data. Parameters such as ”modality”, ”thresholds”, ”classes”, ”trials”, and ”show_on_rest” specify the details of this feedback. This provides visual feedback based on the thresholds of the acquired signals. The runs, comprising offline (calibration) and online (evaluation) phases, involved a total of 20 tasks, which can be adjusted if needed by modifying the setup. These tasks were categorized based on the type of kinesthetic imagination: 10 tasks each for classes [771] ”both-hands” and [773] ”both-feet”, each MI trial lasted from 4.5s to 5.5s. The tasks were presented in random order to minimize the risk of habituation and thereby prevent a decline in feature quality. For the calibration session, the completion time was approximately 5 minutes, while the duration of the evaluation phase varied depending on the classifier’s performance and the subject’s responses. During the experiment, participants were presented with tasks on the screen 3.3, alternating between figures and colors, to indicate the actions to be performed. A red or blue circle would appear, signaling the participant to start imagining continuous movement of both feet or both hands, respectively, until the corresponding progress bar at the top of the screen was fully filled. Before beginning the tasks, participants were instructed to rhythmically move their hands or feet at a gradually slowing pace until the movement ceased entirely, leaving only the sensation of muscle contraction and relaxation. This preliminary exercise was crucial, as the experiment required participants with no prior experience in Brain-Machine Interface (BMI) systems, helping them to better prepare for the tasks ahead. To minimize movement and ocular artifacts, participants were asked not to blink or move their heads while filling the bars. Additionally, before each task began, a fixation cross would appear at the center of the screen for 1 second, signaling the imminent start of the task and providing a focal point to enhance concentration and reduce distractions from external movements in the room, which was kept as quiet as possible. In the offline runs, the bars would automatically fill in accordance with the task cue. In contrast, during the online runs, participants controlled the bars themselves, which could rise at varying speeds, fluctuate, or not move at all, depending on the output from the BMI decoder, regardless of the participant's effort. To maintain a high level of concentration, a relaxation period was provided between tasks, allowing participants to prepare for the next cue. Since the tasks were presented randomly, limb relaxation was mandatory to prevent muscle contractions from one area from interfering with data collection for the other. In case of fatigue, participants were allowed to take short breaks between runs, particularly to rest their eyes. However, they were instructed not to touch or remove the EEG headset due to the presence of conductive gel; any contact between different electrode areas could cause a short circuit, resulting in inaccurate signals. To optimize the evaluation process, two threshold parameters—one for the ”both-hands” task and another for the ”both-feet” task—could be adjusted as needed. If a participant had difficulty raising the bars, the thresholds could be lowered and tested in the next run. Conversely, if a bar rose too quickly without being adequately controlled, the threshold parameter could be increased. Adjusting these values was a highly individualized process, requiring careful consideration on a case-by-case basis. If the threshold was set too high, the task would take longer to complete, offering greater control to the participant. On the other hand, if the threshold was too low, the bar would rise too quickly, possibly escaping the participant's true intentions.

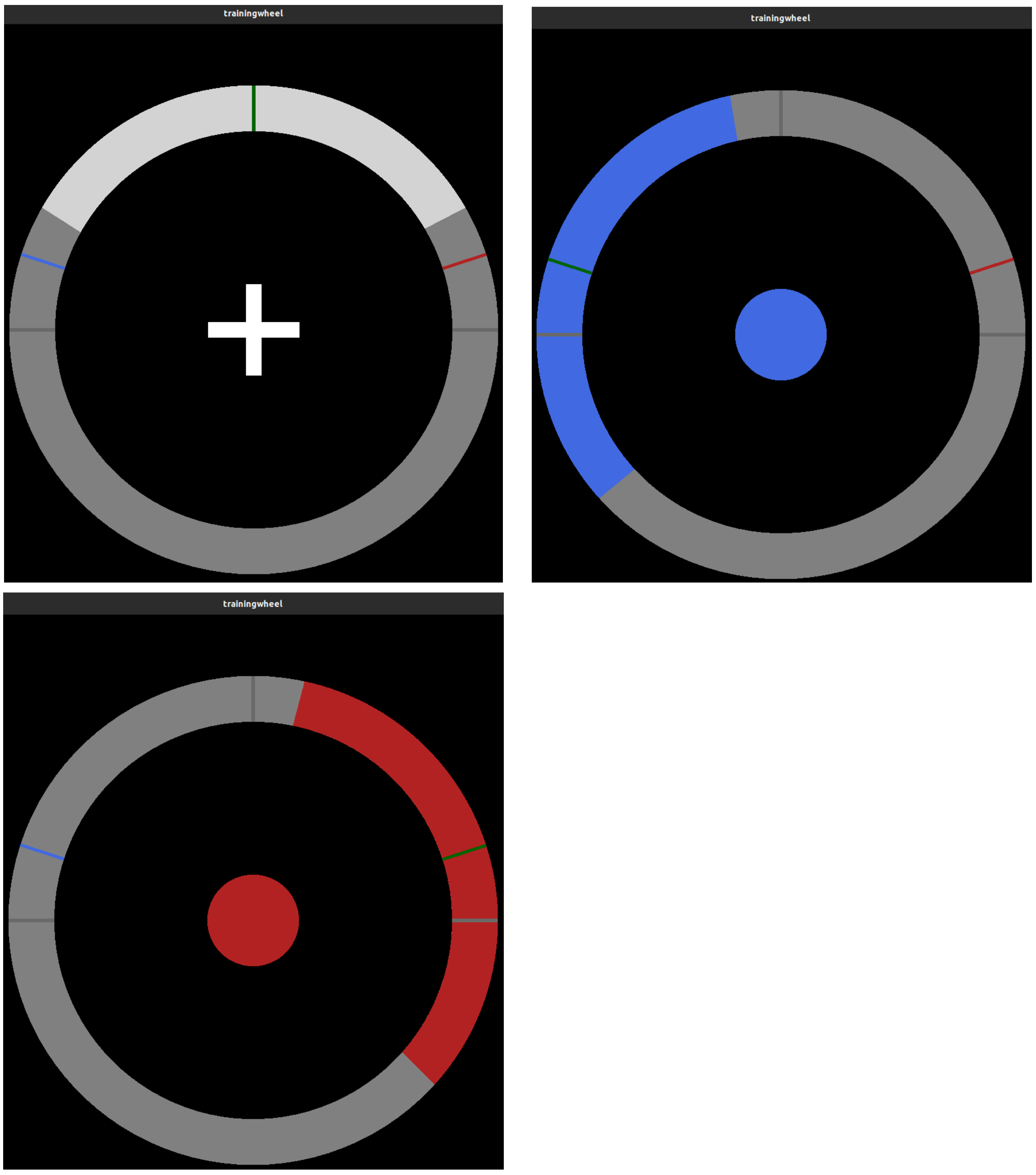


Figure 3.3: A cross appeared at the center of the screen for 1 second to signal the imminent start of the task and enhance concentration. Subsequently, a red or blue circle indicated the participant to begin imagining continuous movement of both feet or both hands, respectively, until the progress bar at the top of the screen was fully filled.

To distinguish between the two BMI classes, it was essential to develop classifiers that could identify key features. To achieve this, attention was directed towards the brain regions involved in the action, specifically the motor-sensory cortex. In this context, a Gaussian classifier was utilized to analyze and differentiate these regions effectively. According to the standard of the helmet used (Emotiv Epoc X), we mainly refer to the fronto-center-parietal electrodes (“AF3”, “F7”, “F3”, “FC5”, “T7”, “P7”, “O1”, “O2”, “P8”, “T8”, “FC6”, “F4”, “F8”, “AF4”), i.e., those in which the differences between the hands are most noticeable (especially the electrode areas FC5 and FC6) as for the feet does not present any relevance but F3 and F4 can be considered even though they are not directly over the primary motor area, they are located in the frontal lobe and can provide data on motor activity. In particular, F4 may be slightly closer to the motoric representation of the legs and feet than F3, although both are more involved in general control of the extremities. Being damped and noisy by their nature, it is necessary to increase the information content of the EEG signals by applying a spatial filter (i.e., a sharpening filter that can increase the signal-to-noise ratio and consequently the level of signal sharpness) and to highlight more the potential field variations along the scalp surface. The Laplacian filter has been created and will be discussed in the following chapter 4.1. Without this specific filter, excessive noise would have remained in the data, negatively impacting the spectral analyses 4.2. This would have rendered the model and the Brain-Machine Interface (BMI) nearly unusable.

Chapter 4

Inside of Experimental Setup

In the following chapter we will discuss the experiments that were completed providing also a more detailed study of the results obtained starting from an original EEG signal up to a transformed one, we will discuss the type of classifier used providing the results obtained starting from the first acquisition session up to the 'last one.

4.1 Laplacian Filter

In the case of this experiment, a Laplacian filter with a 14x14 matrix was created through a Python script 4.1, initially considering only two EEG channels (FC5 and FC6); whereas, literature suggests the use of a larger number of channels for a more accurate four channels were chosen for the Laplacian based on their relevance to motor imagery analysis and their ability to provide an accurate representation of EEG signals. The Laplacian thus created, having been generated through a Python script that saves with a .mat extension, in order to insert it, it was necessary to modify the wc_save_laplacian so that it was saved in .dat. The four channels initially chosen were FC5, FC6, T7 and T8 4.1, where the last two are located in the temporal lobe and are more associated with auditory and cognitive processing, could offer information on how motor and sensory activities are integrated with foot control.

```
PROGRAM CREATE_LAPLACIAN_MATRIX

DEFINE channels and adjacency matrix A

FUNCTION add_edge(i, j):
IF valid indices:
A[i, j] = 1
A[j, i] = 1

FUNCTION main:
```

```
FOR i:
IF general connections:
CALL add_edge(i, i+1)
IF connections at distance 6:
CALL add_edge(i, i+6)

Double connections for FC5, FC6, T7, T8

Compute D = degree matrix
Compute L = D - A

Save L to 'laplacian_4CH_T78.mat'
PRINT "Saved to: 'laplacian_4CH_T78.mat'"

IF __name__ == '__main__':
CALL main()
```

Listing 4.1: Pseudo Code to create a Laplacian Matrix

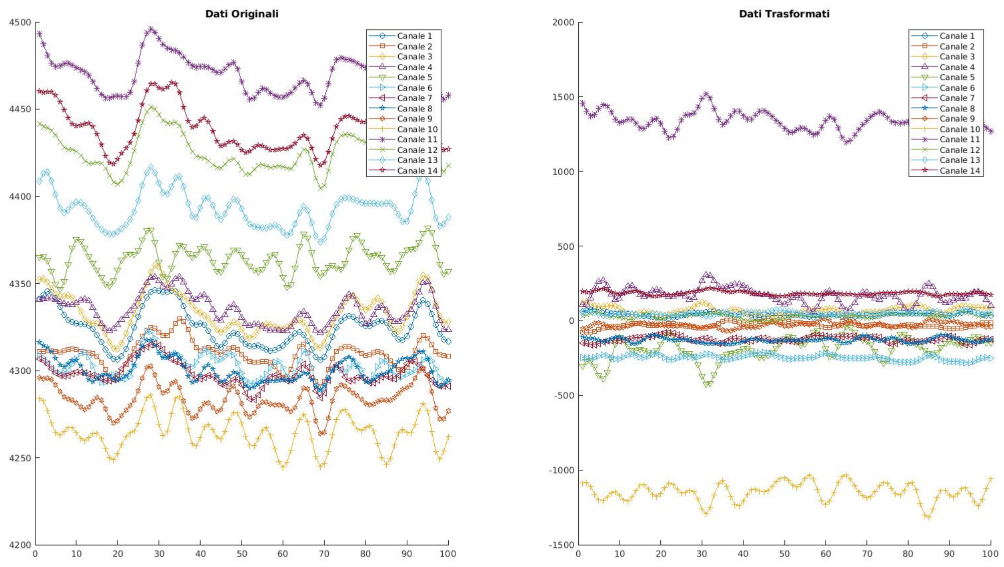


Figure 4.1: Left: Graph of the original EEG data from the 14 channels. These data represent the brain's electrical activity recorded before filtering
Right: Graph of EEG data after applying the Laplacian filter. This filter enhances local features and rapid changes in the signals,

- Original Data : The time series show significant variation among the various channels, with fairly large amplitudes. The differences between the channels are clearly visible and reflect the EEG signals acquired before transformation.

- Transformed Data : After the application of the Laplacian matrix, the signals underwent a noticeable change, the amplitudes of the signals have been reduced, and the values are centered around zero, which is typical for a Laplacian transformation, in that a weighted average is subtracted weighted of the neighboring signals, channels FC5, FC6, T7, and T8, which have been emphasized by doubling their connections in the Laplacian matrix, show signs of greater variation in the transformed data. This was the desired and expected effect. The coherence of the signals is maintained, indicating that the transformation was applied correctly without introducing unwanted artifacts.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Row 4 (FC5)	0	0	-2	10	-4	0	0	0	0	-4	0	0	0	0
Column 4 (FC5)	0	0	-2	10	-4	0	0	0	0	-4	0	0	0	0
Row 11 (FC6)	0	0	0	0	-4	0	0	0	0	-4	10	-2	0	0
Column 11 (FC6)	0	0	0	0	-4	0	0	0	0	-4	10	-2	0	0
Row 6 (T7)	0	0	0	0	-2	3	0	0	0	0	0	-1	0	0
Column 6 (T7)	0	0	0	0	-2	3	0	0	0	0	0	-1	0	0
Row 10 (T8)	0	0	0	-4	0	0	0	0	-2	10	-4	0	0	0
Column 10 (T8)	0	0	0	-4	0	0	0	0	-2	10	-4	0	0	0

Table 4.1: Matrix values for specific rows and columns of FC5,FC6,T7 and T8.

The FC5 and T8 channels have a greater weight on themselves than the other channels, suggesting that the influence of the signal is predominant over neighboring signals. The weights present in the matrix of the Laplacian confirm that the channels were influenced correctly by their neighbors. There is no unwanted doubling. The study on signal energy analysis was applied. The decrease in the energy of the transformed signals is normal and consistent.

Energy of the original signals: 1.0e+12							
Columns 1 through 7	0.9501	0.9425	0.9552	0.9550	0.9659	0.9383	0.9350
Columns 8 through 14	0.9385	0.9303	0.9222	1.0169	0.9933	0.9785	1.0023
Energy of the transformed signals: 1.0e+11							
Columns 1 through 7	0.0025	0.0031	0.0408	0.9320	0.1991	0.0351	0.0113
Columns 8 through 14	0.0144	0.0046	0.8658	1.0261	0.0013	0.0014	0.0210

Table 4.2: Energy of the original and transformed signals with corresponding column values of channel FC5,FC6,T7 and T8.

