FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Digital Twin for Drone Control using a Brain-Computer Interface

Diana Cristina Teixeira Ramos

Mestrado em Engenharia de Software

Supervisor: Professor Doutor Gil Gonçalves

Second Supervisor: Ricardo Faria

July 28, 2021

Digital Twin for Drone Control using a Brain-Computer Interface

Diana Cristina Teixeira Ramos

Mestrado em Engenharia de Software

Approved in oral examination by the committee:

Chair: Prof. Nuno Honório Rodrigues Flores

External Examiner: Prof. Paulo Maio

Supervisor: Prof. Gil Manuel Magalhães de Andrade Gonçalves

July 28, 2021

Abstract

With the increasing technological evolution and the appearance of new needs derived from the various modernization movements, among them the urban air mobility mission, there is also an exponential attraction for unmanned aerial vehicles, often referred to as *drones*. These vehicles are not only associated with transportation tasks, but may also be related to the execution of tasks, data collection and monitoring services; being particularly crucial for the execution of high risk and/or almost unreachable operations, allowing the operator to accomplish them remotely and safely.

The brain-computer interface technologies have gained popularity due to their increasing applicability in a wide variety of situations, among them the remote control of drones. Considering that these vehicles are considered critical systems and their control should be both cautious and stable, brain-computer interfaces provide an alternative to the manual control. Since these technologies are directly linked to the emotional and cognitive state of the operator, situations of anxiety and distractions can influence the control and stability of the drone.

This thesis proposes a decision making system that considers the emotional state of the operator, deciding whether the command formulated by the operator should be sent to the drone. By developing a digital copy of the operator, through the *digital twin* technology, with predictive capabilities for emotional detection (visual and cognitive) of the operator, the proposed solution should be able to identify if the operator is in a stable state to send commands. As soon as these situations are detected, the system will calculate the necessary information, corresponding to the desired command, and send it to the drone. Additionally, the communication between the solution and the drone is established through a ROS 2 client node, that connects to a server node, responsible for managing one or more drones.

The validation of the solution is comprised in four scenarios that aim for an incremental level of robustness and security of the system, the first being an experiment without the execution of the developed solution and the last one the integration of the solution with all its functionalities. In addition, validation was performed in a free scenario in order to evaluate whether the solution detects rapid emotional changes in response to external events. For this purpose, an *arena* was created for a controlled execution of the experiments with the *crazyflie* drone. Additionally, a scenario where two client nodes, simulating two drones, was tested in order to evaluate the communication between multiple client nodes and the server.

Results show that the *digital twin* is able to detect emotions, classified through cognitive and visual inputs, in real time, accurately and with the ability to identify mood changes, adjusting to various scenarios when defining and making a decision. Overall, the system is a reliable and safe platform for controlling drones, allowing this to be ensured through the formulation of mental commands. This solution fits in the context of the use of drones, but may enhance the control of other critical and high-risk systems.

Keywords: drone, brain-computer interface, digital twin, ROS2, decision making, emotional state.

Resumo

Com a crescente evolução tecnológica e aparecimento de novas necessidades derivadas dos vários movimentos de modernização, entre eles a missão da mobilidade urbana aérea, existe, igualmente, uma atração exponencial por veículos não tripulados, frequentemente denominados de *drones*. Estes veículos não são só associados a tarefas de transporte, como podem estar relacionados com execução de tarefas, recolha de dados e serviços de monitorização; sendo particularmente cruciais para a execução de operações de risco elevado e/ou de difícil acesso, permitindo que o operador as consiga cumprir de forma remota e com segurança.

As tecnologias de interface cérebro-computador têm ganho popularidade devido à sua crescente aplicabilidade nas mais variadas situações, entre eles o controlo remoto de drones. Tendo em conta que estes veículos são considerados sistemas críticos e o controlo dos mesmos deverá ser tanto cauteloso como estável, as interfaces cérebro-computador constituem uma alternativa ao modo manual. Visto que estas tecnologias estão diretamente vinculadas ao estado emocional e cognitivo do operador, situações de ansiedade e distrações podem influenciar o controlo e estabilidade do drone.

Esta tese propõe um sistema de tomada de decisão, tendo em consideração o estado emocional do operador, decidindo se o comando formulado por este deve ser enviado para o drone. Ao desenvolver uma cópia digital do operador, através da tecnologia *digital twin*, com capacidades preditivas para deteção emocional (visual e cognitiva) do mesmo, a solução proposta deverá ser capaz identificar se o operador está num estado estável para enviar comandos. Assim que estas situações são detetadas, o sistema calculará a informação necessária, correspondente ao comando pretendido, e a enviará para o drone. Adicionalmente, a comunicação entre a solução e o drone é estabelecida através de um nó cliente em ROS 2, conetando-se a um nó servidor, responsável pela gestão de um ou mais drones.

A validação da solução é compreendida em quatro cenários que visam um nível incremental de robustez e segurança do sistema, sendo que o primeiro será uma experiência sem a execução da solução desenvolvida e o último a integração da solução com todas as suas funcionalidades. Além disso, foi realizada a validação num cenário livre de forma a avaliar se a solução deteta rápidas mudanças emocionais em resposta a eventos externos. Para este propósito, foi criada uma *arena* para a realização controlada das experiências com o drone *crazyflie*. Adicionalmente, foi testado um cenário onde são compreendidos dois nós cliente a simularem dois drones, de forma a avaliar a comunicação entre múltiplos nós cliente e o servidor.

Os resultados mostram que o *digital twin* é capaz de detetar emoções, classificadas através de entradas cognitivas e visuais, em tempo real, de forma eficaz e com capacidade de identificar mudanças de humor, ajustando-se aos vários cenários ao definir e concretizar uma decisão. De forma geral, o sistema é uma plataforma fiável e segura para o controlo de drones, permitindo que este seja assegurado através da formulação de comandos mentais. Esta solução enquadra-se no contexto de utilização de drones, mas poderá potencializar controlo de outros sistemas críticos e

de alto risco.

Palavras-chave: drone; interfaces cérebro-computador; digital twin, ROS2; tomada de decisão; estado emocional.

Acknowledgements

I would like to thank my advisor Professor Gil Gonçalves for his guidance during this thesis and for inciting me with interesting and innovative ideas for this project. I would like to acknowledge my supervisor, Ricardo Faria, for the daily support and availability he has given me.

I would like to acknowledge the Faculty of Engineering of the University of Porto for the support and preparation for the development of this thesis.

I would like to thank Cagpgemini Engineering for accepting me in this project, for making available all resources whenever I needed and for giving me access to their amazing laboratory facilities. With Capgemini Engineering, it was possible to integrate this work within a real use case and demonstrate the value of this thesis in the best way possible.

I am also grateful to my fellow colleagues at Capgemini Engineering, Jorge Godinho and Rui Carvalho, who are part of the project's team and always made me feel well integrated. My special thanks to Matheus Sanches, who was my primary source of support during this thesis and that encouraged and assisted me in whatever I needed.

Finally, I'm deeply indebted to my family that have been present in every stage of my education and academic years and that have always supported me in my decisions with proud. I am extremely grateful for being provided with all means and resources to follow my own path which led to the development of this thesis.

Diana Cristina Teixeira Ramos

"One has to watch out for engineers
they begin with the sewing machine and
end up with the atomic bomb."

Marcel Pagnol

Contents

List of Figures

List of Tables

Abbreviations

UAV	Unmanned Aerial Vehicle
UAM	Urban Air Mobility
BCI	Brain-Computer Interface
R&D	Research and Development
DT	Digital Twin
ROS	Robotic Operating System
CRISP-DM	Cross Industry Standard Process for Data Mining
EEG	Electroencephalography
hBCI	hybrid Brain-Computer Interface
MI	Motor Imagery
LOA	Level of Support
SVM	Support Vector Machine
LDA	Linear Discriminant Analysis
PLM	Product Lifecycle Management
PID	Proportional-Integral-Derivative
RL	Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
PaaS	Platform-as-a-Service
DDS	Data Distribution System
CRTP	Crazy Real-Time Protocol
API	Application Programming Interface
k-NN	K Nearest Neighbors
INTELLI	International Conference on Intelligent Systems and Applications
ICAR	International Conference on Advanced Robotics

Chapter 1

Introduction

An Unmanned Aerial Vehicle (or UAV), or usually labelled as *drone* in this thesis, is an aerial vehicle (or aerial drone) that does not require a human pilot to control it [5]. The common believe is that these vehicles are employed only by governments, for military affairs as they are frequently designated for battlefield surveillance, missile launching etc.; however, this technology has been gaining exponential popularity through the years and higher demand in many other areas. The most significant evolution was noted by the increasing interest of the end-consumer, civilians, as these vehicles are accessible to the public via technological stores and can be acquired by anyone. Typically, these consumers use drones for entertainment purposes (i.e., photography, cinematography); thus, there are cases where the drone is acquired as an assistive device to aid consumers with less motor skills on their everyday routines ([37]).

Furthermore, drones have attracted companies due to their visionary utility and are discovered to be relevant and cost-effective systems. One major application is providing monitoring services: target searching and surveillance for security purposes, and others. As a system whose sole fundamental purpose is to fly, companies have adopted more creative solutions for it. *Amazon*, a dominant e-commerce company, has already developed drone's systems for package delivering [10], where it is claimed that a fully autonomous drone is assigned up to a 5 pound-parcel to be delivered in 30 minutes. This is one example of an innovative idea that meets the *Urban Air Mobility* (or UAM) mission ([18]). It is expected that, by the year of 2030, 60% of the world's population will be urbanized ([8]). Due to the emerging new requirements that meet this new reality, the concept of UAM dictates that drones will be a crucial technology to boost this modernization as they will be the prime mechanism of transportation. The main goal is to connect various points of the city by creating an airline network to allow multiple drones (swarms) to perform individualized operations in parallel.

Moreover, considering that a singular drone, being employed in an indoor mission environment or an outdoor one, is an efficient and effective mechanism for completing simple tasks with

success, when having a set of tasks with higher level of complexity (and/or criticality), an individual drone might be insufficient due to the amount of time and resources provided. In contrast, a drone swarm is a network of drones that are allocated to perform connected operations in a more efficient and organized way and is useful when the operator [1] has these kind of tasks at hand.

1.1 Motivation and Problem Overview

When managing one drone, an operator is responsible for performing standard operations with maximum security and precision, i.e., taking-off and landing the drone, and even the everyday tasks, although simple, can provide some complexity. When adding unsafe and critical operations to the task log, the control complexity increases significantly. The operator needs abilities at its peak, full attention and focus when performing these operations, to provide a reliable and stable control. Furthermore, the inclusion of a swarm of drones for performing these critical-safety tasks is another variable that will require synchronization to avoid drone collisions and will increase the operator's mental workload.

Hand control allows operators to remotely send commands to the drones; however, these are critical systems and especially when they are organized in a swarm, operators need to be cautious with the commands they deliver. The Brain-Computer Interface (or BCI) technology is an alternative control mechanism. Fundamentally, these interfaces are recording systems of the human's brain activity that allow data to be collected, analyzed, processed and, ultimately, classified as corresponding drone commands. The main problem in this procedure is the fact that, as mentioned above, the operator must be fully focused to, consciously, decide the next commands of the drone to achieve the desired goal. The mental state of the operator is not consistent as he is prone to stress, fatigue, increasing mental workload and other degrading emotions, transforming once a stable control into an uncertain and insecure one. The operator, when in an unsuitable state to send commands (under the influence of a degrading emotion), can potentially mislead the drone or swarm of drones with the wrong commands. In addition, BCI's require operators to have previous experiences to learn to formulate commands, which implies multiple training sessions. Even so, these command classifications are error-prone and contribute to an unreliable control.

This project is placed in the context of the Research and Innovation (or R&D) department of Capgemini Engineering, a worldwide leader in engineering consulting. As UAM has being gaining more attention such as the automation of drone systems, this company has been making efforts for developing innovative solutions that accommodate the demands of these sectors. This project is one example.

[1] Human operator that has the required training and licensing to control certain types of drones, according with these standards [2].

1.2 Research Questions

Through the analysis of the research problem and its background insights, specific research questions emerge to be addressed by this master thesis:

RQ1 How to estimate the emotional states of the operator using a BCI?

RQ2 How to reduce or avoid the operational impact on the drone or drone swarm when the operator sends a command under the influence of a negative emotion?

1.3 Thesis Statement

According to the research questions mentioned in section 1.2, the proposed thesis can be addressed in the following manners:

TS1 Regarding **RQ1**, the first hypothesis of this work is that: by adopting a (cognitive) digital twin (or DT) [31] to virtually represent the operator and using machine learning techniques and recorded data from a BCI, it is possible to process, filter and predict the mental emotional condition of the operator.

TS2 Regarding **RQ2**, the second hypothesis of this work is that: with the goal of validating the formulated commands, a digital twin is composed by a main component, a cognitive DT, for mental emotion detection and a complementary component, a visual DT, for visual facial expression detection. In this way, the digital twin can detect whether the operator has high mental workload and/or impactful emotions, at both mental and visual levels. Then, it will decide whether the commands produced should or should not be sent to the drone or swarm of drones. The visual DT aims for minimizing classification errors from the cognitive DT, preventing some commands to be sent to the drone or swarm of drones under the influence of a mental negative emotion. Additionally, a Robotic Operating System (or ROS) 2 client node can be used to send the commands to the drone or swarm of drones.

1.4 Goals

According to the context and research problem explained above, the set of sequential goals defined for this master thesis are the following:

1. Getting acquainted with the technologies required for the research, implementation and validation of this topic, particularly learn and operate with the third-party application involved (introduced in section 3.2.1), relevant to the training (of mental commands) and creation of profiles for such testing scenarios;

2. Design and development of a machine learning subsystem, based on a data mining methodology (introduced in section 1.5), for data analysis and building, training and testing a cognitive emotion prediction model (cognitive digital twin), implemented in python, for real-time cognitive recognition based on data gathered by a BCI (introduced in chapter 3);

3. Research and integration of an visual digital twin, built under a machine learning procedure in python, for real-time emotion prediction model based on data gathered by a webcam (introduced in chapter 3);

4. Design and development of a decision making component implemented in python, that will retrieve real-time predictions from both visual and cognitive digital twins and operate as a layer for deciding whether the mental commands classified should be sent to the drones (introduced in chapter 3);

5. Create a connection between this solution and the real drones, implementing a communication channel for sending the required information, resulted from the decision making system, based on a client-server pattern for the ROS2 (whole solution explained in section 3.1.2.1).

1.5 Research Methodology

The main part of the solution that this thesis proposes involves data mining techniques and machine learning, which can be integrated by various methodologies to ease, structure and organize the research plan.

The methodology adopted in this project was one of most common ones: the Cross Industry Standard Process for Data Mining (or CRISP-DM) methodology. Although this one has a clear separation of steps required for achieving the project's goals, the project can also be changed to meet new emerging requirements or, as new knowledge is being gathered throughout the project, there is the need of refining previous procedures, and so, although this represents a sequential methodology, it is also characterized by its agile development.

As illustrated by figure 1.1, this methodology depicts six main tasks, which are briefly explained bellow. For more detailed information and adaptation to this specific project, the integration of this methodology and the detailed implementation are described in chapter 4.

1. **Business understanding:** first step is to assess the project at a more high-level perspective and outline an implementation plan according to the business goals and specified requirements (described in section 1.4).

2. **Data understanding:** second step is to collect data from the required sources, analyze its format and assess the value and contribution of each fetched feature for the dataset as a whole, evaluating as well the quality of the information, i.e., finding missing values (described in section 4.2.1.1 and appendix B).

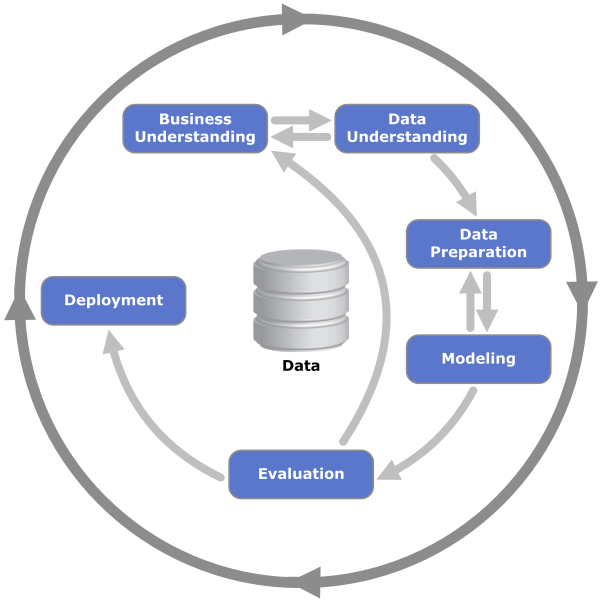


Figure 1.1: Crisp-dm methodology diagram (from [39]).

3. **Data Preparation:** third step is to take the knowledge learnt from the data's insights and select and perform modifications to the dataset to prepare it for the modeling phase. Data preparation is a crucial step that has a direct impact on the outcome and performance of the predictions models (described in sections 4.2.1.2.1, 4.2.1.2.2, 4.2.1.2.3 4.2.1.2.4).

4. **Modeling:** fourth step is to model and manipulate data, using the prepared dataset from the previous phase as an input, to build a prediction model by training and testing a machine learning algorithm. From the performance results of the algorithm's validation, it is possible to assess the model in terms of quality of the predictions and choose the best fitted for the data mining problem at hand (described in section 4.2.1.3).

5. **Evaluation:** fifth step comprehends that, with the chosen prediction model, its outputs and performance evaluation, it is possible to gather new knowledge and align it with the business and data mining goals established in the beginning of the project (described in section 4.2.1.3).

6. **Deployment:** this last step is related to the deployment of the resulted product, maintenance and production of last reviews and conclusions for the overall project (described in chapters 5 and 6).

1.6 Document Structure

This document is divided in six chapters as it follows:

Chapter 1 - this is the current chapter where the context, research problem, research questions, thesis and methodology were presented, in addition of the motivation for this thesis and goals.

Chapter 2 - this chapter (2) is related to literature review of the state-of-the-art technologies that cover this thesis' research area (BCI and DT) and problem, where there are described studies implemented in specific use cases, which can be useful for the thesis solution.

Chapter 3 - this chapter (3) displays the solution as a whole and its individual components at a more high level perspective, showcasing the required materials and experimental setup are explained.

Chapter 4 - this chapter (4) explains the implementation, design choices and all details and efforts for reaching the proposed goals.

Chapter 5 - this chapter (5) outlines the verification and validation plan, where the conducted experiments are described with details and results are shown.

Chapter 6 - this chapter (6) presents the in-depth findings of this research topic from the results registered during the experiments and gives an answer to the research questions placed in the start of the thesis.

Chapter 2

State-of-the-Art

This chapter is related to the state-of-the-art of both digital twin and brain-computer interface technologies that are crucial for the understanding and development of the proposed solution. The first section is the literature review regarding the brain-computer interfaces and their usages on drone management, other research areas, in addition to mental workload analysis and emotion classification based on BCI data. The second section reviews related works where digital twin was an applicable solution for drone problems and for other similar industry use cases. Each study is followed by its paper title, context, research problem, overview of the solution, a more detailed explanation of the solution and validation.

2.1 Brain-Computer Interfaces

Brain-machine interfaces were a great invention phenomenon at the time they started to be conceptualized. Even nowadays, a time characterized by its modern, digital and exponential technological advancements, BCI is not a common topic, off-the-shelf technology, and is being mostly used for research purposes.

BCI's are defined as "*a device that connects the brain to a computer and decodes in real time a specific, predefined brain activity*" [38]. They can use direct or indirect methods to do so, namely by evaluating the nerve cells activity or by assessing the levels of blood oxygen for these cells [38]. Essentially, these mechanisms are built under the concept of mind controlling something, depending on the subject, which, at first glance, most people will resemble it as some science-fiction theory of some sort.

These mechanisms are around since 1924, when appeared the first record of the brain activity with support of Electroencephalography (or EEG) when assessing a subject with a probable brain tumor with needles electrodes [38]. The first concrete device for this type of recording was composed of wires connected to the scalp of the patients. Obviously that these devices evolved

drastically. Currently there are incomparably more refined headsets that are comfortable and effective (for instance, the EMOTIV EPOC+ headset [25]).

This section of the state-of-the-art describes how the BCI mechanism can be implemented to solve some diverse problems. Not also apply to drone-related use cases but also to other industry examples (games and others).

2.1.1 Switching between manual control and brain-computer interface using long term and short term quality measures [37]

Accessibility is a complex topic that is growing in popularity and more systems are designed and developed with this matter in mind. When reading about this subject, it is common to generalize to the portion of humanity with blindness or with severe ocular diseases; however, there are individuals with motor disabilities that also require supportive systems on their everyday lives.

Assistive devices are designed to help people extending their capabilities in performing certain activities they would no longer be able to do by themselves. One crucial point on these systems is how this individual can control them since their motor qualifications is limited. Most supportive systems require a single input form from physical movement [37], such as a finger motion, yet, besides of the physical impairments that restrict the ways of controlling such mechanisms, the quality of these signals degrades, due to inner factors of the individual, i.e., build-up fatigue or inconsistent motion triggered by spams.

A study was presented in Frontiers in Neuroscience conference that showcased a rather innovative way of surpassing the problem mentioned above [37]. The goal of this study was to propose a hybrid brain-computer interface (or hBCI) that would enhance assistive device's functional capabilities by assessing the signal quality and improving the functionality consistency through a mix of cognitive and physical input forms.

This proposal is composed by a BCI which is the cognitive signal provider and a joystick as the physical signal provider. Both of these mechanisms have their disadvantages on account of those internal factors, so the first step was to pick the short-term and long-term quality metrics (for instance, BCI instability, joystick shaking and others) that would more effortlessly distinguish odd signals and, consequently, detect inefficiencies on the system functioning [37].

As illustrated in figure 2.1, the individual equips with both cognitive and physical providers and uses both when performing an operation, thus only the mechanism with the best quality signal gets the control. When performing some operation, the system collects signal data from both providers and quality measurements are constantly being computed considering the metrics defined. When the quality of the signals of the current provider drops below the threshold, that is, the limit value accepted for the quality standard, then the system sends feedback to switch from the current technology to the other. For instance, if the BCI is the controlling technology and the individual breaks his concentration, which leads to a decrease in the signal quality, the system might switch the mode from the BCI to the joystick, turning the physical device the dominant control mode.

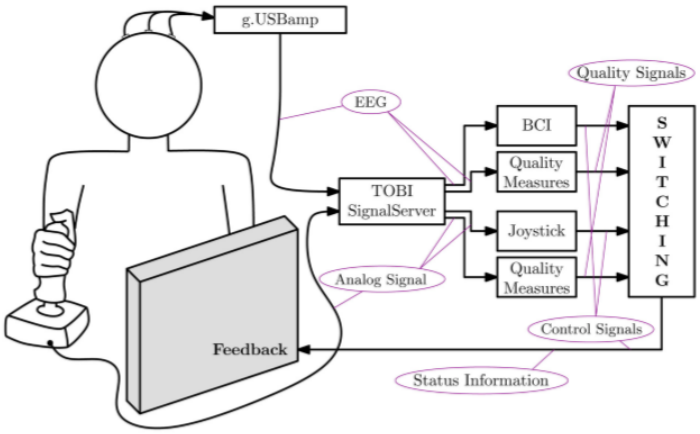


Figure 2.1: Overview of the feedback system (from [37]).

To validate this approach, each individual would have access to a car game, where they would drive it and collect as many coins on their path as they could and surpass the obstacles that occasionally appeared. They would rely on both modes of operation, each at a time.