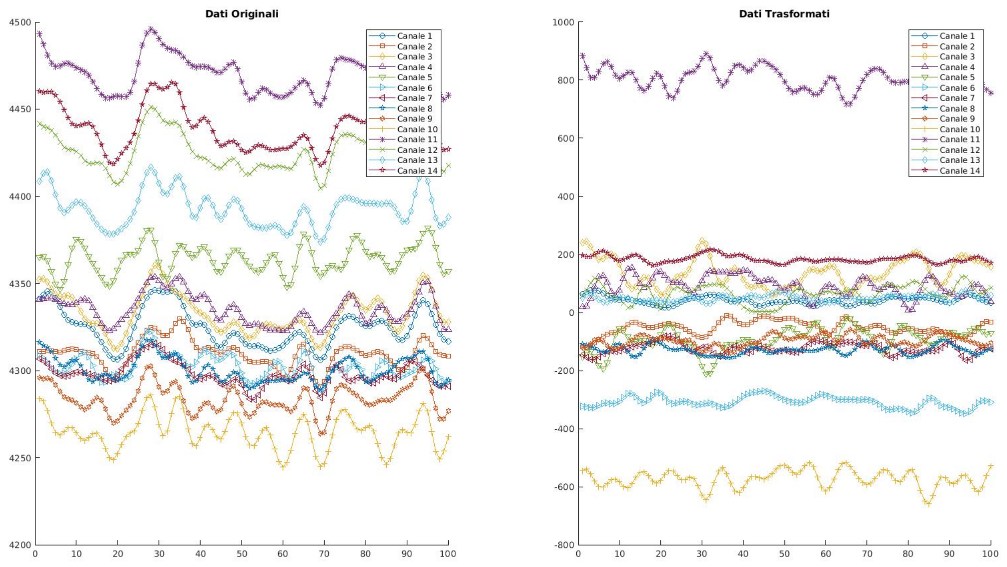


Figure 4.2: Left: Graph of the original EEG data from the 14 channels. These data represent the brain's electrical activity recorded before filtering
Right: Graph of EEG data after applying the Laplacian filter. This filter enhances local features and rapid changes in the signals,

The plot 4.2 clearly shows a visual difference between the original and transformed data. The original signals have higher energy levels, as observed in the range of 4200-4500, while the transformed signals are scaled differently, showing a range from -800 to +1000. Original signal energies: $1{,}0 \times 10^{12}$ range, with values around 0.9 to 1.0 for most channels the transformed signal energies: $1{,}0 \times 10^{10}$ range, but with drastically different energy for channels like the third (F3) and fourth (FC5), which seems to be much higher than the others. The drastic energy shift

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Row 4 (FC5)	0	0	-4	8	-2	0	0	0	0	-2	0	0	0	0
Column 4 (FC5)	0	0	-4	8	-2	0	0	0	0	-2	0	0	0	0
Row 11 (FC6)	0	0	0	0	-2	0	0	0	0	-2	8	-4	0	0
Column 11 (Fc6)	0	0	0	0	-2	0	0	0	0	-2	8	-4	0	0
Row 3 (F3)	0	-2	8	-4	0	0	0	0	-2	0	0	0	0	0
Column 3 (F3)	0	-2	8	-4	0	0	0	0	-2	0	0	0	0	0
Row 3 (F4)	0	0	0	0	-2	0	0	0	0	0	-4	6	0	0
Column 3 (F4)	0	0	0	0	-2	0	0	0	0	0	-4	6	0	0

Table 4.3: Matrix values for specific rows and columns of FC5,FC6,F3 and F4.

Energy of the original signals: 1.0e+12							
Columns 1 through 7	0.9501	0.9425	0.9552	0.9550	0.9659	0.9383	0.9350
Columns 8 through 14	0.9385	0.9303	0.9222	1.0169	0.9933	0.9785	1.0023
Energy of the transformed signals: 1.0e+10							
Columns 1 through 7	0.0251	0.0599	1.6337	5.8761	0.4977	0.5245	0.0113
Columns 8 through 14	0.1440	0.1000	2.1646	3.8310	0.0515	0.0145	0.2103

Table 4.4: Energy of the original and transformed signals with corresponding column values of channel FC5,FC6,F3 and F4.

between original and transformed signals can indicate the removal of noise, highlighting certain frequency bands, or isolating key components in the EEG signals. Channels like FC5 and FC6 exhibit significantly higher energy in the transformed state, suggesting that these components were emphasized by the transformation. comparing the two plots 4.1 and 4.2 the original data are the same in both plots, the same gdf file was considered; the original signals the channels show oscillations in the range between approximately 4250 to 4500. The amplitude of the fluctuations is significant for all four channels. if we taking into consideration the first plot 4.1 shows that transformed signals, the amplitude is compressed, with values ranging between

-600 to +800. in the other plot 4.2 the range is a bit larger, with oscillations between -1000 and +1500 for some channels, especially for T7 and T8. This suggests that these temporal channels are more impacted by the transformation, leading to greater fluctuations in the transformed data compared to the frontal channels. The channels F3 and F4 experience a high transformation impact, likely due to their high original energy levels, which shows a smaller energy reduction compared to the first plot. T7 and T8 still retain some stronger signal characteristics after transformation. The transformed signals F3 and F4 (frontal channels)are highly compressed, indicating significant noise reduction or signal simplification, the temporal channels (T7 and T8) seem to retain more dynamic activity after the transformation, as seen by their greater amplitude in the transformed graph. This might suggest that the activity of temporal regions (perhaps related to auditory or memory processes) remains more prominent after the transformation. The energy reduction is significant in both cases, but the temporal channels appear to retain slightly more energy and dynamic range after the transformation. Without this specific Laplacian filter,

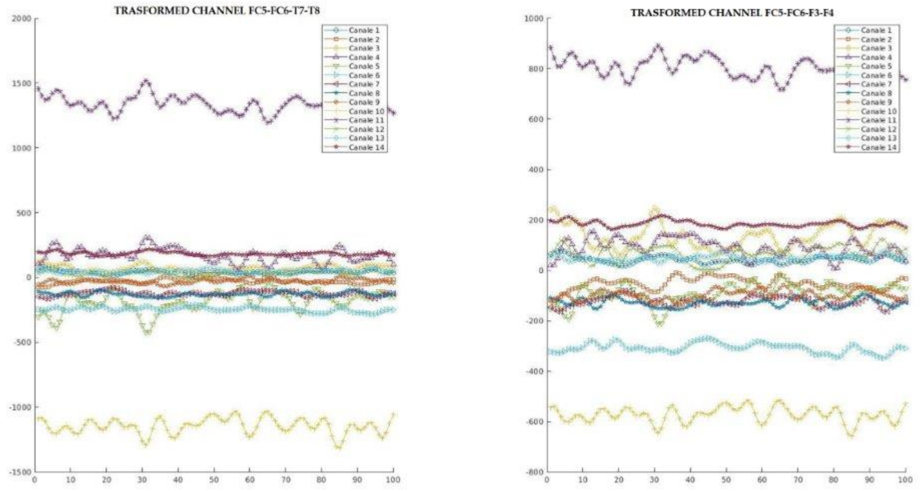


Figure 4.3: Transformed EEG signals are shown in both plots, calculated using a Laplacian mask. The left plot uses electrodes FC5, FC6, T7, and T8, while the right plot shares FC5 and FC6 but replaces T7 and T8 with F3 and F4. Each line represents the transformed data from different channels (Canale 1 to Canale 14) across 100 time points

there would have been excessive noise affecting the spectral analyses. This noise would have compromised the model and rendered the BMI nearly unusable. The impact on the model and BMI will be further discussed in the following sections.

To properly execute evaluation_wheel, which includes the processing node (referenced as 4.2), we need the Laplacian filter (discussed earlier), the classifier, and the specification of the number of channels for the helmet device in use. The Emotiv EPOC X headset has a samplerate and framerate of 256Hz with 1 sample per frame, while the rosneuro_processing framework supports a framerate of 16Hz, a samplerate of 512Hz, and 32 samples per frame. Therefore, the configuration of SmrBci.cpp was modified to reflect this frequency. The Gaussian Classifier, originally designed for 16 EEG channels, had to be adapted to the Emotiv EPOC X headset, which uses 14 channels (”AF3”, ”F7”, ”F3”, ”FC5”, ”T7”, ”P7”, ”O1”, ”O2”, ”P8”, ”T8”, ”FC6”, ”F4”, ”F8”, ”AF4”). Consequently, the eegc3_smr_newsettings was modified, and both eegc3_smr_erds (which calculates ERD/S on epochs extracted from data with respect to baseline) and eegc3_smr_select_up_to_best_16 (which computes the discriminant power) had to be adjusted accordingly.

```
<!-- processing node -->
<node name="smrbci" pkg="rosneuro_processing" type="test_smrbci"
 output="screen">
<param name="~lap_path" value="$(arg lap_path)"/>
<param name="~decoder_path" value="$(arg decoder_path)"/>
<param name="~n_channels" value="$(arg n_channels)"/>
</node>
```

Listing 4.2: processing_node

4.2 Spectral Density

The direct analysis of signals in the time domain does not effectively capture features that optimize the distinction between the two motor imagery (MI) tasks in a Brain-Machine Interface (BMI). As supported by findings in the literature[18], cognitive processes related to motor activity not only elicit responses resembling event-related potentials (ERPs) but also exhibit distinct neural dynamics. Specifically, motor imagery is associated with characteristic changes in brain activity observable through spectral analysis. During motor imagery, a phenomenon known as event-related desynchronization (ERD) occurs, where groups of neurons, particularly in the primary motor and somatosensory cortices, cease to operate in synchronized patterns. This desynchronization results in a measurable decrease in power, most notably in the alpha band (8–12 Hz), as well as other frequency bands like beta (13–30 Hz). The amplitude reduction during ERD reflects decreased neuronal synchronization compared to the resting state, which is indicative of active cognitive processing related to motor tasks. Once the motor imagery task concludes, the neurons return to a more synchronized state in a process called event-related synchronization (ERS). This re-synchronization leads to an increase in signal power, bringing the activity levels back to those observed during rest. The transition between ERD and ERS is a key feature in distinguishing motor imagery from resting or inactive states and serves as a reliable marker for decoding intentions in BMI systems. By analyzing these frequency-specific changes rather than time-domain signals alone, it becomes possible to identify subtle variations in brain activity that correlate with motor imagery tasks. This spectral approach enhances the ability to detect meaningful features, thereby improving the classification and reliability of BMI systems in distinguishing between different motor intentions. While analyzing brain signals in predefined frequency bands, such as alpha or beta, can reveal certain neural patterns, this approach often overlooks important nuances because the exact configuration of these bands can vary significantly between individuals. Motor imagery (MI) tasks in brain-machine interfaces (BMI) are a prime example, where some individuals may modulate only specific sub-bands within the broader alpha or beta ranges, rather than the entire band as described in typical event-related desynchronization (ERD) or event-related synchronization (ERS) phenomena. This variability poses a challenge for real-world BMI applications, where accurate and reliable control depends on capturing the subtle frequency shifts that reflect each user's unique brain activity. Simply aggregating the data across entire frequency bands reduces the spectral precision and can result in a loss of meaningful information, limiting the BMI's ability to effectively decode motor intentions. To address this, researchers have developed methods to increase spectral resolution, similar to how spatial resolution has been enhanced in EEG analysis. By focusing on smaller sub-bands and resolving power changes at more granular frequency levels, it becomes possible to capture more precise spectral signatures associated with motor imagery. This refined approach ensures that the analysis is not confined to broad, generalized bands but instead taps into the multiple, distinct frequency components that each user may be modulating during motor tasks. This spectral refinement is achieved by analyzing the variation in the power spectrum, which describes how the signal's power is distributed across different frequencies over a given period of time. Instead of treating frequency bands as homogeneous, the power spectrum is broken down into discrete frequency components, enabling a more detailed examination of the power fluctuations in shorter time intervals. By dividing the data into time blocks, we can generate a spectrogram that links the temporal evolution of brain activity with its corresponding spectral characteristics. This approach not only preserves the temporal information of the signal but also provides insights into how specific frequencies behave during critical moments of motor imagery. Such enhanced spectral methods are crucial in improving the performance of MI-based BMI systems, allowing for better decoding of brain signals by considering each user's unique spectral fingerprint. This can lead to more responsive and reliable BMI applications, where individualized frequency patterns are leveraged for more accurate control. Since it is practically impossible to construct a perfect spectrum (as an infinite period of time would be required to calculate the fast Fourier transform (FFT)), we aim to create an approximation as close as possible to the true spectrum using spectral averaging methods, such as Welch's method[19]]. This algorithm, used for feature extraction, generates the power spectral density (PSD). The signal is segmented into a finite number of windows, with the length determined by the algorithm's specific requirements. These windows are further divided into overlapping segments over shorter time intervals. For each segment, the spectrum is computed using the FFT for the target frequencies, and the resulting values are averaged, providing a representative spectral value for that portion of the signal. Overlapping the windows is essential to enhance spectral resolution.

$$\mathbf{R}\left\{\frac{\mathbf{1}}{\mathbf{2}\pi}\int_{\mathbf{0}}^{\infty}\mathbf{2F}(\omega)\mathbf{e}^{\mathbf{i}\omega\mathbf{t}}\,\mathbf{d}\omega\right\}$$

At the conclusion of the process, a three-dimensional matrix will be generated, encompassing dimensions for windows, channels, and frequencies. This matrix will encapsulate all the spectral information required to assess the subject's mental state on a per-window basis and to identify the task's start and end periods. The Welch method, used to derive this matrix, offers several advantages over methods such as ERD/ERS. Firstly, it provides a substantially finer spectral resolution. By averaging over multiple segments, the Welch method mitigates the influence of noise. Even if high-power noise persists after initial filtering, averaging across different segments diminishes its effect. Additionally, the Welch method enhances computational efficiency compared to methods like Bartlett's, because it allows for reuse of previous FFT calculations in subsequent analyses. This reuse of FFT results speeds up the spectral computation process, making it a more practical choice for real-time or large-scale spectral analysis. Finally, a base-ten logarithmic transformation is applied to the spectrum to convert it from a linear scale to a logarithmic scale. This transformation simplifies the visualization of the spectrum's trends and aids in model development. Utilizing a PSD-based method not only facilitates a more detailed analysis of sub-band dynamics but also enhances spectral clarity, providing more accurate information across different time periods. This improved precision allows for the identification of specific windows of interest. Once these windows are computed, they are categorized based on the task, transitioning from a three-dimensional matrix to a four-dimensional matrix. This transition enables the detection of differences in the PSDs and serves as a foundation for developing a classifier based on these variations.