This study concludes that using a combination of two input signals, cognitive and physical, helped individuals increase their points on the online car game; however, the experiments would need to be applied to real patients and real use cases. The main issue to point out is that this system has into consideration many parameters that are adjusted according to the user's profile, thus, each patient is unique to its condition so one crucial attribute of this system would be its parameters and weighting methods to be flexible with the intention of each caretaker portray their patient in detail. Additionally, this framework can be expanded by applying more quality parameters, weighting methods and more input signals all together.

2.1.2 A Real-time Control Approach for Unmanned Aerial Vehicles using Brain-computer Interface [50]

BCIs have been around for a while and have demonstrated to be a multi-purposed technology when implemented on many sectors beyond the medical scope, one of them being the drone's control systems. There are many studies that give an overview of these systems and many perspectives on their development, evaluation and value; nevertheless, signal processing and classification, which are main data mining tasks that compose these systems, involve common algorithms, for instance SVM, that give acceptable and reliable outcomes.

When applying these same mechanisms on high-speed control systems, the problem becomes apparent. SVM's time complexity is $O(nd^2)$ where n is the number of data points and d the dimension, which means that the larger the dataset, the higher will be the training time with it [12]. In summary, these popular algorithms are not adequate for these types of situations due to their low-speed classification. When handling critical systems like drone's, not also we need to consider having reliable models that produce accurate predictions but also the runtime that they need to do so.

To tackle this problem, it was made a study where the authors developed a classification methodology that, by mixing the Common Spatial Paradigm (CSP) and the Linear Discriminant Analysis (LDA) algorithms, they were able to improve classification precision in real time and validated the approach with a fixed-wing drone use case.

For this procedure, an EEG headset will record the brain activity, following an acquisition protocol. This protocol incorporates a motor imagery (or MI) acquisition mechanism that involves four tasks as illustrated in figure 2.2. Motor imagery tasks are based on visualizing physical movements instead of performing them, in other words, the subject would imagine certain motions like moving one hand. The outcome for distinctly identify each movement and tasks now lays on the algorithms of classification.

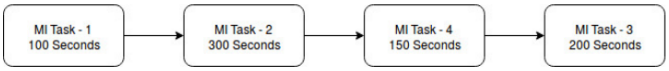


Figure 2.2: Workflow of motor imagery tasks (adapted from [50])

With the data in hand, the classification is a procedure with great importance for this methodology. Choosing CSP and LDA algorithms had a reason behind it. First, the authors identified points of interest: handling noise, frequency band and channel selection [50]. CSP is a feature extraction algorithm that uses spatial filters to differentiate the classes of data more distinctly and that copes well with noise [7]. The authors describe those spatial filters "*result in optimal variances for the classification of two motor imagery signals*" [50]. Secondly, the authors applied LDA algorithm for training models which is a binary classification algorithm that aims to divide the dataset in two classes. Another approach was also explored due to the limitations of the binary classification methodology, because it only considers two classes as a classification discriminant; however, when having more than these two classes, another algorithm must be used. In this case the authors explored the SVM algorithm with the non-linear kernel.

To validate this methodology, a fixed-wing drone was chosen. As soon as the LDA algorithm gave its predictions, the outcomes would be sent to the drone. The experiment involved 14 individuals with no previous experience with the BCI technology and needed to be submitted for training for the usage of the BCI. In a general appreciation, the approach proceeds as the authors wanted as it gives a good overall real time accuracy. The performance evaluation was based on three metrics: the LDA accuracy, the SVM accuracy, the average focus time for some imagery tasks and the maximum reached focus time for each subject. Both accuracy evaluation did not go below 0.77, from 0 to 1, although, one of the individuals had 505 seconds of maximum focus time, which is significantly higher than the others (the second had about 367 seconds) [50], due to the fact that he/she does Yoga, which is a known sport to increase mental and physical stability [9].

2.1.3 Brain Computer Interface system based on indoor semi-autonomous navigation and motor imagery for Unmanned Aerial Vehicle control [47]

As commonly perceived by most people and probably being the main usage for it, drones are featured as a system with significant impact for outdoor activities. Thus, these systems can also be

implemented for indoor tasks, for instance, target searching. This is an important one considering the effect that it can give at the level of an emergency, i.e., searching for someone on a fire situation, inside a building.

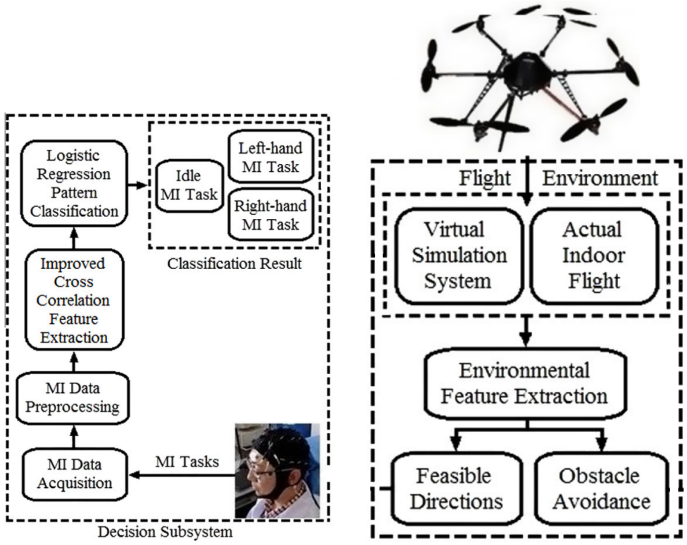
Although this may seem a very alluring solution, problems can arise unexpectedly and add more complexity to the task at hand. The human operator is a crucial piece for smoothing out the procedure, in other words, having a fully automated system might not correspond to the expected results. The human operator plays an important role on the system; nevertheless, the addition of the human interceptor also rises some complications. Depending on the operator's profile, it will reflect on the control system of the drone. The system itself lacks a stable and standard control system.

To fill this gap, a study was made to develop a decision-making system, featuring MI and a semi-autonomous system through a BCI equipment [47], to counter balance the computational costs required to give some level of autonomy for the drone. This enabled the drone to detect and dodge obstacles at its course and provide some feedback regarding which path to follow.

This system is divided in two subsystems: the decision-making and the semi-autonomous navigation. The first one is responsible for fetching the data recorded by the BCI equipment and classify it as commands, i.e., by applying data mining techniques. Figure 2.3a illustrates how this subsystem is organized and the workflow for classifying the data. Two vital phases are the feature extraction and classification tasks. For the feature extraction step, the authors applied an improved cross-correlation algorithm which, for the nature of this context, was the most beneficial in terms of determining more valuable information and provides an effective noise reduction [47]. For the classification task, the authors opted for a logistic regression algorithm due to its "*low model complexity and low risk of overfitting*" [47], converting the output into drone commands.

The semi-autonomous navigation subsystem (figure 2.3b) has a higher level of intelligence since the human operates in it. As part of the experiment, which is detailed above, there are two types of flight: simulated and non-simulated indoor environment. This system collects those environmental variables from both modes so that the drone can dodge the obstacles and give the feedback to the operator in terms of possible directions to follow. These directions are displayed in a real-time video screen so that the operator acknowledge them and perform MI tasks (for example, moving the left hand) accordingly with the direction they want to take. This data enters the decision making subsystem and results in drone commands, which are sent to the drone itself and the loop continues.

To validate this approach the authors performed a set of experiments. The subjects had to go through an initial experience, by following some random MI tasks that appeared on the screen and then performed some simulated flight-related tasks following defined paths to calibrate the system (*MI experiment*). Now for the actual goal of this study, the authors then performed an indoor target searching experiment. The individuals that participated in the *MI experiment*, plus others who had not were included. The aim was to take off the drone from its initial position and find the target that was hidden somewhere on the site which map was unknown to all subjects. This task was split into two parts: (1) all subjects equipped with the BCI and performed the experiment and (2)



(a) Architecture of the decision-making subsystem (adapted from [47])

(b) Architecture of the semi-autonomous navigation subsystem (adapted from [47])

Figure 2.3: Subsystem's architecture.

only the left-out individuals from the *MI experiment* performed without the BCI equipment, via a mobile phone.

Regarding the *MI experiment*, results indicate that the algorithms for feature extraction and classification were highly accurate and effective (lowest value for accuracy was 0.91) and outperformed other comparable algorithms (i.e., CSP) [47]. The subjects that were not present in the *MI experiment* had more difficulties adapting to the MI tasks (the trajectories were not as straight), but they improved their control over time, which indicates that unprepared individuals can learn and handle the system successfully within limited time. Following this narrative, the subjects that were present in the first experiment had smoother trajectories and had a better control of the drone, proving that this system reached its expectations.

2.1.4 Mental workload and Emotion analysis on the human brain

As the main goal of this thesis is based on defining a system capable of discriminating the operator's emotional states, in this section it is presented two studies that analyzed the emotion spectrum of subjects and the impact of increasing mental workload on drone operator's via a BCI.

2.1.4.1 Mental Workload Assessment for UAV Traffic Control Using Consumer-Grade BCI Equipment [13]

Critical systems are defined as applications with strong reliability and quality-persistence as they cope with tasks which might outcome in significant losses when there is an alteration on their internal state [34]. For instance, the failure of systems that handle big volumes of sensitive data

can lead to loss of important data or the failure of safety-critical systems can lead to injuries or even death.

One example of a critical system is a drone. Drones have an important role on many scenarios, because they perform operations that are physically unreachable for humans, thus they are more intelligent systems that require a different control. Without proper control of the flight, there isn't a way of ensuring that the system won't drift from its initial operation in response of some external factor and this is where the human layer has an important role. When performing such critical tasks, these systems must be consistently at its peak of performance, which is exhausting for the human operator as he is prone to fatigue and other natural deterioration that lowers the expected efficiency. One possible solution is implementing a system with multiple Levels Of Autonomy (or LOAs), that is, the automation of a system in different degrees in which each one gradually discriminates the level of support from external entities to perform some operation.

It was published a paper that targeted the implementation of a system with a LOA's framework to support drone's flight control. Since the drone's human operator performance could be affected by accumulated fatigue, the goal was to give the drone some degree of autonomy to decrease the impact of the operator's inefficiencies [13], by separating this autonomy control in different levels that would switch accordingly with the operator's mental state.

Before jumping into how the authors developed this solution, it is also important to mention how they uncovered a method of evaluating the operator's mental workload. *Mental workload*, or *cognitive workload*, is defined as the cognitive effort required from an operator to perform a certain task [17]. The authors opted to use the BCI technology to read brain waves signals to evaluate the operator's mental state. Now concerning the LOAs, the system needs to acknowledge when to switch from one level to another, which implies that the system must adapt itself to the operator or, in other words, it needs to develop prediction capabilities. A prediction model was developed for this purpose with support of the Support Vector Machine (or SVM) classification algorithm.

There were four tasks with different difficulties that contributed for the operator's cognitive assessment: M1 (a drone would follow a path and detect and dodge obstacles), M2 (two drones that would have theirs paths intersected), M3 (five drone flights in which three of them would intersect each other) and M4 (six drone flight that needed the operator's assistance). Due to the ease of successfully completing the task M1, the data resulted from this mission was considered the baseline for future comparisons and, because of the increasing mental pressure, the data fetched by M4 was considered a reference. To validate this approach, ten subjects performed the four tasks above, each one equipped with the *Emotiv* Headset ([25]).

Two of the ten participants only successfully completed one mission, which made the training portion of the model harder due to the classes' imbalance. The accuracy of the predictions of both the sets; test set and validation set; are similar which leads to the conclusion that the model is not overfitted and should make accurate predictions when tested by other subject's test cases. In the future, the author's aspiration is to perform online assessment, instead of offline evaluation performed this study.

2.1.4.2 Detecting Emotion from EEG Signals Using the Emotive Epoc Device [40]

There is a growing urge of software to be supported with facial and voice data collection from users for emotion recognition. Although these systems can classify human emotions, based on these data streams, with satisfactory results of accuracy [40], there is a need to rely on other data streams. Facial and voice data are manipulated by the brain and therefore filtered, while the processing of human brain waves allow the further analysis of inherent emotions [40].

It was developed an approach for emotion recognition based on EEG signals as input, triggered by specific sounds. By means of machine learning techniques, the goal was to classify emotions as levels of *arousal* (or excitement) and *valence* (defined what is a positive and negative emotion). As displayed by figure 2.4, *arousal* and *valence* can be organized as axis and can define multiple categories of emotions; for instance, low levels of *arousal* and negative values of *valence* can mean the brain is in a *sleepy* state.

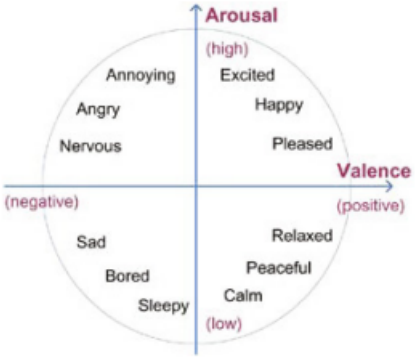


Figure 2.4: Emotional levels based on *arousal* and *valence* (adapted from [40])

Values of *arousal* and *valence* can be computed according to the *alpha* (8-12Hz) and *beta* (12-30Hz) frequency bands of EEG signals, as described in [40]. For this purpose, the authors chose the *Emotiv Epoc* headset as a BCI for data collection, which allows *band power* data stream to be recorded that included the *alpha* and *beta* bands. This study involved six subjects (two males and three females). They were subjected to a sets of sounds [1] (in sessions of five seconds) (from IADS library of emotion-annotated sounds [40]), that would trigger stimuli from multiple positions on the map (figure 2.4). Between these sessions, a 10-second silent rest was inserted to set a *neutral* emotional state.

The authors evaluated two classifiers: Linear Discriminant Analysis (or LDA) and SVM, on the classification of the emotional states of the subject in the following classes: *happiness, anger, sadness, and calm*. The resulting dataset was split in 90% for training the algorithms and 10% for validating the models. Average accuracies for depicting high-low *arousal* and positive-negative *valence* were 77.82% and 80.11% and the best accuracies were 83.35% and 86.33%, both with the SVM (radial function kernel) [40]. Overall, the models were able to distinguish the emotional states without any specific training of each emotion by the subjects.

[1] Sounds that were stimulates high/low values of valence and arousal [40].

2.2 Digital Twin

Nowadays, a digital twin is described as a virtual representation that carries information to realistically behave, change and look like some physical hardware. In fact, it is formally introduced as "*a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level*" [31].

The digital twin concept was not always this straightforward and was refined overtime. The idea was once mentioned in 2002 by Michael Grieves when describing a model for Product Lifecycle Management (or PLM) which, at first sight, seem two very different subjects. The interesting part was that Grieves hypothesis was that there were two models of the same system: one physical and one virtual that carried all specifications of the first one and there were links between the two spaces. Both models would be synchronized and there would exist a data transferring between them [31]. This narrative already has some similarities with the digital twin we know of today as it covers the key points of the technology. To this point forward, this model of PLM became to be referred as the *mirrored spaces model* until it was adopted by NASA in aerospace projects in which they associated with the jargon *digital twin*.

The premise of this technology is somehow restricted to its purposes but also abstract enough to be applicable to many industry cases. It is constantly evolving to serve for each project needs. One variant that derives from it is the digital twin environment [31] with predicting capabilities which is the branch where this thesis and related work fit into. The main goal is to have a simulated environment with high-fidelity physics for training the digital twin and evaluate if it learned from the experiences. The digital twin should gain predictive capabilities to anticipate the hardware's response or behavior in situational events during runtime.

This section of the state-of-the-art is specifically regarding the application of the digital twin environment with multiple industry use cases, targeting the drones as the test subject.

2.2.1 Improving Prediction Capability of Quadcopter Through Digital Twin [35]

When operating complex systems that are connected by many components, due to slight differences in the manufacturing process (it is not possible to have two identical pieces), each part of the system is associated with a range of measurements which represents an *uncertainty* rate [35]. This is one of the main problems when dealing with physical mechanisms: we can think of the final system as a whole; however, we need to consider the irregularities on each of its fragments.

Another concern is monitoring and optimizing these systems' processes [35], which is an increased interest when we have such diversities between the same system and is harder to predict whether the system is functioning in a correct way and where it is failing, if that is the case. In other words, having a standardized monitoring/predicting process, where the performance of the system is expected to be the same for every similar system, as well as for the fault tracking, is not very efficient or even accurate.

A research work emerged from this context with the intent of proposing a framework to improve the estimates of certain measurements of physical systems, more specifically a quadcopter

or drone, by implementing a virtual layer, i.e., a digital twin, that would represent the real device and predict its performance [35].

This approach implies that each piece of the quadcopter has its own prediction models that should learn with each experiment, be updated through time and, ultimately, accurately anticipate some metrics that are valuable to the end-user. An important metric that is highlighted in this study is *endurance*, between many others. As explained above, because we are handling diversities on the same version of a component, another mentioned metric is the *maximum range* [35].

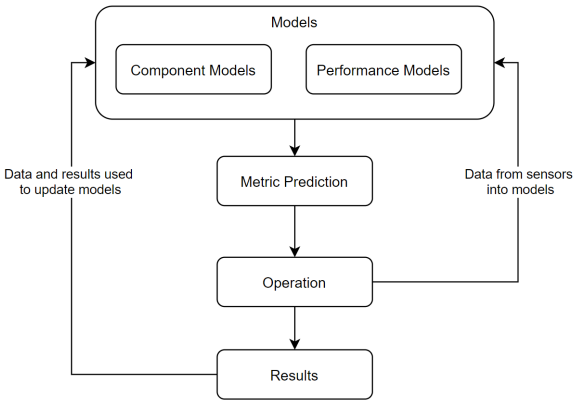


Figure 2.5: Proposed approach to model update and real-time model estimation (adapted from [35])

As demonstrated in figure 2.5, in a closed loop, the models receive real-time information from the sensors of the quadcopter, proceed to predict the wanted metrics and perform the operation. Once the operation ends, new data collected during the experiment is compared to the earlier predicted data and the models are adjusted to become better fitting. It is important to emphasize that each of these components have their own models, which are developed considering many inherent variables and aerodynamic theories.

To validate this approach, the team conducted two experiments to predict the quadcopter power consumption in hover condition. The *experimental flight* is just a normal flight where the quadcopter has some mission details with tasks or requirements it needs to fulfil. While the mission is occurring, more and more information is being fetched by the sensors and is going to be used to refine the component's models hyper parameters, i.e., what they refer as *coefficient calibration*. The other experiment is supported by a *digital flight* that continuously calculates the metrics based on these same requirements and real time data collected by the sensors. At the end, the power consumption values between the *experimental flight* and the *digital flight* are compared.

As the number of experiments increases, the prediction of power consumption improves and the model gets less affected by noisy data, provided by situational conditions from the experimental flights. When the process is completed, as the refined model is now comparable to the quadcopter, it turns into its digital twin.

2.2.2 Reinforcement Learning for UAV Attitude Control [36]

Control systems are mechanisms that are usually required by autopilot technologies. More particularly, when implementing autopilot functionalities on drones, urges the need of considering two aspects: (1) maintaining some support and stability on the flight, which is an inner task, or *inner loop* [36], and (2) an *outer loop* [36], or mission tasks like having flight information provided by sensors, illustrated by the usage of the way-point navigation or GPS navigation.

Most flight control technologies that are required for the first aspect mentioned and responsible for controlling the assets of drones, are based on the Proportional-Integral-Derivative (or PID) control systems [36]. These systems are highly reliable due to their process of systematically computing and updating parameters regarding their components and are closed to the optimal performance when a stable environment is achieved, with no influences from external factors [36]. Nonetheless, when the environment is not stable and have other unknown variables that impact the system, like aerodynamic factors, the performance of this controller decreases significantly. This is a major concern when developing an autopilot system. Realistically, while drones are performing some task or mission, being indoors or outdoors, it is inevitable that external events occur and triggers a change of the current state of the system. PID controllers are not capable enough to adapt to issue this concern.

A study emerged to address this issue and is based on a refined solution that promises delivering more realistic approaches when handling attitude control systems. This study describes a modern simulator, called GymFC, that realistically and accurately implements possible external dynamics supported by physic laws. The goal of GymFC is to provide a training platform to put drone's control systems under test. A virtual representation (digital twin) of the hardware is used to learn from the experiences of the simulation and be deployed to the physical hardware if the desired outcomes are achieved.

As illustrated in figure 2.6, GymFC is composed by many tiers: a digital twin tier, a communication tier and an environment interface tier, which were developed during this study in addition of the usage of the gazebo simulator as a baseline for visualization.

The digital twin layer has a crucial role in this system. Reminding that the main goal of GymFC is to have a realistic and accurate flight simulation so that the drone can virtually learn from the experience and gather knowledge to apply to real case scenarios for the sole purpose of learning attitude control policies, when we have the simulated environment, we will also need the closest digital representation of the hardware to simulate its exact behavior. This layer functions around the collaboration of the gazebo simulator and an aircraft plugin, in addition of its endpoints established for the communication layer. Furthermore, the communication layer provides an indirect connection from the digital twin and the higher layers, such as the environment interface and the agent itself. The last layer connects the rest of the system to the agent.

The evaluation of the simulator was based on training controllers with Reinforcement Learning (or RL) with the support of algorithms from the neural-network family, for instance, Deep Deterministic Policy Gradient (or DDPG), Trust Region Policy Optimization (or TRPO), and Proximal

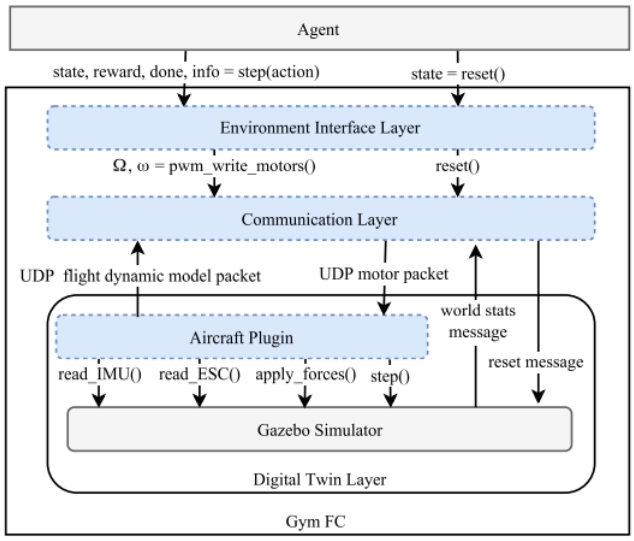


Figure 2.6: Architecture of the GymFC simulator (from [36]).

Policy Optimization (or PPO). The goal is to experiment and discover which one of these RL algorithm has the best performance. To analyze and compare the algorithms, some performance measurements were defined, some of them being the stability of the response until the simulation reaches mid duration.

With the same conditions established for all evaluation simulations, the authors distinguished the PPO algorithm as this had a better performance than PID in almost all metrics and has proven to be a more refined attitude controller than PID itself. In conclusion, this study helped figuring out that RL can be applied to correctly train attitude controllers. Although the digital twin was a small part of this study, it provides interchangeable information to pass from the digital world to the physical hardware and one of the main goals of the authors for the future is to transfer all trained data to the physical hardware and perform more analysis.

2.2.3 Simulation and Digital Twin Support for Managed UAV Applications [32]

One of the sectors where drones are becoming more prominent is the smart city industry. This concept is defined as the modern futuristic cities based on a technological foundation and networking to improve the commodity of the infrastructure for the locals. drones play a crucial role because they can be used for most monitoring activities (surveillance) and other services [48].