4.3 Signal Processing

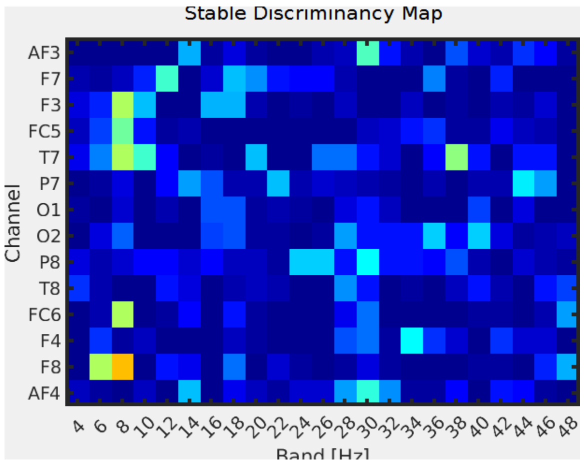
Signal processing is a crucial component in Brain-Computer Interfaces (BCIs), facilitating the transformation of raw neural signals into meaningful commands for control applications. BCIs enable direct communication between the brain and external devices by bypassing conventional neuromuscular pathways. The effectiveness of a BCI system heavily relies on its ability to accurately interpret brain signals, which are often complex and noisy. The following section will discuss the Feature extraction and classification.

4.3.1 Feature Extraction

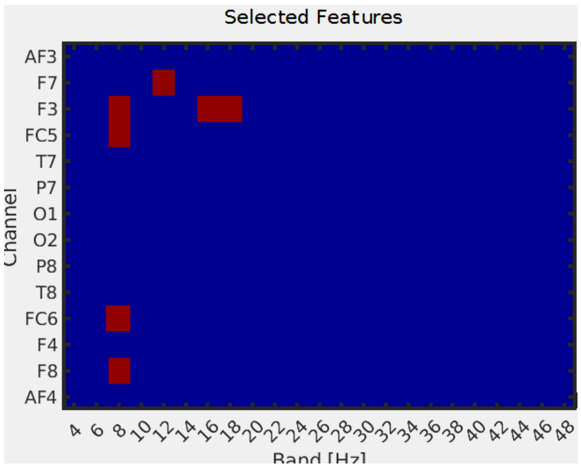
In order to develop an effective model for Brain-Computer Interface (BCI) control, it is essential to identify features that optimally distinguish the execution of one task from another. One of the main goals of this process is to analyze the evolution of neural patterns resulting from neuroplasticity, when training a BMI model, understanding the evolution of neural activity over time is crucial. The brain's plasticity means that neural pathways can strengthen or change as a result of repeated training. By analyzing these changes, it is possible to identify neural correlates that are consistently linked with task execution. These patterns help in refining the feature set to capture meaningful data over time, enabling better task discrimination. or the brain's ability to reorganize itself in response to experience. Additionally, it is important to minimize the risk of overfitting and reduce the computational load of algorithms during real-time (online) use by designing a classifier that relies on a limited, yet highly informative, set of features. As discussed earlier4.2, features like Power Spectral Density (PSD) are often extracted from neural signals, offering high-dimensional data. While this provides a large number of potential features, many of them do not significantly contribute to distinguishing between different tasks. In fact, relying on non-informative features can lead to poor model performance, including incorrect predictions. To address this, statistical feature selection methods, grounded in neurophysiological knowledge, are used to pinpoint the most relevant features for decoding motor intentions. Even though PSD analysis spans a broad frequency range, only a limited subset of frequency bands is actually useful for decoding motor intentions. In particular, movement-related brain activity is often reflected by changes in the spectral power within specific frequency ranges, such as the mu (μ) and lower beta (β) waves, which are typically localized in the motor cortex. Knowing this, feature dimensionality can be drastically reduced by selecting specific channel-frequency combinations that are most relevant for the task. Using a large number of features can lead to overfitting, where the model performs well on training data but fails to generalize to new data. Overfitting is a major issue in real-time applications like BMI control. By carefully selecting features that are statistically significant and physiologically relevant, we can build a simpler and more robust model that avoids this pitfall. Moreover, reducing the number of features helps in lowering the computational burden during online processing, allowing for faster, real-time control. Each individual exhibits unique neural responses, making it essential to consider these differences during the calibration phase. This is why creating a single, generalized model that works for all subjects is not feasible. To determine the specific frequencies and channels that show the greatest variation between the two motor imagery (MI) tasks, various statistical techniques can be employed. Among these, the Fisher Score (FS) is the most commonly used method due to its efficiency and speed of application.

$$F_S(k) = \frac{|\mu_2(k) - \mu_1(k)|}{\sqrt{\sigma_2^2 + \sigma_1^2}}$$

For each calibration run carried out, this value is calculated for every channel/frequency pair and a map is generated. This mapping offers an initial visual representation of the channel-frequency pairs that most effectively differentiate between the two tasks, indicated by higher coefficients compared to others. After acquiring the corresponding matrices from the trials, a single composite matrix can be created by averaging these individual pairs. This step enhances the features that remain consistent throughout the trials while minimizing the influence of sporadic variations, allowing for a clearer assessment of their stability 4.4a. This statistical measure helps to reveal the temporal evolution of channel-frequency pairs where the difference in activity (PSD) between the both-hand and both-feet tasks is most pronounced. Naturally, these values are closely tied to ongoing modulations and may fluctuate in intensity over time.



(a)

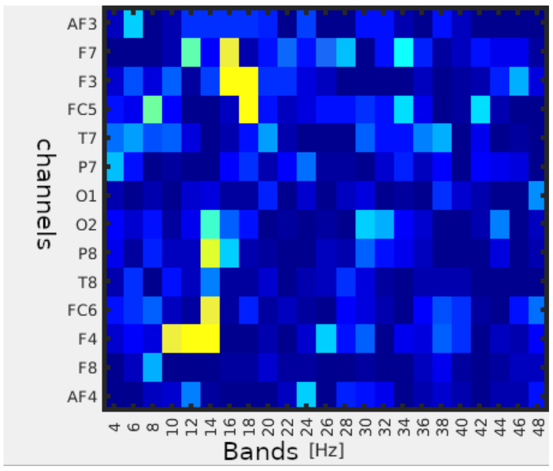


(b)

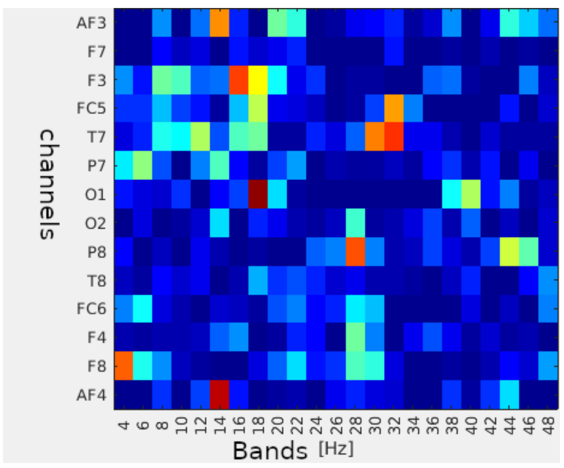
Figure 4.4: Comparison between the Stable Discriminancy Map and Selected, Features Fisher maps generated by averaging 3 calibration sessions. (a) the stable discriminancy map, with regions of higher discriminancy represented by warmer colors ; (b) the selected features highlighted in red on the frequency bands and channels

4.3.2 Feature Classification

After generating the average Fisher Score (FS) map between the classes, the subsequent step is to select the most relevant features. This selection process should not rely solely on the highest FS values but must be guided by a thorough understanding of the theoretical framework. As previously highlighted, only specific channels—mainly those positioned over the central and frontal regions—and certain frequency bands, particularly the mu (μ) and lower beta (β) waves, are indicative of motor imagery (MI). If significant indices appear outside these regions or bands, they should be excluded, as they are likely caused by random factors unrelated to intentional modulation. Such factors might include eye movements, often detected by the Fp1 and Fp2 electrodes (which are not part of the Emotiv Epoc X setup), distractions, variations in attention, stress, or muscle artifacts, typically observed in the 30-40 Hz range. Furthermore, the selected features should remain consistent, or at least partially so, across each run analyzed. This consistency is crucial both for studying the progression of neurophysiological patterns and for ensuring a stable model that is not influenced by temporary fluctuations, thereby achieving high performance, which can only be guaranteed by continuous properties.



(a) Good features



(b) Bad features

Figure 4.5: (a) shows a strong activation in the sensory-motor cortex electrodes (such as FC5, FC6, F3, F4, etc.) with prominent activity concentrated in the μ (8-12 Hz) and β (12-30 Hz) bands. These are typical frequency ranges associated with motor imagery or movement-related tasks, which are key for BCI applications. The yellow areas indicate higher signal power in these channels and frequency bands, implying these features are likely useful for classifying motor tasks in a BCI system. (b) this heatmap displays less organized and more dispersed activation patterns across channels and frequency bands. There is high activity in a few non-sensory-motor electrodes (such as O1, O2, P7), and the activity is spread across various frequency bands, including higher β and γ bands (30+ Hz). This suggests less focused motor-related signal extraction, which could make it more difficult for a BCI system to accurately classify motor tasks.

After confirming that all these conditions are met, we proceed to the actual feature selection process. This is primarily based on the strength and recurrence of the channel-frequency pairs involved in the study. To avoid overwhelming the computational load during online phases—where a large number of covariance matrices will need to be computed—and to minimize overfitting, only a small set of features is selected, typically between three and ten, with a focus on the most robust ones. Once the features are identified, a feature vector is created, containing both the feature values and the corresponding indices that define the task. This vector is then provided to the machine learning algorithm for classifier construction. The aim of the classifier is to establish a set of parameters that maximize the statistical distance between the distributions of the two classes, thereby reducing uncertainty in classification decisions 4.6. Based on several studies [20], Linear Discriminant Analysis (LDA) has proven particularly effective in motor imagery (MI) classification. LDA, both in its linear form (when the covariance matrices of the two classes are equal) and quadratic form (when the covariance matrices differ), is especially useful due to the quasi-Gaussian nature of the power spectral density (PSD) of the two classes. This assumption is crucial for the algorithm's application. LDA works by finding a projection that best separates the two classes, maximizing the ratio of between-class variance to within-class variance, which is ideal for tasks like MI classification where distinguishing between different brain activity patterns is essential. The main goal of Linear Discriminant Analysis (LDA) is to project data onto a lower-dimensional space, typically a one-dimensional line in the case of two classes, while maximizing class separability. The formula for LDA can be described step by step:

1. Computing the mean of each class: Let μ_1 and μ_2 be the means (centroids) of the two classes:

   $$\mu_1 = \frac{1}{N_1} \sum_{i \in C_1} \mathbf{x}_i, \quad \mu_2 = \frac{1}{N_2} \sum_{i \in C_2} \mathbf{x}_i$$

   Where: $\mathbf{x}_i$ is a data point belonging to class C_1 or C_2 ; N_1 and N_2 are the number of samples in classes 1 and 2, respectively.

2. Within-class scatter matrix: S_W represents the variance within each class and is given by:

   $$S_W = \sum_{i \in C_1} (\mathbf{x}_i - \mu_1)(\mathbf{x}_i - \mu_1)^T + \sum_{i \in C_2} (\mathbf{x}_i - \mu_2)(\mathbf{x}_i - \mu_2)^T$$

   This measures how much the samples deviate from their class means.

3. Between-class scatter matrix: S_B captures the distance between the means of the two classes:

   $$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

   This measures the distance between the class means.

4. LDA Projection Vector: To maximize class separability, LDA looks for a projection vector $\mathbf{w}$ that maximizes the ratio of the between-class scatter to the within-class scatter. This ratio is known as the Fisher criterion and is expressed as:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$$

   To solve for $\mathbf{w}$, we take the derivative of $J(\mathbf{w})$ and set it to zero, which results in the following generalized eigenvalue problem:

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

   Where $\mathbf{w}$ is the eigenvector corresponding to the largest eigenvalue λ. Thus, the optimal projection vector $\mathbf{w}$ is the eigenvector of $S_W^{-1} S_B$ that corresponds to the largest eigenvalue, as it maximizes the separation between the classes.

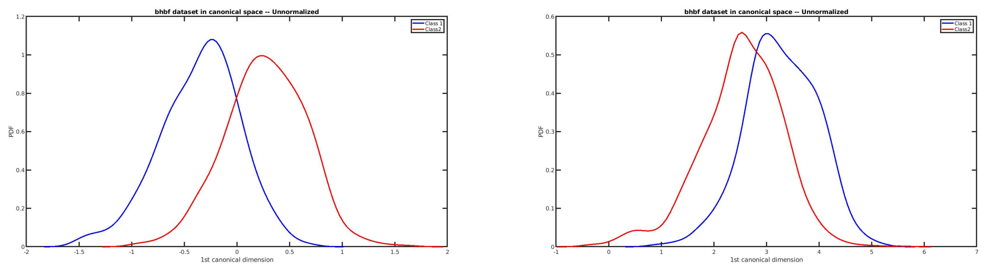
5. Classifying a New Sample: Once $\mathbf{w}$ is obtained, a new data point $\mathbf{x}$ can be projected onto $\mathbf{w}$ and classified based on which side of the projected space it lies:

$$y = \mathbf{w}^T \mathbf{x}$$

   This allows the classification of the new point based on how close it is to the means of the two classes in the projected space. In summary, LDA maximizes the between-class variance while minimizing the within-class variance by solving the eigenvalue problem. The key formula here is:

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

Obviously, no matter how well a single model performs, its effectiveness alone is insufficient for ensuring proper use in brain-machine interfaces (BMI). To address this, validation methods are employed to identify the best possible parameters. When a classifier is built, the algorithm estimates the statistical distributions of the two classes and visualizes them on a graph. Additionally, the rejection rate during each calibration run is evaluated. This parameter reflects how often the classifier predicts outcomes with an accuracy below 55%, which indicates a very low level of confidence (approaching random classification). A model with such low predictive accuracy cannot be relied upon for real-time applications. Among the classifiers generated during calibration, the best model is the one that exhibits the greatest separation between classes and the lowest rejection rate. Other key metrics for evaluating the quality of a classifier are summarized in a confusion matrix, which details the percentages of correct classifications (true positives, TP, for the first class and true negatives, TN, for the second) and incorrect classifications (false positives, FP, and false negatives, FN). These values provide insight into the algorithm's ability to distinguish between the two tasks and make accurate predictions. From these four metrics, the overall accuracy of the model, which reflects its ability to correctly classify samples, can be calculated. Regarding accuracy, particular attention must be given to the risk of overfitting. If a classifier shows very high accuracy (over 90%), it could mean that the model is overly dependent on the specific data used during its creation. This can lead to poor generalization, causing significant errors in interpreting the user's intentions during real-time use of BMIs



(a) Good distributions (b) Bad distributions

Figure 4.6: (a) shows an example of well-separated distributions differently from (b). A nice division leads to better classification from the model.