However, the complexity for managing these drones at the scale of an entire city is very high. Not also arises the concern about developing stabilized control systems to avoid collisions and other possible unsafe scenarios, i.e., when performing critical tasks, but also about privacy, which has been a dilemma for the population when accepting these systems in their lives and urban air mobility management. Other management approaches target territorial domains for drones where the state of the drone itself should be rather the prioritized information.

At the light of this subject, it was made a study aiming to make the usage of drones a more trustworthy technology for its collaborators [32]. The main contribution mentioned on this paper is the development of a simulation environment built with the Platform-as-a-Service (or PaaS) paradigm, based on a shared platform and with the support of the digital twin technology.

PaaS is almost an intrinsic pattern that developers look for in cloud computing. According to Grohmann, PaaS is a "*provision of a complete platform, i.e., hardware and software, as service*" [33] to "*develop and to provide SaaS solutions or to integrate them with traditional software applications*" [33]. This means that it includes all the infrastructure, development environment and other required tools for teams to develop a system for the cloud [11]. In the context of this work, as illustrated in figure 2.7, the authors developed the management controller which is a cloud service provider that is responsible for managing all client calls. Each physical drone is connected to this service through their official software, which represents the different nodes with isolated runtime, saved on the system's repository. Artifacts like flight data, flight capabilities and drone resources are handled by what they call the *descriptors* [32]. The management controller is also responsible for allocating all necessary data for each client request, i.e., drone application execution. Each drone has its own flight-related information (restrictions, flight plan, etc.) that runs under a unique environment, specially made for maintaining its runtime isolation.

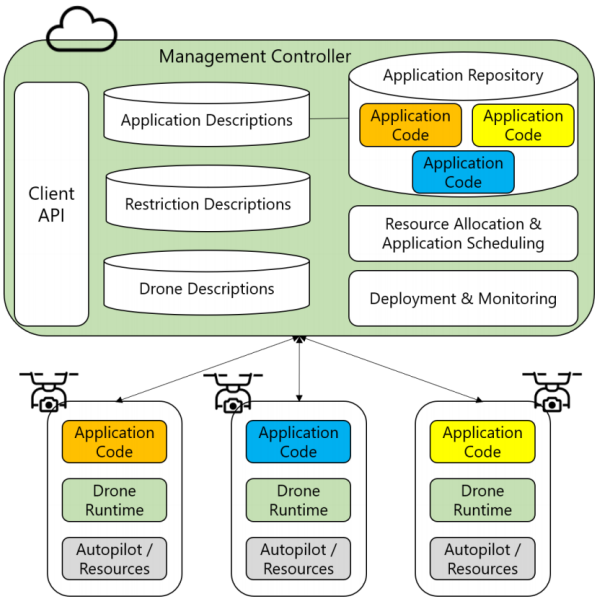


Figure 2.7: PaaS architecture for drone usages (from [32]).

Another component of this approach is the usage of the digital twin technology. PaaS gives the ability to manage and deploy drone applications, yet, part of the problem is finding a way of testing and uncovering the underlying errors that might lead to drone failures in the future. There are some virtual representations: of the drone (what they refer as the *v-drone*) and of the controller (what they refer as the *v-controller*). The *v-drone* acts just like the real hardware and runs the same code. When having a simulation with a swarm of *v-drones*, they connect via the *v-controller* for

management and exchange information through communication channels directly with each other and/or through the *v-controller*. To specifically proceed to the testing part, a new environment is introduced into the system.

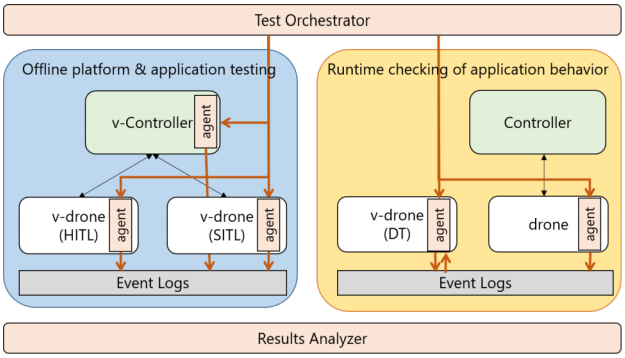


Figure 2.8: Overview of the simulation and digital twin for PaaS (from [32]).

As described in figure 2.8, the *test orchestrator* communicates with an *agent* interface to prepare the test environment for all the entities. Another important aspect is the log feature of the *agent* that allows to save all runtime logs when performing the test and be analyzed by the *results analyzer* to assess whether the drone or the system itself is compliant or if there is any exceptional behavior.

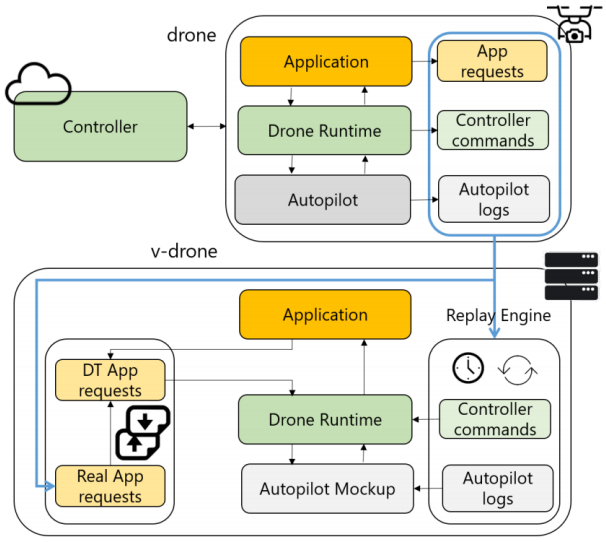


Figure 2.9: *V-drone* configuration (from [32]).

The digital twin configuration is displayed in figure 2.9. Both drones, virtual and real, run with a small lag from each other and the *v-drone* will have a *replay engine* with information captured from the real drone. The *agent* entity will receive information from the application and compare it with the real drone's application data. In case of dissimilarities between them, it means that there is some error that might lead to a failure and the system sends a signal.

To validate this approach, the authors developed a python script containing all configurations for the digital twin as well as for the tests. The drone would perform some scripted movements and the *v-drone* the configurations to equal to the real one. At the end, the assertions and evaluation were performed. Besides the success on the experiments of this framework, the authors are still improving and considering other options, for instance, a predictive digital twin for runtime simulation.

2.3 Summary

This chapter presented the state-of-the-art of two technologies: (1) brain-computer interfaces and (2) digital twin mechanisms, where drone-related use cases and others were depicted.

Brain-computer interfaces are a technology that even though it has been around for almost a century, they are still being discovered as a viable solution for many research areas. These mechanisms connects our thoughts to science by allowing the recording of the human brain waves and by discriminating the factors that compose our cognitive ideas, materializing them into something usable. The human brain is a complex organ that is constantly active, producing considerable amounts of data and we organically interpret it with so much ease that when we try to replicate the process in computational environments, there is an emphasized effort to produce the same accurate outcomes. As shown by the studies conducted with experiments, data analysis is a crucial step to acquire this same successful scenario when applying brain-computer interfaces in a research problem. This is also a task determined by the context of the problem, that is, some studies refer that SVM, a popular classification algorithm for machine learning, is not appropriate to be applied when we have a large dataset and so, other algorithms should be considered. Studies show how a BCI can be used for measuring the mental workload of human subjects.

The digital twin is a small component that runs on the fourth industrial revolution machinery, as it envisions the automation of procedures by providing a digital representation of a system. From a more fundamental level, humans are intelligent beings that have always created refined and innovative systems to meet life's needs for improvement; thus, this cycle is ongoing since human's existence and one concern right now is how to pass on that intelligence to systems. By providing simulated environments, researchers and other entities can validate or perform tests on a system without really compromising its physical integrity. Furthermore, with the same concept, it is possible to monitor and evaluate one's status without being physically present. Creating simulated environments, beyond the complexity of combining high-reliable physics and even replicating the system itself, predicting ones behavior is the main reason for employing digital twins and the most intricate task of all. Applying data analysis techniques for creating learning models is almost an inevitable solution for the data that these models produce and expect to receive (i.e., the predictions for the hardware's behavior).

Overall, these studies give an overview of how researchers have implemented BCIs to intercept our thoughts and organize them as commands for piloting drone's, which is a research problem of this thesis. Ultimately, studies also suggest the application of digital twins for simulating and

monitoring swarm of drone's, retiring the human operator to a more supervisory control role. The application of data analysis and the determination of the most optimal data mining techniques have a crucial and effective role for both research areas. Considering that this thesis' problem and solution focuses on analyzing data collected directly from the brain waves, it is important to study and learn data analysis techniques to discriminate human emotions and what are the desired drone's commands, which also connects to the idea of using a digital twin to intercept, learn and redirect the classified commands to the drone or swarm of drones. In addition, even though these are two prominent research areas, there is not a published study that embraces both BCIs and digital twin for solving drone's research problems or other industry use cases, which makes this thesis more unique and this state-of-the-art as an inspiration.

Chapter 3

Solution Overview

This chapter is divided into two sections: (1) an overview of the architecture of the proposed solution, often mentioned in this thesis as the *decision making system*, and (2) the research methodology adopted during this work. First, design choices and high-level insights regarding each component or subsystem of the proposed solution are explained. Second, the research methodology includes a detailed explanation of the needed hardware and tools to implement and validate the proposed solution, such as the BCI chosen, how was conducted the training of mental commands by the operator and the drone-related hardware, tools and spaces assembled for the validation of the thesis, including the complementary communication subsystem adopted for the project.

3.1 Decision Making System Overview

As mentioned in chapter 1, the core goal of this thesis is the implementation of a system, called *decision making system*, that captures real-time data from brain-wave activity of a drone operator, processes and classifies it according to a set of emotional (mental) states. Additionally, this system should also capture the real-time visual facial expressions of the operator via a camera and identify them as a set of emotional (visual) states. Ultimately, it decides whether the command formulated by the operator, during this period, is valid and formulated while the operator was in a stable emotional environment to be sent as instructions to the drone or swarm of drones. Considering that this solution is integrated in a use case involving real drones, it should be established communication with the drone or swarm of drones through a ROS 2 client node for transferring necessary data for the drone to execute the desired operation.

Since this system will handle with multiple technologies and different types of hardware and data, it was assembled a set of components and subsystems that represent different kinds of operations (figure 3.1): the digital twin subsystem, containing a cognitive and visual digital twins and a decision component; and a ROS2 client-server subsystem, containing a ROS2 client node that communicates with an external server node, that is connected to the project.

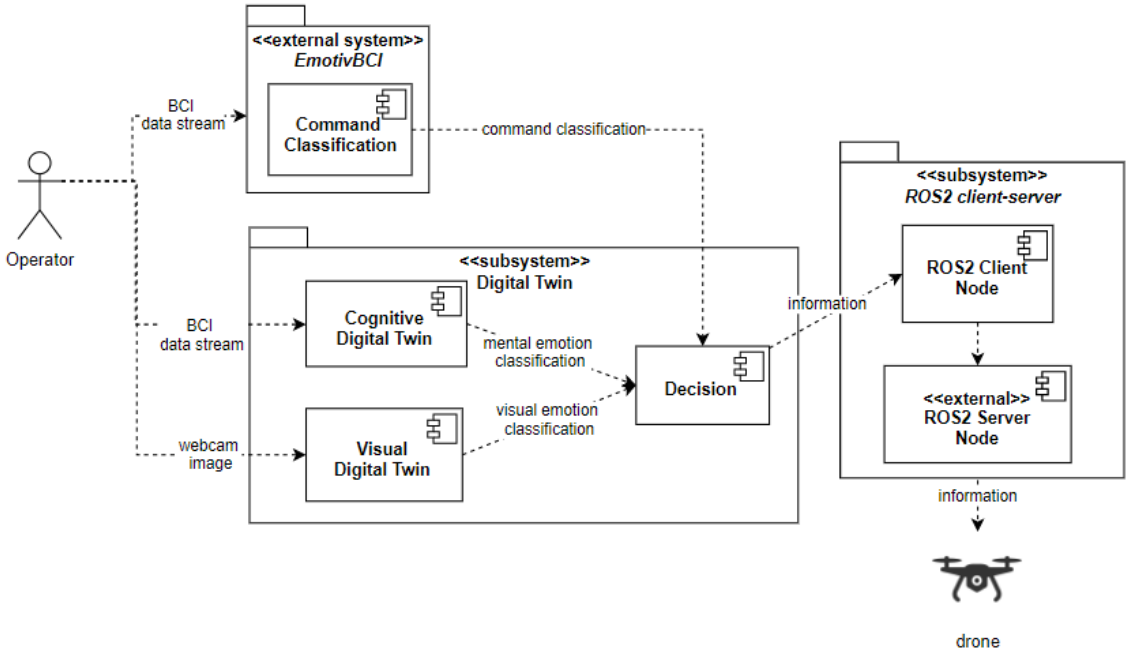


Figure 3.1: Decision making system architecture diagram.