4.3.3 Accumulation Framework (AF)

The output generated by the model when data are processed through the algorithm consists of samples, meaning that the classification occurs on individual time samples within each window of the PSD. However, the model's output is often not continuous, producing task predictions that do not align with the expected frequency of kinesthetic imagination. This discontinuity typically arises due to classifier bias, which is caused by small differences in the class distributions stemming from poorly informative features or the presence of high-amplitude artifacts that were not properly filtered. Additionally, the model's predictions do not account for previous outcomes over time. As a result, this type of erratic classification is not suitable for real-time use in brain-machine interfaces (BMI). To address these issues, a method is needed that can accumulate evidence over time, or identify, with high certainty, the user's intent to execute a particular task based on information gathered from previous moments. This technique is known as an accumulation framework (AF), and it typically involves one or more mathematical functions, which are fine-tuned by the researcher to optimize classification performance. Various types of AFs can be used for real-time classification, but in this experiment, an algorithm based on an Exponential Smoothing function was applied. Specifically, the prediction outcome is influenced by both the current smoothing parameters and the probability values generated by the classifier at the current and previous time points. A critical aspect of AF is striking a balance between the current prediction and those made in the past. Exponential smoothing achieves this by incorporating previous predictions to inform future ones, with the outcome being adjusted based on a parameter called α. This parameter (which can be adjusted according to the subject's performance) regulates how quickly the system accumulates evidence. Once the output of the accumulation framework reaches or surpasses the threshold set to define a class, the probabilities for each task are reduced to 50%, which is the minimum value, as the binary nature of the tasks prevents a lower percentage. At this point, the command is sent to the BMI (for instance, turn right or left depending on whether the task progression involves both hands or both feet). Choosing the right parameters in this process has a significant impact on the user experience when controlling the BMI. Fine-tuning these parameters can make it easier or harder to complete certain tasks (depending on difficulty), grant the user more control over timing, or mitigate the impact of misclassifications

$$D(t) = D(t-1) \cdot \alpha + pp(t) \cdot (1-\alpha)$$

```xml
<!-- integrator node -->
<node name="integrator" pkg="rosneuro_integrator" type="integrator"
output="screen">
<rosparam param="plugin" subst_value="True">$(arg integratorplugin)</
rosparam>
<rosparam param="alpha" subst_value="True">$(arg alpha)</rosparam>
<remap from="/smr/neuroprediction" to="/smrbci/neuroprediction"/>
<remap from="/integrated" to="/integrator/neuroprediction"/>
</node>
```

Listing 4.3: ROSNeuro_integrator

A major challenge for BMIs arises when the user's neurophysiological patterns become unstable during use. In such cases, the model struggles to distinguish accurately between different classes, resulting in a bias influenced by the initial parameters set during its development. This bias can make it difficult to complete one task and cause the system to incorrectly switch to another. Additionally, exponential smoothing can become unstable when faced with a high number of misclassified samples, failing to manage the absence of motor imagery signals effectively. These fluctuations can be managed by selecting a more resilient framework [21], incorporating a progressive analysis of features, and performing periodic re-calibration. If the model is no longer able to align with the user's intentions, it should either be re-calibrated or adjusted using the new data, while retaining the previous model or rebuilding it from scratch if necessary.

4.4 Unmanned Aerial Systems (UAVs)

The standard n-rotor is made of a rigid body with n propellers spinning about their own axis (including the special cases of all parallel or all different axes). In particular, the number n of propellers and the axes mutual orientations determine if the n-rotor is an under-actuated or fully-actuated system. In our case study we have $n = 4$.

4.4.1 Kinematics and Dynamics of the n-Rotor System

To describe the dynamics of the n-rotor system, we define the body frame F_B with the origin O_B at the center of mass (CoM) of the platform. The position of O_B in the inertial world frame F_W is represented by the vector $\mathbf{p} \in R^3$ and the orientation of the body frame F_B relative to F_W is given by the rotation matrix $\mathbf{R} \in SO(3)$. The state of the vehicle is represented as $X = (\mathbf{p}, \mathbf{R}) \in R^3 \times \mathrm{SO}(3)$ describes the complete pose of the vehicle in the inertial frame F_W. The full pose of the vehicle in F_W is described by the position and orientation of the platform. The twist of the platform is given by the pair (v, ω), where:

$$v = \dot{p} \in R^3, \quad \omega \in R^3$$

Here, v denotes the linear velocity of O_B in F_W, and w is the angular velocity of F_B relative to F_W. The platform's kinematics can be expressed as:

$$\begin{aligned} \dot{p} &= v \\ \dot{R} &= R[\omega]_\times \end{aligned}$$

where $R[\omega]_\times$ represents the skew-symmetric matrix associated with the vector w.

4.4.2 Forces and Moments from the Propellers

The dynamics of the system are analyzed using the Newton-Euler method. The i-th propeller, where $i = 1, \ldots, n$, rotates around its axis with a controllable spinning rate $\omega_i \in \mathbf{R}$. There are two types of propellers: those that spin clockwise (CW) and those that spin counterclockwise (CCW) with respect to their axis, denoted as $\hat{u}_{z_i}$. The angular velocity in the body frame F_B is ddefined as:

$$\begin{array}{ll} -\omega_i \hat{u}_{z_i} & \text{IF CW Rotation} \\ +\omega_i \hat{u}_{z_i} & \text{IF CCW Rotation} \end{array}$$

We define the control input as $u_i = \omega_i|\omega_i| \in \mathbf{R}$. Each propeller generates a thrust force $f_i \in \mathbf{R}^3$which in the body frame F_B, is given by:

$$f_i = c_{f_i} u_i \hat{u}_{z_i}$$

where $c_{f_i} \in \mathbf{R}^+$ is a constant parameter. Additionally, the propeller produces a drag moment $\tau_i^d \in \mathbf{R}^3$, expressed as:

$$\tau_i^d = c_{\tau_i} u_i \hat{u}_{z_i}$$

where $c_{\tau_i} \in R$ is a constant parameter (positive for CW and negative for CCW). Considering now the entire UAV 4.7 of a quadocopter. The thrust moment generated by the i-th propeller is represented as $\tau_{t_i} = p_i \times f_i \in \mathbf{R}^3$, where p_i is the position vector of the propeller. The total control force $f_c \in \mathbf{R}^3$ and the total control moment $\tau_c \in \mathbf{R}^3$ applied at O_B are:

$$f_c = \sum_{i=1}^{n} f_i = \sum_{i=1}^{n} c_{f_i} \hat{u}_{z_i} u_i \tag{4.1}$$

$$\tau_c = \sum_{i=1}^{n} (\tau_{t_i} + \tau_{d_i}) = \sum_{i=1}^{n} \left(c_{f_i} (p_i \times \hat{u}_{z_i}) + c_{\tau_i} \hat{u}_{z_i} \right) u_i \tag{4.2}$$

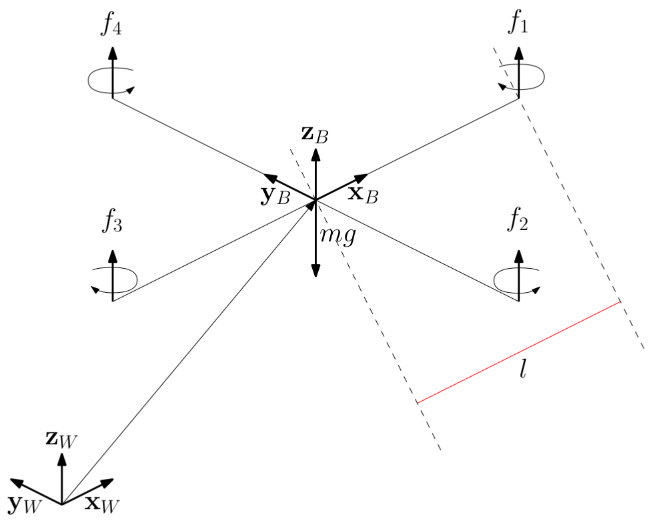


Figure 4.7: Reference frames of an UAV

Introducing the control input vector $u = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix}^\top \in R^n$, equations (4.1) and (4.2) can be simplified as:

$$f_c = Fu, \quad \tau_c = Mu,$$

where the control force input matrix $F \in \mathbf{R}^{3\times n}$ and the control moment input matrix $M \in \mathbf{R}^{3\times n}$ depend on the geometric and aerodynamic parameters introduced earlier.

4.4.3 Dynamics of the n-Rotor System

Neglecting second-order effects (like gyroscopic and inertial effects), the dynamics of the n-rotor system are described by the Newton-Euler equations:

$$\begin{aligned} m\ddot{p} &= -mge_3 + Rf_c = -mge_3 + RFu \\ J\dot{\omega} &= -\omega \times J\omega + \tau_c = -\omega \times J\omega + Mu \end{aligned}$$

where $g > 0$ is the gravitational acceleration, $m > 0$ is the total mass of the platform, and $J \in \mathbf{R}^{3\times 3}$ is its positive-definite inertia matrix. The vector e_i represents the i-th canonical basis vector of $\mathbf{R}^3$ for $i = 1,2,3$.

4.4.4 General and Standard Cases of Control Matrices

For a quadrotor with tilted propellers (propellers are tilted with an angle α along the axis orthogonal to the arm of length l), the F and M matrices are expressed as:

$$F = c_f \cdot \begin{bmatrix} 0 & s_\alpha & 0 & -s_\alpha \\ s_\alpha & 0 & -s_\alpha & 0 \\ c_\alpha & c_\alpha & c_\alpha & c_\alpha \end{bmatrix}$$

$$M = l \cdot c_f \cdot \begin{bmatrix} 0 & c_\alpha & 0 & -c_\alpha \\ -c_\alpha & 0 & c_\alpha & 0 \\ s_\alpha & -s_\alpha & s_\alpha & -s_\alpha \end{bmatrix} + |c_\tau| \cdot \begin{bmatrix} 0 & s_\alpha & 0 & -s_\alpha \\ -s_\alpha & 0 & s_\alpha & 0 \\ -c_\alpha & c_\alpha & -c_\alpha & c_\alpha \end{bmatrix}$$

where $c_\alpha = \cos(\alpha)$ and $s_\alpha = \sin(\alpha)$. In the case of the standard coplanar quadrotor, when $\alpha = 0$, the resulting expressions are:

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_f & c_f & c_f & c_f \end{bmatrix}$$

$$M = \begin{bmatrix} 0 & l\cdot c_f & 0 & -l\cdot c_f \\ -l\cdot c_f & 0 & l\cdot c_f & 0 \\ -|c_\tau| & |c_\tau| & -|c_\tau| & |c_\tau| \end{bmatrix}$$

4.4.5 System coupling and actuation properties

It is useful at this stage to provide a unified overview of the kinematics and dynamics equations both with a $X = (p,R) \in \mathbf{R}^3 \times SO(3)$ representation (as given in detail above):

$$\dot{p} = v \tag{4.3}$$

$$\dot{R} = R[\omega]_\times \tag{4.4}$$

$$m\ddot{p} = -mge_3 + RF_u \tag{4.5}$$

$$J\dot{\omega} = -\omega \times J\omega + M_u \tag{4.6}$$

and with a $X = (p,q) \in \mathbf{R}^3 \times \mathbf{S}^3$, where we use the quaternion representation for the pose orientation and which can be retrieved from the previous analysis by replacing the rotation matrices and employing quaternion algebra:

$$\dot{p} = v \tag{4.7}$$

$$\dot{q} = \frac{1}{2} q \circ \begin{bmatrix} 0 \\ \omega \end{bmatrix} = \frac{1}{2} M(q) \begin{bmatrix} 0 \\ \omega \end{bmatrix} \tag{4.8}$$

$$m\ddot{p} = -mge_3 + R(q)F_u \tag{4.9}$$

$$J\dot{\omega} = -\omega \times J\omega + M_u \tag{4.10}$$

We can highlight here how the rotational dynamics ((4.6))-((4.10)) influences, through ((4.4))-((4.8)), the translational dynamics ((4.5))-((4.9)). This cascaded dependency of the translational dynamics on the rotation implies the emergence of a coupling between the control force and the control torque. If we go on definition of under and fully actuated systems where: , we can see that the affine form $\ddot{x} = f_1(x,\dot{x},t) + f_2(x,\dot{x},t)u$ corresponds to:

$$\begin{bmatrix} \ddot{p} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{g}{m} e_3 \\ -J^{-1}(\omega \times J\omega) \end{bmatrix} + \begin{bmatrix} \frac{1}{m} RF_u \\ J^{-1} M_u \end{bmatrix}$$

Full actuation is attained if the following rank condition is satisfied:

$$\operatorname{rank} \begin{bmatrix} m^{-1}RF \\ J^{-1}M \end{bmatrix} = \operatorname{rank} \begin{bmatrix} m^{-1}RF \\ J^{-1}M \end{bmatrix} \in \mathbf{R}^{6\times n}$$

where this matrix must have rank 6. In detail, this means:

$$\operatorname{rank}\begin{bmatrix} m^{-1}R & 0 \\ 0 & J^{-1} \end{bmatrix}\begin{bmatrix} F \\ M \end{bmatrix} = \operatorname{rank}\begin{bmatrix} F \\ M \end{bmatrix}$$

The quadrotor, whether coplanar or tilted, is always underactuated since $n = 4$. In contrast, the hexarotor can be fully actuated only if the matrix F has its first two rows non-zero. This necessitates a tilted rotor configuration, where the rotors are tilted appropriately. An example is the following case:

- **Coplanar Quadcopter**: All propellers' z-axes are aligned with the body frame z-axis.

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ * & * & * & * \end{bmatrix}, \quad M = \begin{bmatrix} 0 & * & 0 & * \\ * & 0 & * & 0 \\ * & * & * & * \end{bmatrix}$$

For the coplanar configuration, the rank conditions are:

$$\operatorname{rank}(F) = 1, \quad \operatorname{rank}(M) = 3, \quad \operatorname{rank}\begin{bmatrix} F \\ M \end{bmatrix} = 4$$

- **α-Tilted Quadcopter**: All propellers' z-axes are tilted by an angle α around the arm-axis.

$$F = \begin{bmatrix} 0 & * & 0 & * \\ * & 0 & * & 0 \\ * & * & * & * \end{bmatrix}, \quad M = \begin{bmatrix} 0 & * & 0 & * \\ * & 0 & * & 0 \\ * & * & * & * \end{bmatrix}$$

For the α-tilted configuration, the rank conditions are:

$$\operatorname{rank}(F) = 3, \quad \operatorname{rank}(M) = 3, \quad \operatorname{rank}\begin{bmatrix} F \\ M \end{bmatrix} = 4$$

- **β-Tilted Quadcopter**: All propellers' z-axes are tilted by an angle β around the orthogonal to the arm-axis.

$$F = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}, \quad M = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

For the β-tilted configuration, the rank conditions are:

$$\mathrm{rank}(F) = 3, \quad \mathrm{rank}(M) = 3, \quad \mathrm{rank}\begin{bmatrix} F \\ M \end{bmatrix} = 4$$

An additional parameter that impacts the control effectiveness of a coplanar multirotor is the angle γ between the platform's arms. When the propellers are tilted (as in the α-tilted configuration), any adjustment to the angle γ alters both the strength and direction of the force component that acts in the plane of the propellers 4.8.