Considering that this system handles large amounts of data, the whole solution was developed in python 3.8, a high-level, object-oriented, programming language in script form that delivers dynamic and robust data processing and provides multiple useful libraries that were often used in this project (i.e., *pandas*, *pyplot*, *numpy* and others). This project was developed in a visual studio code environment and under the Windows operating system.

3.1.1 Digital Twin Subsystem

The digital twin is a technology that aims to mimic a real-world component. Although often associated with physical hardware, this concept can be also applied in this solution. The human being is rather a more complex entity and develop its own traits through time, as well as learning how to react to different scenarios and situations. Machine learning is a useful technique to build the operator's profile since it also requires data and learning mechanisms for behavior recognition. The digital twin can benefit from these technologies to capture the essence of the operator, including its naturally involuntary behaviors and inherent reactions, and associate them with a certain emotional state. In order words, this *decision making system* is composed by a software subsystem called the *digital twin* which aims for learning the behaviors of the operator and recognize its patterns by developing prediction models. Consequently, the digital twin will standardize the operator's reactions and associate them with multiple *positive* and *negative* emotion categories. For this purpose, this subsystem is divided into three parts: (1) the *cognitive digital twin*; (2) the *visual digital twin* and (3) a decision component.

Since the primary goal of this thesis is to detect whether the operator is in a suitable, cognitive, state to send commands to the drone or swarm of drones, at its core, the first component is the main

one for the composition of the digital twin of the operator. This component requires the training of a machine learning algorithm with real-time data from the BCI that will output a mental emotion classification (more details are explained in section 4.2.1). In conjunction with this cognitive digital twin, it was developed an additional component to improve the accuracy and security of the resulting decision of this subsystem: the *visual digital twin*. Although being the core of this thesis and a suitable proposition for the resolution of the presented problem, it is inevitable that the cognitive digital twin will have its uncertainties on predicted emotions or else a perfect model can be associated with an overfitted one. This new component aims for filling the possible failures of this core solution by, similarly to the *cognitive digital twin*, standardize the visual features of the operator through camera footage and classify the current facial expressions as emotional states. For this purpose, solutions from open-source repositories are searched and the best fitted one is included within this system. As opposed to the first digital twin, which gathers knowledge of a certain operator profile, this component is more generic and not exclusive to the operator's character (more details are explained in section 4.2.2).

The second part of this subsystem is the evaluation of the operator's mental and visual emotional states to accept or reject the formulated mental commands to be sent to the drone or swarm of drones. For this purpose, the decision component will be receiving the classified mental emotion and the visual emotion as complementary information. Since this decision concerns a certain point in time where the operator has a certain disposition and mood, both the prediction models should be executed in parallel to ensure that the classifications they provide are from the same periods of time. In this way, for a certain moment, the system should record information (both mental and visual inputs), fed it to the digital twins and the system will evaluate if the operator is in a suitable overall emotional state to formulate valid commands (more details are explained in section 4.2.3). In a positive scenario, meaning that the visual and cognitive digital twins output and overall positive classification, the decision component should acquire the classified command from the *Emotiv* system [1], as depicted by figure 3.1 (further explained in section 4.2.1.1) and translate it into a compatible data scheme, so that the drone can process it and execute (further details on section 4.2.3.1).

3.1.2 ROS2 Client-Server Subsystem

The third and final part of the system is the creation of a ROS 2 client node (using the ROS2 *Foxy* distribution). This represents another additional software layer that aims to optimize the communication between the operator, the system and the drone or swarm of drones. Firstly, ROS is a framework for building robotic-related applications and increases the value of this proposed solution due to its generality of use, providing more diversity and growth space, and ease of adding more drones to the whole system (more details about the overall ROS2 subsystem are explained in section 3.1.2.1 and the concrete implementation on section 4.3). Secondly, another development

[1] External system whose implementation it not integrated in the proposed solution, but rather is needed for the overall functionality of the *decision making system* and is crucial for acquiring the BCI recorded information and command classification.

environment is needed (visual studio community version) and multiple other dependencies need to be fulfilled to install, build and execute ROS2 projects on Windows (as detailed in [42]). This ROS2 node is part of another subsystem of the solution and ultimately connects to an external, pre-existing, component (a server node), that already provides a platform for drone management. This subsystem is implemented under a client-server ROS2 architecture (as further detailed in section 3.1.2.1). For more information about this type of architecture and how it works, check appendix C.

3.1.2.1 ROS2 Server Node

Section 3.1 details the architecture of the proposed solution, comprising all components necessary to achieve the goals of this thesis; thus, defining a ROS2 client node is not the only artifact required to achieve real-time communication with the drone or swarm of drones. This section details the remaining needed components for establishing this communication channel, that are part of an existing subsystem that manages the hardware. As mentioned above, this subsystem was developed under a client-server ROS2 architecture, which means there is a server node that provides services and client nodes that request those services. Considering this, the client node implemented in the proposed solution does not comprises functionalities to send information directly to the drone, but rather connects to this subsystem, requesting certain services, allowing information to be received by it and forwarded to the drone or swarm of drones (for more information about this architecture, see appendix C).

As described by figure 3.2, the server node that provides services for the *decision making system* is called the *base station* and runs under a Linux environment. This node has its own subsystem, containing two additional components: (1) the *micro-ros agent* and (2) the *bridge*.

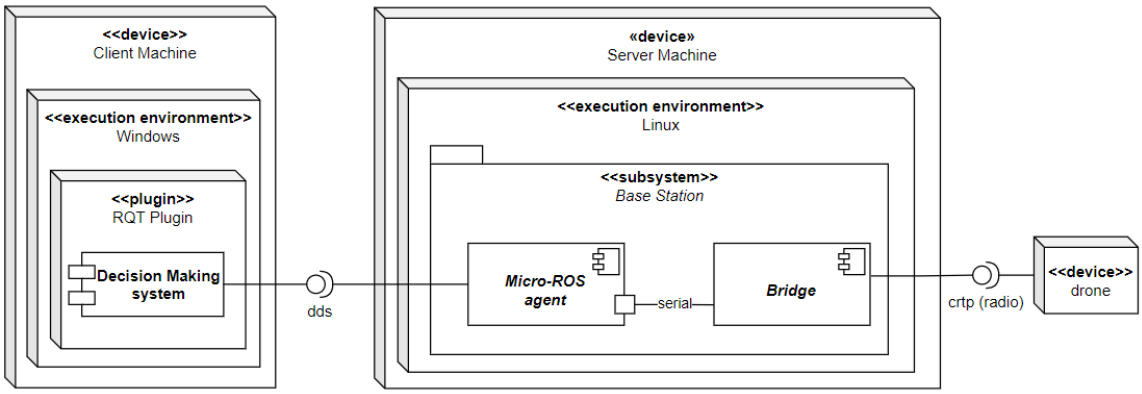


Figure 3.2: ROS2 system deployment diagram.

The *decision making system* will provide a graphical user interface such as a RQT plugin [2], to accommodate all operations required and developed in this thesis. This plugin will be developed with the support of *QT Creator* [3]. This will serve as a control panel for the operator to manage all

[2] ROS' graphical user interface framework for creating adapted interfaces in the form of plugins [49].
[3] Multi-platform editor for building applications and graphical user interfaces [19].

drone executions and to run experiments methodically. When executing any functionality from the plugin, the communication between the server and the client machine is accomplished through the Data Distribution System (or DDS) [4]. Since ROS2 is not compatible with the drone's microcontrollers, a *micro-ros agent* was used to translate all information from the proposed, ROS2-based, system to micro-ros, a lighter and simpler version of ROS2. This *agent* is connected to a python *bridge*, through a virtual serial port.

The *bridge* is the interface that will allow messaging between the entire subsystem and the drone. All data that results from the *decision making system* and that enters into the server machine, through the invocation of services, will be received by this component. Ultimately, the server machine will connect to the *crazyradio PA* (figure 3.3), an USB radio dongle with a maximum range of 1 km, that will allow to send radio messages containing the required information to the drone or swarm of drones. In addition, this communication is possible due to the usage of a python library provided by *Bitcraze* called *cflib*, which is based on the *Crazy Real-Time Protocol* (or CRTP).

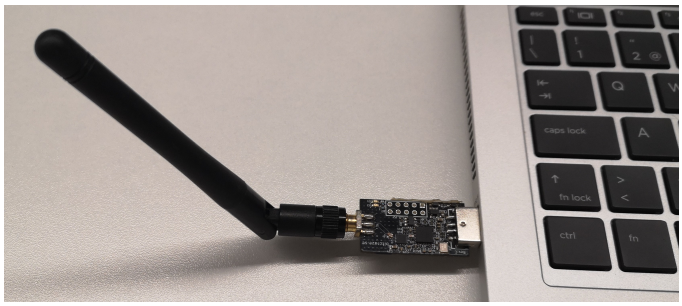


Figure 3.3: *Crazyradio PA* USB dongle.

3.2 Research Methodology

This section is related to the research methodology that was followed in this thesis. It is divided into two themes: (1) the BCI related contents, where it is detailed the chosen headset and how the training of the mental commands was conducted and (2) the experimental setup, including insights of the hardware required for the validation of the proposed solution in a real-case scenario, such as the drone model selected and all its derivative components to execute the experiments.

3.2.1 Brain-computer Interface Headset

The BCI selected for this thesis was the *Emotiv Epoc+* headset ([25]), developed by the *Emotiv* company. It is one of the most popular devices on this market and commonly used on scientific research (i.e., in study [47]), due to its portability, reliability and connection with multiple applications and functionalities that composes a complete platform for brain-activity examination.

[4] End-to-end middleware based on a publish-subscriber pattern for data transferring [27].

Figure 3.4: *Emotiv Epoc+* hardware (from [25]).

This model has a built-in gyroscope based on 3-axis as well as an accelerometer and multiple other artifacts. Regarding the physical specifications, the device has two electrode arms containing seven sensors, two reference ones on each (as showed on figure 3.4). For achieving good coverage and quality signal, the operator will need to position these references on the correct location so that the remaining sensors can be identified. Each sensor has its specific location to be fitted and the headset already organizes them through the connectors on each arm. The operator will only have to place the device on a comfortable position and readjust each sensor. Each sensor assembly is composed by mainly the copper-based sensor and a felt tip. To make the connection, a saline solution is required for dumping these felt tips; otherwise, the quality of the contact will be low or even none.

In addition, *Emotiv* provides multiple software applications for exploring the features of their headsets, which vary according to the user's license and type of hardware. In this thesis, it is used the *EmotivBCI* application ([21]) with the free-of-charge license. This application allows the operator to create multiple profiles with customizable demographics, for instance, level of education, age, gender and others (table 3.1 displays the operator's profile). For each created profile, the operator can perform multiple operations: training of mental commands, training of facial expressions and monitoring of real-time data streams.