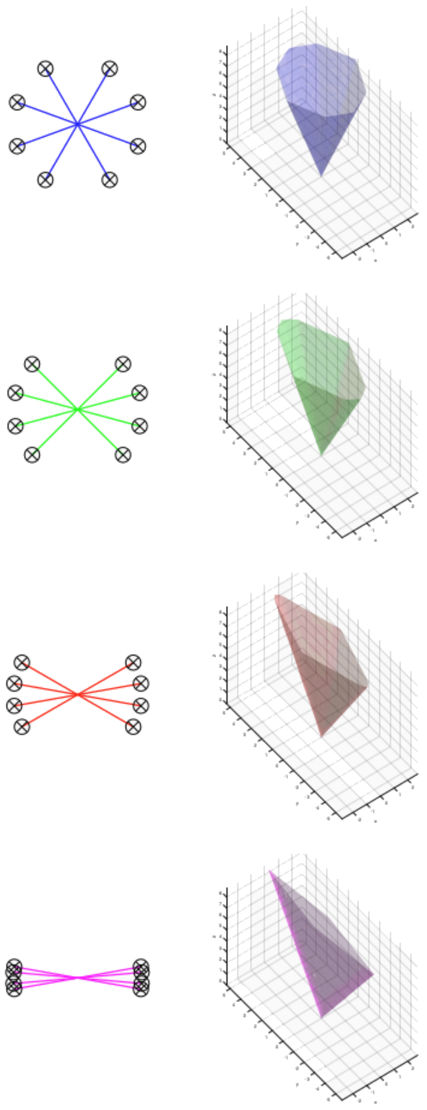


Figure 4.8: Tilted octorotor: force cones of the α-tilted octorotor, for different choice of the γ-angles among the arms.

In general, for a multirotor platform, the following holds:

$$\begin{aligned} \text{Control force input matrix} \quad F \in \mathbf{R}^{3\times n} \quad \text{with} \quad 1 \leq \operatorname{rank}(F) \leq 3 \\ \text{Control moment input matrix} \quad M \in \mathbf{R}^{3\times n} \quad \text{with} \quad 1 \leq \operatorname{rank}(M) \leq 3 \end{aligned}$$

For commonly analyzed platforms,full orientation control is typically achieved: $\operatorname{rank}(M) = 3$. Given this background, it is possible to find matrices A and B such that:

$$A \in \mathbf{R}^{n\times 3}, \quad \operatorname{Im}(A) = \operatorname{Im}(M^\top) = (\ker(M))^\perp$$
$$\operatorname{Im}(B) = \ker(M) \quad \text{where} \quad B \in \mathbf{R}^{n\times(n-3)}$$

The matrix $T = [A|B] \in \mathbf{R}^{n\times n}$ is of full rank and can be regarded as a change of basis matrix in $\mathbf{R}^n$. This can be applied to the input vector u as follows:

$$u = T\tilde{u} = [A|B] \begin{bmatrix} \tilde{u}_A \\ \tilde{u}_B \end{bmatrix} = A\tilde{u}_A + B\tilde{u}_B$$

This decomposition results in:

$$f_c = Fu = FA\tilde{u}_A + FB\tilde{u}_B =: f_c^A + f_c^B$$
$$\tau_c = Mu = MA\tilde{u}_A + MB\tilde{u}_B =: \tau_c^A$$

From this, we can define the following spaces:

$$f_c \in \mathscr{F} := \operatorname{Im}(F) \subseteq \mathbf{R}^3$$
$$f_c^A \in \mathscr{F}_A := \operatorname{Im}(F_A) \subseteq \mathscr{F}$$
$$f_c^B \in \mathscr{F}_B := \operatorname{Im}(F_B) \subseteq \mathscr{F}$$

Thus, we have:

- $\dim(\mathscr{F}_B) = \dim(\mathscr{F})$, it indicates that the control force is entirely independent of the control moment, resulting in an uncoupled system.
- $0 < \dim(\mathscr{F}_B) < \dim(\mathscr{F})$, at least one component of the control force can be freely selected, indicating a partially coupled system.
- $0 = \dim(\mathscr{F}_B)$, the control force is completely dependent on the control moment, leading to a fully coupled system.

An numerical example in the following case:

- **Coplanar Quadcopter:** All propellers' z-axes are aligned with the body frame z-axis.

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}, \quad M = \begin{bmatrix} 0 & 0.02 & 0 & -0.02 \\ -0.02 & 0 & 0.02 & 0 \\ -0.1 & 0.1 & -0.1 & 0.1 \end{bmatrix}$$

$$\mathrm{Im}(M^\top) =< (\left(\begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \right) >, \quad \ker(M) =< \left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right) >$$

$$F_A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad F_B = \begin{bmatrix} 0 \\ 0 \\ 0.4 \end{bmatrix} \Rightarrow \begin{cases} \dim(F_B) = \dim(F) : \text{Uncoupled system} \\ f_c = Fu = FA\tilde{u}_A + FB\tilde{u}_B = f_c^B \end{cases}$$

Decomposition of the force vector yields:

$$u = A\tilde{u}_A + B\tilde{u}_B = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 1 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} d = \begin{bmatrix} -b-c+d \\ a+c+d \\ b-c+d \\ -a+c+d \end{bmatrix}$$

and then

$$f_c = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix} \begin{bmatrix} -b-c+d \\ a+c+d \\ b-c+d \\ -a+c+d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.4d \end{bmatrix}$$

- **α-tilted Quadcopter:**
  All propellers' z-axes are tilted by an angle α around the arm-axis.

$$F = \begin{bmatrix} 0 & -0.02 & 0 & 0.02 \\ -0.02 & 0 & 0.02 & 0 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}, \quad M = \begin{bmatrix} 0 & 0.01 & 0 & -0.01 \\ -0.01 & 0 & 0.01 & 0 \\ -0.1 & 0.1 & -0.1 & 0.1 \end{bmatrix}$$

$$\mathrm{Im}(M^\top) =< (\left(\begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \right) >, \quad \ker(M) =< \left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right) >$$

$$F_A = \begin{bmatrix} -0.03 & 0 & 0 \\ 0 & 0.03 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad F_B = \begin{bmatrix} 0 \\ 0 \\ 0.4 \end{bmatrix} \Rightarrow \begin{cases} 0 < \dim(F_B) < \dim(F) : \text{Partially coupled system} \\ f_c = Fu = FA\tilde{u}_A + FB\tilde{u}_B = f_c^A + f_c^B \end{cases}$$

4.4.6 Platform control

An illustration of the fundamental control actions for a quadcopter to generate thrust and achieve roll, pitch, and yaw movements is presented in 4.9

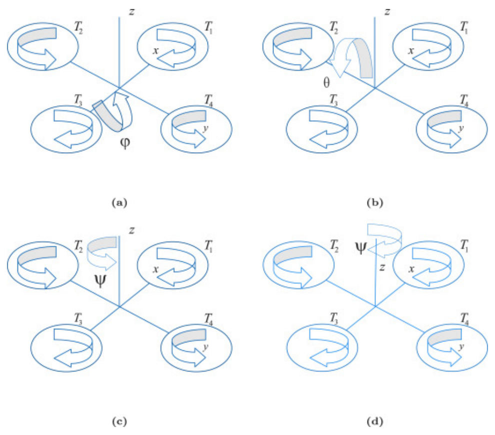


Figure 4.9: Representation of the quadcopter's throttle and angular motions in terms of roll (ϕ), pitch (θ), and yaw (ψ). The diagrams illustrate the corresponding rotational movements of the quadcopter around its primary axes. (a) shows the roll (ϕ) around the x-axis, (b) depicts the pitch (θ) around the y-axis, (c) demonstrates the yaw (ψ) around the z-axis with respect to motor thrusts, and (d) highlights the yawing motion by rotating around the z-axis, driven by the differential thrust of the motors.

A full control scheme is given in 4.10 where the idea of cascaded control is show:This controller is divided into two primary blocks: one focused on position control and the other on attitude control. The position controller takes a position reference p_{ref} as input, along with feedback signals regarding position (and velocity). The attitude controller, on the other hand, considers information related to the body frame orientation (reference rotations and feedback signals). Crucially, the position controller may generate a desired rotation to achieve the position reference, which must be aligned with the orientation reference within the attitude controller. These two blocks produce a wrench pair (force and moment), which is then translated through the wrench mapper into the actual commands for the *n*-rotor (rotational speed of the propellers). It is worth noting that the vertical force needed to counteract gravity may vary based on the platform's orientation. One solution could be to account for (potentially non-zero) roll and pitch angles to achieve precise vertical compensation.

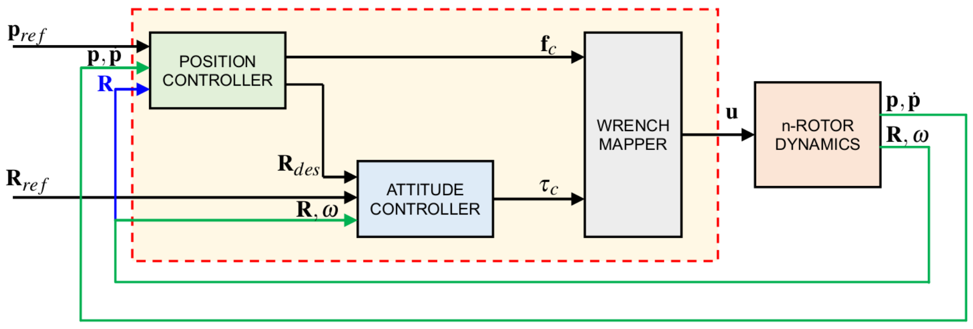


Figure 4.10: Cascaded control scheme.

In the context of the quadrotor, the degrees of freedom for control are constrained to four due to the structural underactuation of the platform. For instance, we can consider the three translational positions and the yaw angle (heading direction), while roll and pitch are determined by the position controller in the y-x plane. Again, this linkage highlights the coupling between translational and rotational dynamics. The corresponding control scheme is illustrated in 4.11, which can be implemented using a set of standard Single-Input Single-Output (SISO) controllers (PID controllers), one for each variable.

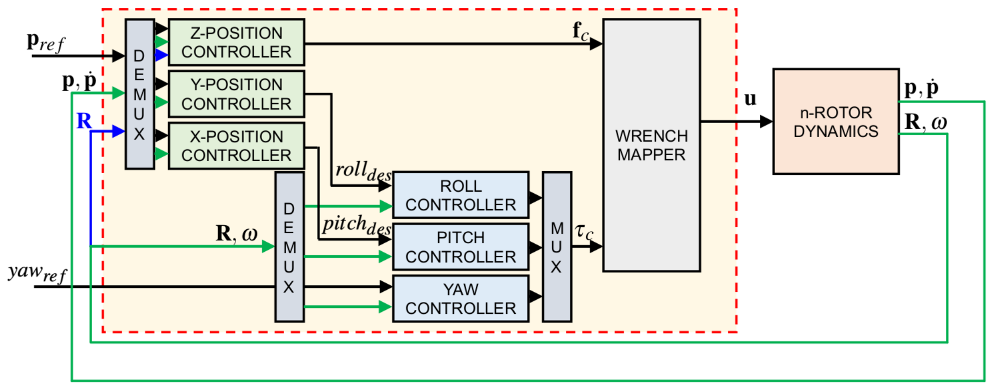


Figure 4.11: Cascaded control scheme for the quadrotor.

4.4.7 Decoupled control

The dynamics of the quadrotor are represented by several key equations that describe its motion starting with angular momentum and control inputs where the first equation is:

$$J\dot{\omega} = -\omega \times J\omega + M_u$$

- J': Inertia matrix, which describes how mass is distributed in the quadrotor. In this case, it's a diagonal matrix indicating different moments of inertia along the principal axes.
- $\dot{w}$: Angular acceleration vector, representing the rate of change of the angular velocity.
- w: angular velocity vector of the quadrotor.
- M_u: control torque vector that includes the input torques applied to control the quadrotor.

The rotation matrix w_B transforms the angular rates to the body frame:

$$w_B = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

- $\dot{\phi}, \dot{\theta}, \dot{\psi}$: Time derivatives of the roll, pitch, and yaw angles, respectively.
- w_x, w_y, w_z: Angular velocities in the body frame, transformed by the rotation matrix.

and we consider a diagonal inertia matrix

$$J = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}$$

with control inputs defined by thrust f_c and torque τ_c in the body frame:

$$f_c = Fu = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}, \quad \tau_c = Mu = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}$$

- f_c: Force control input, where thrust T acts in the z-direction.

- τ_c: Torque control input representing roll, pitch, and yaw torques.

The equations describing translational dynamics are:

$$\dot{p} = v$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix}^{-1} \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}$$

$$\dot{v} = \ddot{p} = -\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{1}{m} \begin{bmatrix} * & * & s\psi s\phi + c\psi s\theta c\phi \\ * & * & -c\psi s\phi s\psi s\theta c\phi \\ * & * & c\theta c\phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

$$\begin{bmatrix} \dot{w}_x \\ \dot{w}_y \\ \dot{w}_z \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} w_y w_z \\ \frac{J_z - J_x}{J_y} w_z w_x \\ \frac{J_x - J_y}{J_z} w_x w_y \end{bmatrix} + \begin{bmatrix} J_x^{-1} & 0 & 0 \\ 0 & J_y^{-1} & 0 \\ 0 & 0 & J_z^{-1} \end{bmatrix} \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}$$

Assuming small angles and similar inertia moments simplifies the model further to:

$$\begin{aligned}
\dot{p} &= v \\
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} \\
\dot{v} = \ddot{p} = -\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{1}{m} &\begin{bmatrix} s\psi s\phi + c\psi s\theta c\phi \\ -c\psi s\phi s\psi s\theta c\phi \\ c\theta c\phi \end{bmatrix} \frac{T}{m} \\
\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \dot{w}_x \\ \dot{w}_y \\ \dot{w}_z \end{bmatrix} &= \begin{bmatrix} \frac{\tau_\phi}{J_x} \\ \frac{\tau_\theta}{J_y} \\ \frac{\tau_\psi}{J_z} \end{bmatrix}
\end{aligned}$$

This simplified model can be employed to devise the control action, revealing that the inputs—thrust T and torques τ_ϕ, τ_θ, and τ_ψ—are the four control variables afforded by the quadrotor's actuation properties. A preliminary selection for the controller references can now be established by incorporating two loops: one for attitude (requiring desired attitude profiles $\phi_{\text{des}}(t)$, $\theta_{\text{des}}(t)$, $\psi_{\text{des}}(t)$) and another for elevation (necessitating a desired elevation profile $p_{z,\text{des}}(t)$). The attitude controller may consist of three independent Proportional-Derivative (PD) controllers:

$$\begin{aligned}
\tau_\phi &= k_{\phi,p}(\phi_{\text{des}} - \phi) + k_{\phi,d}(\dot{\phi}_{\text{des}} - \dot{\phi}) \\
\tau_\theta &= k_{\theta,p}(\theta_{\text{des}} - \theta) + k_{\theta,d}(\dot{\theta}_{\text{des}} - \dot{\theta}) \\
\tau_\psi &= k_{\psi,p}(\psi_{\text{des}} - \psi) + k_{\psi,d}(\dot{\psi}_{\text{des}} - \dot{\psi})
\end{aligned}$$

- $k_{\phi,p}; k_{\phi,d}$: Proportional and derivative gains for roll control.