3.2.1.1 Mental Command Training

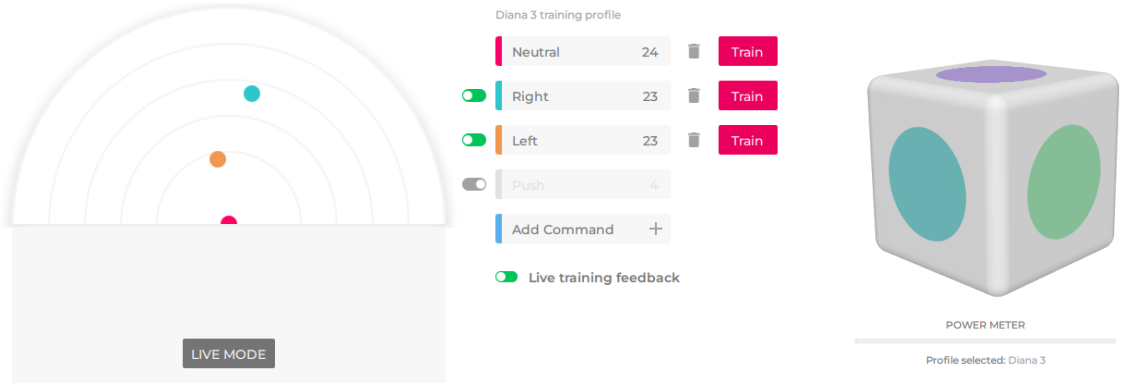
As a fundamental task of this thesis, the operator must learn how to formulate and send commands through the mind. For this purpose, he must create a profile on the *EmotivBCI* with his specific demographics (table 3.1), as described above, that will provide additional information for delivering better results.

For the concrete training of the mental commands, the operator needs to establish strategies for reproducing each desired command whenever he wants. Considering that the main goal is to forward commands uniquely through the detection of the brain activity, it is important to highlight that these strategies will be based entirely on the formulation of thoughts, excluding any kind of

Table 3.1: Demographic's information regarding the operator.

Demographic Key	Value
Gender	Female
Year born	1998
Handedness	Right-handed
Education	Bachelor's degree
Nature of occupation	Engineering
Language Ability	Bilingual
Musical Ability	Single instrumentalist
Country of residence	Portugal
Country of cultural influence	Portugal
Meditation	No

muscle motions of the head, for instance, rotation of the head and facial muscle motions, that would compromise the training of these mental commands. For accurate detection of commands, the application is supported by a machine learning prediction model based on pattern recognition to build a profile and refine it each time the user has a training session. Figure 3.5 displays the panels for mental command training and formulation.



(a) *EmotivBCI* mental command training panel. (b) *EmotivBCI* live mode.

Figure 3.5: *EmotivBCI* mental command training and testing.

According to 3.5a, the application will display a map containing the *neutral* state, which is mandatory to have, and the remaining desired commands. This *neutral* state represents a neutral mental condition, where the operator defines a *relaxed* state, meant to provide no command, typically by remaining still and think of anything. The algorithm of detection will compare the *neutral* state to the remaining commands and define their relative positions on this map. The ideal scenario is to have the commands with as much distinct and separable positions as possible, meaning that the model can discriminate each trained command with distinction. In this work, two mental commands were trained: *right* and *left*, with about 23 training sessions. For each command training, a cube appears in the middle of the screen to simulate the motion of the command (similarly to

figure 3.5b) and the session lasts eight seconds. If the operator is not satisfied with the training session, he can discard it.

The strategy adopted for the formulation of the mental commands that best fitted the operator and that provided a robust procedure for reproducing them anytime was visualizing the cube moving to the desired direction of the command, for instance, if the *right* command was being trained, then the operator would visualize the cube moving to the right.

The main approach here is to train the *neutral* state interleaved with the remaining commands to keep the map updated. Each command is trained sequentially, meaning that only when there is enough confidence for performing the current trained command that another one can be added to the profile. These training sessions were conducted in multiple days, to incorporate different states of mind, adapting according to the operator's disposition and validate if the current strategy is valid enough to be used at whatever time.

For the practice of the mental commands, the operator accesses the *live* mode (figure 3.5b), where the same cube appears and the operator can reproduce whatever command he wants, as opposed to the 8-second training session with individualized command formulation. Before any demonstration or validation of the proposed solution, the operator was submitted to a *warm-up live* mode session, to switch to a control mode mindset and practice the desired commands for the mission.

3.2.2 Experimental Setup

Considering that this system is validated with a real-world use case, where the goal is to control a drone in a live setting, these sections detail how it was arranged the experimental setup, including insights on the hardware and tools that were used, such as the model of the drone and the area of flight.

3.2.2.1 *Crazyflie* Quadcopter

The drone model selected to validate the solution of this thesis is the *crazyflie* 2.1 quadcopter (figure 3.6).

With a dimension of 92 mm of width, 92 mm of height and 29 mm of depth, this quadcopter has a low weight of 27 g, making it a suitable drone for indoor experiments and eases the control of unexpected situations, without significantly disrupting its surroundings. Regarding the flight specifications, with a fully charged battery, the quadcopter holds about seven minutes and needs forty minutes to charge the battery (for more information, check [14]).

3.2.2.2 *Arena* Location

To execute all experiments in a secure setting, it was assembled a physical environment for drone flight demonstrations, composed by a four square meter indoor zone, called the *arena* (figure 3.7b). In the *arena*, a system similar to a GPS is established, using the *loco positioning system* [16], provided by the *Bitcraze* company ([15]), for locating the drone's absolute position in the

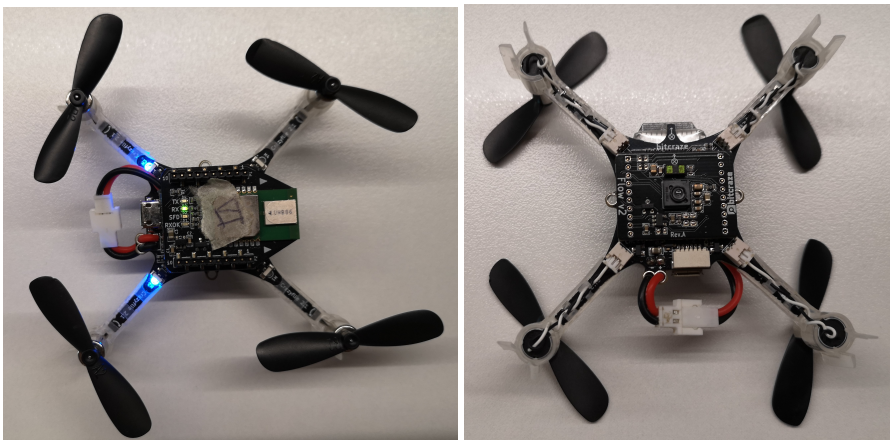


Figure 3.6: *Crazyflie* 2.1 quadcopter top perspective, top and bottom views.

3-dimensional space. In each vertex are positioned a set of *anchors* (figure 3.7a), which serve as reference guides, that communicate via radio messages with the drone's *tags* [5]. Both modules allow the computation of the absolute location of the drone [16].

(a) *Anchor* hardware. (b) *Arena* composition.

Figure 3.7: *Arena* composition hardware.

3.3 Summary

The core goal of this thesis is the implementation of a digital twin of the operator to predict his cognitive state and evaluate whether he is in a suitable state to formulate and send commands to the drone or swarm of drones. Furthermore, two additional software components are proposed to increase the system's reliability and security, crucial whenever the cognitive digital twin has its failures. For this purpose, the proposed solution is developed in a Windows machine with a python

[5] Hardware component of the *crazyflie* drone.

3.8 environment, benefiting from its useful data processing libraries to deliver a robust and quick result in real-time. There are two subsystems in this solution, being (1) the digital twin (containing the cognitive and visual digital twins and a decision component) and (2) a ROS2 communication channel for establishing message sending between the system and the hardware, so that the drone can execute the reproduced command.

Considering that the selection of the BCI headset is a crucial step that will influence the course of this work, *Emotiv Epoc+* device has proven to be a suitable one due to its ease of use and connection to multiple applications of interest. One of them is the *EmotivBCI* application that, among many functionalities, has a section for mental command training and practice, based on a built-in machine learning model for command pattern recognition. The operator creates his profile, inserts his demographic information and creates his mental command profile with three mental commands: *neutral*, *right* and *left*.

The experimental setup includes a single or multiple *crazyflie* quadcopters, a suitable drone model for indoor missions, an *arena*, a specially designed area for flight testing and an end-to-end communication platform based on the ROS2 architecture to deliver messages from the *decision making system* to the drone.

Lastly, for the purpose of establishing a real-time connection between the proposed solution and the *crazyflie* quadcopter, a ROS2 client node is created on the *decision making system* and communicates with the *base station*, i.e., the server machine. This machine contains an *agent* that will translate information from a ROS2 scheme to a lighter and compatible distribution, micro-ros. In addition, the machine connects to an USB radio dongle that will allow information to be sent to the drone through the opening of a *bridge*, based on *cflib*, a python library for radio communication.

Chapter 4

Implementation

In this chapter, implementation of each component/subsystem and step to develop the proposed solution is explained in detail in the following order: (1) the connection with the chosen headset and base code; (2) the digital twin subsystem, including the cognitive digital twin (data acquisition, preparation and modeling), the visual digital twin and decision component (command handling and computation of coordinates) and (3) the ROS2 client-server communication subsystem. Each of these components are associated with their corresponding activity diagrams.

4.1 Headset Connection with the Brain

As a starting point, since the *Emotiv Epoc+* headset uniquely communicates with the *Emotiv* server (or *Emotiv Cloud*), the system receives data captured from this device through an Application Programming Interface (or API). The *Cortex API* ([22]) is a wrapper built by *Emotiv* to aid developers to start their third-party applications by providing multiple tools, for instance the subscription of data streams (see appendix A), and communicates over *JSON* requests and web sockets.

Following the official guide of *Emotiv* ([24]) for the development of such applications, the first step is to choose a suitable license starting from a cost-free API access with the core functionalities, an academic one for access to extra applications and EEG research and a business one for custom software. For this thesis, the cost-free license was selected as it covers all functionalities needed.

The second step is to install the *Emotiv App*, which manages all *Emotiv* devices, allows to create new simulated ones, has quick access to other applications and is responsible for the login on the application, interaction required to access the headset and all its functionalities.

The third and fourth steps are to register what it is called the *Cortex App* and receive its ID, client ID and client secret. Since the purpose is to create a third-party application to access the server's information, this mechanism allows these applications to connect to the user's account without their credentials, by entering the client's generated ID and secret.

The *Emotiv* company provides basic examples to demonstrate the multiple features of the *Cortex API* in various languages (Python, C++, etc.), for instance, how the subscription of data streams is conducted (see appendix A for more information). The connection between the *Cortex API* is achieved through the creation of a web socket client that connects to the *localhost* address, port 6868, with the support of the web socket secure and *JSON-rpc 2.0* protocols, as described in [20]. Requests incorporate the required method of the service, the parameters (as needed) and the id of the request. The response incorporates the result of the function called (if successfully executed) an error code and message (if unsuccessfully executed) and the matched id of the request. For example, for getting information about the logged in user, as described in [3], the request is composed with the following structure:

```
{
    "id": 1,
    "JSONrpc": "2.0",
    "method": "getUserLogin"
}
```

Listing 4.1: *JSON* request example.

```
{
    "id": 1,
    "JSONrpc": "2.0",
    "result": [{
        "currentOSUId":"501",
        "currentOSUsername":"jsnow",
        "lastLoginTime": "2019-11-28
            T12:09:17.300+07:00",
        "loggedInOSUId":"501",
        "loggedInOSUsername":"jsnow",
        "username":"jon.snow"
    }]
}
```

Listing 4.2: *JSON* response example (from [3]).