- $\theta_{\text{des}}; \dot{\phi}_{\text{des}}$: Desired roll angle and rate.

For the elevation controller, a basic approach derived from the model equations under hovering conditions ($\phi \approx 0$, $\theta \approx 0$) can be utilized:

$$\ddot{p}_z \approx -g + \frac{T}{m} \quad \Rightarrow \quad T = m\left(\ddot{p}_{z,\text{des}} + g\right)$$

Alternatively, a more sophisticated PD controller can be employed, incorporating information about roll ϕ and pitch θ angles, alongside a nonlinear compensation term for gravity and the roll and pitch inclinations, and possibly a feedforward term:

$$T = \frac{m}{\cos\theta\cos\phi}\left(g + \ddot{p}_{z,\mathrm{des}} + k_{z,p}(p_{z,\mathrm{des}} - p_z) + k_{z,d}(\dot{p}_{z,\mathrm{des}} - \dot{p}_z)\right)$$

The main limitation of this control design is the challenge of deriving references for the attitude based on a desired trajectory to follow. The quadrotor selected for simulation in Gazebo is the IRIS model, which features slightly asymmetric properties, providing nearly double the rolling capabilities compared to pitching. The input **u** consists of the squared rotor spinning rates:

$$\mathbf{u} = \begin{bmatrix} \omega_1^2 & \omega_2^2 & \omega_3^2 & \omega_4^2 \end{bmatrix}$$

To summarize, a **wrench mapper** can be developed to translate the desired common thrust T and the desired body torques τ_ϕ (roll), τ_θ (pitch), and τ_ψ (yaw) into the required input vector **u**. The attitude controller comprises three independent PD controllers:

$$\begin{aligned}
\tau_\phi &= k_{\phi,p}(\phi^{\mathrm{des}} - \phi) + k_{\phi,d}\frac{sN_\phi}{N_\phi + s}(\dot{\phi}^{\mathrm{des}} - \dot{\phi}) \\
\tau_\theta &= k_{\theta,p}(\theta^{\mathrm{des}} - \theta) + k_{\theta,d}\frac{sN_\theta}{N_\theta + s}(\dot{\theta}^{\mathrm{des}} - \dot{\theta}) \\
\tau_\psi &= k_{\psi,p}(\psi^{\mathrm{des}} - \psi) + k_{\psi,d}\frac{sN_\psi}{N_\psi + s}(\dot{\psi}^{\mathrm{des}} - \dot{\psi})
\end{aligned}$$

where N represents a filter coefficient required for implementing the derivative term. Additionally, the PD controllers have saturated outputs. The elevation controller is also a PD controller but incorporates a nonlinear compensation term for gravity and the roll and pitch inclinations, along with a feedforward term:

$$T = \frac{m}{\cos\theta\cos\phi}\left(g + \ddot{p}_z^{\mathrm{des}} + k_{z,p}(p_z^{\mathrm{des}} - p_z) + k_{z,d}(\dot{p}_z^{\mathrm{des}} - \dot{p}_z)\right)$$

the equations represent the mathematical model governing the dynamics of a quadrotor UAV, describing how the control inputs (thrust and torques) affect its position and orientation. The control strategy consists of a hierarchy of PD controllers for regulating roll, pitch, yaw, and elevation, translating the desired states into motor commands through a wrench mapper. The approach allows for decoupled control, simplifying the complex interactions between the quadrotor's motions.

Chapter 5

Code

This chapter will discuss and include the scripts and codes that enabled me to plot and simulate the IRIS drone via Gazebo. The drone simulations were started by means of launchers executed via the terminal where the Ubuntu operating system is installed, Matlab scripts were executed and additional external libraries such as Biosing and Eeglab had to be installed.

Before initiating control of the drone via BCI, a takeoff sequence is executed, managed by a separate TakeOFF class, this choice was made in order to have a safe takeoff. The drone can move left or right based on commands received from the BCI system processed by a ROS node. This operating procedure was achieved by initializing a ROS node that subscribes to the topic /smrbci/neuroprediction, which provides the predictions generated by the BCI, the node also subscribes to the topic /mavros/local_position/pose, which provides the current position of the drone, at present with this methodology it does not include PID controllers, such a control system is still in a state of development with the goal of adjusting the orientation and movement. At the current state three functions named respectively: waypoint_right, waypoint_left and neuro_callback have been included; the first two 5generate waypoints for the drone's movement to the left and right respectively with respect to the current position, the function uses the drone's current position to calculate a new position (waypoint) in which the drone moves; the neuro_callback function is called to determine the direction of movement by exploiting msg.hardpredict.data. The drone's behavior is consistent with the received predictions when: Neural prediction indicates (0,1), the drone moves to the right; the neural prediction indicates (1,0), the drone moves left. the proposed scheme is the one that was used in the simulation without the use of a PID 5.1

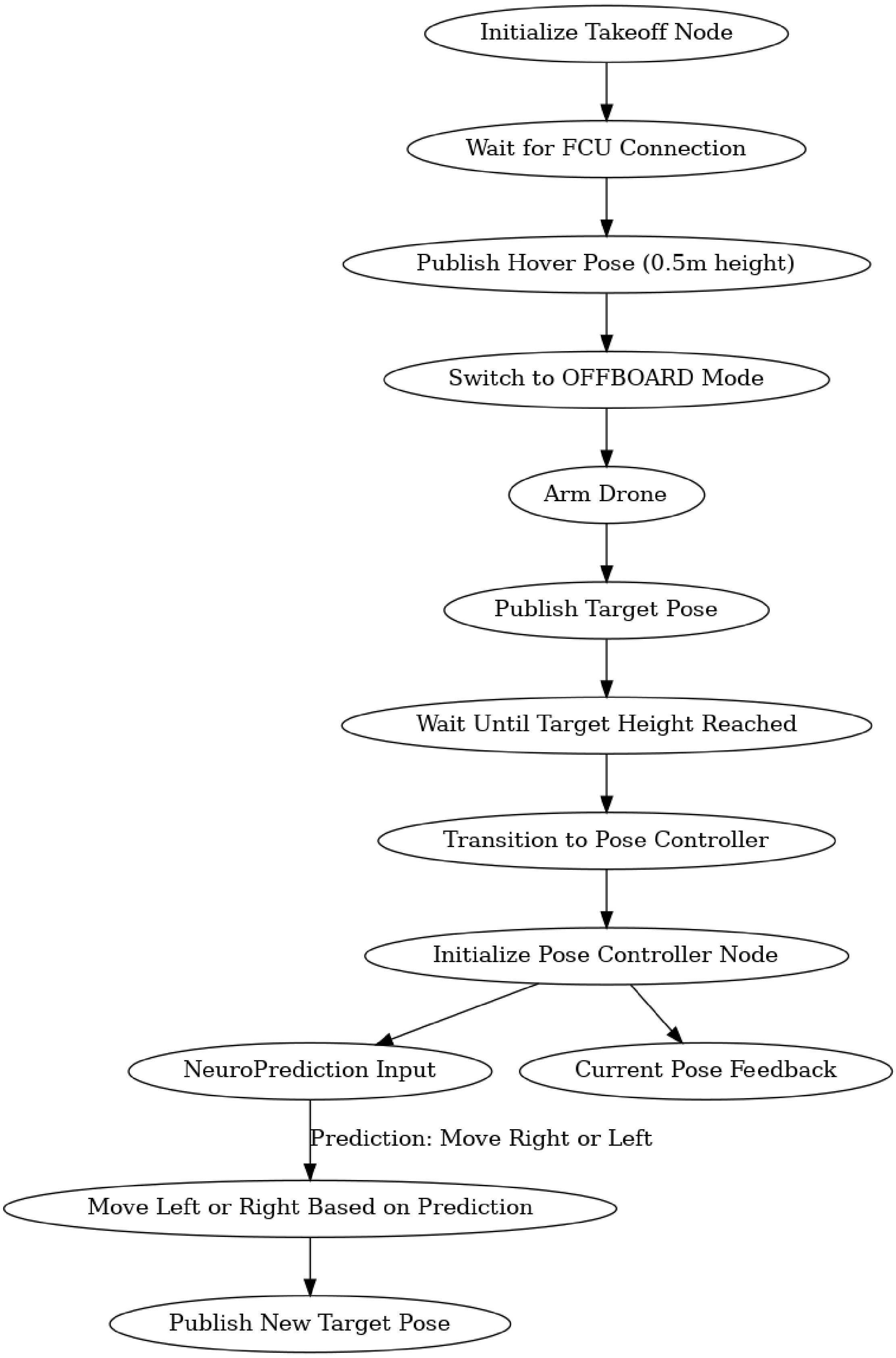

Initialize Takeoff Node
Wait for FCU Connection
Publish Hover Pose (0.5m height)
Switch to OFFBOARD Mode
Arm Drone
Publish Target Pose
Wait Until Target Height Reached
Transition to Pose Controller
Initialize Pose Controller Node
NeuroPrediction Input
Current Pose Feedback
Prediction: Move Right or Left
Move Left or Right Based on Prediction
Publish New Target Pose

Figure 5.1: Flowchart

To run the simulation using Gazebo, PX4 and ROSNeuro, a launcher was created that runs in Ubuntu only once via terminal:

```
<?xml version="1.0"?>
<launch>
<!-- Include the MAVROS node with SITL and Gazebo -->
<include file="$(find px4)/launch/mavros_posix_sitl.launch"> </
include>

<!--node pkg="takeoff" type="takeoff.py" name="takeoff" required="
true" output="screen" />-->

<node pkg= "takeoff" type="Pose_controller_not_PID.py" name="
bci_controller" required="true" output="screen" />

<!-- Our node to control the drone -->
<!-- Feedback arguments -->
<arg name="modality"     default='control'/>
<arg name="thresholds"   default='[0.9, 0.9]'/>
<arg name="classes"      default='[771, 773]'/>
<arg name="show_on_rest" default='True'/>

<!-- processing arguments-->
<arg name="lap_path"      default='$(env HOME)/ros_neuro_ws/recorder/
laplacian14.dat'/>
<arg name="decoder_path"  default='$(env HOME)/ros_neuro_ws/recorder/
classifier14_2.dat'/>
<arg name="n_channels"    default='14'/>
<arg name="eog_threshold" default='43.0'/>

<!-- control framework arguments -->
<arg name="integratorplugin"   default="
rosneuro::integrator::Exponential"/>
<arg name="alpha"              default='0.98'/>

<!-- aquisition node
<node name="rosneuro_windows" pkg="rosneuro_windows" type="
rosneuro_windows.py" output="screen" >
</node>
-->
<!-- processing node -->
<node name="smrbci" pkg="rosneuro_processing" type="test_smrbci"
output="screen">
<param name="~lap_path" value="$(arg lap_path)"/>
<param name="~decoder_path" value="$(arg decoder_path)"/>
```

```
  <param name="~n_channels" value="$(arg n_channels)"/>
  </node>

  <!-- integrator node -->
  <node name="integrator" pkg="rosneuro_integrator" type="integrator"
  output="screen">
  <rosparam param="plugin" subst_value="True">$(arg integratorplugin)</
  rosparam>
  <rosparam param="alpha" subst_value="True">$(arg alpha)</rosparam>
  <remap from="/smr/neuroprediction" to="/smrbci/neuroprediction"/>
  <remap from="/integrated" to="/integrator/neuroprediction"/>
  </node>

  <!-- neurowheel node -->
  <node name="controlwheel" pkg="rosneuro_feedback_wheel" type="
  controlwheel"  output="screen">
  <rosparam param="thresholds" subst_value="True">$(arg thresholds)</
  rosparam>
  <rosparam param="classes" subst_value="True">$(arg classes)</rosparam
  >
  </node>

  <!--node pkg="rqt_reconfigure" type="rqt_reconfigure" name="
  reconfigure" />-->

  </launch>
```

After presenting the launcher, the subsequent codes present the takeoff, which as shown by the flowchart is separate and after reaching the set target the controller will be executed via BCI. The first code is only the TakeOFF, the last is DroneController BCI.

```
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import PoseStamped
from mavros_msgs.msg import State
from mavros_msgs.srv import CommandBool, CommandBoolRequest, SetMode,
 SetModeRequest

current_state = State()

def state_cb(msg):
global current_state
current_state = msg

class TakeOFF:
def __init__(self):
rospy.init_node("takeoff", anonymous=True)
rospy.Subscriber("mavros/state", State, state_cb)

self.local_pos_pub = rospy.Publisher("mavros/setpoint_position/local"
, PoseStamped, queue_size=10)
rospy.wait_for_service("/mavros/cmd/arming")
self.arming_client = rospy.ServiceProxy("mavros/cmd/arming",
CommandBool)

rospy.wait_for_service("/mavros/set_mode")
self.set_mode_client = rospy.ServiceProxy("mavros/set_mode", SetMode)

self.rate = rospy.Rate(10)

def takeoff_sequence(self):
global current_state

# Wait for FCU connection
while not rospy.is_shutdown() and not current_state.connected:
self.rate.sleep()

pose = PoseStamped()
pose.pose.position.x = 0
pose.pose.position.y = 0
pose.pose.position.z = 0.5
```

```python

# Send a few setpoints before starting
for _ in range(10):
if rospy.is_shutdown():
return
self.local_pos_pub.publish(pose)
self.rate.sleep()

offb_set_mode = SetModeRequest()
offb_set_mode.custom_mode = 'OFFBOARD'

arm_cmd = CommandBoolRequest()
arm_cmd.value = True

last_req = rospy.Time.now()
reached_target_height = None
start_time = rospy.Time.now()

while not rospy.is_shutdown():
if current_state.mode != "OFFBOARD" and (rospy.Time.now() - last_req)
 > rospy.Duration(0.5):
if self.set_mode_client.call(offb_set_mode).mode_sent:
rospy.loginfo("OFFBOARD enabled")
last_req = rospy.Time.now()
else:
if not current_state.armed and (rospy.Time.now() - last_req) > rospy.
Duration(0.5):
if self.arming_client.call(arm_cmd).success:
rospy.loginfo("Vehicle armed")
last_req = rospy.Time.now()

self.local_pos_pub.publish(pose)
self.rate.sleep()

# Check if the drone has reached the target height or if a certain
time has passed
if rospy.Time.now() - start_time > rospy.Duration(20.0):
reached_target_height = True

if reached_target_height:
rospy.loginfo("Target height reached, transitioning to
DroneController...")
break
```

Listing 5.1: TakeOFF Python-ROS

```
#!/usr/bin/env python3

import rospy
from rosneuro_msgs.msg import NeuroOutput
from geometry_msgs.msg import PoseStamped, Twist
from mavros_msgs.msg import State, AttitudeTarget
from std_msgs.msg import Header

class DroneController:
def __init__(self):
# Node Subscriber of neuroprediction - RosNeuro
self.neuro_output_sub = rospy.Subscriber("/smrbci/neuroprediction",
NeuroOutput, self.neuro_callback)

self.pose_pub = rospy.Publisher("/mavros/setpoint_position/local",
PoseStamped, queue_size=10)
self.rate = rospy.Rate(10)

self.current_pose = None
self.target_pose = None

# Subscriber for current pose
self.pose_sub = rospy.Subscriber("/mavros/local_position/pose",
PoseStamped, self.pose_callback)

def pose_callback(self, msg):
self.current_pose = msg

def waypoint_right(self):
if self.current_pose is not None:
x = self.current_pose.pose.position.x + 1.5
y = self.current_pose.pose.position.y + 1.0
z = self.current_pose.pose.position.z + 0.1
pose = PoseStamped()
pose.header.stamp = rospy.Time.now()
pose.header.frame_id = "map"
pose.pose.position.x = x
pose.pose.position.y = y
pose.pose.position.z = z
return pose
else:
```

```
return None

def waypoint_left(self):
if self.current_pose is not None:
x = self.current_pose.pose.position.x - 1.5
y = self.current_pose.pose.position.y + 0.5
z = self.current_pose.pose.position.z + 0.1
pose = PoseStamped()
pose.header.stamp = rospy.Time.now()
pose.header.frame_id = "map"
pose.pose.position.x = x
pose.pose.position.y = y
pose.pose.position.z = z
return pose
else:
return None

def neuro_callback(self, msg):
print("Received prediction:", msg.hardpredict.data)
# Assume hardpredict.data contains the prediction
if msg.hardpredict.data == (0,1):
self.target_pose = self.waypoint_right()
rospy.loginfo("Drone moving right")
elif msg.hardpredict.data == (1,0):
self.target_pose = self.waypoint_left()
rospy.loginfo("Drone moving left")
else:
self.target_pose = None

def run(self):
while not rospy.is_shutdown():
if self.target_pose:
self.pose_pub.publish(self.target_pose)
self.rate.sleep()

if __name__ == '__main__':
rospy.init_node('bci_controller', anonymous=True)
controller = DroneController()
controller.run()
rospy.spin()
```

Listing 5.2: DroneController Python-ROS

This algorithm 1 describes the process of loading an EEG file, applying a Laplacian transformation to the EEG data, and visualizing the results. The algorithm begins by creating a graphical user interface (GUI) for interacting with the user. The GUI is built with a window of size 500×300, featuring a minimalist design without a menu bar, and it is non-resizable. The window is labeled as "Select GDF File," indicating that the primary function of the interface is to allow users to select an EEG data file in the GDF format. This step is important as it makes the process user-friendly by providing a visual interface for file loading. After uploading the GDF file, the core of the algorithm involves applying a Laplacian transformation to the EEG data. This step starts by loading a pre-defined Laplacian mask matrix from a file called laplacian_mask_4ch.mat. The Laplacian matrix is crucial because it will be used to transform the EEG signals by emphasizing local differences between channels, which enhances spatial resolution and helps in identifying relevant neural patterns. The matrix's dimensions are checked to ensure that it is 14×14. This is important because the EEG data is expected to have 14 channels, and the Laplacian transformation matrix must match these dimensions. If the matrix is not of the correct size, an error is displayed, and the process is aborted, ensuring that the transformation is only applied when the matrix is compatible with the data. After the data is loaded, the algorithm proceeds to apply the Laplacian transformation. This is done by multiplying the EEG data by the Laplacian mask matrix, which transforms the EEG signals. The purpose of this transformation is to enhance spatial differences between channels, essentially highlighting local variations in brain activity. After applying the transformation, the data is transposed back to its original format, with n_samples×14 dimensions. Both the original EEG data and the transformed data are saved as .mat files. This step is useful for further analysis or for future processing, as the transformed data represents the localized activity after the Laplacian enhancement. To provide a visual representation of the data, the algorithm generates a figure with two subplots. The first subplot displays the original EEG data, while the second subplot shows the transformed data. Different colors and markers are used for each EEG channel, making it easier to distinguish between the signals. This step is particularly valuable for visually comparing the original and transformed data, helping to identify how the Laplacian transformation has affected the signal. Finally, the algorithm computes the energy of both the original and transformed EEG data. Signal energy is calculated as the sum of the squares of the signal values across all samples:

$$E = \sum_{i=1}^{n\_samples} x_i^2$$

This measure of energy gives a sense of the signal's intensity and can help in understanding how the transformation has affected the EEG data. The energy values are printed to the console for both the original and transformed signals, providing quantitative feedback on the transformation's impact.

Algorithm 1 EEG GUI File Loading and Laplacian Transformation

```
1: Create GUI Window:
2: Create a window with size $(500 \times 300)$
3: Set properties: NoMenuBar, NonResizable, title: Select GDF File
4: Add GUI Elements:
5: Add a button labeled “Load GDF File” for loading the GDF file
6: Add a text field initialized with “No file selected” to display the file name
7: Load Laplacian Matrix:
8: Load the Laplacian mask matrix from laplacian_mask_4ch.mat
9: if Matrix dimensions $\neq 14 \times 14$ then
10:     DISPLAY error: “The Laplacian matrix must be 14x14”
11:     Abort process
12: end if
13: Verify EEG Channel Indices:
14: Set channel indices:
15: $FC5\_idx \leftarrow 4$, $FC6\_idx \leftarrow 11$, $T7\_idx \leftarrow 5$, $T8\_idx \leftarrow 10$
16: or $F3\_idx \leftarrow 3$, $F4\_idx \leftarrow 12$ (excluding $T7$ and $T8$)
17: Print row and column data for these channels from the Laplacian matrix
18: File Selection Callback:
19: Open file selection dialog to choose a .gdf file
20: if No file is selected then
21:     DISPLAY “No file selected”
22:     Return
23: else
24:     DISPLAY the selected file name in the text field
25:     Load EEG data from the selected file using sload
26:     if Number of channels $\neq 14$ then
27:         DISPLAY error: “EEG data must have 14 channels”
28:         Return
29:     end if
30:     Transpose EEG data to format $14 \times n\_samples$
31: end if
32: Apply Laplacian Transformation:
33: Multiply EEG data by the Laplacian mask:
    $transformed\_data = laplacian\_mask \times eeg\_data$
34: Transpose back to format $n\_samples \times 14$
35: Save both original and transformed data as .mat files
36: Plot Data:
37: Create a figure with two subplots
38: Plot the original EEG data in one subplot
39: Plot the transformed EEG data in the other subplot
40: Use different colors and markers for each channel
41: Calculate Signal Energy:
42: Calculate energy of both original and transformed data: $E = \sum_{i=1}^{n\_samples} x_i^2$
43: Print energy values for both original and transformed signals
```

This algorithm 2 describes the process of reading a Laplacian matrix from a .mat file, performing necessary checks, and writing it to a new .dat file with appropriate metadata. The Laplacian matrix, referred to as lapmask, is then extracted from the loaded structure. Once extracted, the algorithm verifies that this matrix is indeed 14×14, matching the expected dimensionality for the 14-channel EEG data. If the matrix is not the correct size, an error is thrown to notify the user that the file is invalid for the intended purpose. This dimensionality check ensures compatibility with the expected data structure and prevents further errors down the line. At this stage, the algorithm attempts to open the destination .dat file for writing. If the destination file cannot be opened (e.g., due to an invalid file path or permission issues), an error is thrown, and the process is halted. This check guarantees that the file is writable and prevents loss of data due to faulty file operations. Once the destination file is successfully opened, the algorithm writes both the metadata (header information) and the actual Laplacian matrix data to the file:

- The function wc_writeheader is called with the file identifier and metadata (such as format, version, and type). This ensures that the file includes all necessary information for future use or interpretation.
- The function wc_writeeigen is then called to write the Laplacian matrix itself into the file, preserving the actual data.

Both of these write operations occur within another try-catch block, allowing the algorithm to gracefully handle any errors that might arise during the file-writing process. If an error occurs, the file is closed, and the error is rethrown for proper debugging and handling

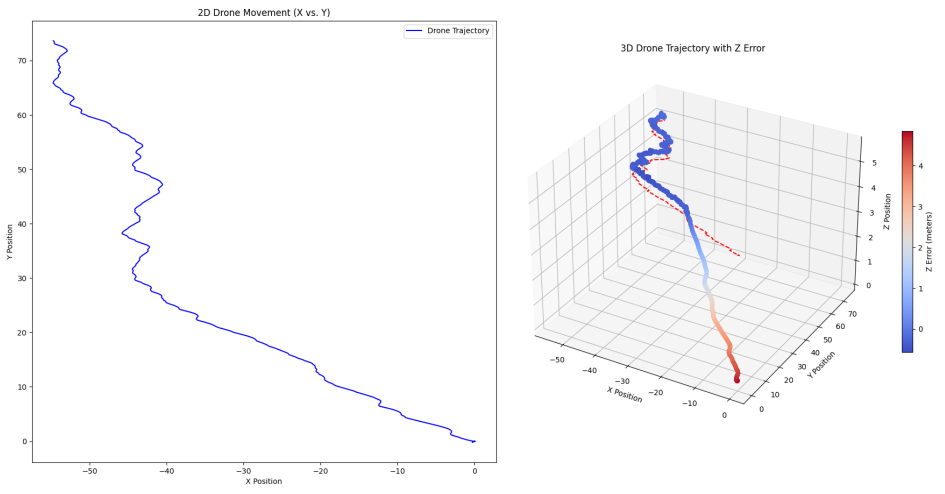
Algorithm 2 Save Laplacian Matrix to .dat File

```
1: Initialize the following variables:
2:     format = 'Grex'
3:     version = '0.5'
4:     type = 'Laplacian14CH'
5:     label = ''
6: if the number of input arguments is less than 2 then
7:     Extract the path and name of the source file.
8:     Set the destination file to the same path with '.dat' extension.
9: end if
10: Try to load the variable from the .mat file:
11:     catch error if file cannot be loaded.
12: Extract the variable name from the structure loaded from the .mat file.
13: if the structure does not contain any variables then
14:     Throw an error that the .mat file contains no variables.
15: end if
16: Extract the Laplacian matrix (lapmask) from the loaded structure.
17: Verify that lapmask is a 14x14 matrix:
18: if lapmask is not a 14x14 matrix then
19:     Throw an error that the matrix format is invalid.
20: end if
21: Open the destination file for writing:
22: if the file cannot be opened then
23:     Throw an error indicating an invalid destination path.
24: end if
25: Try to write the header and Laplacian matrix data to the file:
26:     Call wc_writeheader with the file identifier and metadata.
27:     Call wc_writeeigen to write the Laplacian matrix.
28: catch any errors during file writing:
29:     Close the file and rethrow the error.
30: Close the destination file.
```

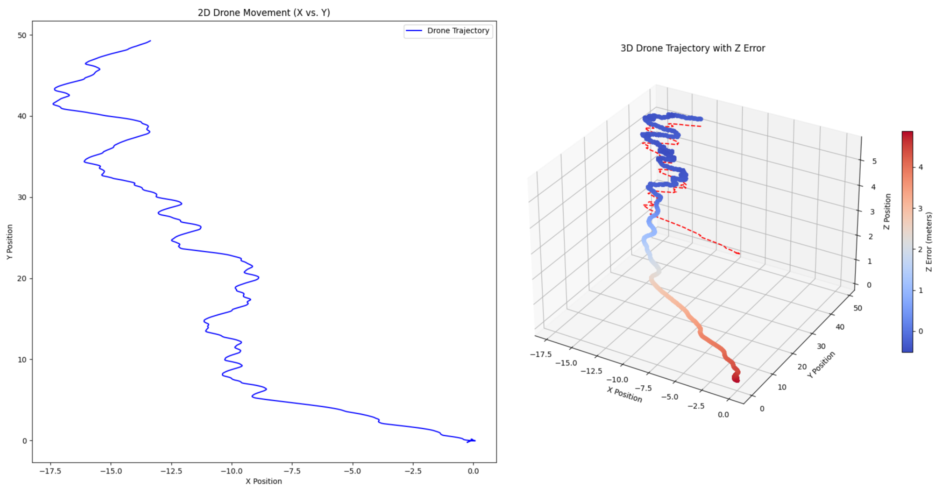
Chapter 6

Results

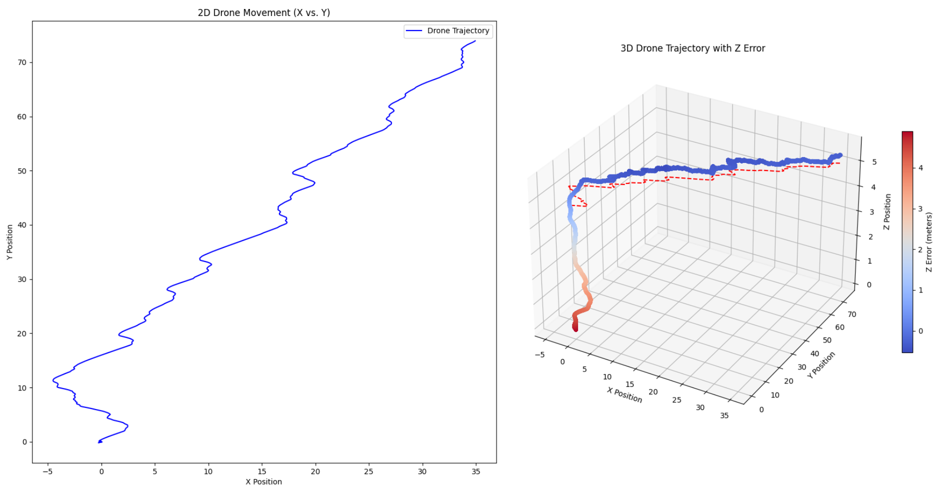
In this chapter, are presented the results of piloting a drone with a brain-computer interface (BCI) system, focusing on offline and online simulations. The key distinction between the two is that offline simulations generate trajectories dynamically based on saved raw EEG data, whereas the online simulation follows a predefined square trajectory . The main objective is to evaluate how different Laplacian filter configurations affect trajectory generation and control accuracy, particularly in the BCI-driven scenario. The offline simulation involved the processing of raw EEG signals recorded in .gdf format. These signals were passed through an integrator, which generated together with the Laplacian filter and classifier a different trajectory for both of the three trials in which only the Laplacian filter was modified. The generated trajectories were observed and compared.



(a) Drone Trajectory with Laplacian filter applied to channels FC5, FC6, F3 and F4.



(b) Drone Trajectory with Laplacian filter applied to channels FC5, FC6, T7 and T8.



(c) Drone Trajectory with Laplacian filter applied to channels FC5 and FC6.

Figure 6.1: Drone Trajectory with different Laplacian filter

The offline simulation results indicate that the choice of Laplacian filter has a profound impact on the drone's control trajectory. The optimized filter produced a trajectory with minimal noise, validating its effectiveness in improving control precision the below figure shows the comparison of trajectories generated by different filters.

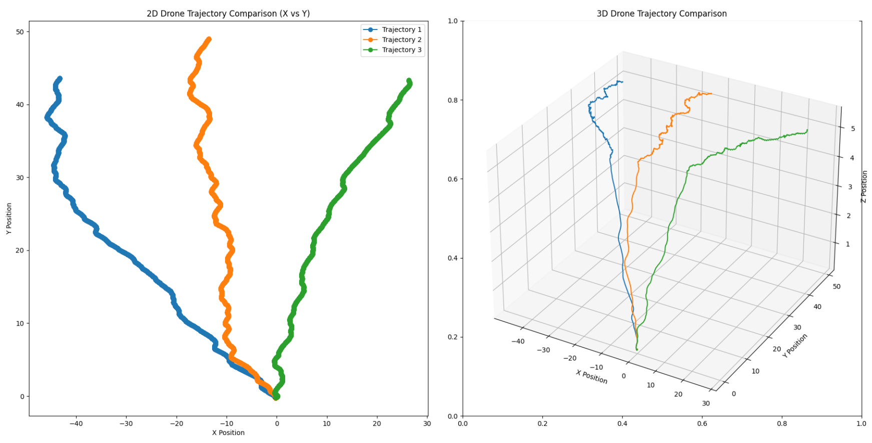


Figure 6.2: Comparison of Drone Trajectory where the blue trajectory is the trajectory where is apply a Laplacian Filter with FC5,FC6, F3 and F4; the orange trajectory where is apply a Laplacian Filter with FC5,FC6, T7 and T8, the green trajectory where is apply a Laplacian Filter with FC5 and FC6

In the 4.1 is possible see the result and the different between a filter where is apply only the FC5 and FC6 with the other two cases, is possible look the difference of Energy of the transformed signals. In the online simulation, noise had a noticeable impact as can be seen in the figure. The Laplacian filter that was used takes as reference the temporal areas T7 and T8 and the motor areas FC6 and FC7, it was not possible to exploit the filter where the four channels of motor imagery activity were taken into account as it was no longer possible to access the laboratories of the University of Catania. Further improvements in filtering and real-time processing could help eliminate these deviations and possible delays due to RAW data transfer altogether. The results of offline and online simulations demonstrate the significant impact of Laplacian filtering on drone control via BCI. In offline simulations, the choice of filter directly influenced the generated trajectory. In the online simulation, the BCI system achieved near-perfect control, generally reflecting the intended trajectory. These results highlight the critical role of filtering techniques in improving the performance and reliability of drone control via BCI.

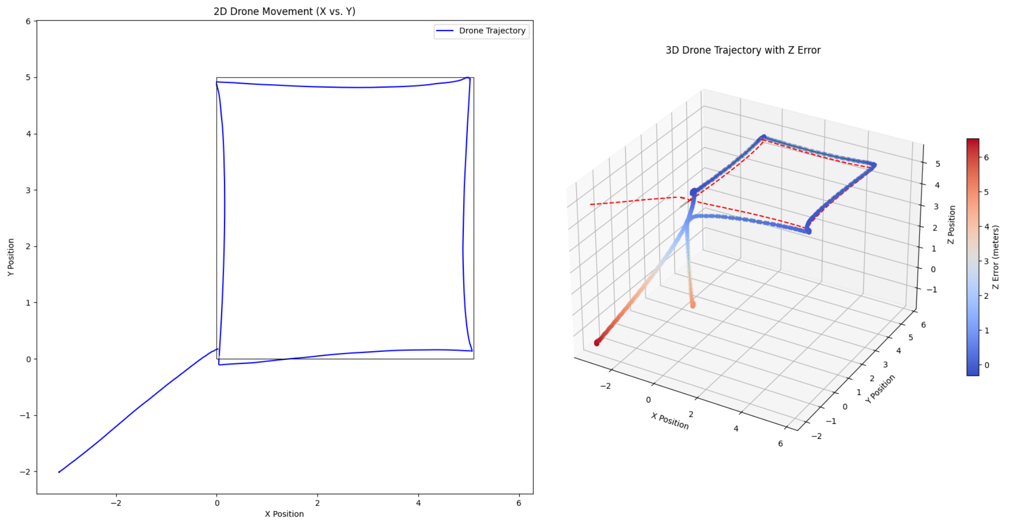


Figure 6.3: The black square represents the ideal trajectory, which is a perfect 5x5 square. The drone was instructed to not exceed an altitude of 5 meters and to complete a square path, where each side of the square is 5 meters in length. The blue line shows the actual trajectory followed by the drone during the simulation. While the drone's movement isn't perfectly aligned with the ideal path, it closely follows the intended square shape.

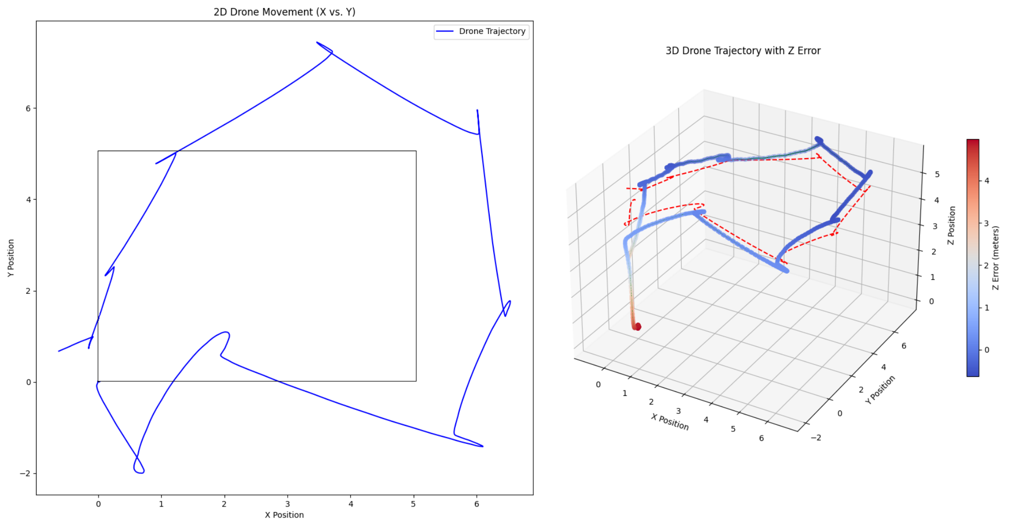


Figure 6.4: Comparison between the ideal flight trajectory (black line) and the actual trajectory (blue line) executed by the drone under the control of a Brain-Computer Interface (BCI) system. The ideal path represents a predefined set of waypoints the drone was supposed to follow, while the blue line shows the real movement of the drone in simulation. Deviations between the ideal and actual trajectories highlight challenges in BCI control, such as noise in brain signal interpretation, response latency, and controller accuracy.

This plot 6.4 is particularly important because it highlights the real-time performance of the drone under BCI control. The deviations of the blue line from the black ideal path indicate how accurately the BCI system was able to command the drone's movements. Several factors might influence these deviations, including:

- Noise in Brain Signals: Since the BCI relies on interpreting brain activity to generate control signals, noise or inaccuracies in the prediction (e.g., from brainwave interpretation errors) could lead to unsteady or imprecise movements, which are reflected in the divergence from the ideal trajectory.
- Latency in Response: The reaction time of the drone to BCI inputs might introduce delays in executing the commands, causing the drone to miss or overshoot target waypoints.
- Controller Accuracy: The BCI controller's prediction accuracy can vary depending on the quality of training data and how well the system is tuned to the individual user. Errors in predicting the intended direction might lead to unintended movements, as seen in the more erratic sections of the blue trajectory.

Chapter 7

Conclusions

This thesis explored the potential of utilizing brain-computer interface (BCI) technology to control drones, specifically examining how external noise might affect control signals. Our primary objective was to test the influence of ambient sounds on the BCI-driven operation of a drone in a real-world environment. However, due to several technical challenges, including difficulties in transmitting raw data to RosNeuro and compatibility issues with the classifier provided by the University of Padua, we were unable to conduct field experiments. Despite these obstacles, we successfully established compatibility between the Emotiv Epoc X headset and the RosNeuro framework, marking a significant advancement in the integration of BCI systems with UAV technology. This achievement lays the groundwork for further development and testing, as it demonstrates the feasibility of linking BCI devices to drone control systems. The simulations carried out were limited to an open-loop system and did not incorporate PID controllers, which restricted our ability to fully assess the dynamics of BCI-controlled flight. Nevertheless, these initial tests provided valuable insights into the BCI's responsiveness and its potential applications in various contexts. Future research should focus on enhancing the robustness of the BCI system, ensuring that it can function effectively not only in simulated environments but also in real-world scenarios. This is particularly important as a BCI-driven UAV system could offer significant benefits for individuals with motor disabilities, granting them new avenues for interaction and control that may not be accessible through traditional means. In conclusion, while this work did not fully achieve the initially set goals, it has laid a crucial foundation for ongoing exploration in the field of BCI technology and UAV systems. Continued efforts to overcome existing technical challenges will be essential in realizing the full potential of BCI applications, ultimately contributing to advancements in assistive technologies and enhancing the quality of life for individuals with disabilities.

Bibliography

[1] (). Emokit: Open source api for emotiv devices, [Online]. Available: https://github.com/openyou/emokit.

[2] S. Ackerman, *Major Structures and Functions of the Brain*. National Academies Press (US), 1992. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK234157.

[3] A. Ghosh, D. Sherman, and P. Brophy, "The axonal cytoskeleton and the assembly of nodes of ranvier," *Trends in Neurosciences*, 2018. DOI: 10.1177/1073858417710897.

[4] G. Schiera, C. Di Liegro, and I. Di Liegro, "Cell-to-cell communication in learning and memory: From neuro- and glio-transmission to information exchange mediated by extracellular vesicles," *International Journal of Molecular Sciences*, vol. 21, no. 1, p. 266, 2019. [Online]. Available: https://doi.org/10.3390/ijms21010266.

[5] A. Biasiucci, B. Franceschiello, and M. M. Murray, "Electroencephalography," *Current Biology*, vol. 29, no. 3, R80–R85, 2019, ISSN: 0960-9822. DOI: https://doi.org/10.1016/j.cub.2018.11.052.

[6] J. J. Vidal, "Toward direct brain-computer communication," *Annual Review of Biophysics*, vol. 2, R157–R160, 1973. DOI: https://doi.org/10.1146/annurev.bb.02.060173.001105.

[7] M. Baraka, A. Abdi T., M. Libe V., *et al.*, "Brain–computer interface: Trend, challenges, and threats," *Brain Informatics*, vol. 10, 2023. DOI: 10.1186/s40708-023-00199-3. [Online]. Available: https://doi.org/10.1186/s40708-023-00199-3.

[8] L. Tonin and J. d. R. Millan, "Noninvasive brain–machine interfaces for robotic devices," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, May 2021. DOI: 10.1146/annurev-control-012720-093904.

[9] S. Sutton, M. Braren, J. Zubin, and E. R. John, "Evoked-potential correlates of stimulus uncertainty," *Science*, vol. 150, no. 3700, pp. 1187–1188, 1965. DOI: 10.1126/science.150.3700.1187. eprint: https://www.science.org/doi/pdf/10.1126/

science.150.3700.1187. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.150.3700.1187.

[10] D. E. Farwell LA, "Talking off the top of your head: Toward a mental prosthesis utilizing event-related brain potentials.," *Electroencephalogr Clin Neurophysiol.*, vol. 70, no. (6), pp. 510–23, 1988 Dec. DOI: 10.1016/0013-4694(88)90149-6.

[11] B. Z. Allison, E. W. Wolpaw, and J. R. Wolpaw, "Brain–computer interface systems: Progress and prospects," *Expert Review of Medical Devices*, vol. 4, no. 4, pp. 463–474, 2007. DOI: 10.1586/17434440.4.4.463.

[12] R. P. N. Rao, A. Stocco, M. Bryan, *et al.*, "A direct brain-to-brain interface in humans," *PLoS ONE*, 2014. DOI: https://doi.org/10.1371/journal.pone.0111332.

[13] O. I. D. Bashi, W. Z. W. Hasan, N. Azis, S. Shafie, and H. Wagatsuma, "Unmanned aerial vehicle quadcopter: A review," *Journal of Computational and Theoretical Nanoscience*, vol. 14, no. 12, pp. 5663–5675, 2017. DOI: https://doi.org/10.1166/jctn.2017.7049.

[14] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: An open-source robot operating system," vol. 3, Jan. 2009.

[15] L. Tonin. (). The first open-source neurorobotic, [Online]. Available: https://github.com/rosneuro.

[16] H. Yuan and B. He, "Brain-computer interfaces using sensorimotor rhythms: Current state and future perspectives," *IEEE transactions on bio-medical engineering*, vol. 61, pp. 1425–35, May 2014. DOI: 10.1109/TBME.2014.2312397.

[17] (). Libxdffileio-dev, [Online]. Available: https://neuro.debian.net/pkgs/libxdffileio-dev.html.

[18] C. Neuper, M. Wörtz, and G. Pfurtscheller, "Erd/ers patterns reflecting sensorimotor activation and deactivation," *Prog Brain Res*, pp. 211–22, 2006. DOI: 10.1016/S0079-6123(06)59014-4.

[19] D.-J. Jwo, W.-Y. Chang, and I.-H. Wu, "Windowing techniques, the welch method for improvement of power spectrum estimation," *Computers, Materials & Continua*, vol. 67, pp. 3983–4003, 2021. DOI: 10.32604/cmc.2021.014752.

[20] M. Z. Ilyas, P. Saad, M. I. Ahmad, and A. R. I. Ghani, "Classification of eeg signals for brain-computer interface applications: Performance comparison," in *2016 International Conference on Robotics, Automation and Sciences (ICORAS)*, 2016, pp. 1–4. DOI: 10.1109/ICORAS.2016.7872610.

[21] L. Tonin, F. C. Bauer, and J. del R. Millán, "The role of the control framework for continuous teleoperation of a brain–machine interface-driven mobile robot," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 78–91, 2020. DOI: 10.1109/TRO.2019.2943072.

[22] A. Barachant, S. Bonnet, M. Congedo, and C. Jutten, "Multiclass brain–computer interface classification by riemannian geometry," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 4, pp. 920–928, 2012. DOI: 10.1109/TBME.2011.2172210.

[23] S.-i. Amari, *Information geometry and its applications*. Springer, 2016.