To access any functionality of the headset, it is required that the user goes through a connection and authorization procedure, followed by the creation of a session. Figure A.1 represents all steps required to initiate a session with the headset. By completing all steps described and adding the credentials to the system, the application will query turned on headsets and connect to the selected one. A *Cortex token* will be generated, which is associated with the license, specific *Cortex App* and the user, required to be sent in most of the service's method calls.

Since the *Emotiv* examples already assembles most function calls in a library and makes the connection with the current headset for the demonstration of core functionalities, the project was built above these code bases with an already functional connection to the *Cortex API* in a synchronized manner.

4.2 The Digital Twin

In the context of this thesis, the digital twin is a virtual representation of the operator and its main goal is to predict, in real-time, his emotional (cognitive and visual) state. This subsystem is composed by 3 components, as described in figure 3.1 and in section 3.1: (1) the cognitive

digital twin (main component); (2) the visual digital twin (complementary component) and (3) the decision component.

4.2.1 The Cognitive Digital Twin

The first component is the cognitive digital twin, which uses data collected by the BCI headset to build a cognitive profile, adapted to the operator. It is the core component of the *decision making system* as it will provide decisive information to ascertain the destination of the command. The remaining components that follow are designated to support the cognitive digital twin and add complementary information for the decision. This approach involves acquiring data from the headset (section 4.2.1.1), submit this information to several transformations (section 4.2.1.2) to create a valid dataset and the creation of a prediction model with the support of machine learning techniques to build the operator's emotional profile at the cognitive level (section 4.2.1.3) (tasks described in figure 4.1).

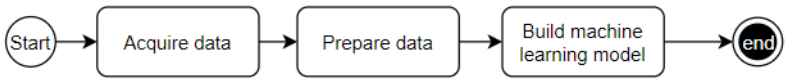


Figure 4.1: Activity diagram showing the workflow of the digital twin.

4.2.1.1 Data Acquisition

After the session is created, it is possible to subscribe to multiple data streams, that represent real-time information that the headset captures and/or computes. This data is composed by individual *data sample* objects, uploaded to the *Emotiv Cloud*, that are continuously sent and match values from a certain point in time. Table 4.1 contains all streams provided by the headset. Each subscription response has an associated id, the specific values of the selected data stream and a numerical timestamp [1] registered by the headset.

Considering all data streams available, in this thesis, four of them are subscribed: (1) the head motion, (2) the band power, (3) the mental command detection and (4) facial expression detection. With the exception of the mental command detection, which is a command prediction resulted from the internal classification model at a certain point in time, the remaining data streams represent a set of features with high value for the creation of the emotional mental profile, that is, values that are directly related with the mental state of the operator. For instance, the rotation of the head can be a sign of discomfort, distraction or even the lack of it a sign of tranquility.

For the classification of the operator's emotional states, a set of emotions were selected to represent positive states (i.e., *calm* and *focused*), meaning that he is in a stable cognitive state to send commands to the drone, as opposed to the negative spectrum of emotions (i.e., *stressed* and *distracted*) that detail an unstable cognitive state and, therefore, unacceptable state to send

[1] The number of seconds that have elapsed since 00:00:00 Thursday, 1 January 1970 UTC.

Table 4.1: Data streams provided by *Cortex* (adapted from [3]).

Data Stream	**Description**
eeg	The raw EEG data from the headset.
mot	The motion data from the headset, captured by the built-in gyroscope
dev	The device data from the headset. It includes the battery level, the wireless signal strength, and the contact quality of each EEG sensor.
eq	The EEG quality of each EEG sensor.
pow	The band power of each EEG sensor. It includes the alpha, low beta, high beta, gamma, and theta bands.
met	The results of the performance metrics detection.
com	The results of the mental commands detection. You must load a profile to get meaningful results.
fac	The results of the facial expressions from muscle motion detection.
sys	The system events. These events are related to the training of the mental commands and facial expressions.

commands. In this work, the same operator simulated all the four emotions, at multiple days, in sessions of 8 seconds, to ensure full concentration on the reproduction of such scenarios.

Similarly to the training of the mental commands (mentioned in 3.2.1.1), the training of the mental emotional states need to follow a certain strategy established by the operator. Even though emotions are natural reactions and associated with specific involuntary or voluntary motions and gestures, these different reactions can be similar enough to induce the prediction model in error. So, the operator needs to have concrete physical and mental dispositions pre-defined for each emotion. For instance, in this case, the *calm* state is defined by low, almost non-existent, motion of the head and facial muscle motions, and the operator is looking straightforward, blinking very slowly, to deliver a calmer and neutral disposition. The *focused* state is characterized by a non-existent or slightly lowering of the head, occasional frowning of the eyes and little blinking, focused on some task (i.e., reading a text without any interruption or sending commands on *live mode* on the *EmotivBCI* application). Regarding the *negative* emotions, the *distracted* one is defined by frequent motions, being facial muscle motions or rotation of the head in high amplitudes and normal amount of blinking, often boosted by scenarios of conversations with other people, looking around at multiple objects for short periods of time in conjunction with intense background noise. Finally, the *stressed* state is characterized by its pre-preparation before recording, physical workout significant enough to trigger a heart race and heavy breathing, in addition to constant physical agitation, small amplitude and often head rotation, facial muscle constant change and frequent blinking.

4.2.1.2 Data Preparation

Each data stream response has its own structure, frequency rate and features, which require multiple steps of data pre-processing. Figure 4.2 gives an overview how data preparation is conducted.

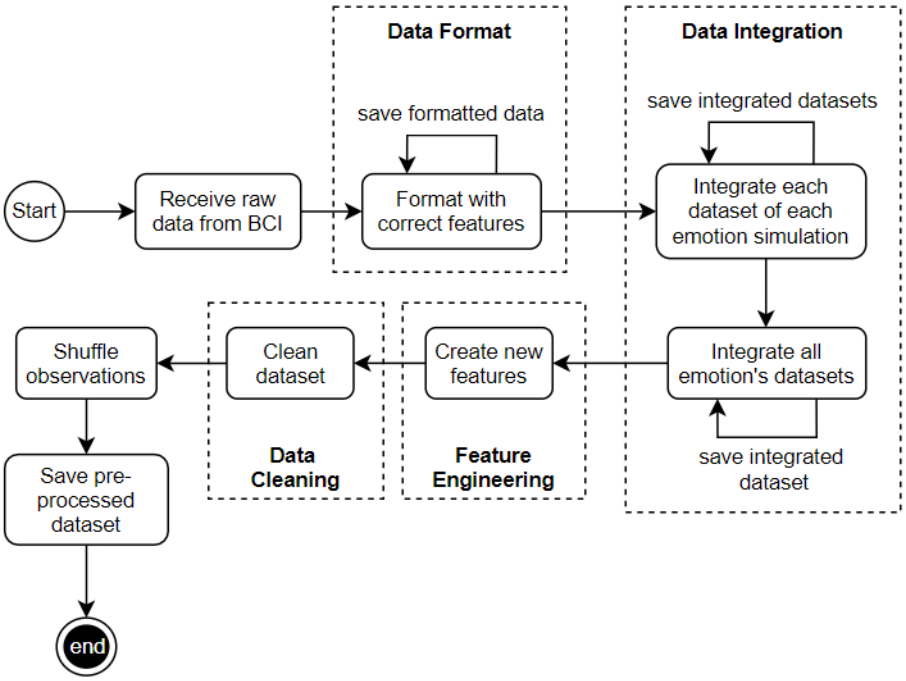


Figure 4.2: Activity diagram showing the workflow of data preparation.

4.2.1.2.1 Data Format

The first step of data preparation is formatting the structure of the received responses. Each *JSON* object is composed by keys, representing a specific feature or information about the object and with a corresponding value pair. These objects are commonly associated with a timestamp, registered by the headset at the time of the recording, the session id and the specified values of a certain data stream, that vary from type to type, which are received as an array.

The valuable information is grouped in these arrays, on each object that is retrieved, but, when handling this data, nested lists add unnecessary complexity. The main goal of this task is to convert *JSON* objects to python *dictionaries*, a common data structure for the collection of key-value pairs and transform these *dictionaries* to a one-dimensional list of objects with the associated feature key, which was implicit on the data. In addition, because the registered timestamp is in seconds, this value is converted to a *datetime* object (readable timestamp), which will compute the instant at a millisecond precision.

Concerning the *motion* data stream, the 12 collected features are presented in table 4.2 with the matching type and description.

Table 4.2: Motion data stream labels (adapted from [4]).

Label	Type	Description
COUNTER_MEMS	number	Increment by 1 for each sample, reset every second.
INTERPOLATED_MEMS	number	0 if sample was received from the headset. 1 if sample was interpolated by Cortex.
ACCX, ACCY, ACCZ	number	X, Y, Z axis of the accelerometer.
MAGX, MAGY, MAGZ	number	X, Y, Z axis of the magnetometer.
Q0, Q1, Q2, Q3	number	Quaternions of the gyroscope.

As exemplified by code listing 4.3, the *mot* array has 12 values that matches the features on table 4.2 in an ordered fashion. Code listing 4.4 is the same object after the formatting transformations.

```
{
    "mot": [
            24,
            0,
            0.636,
            -0.215454,
            -0.671997,
            -0.312256,
            0.965835,
            0.137209,
            0.009766,
            -94.059663,
            -129.799344,
            35.739681
            ],
    "sid": ...,
    "time": 1619707227.7945
}
```

Listing 4.3: Motion raw *JSON* object example.

```
{
    "sid": ...,
    "time": "2021-04-29
        15:40:27.794500",
    "COUNTER_MEMS": 24,
    "INTERPOLATED_MEMS": 0,
    "Q0": 0.636,
    "Q1": -0.215454,
    "Q2": -0.671997,
    "Q3": -0.312256,
    "ACCX": 0.965835,
    "ACCY": 0.137209,
    "ACCZ": 0.009766,
    "MAGX": -94.059663,
    "MAGY": -129.799344,
    "MAGZ": 35.739681
    }
```

Listing 4.4: Motion formatted *JSON* object example.

The *band power* has about seventy features, representing the sensor and band at the level of *theta* waves (4-8Hz), *alpha* waves (8-12Hz), low *beta* waves (12-16Hz), high *beta* waves (16-25Hz) and *gamma* waves (24-45Hz), and is illustrated by object 4.5 (some values were omitted). Code listing 4.6 is the same object after the formatting transformations.

```
{
    "pow": [
            14.925,
            2.526,
            0.387,
            0.411,
            0.525,
            11.512,
        ...
        ],
    "sid": ...,
    "time": 1619707227.8338
}
```

Listing 4.5: Band power raw *JSON* object example.

```
{
    "sid": ...,
    "time": "2021-04-29
        15:40:27.833800",
    "AF3/theta": 14.925,
    "AF3/alpha": 2.526,
    "AF3/betaL": 0.387,
    "AF3/betaH": 0.411,
    "AF3/gamma": 0.525,
    "F7/theta": 11.512,
    ...
}
```

Listing 4.6: Band power formatted *JSON* object example.

The *facial expression* has five features, as described in table 4.3 that present the detection of muscle motions.

Table 4.3: Facial expression data stream labels (adapted from [4]).

Label	**Type**	**Description**
eyeAct	string	The action of the eyes.
uAct	string	The upper face action.
uPow	number	Power of the upper face action. Zero means "low power", 1 means "high power".
lAct	string	The lower face action.
lPow	number	Power of the lower face action. Zero means "low power", 1 means "high power".

Code listing 4.7 is an example of an object of facial expression detection and code listing 4.8 represents the same object after the formatting transformations.

```
{
    "fac": [
            "neutral",
            "neutral",
            0.0,
            "neutral",
            0.0
    ],
    "sid": ...,
    "time": 1619707227.8024
}
```

Listing 4.7: Facial expression raw *JSON* object example.

```
{
    "sid": ...,
    "time": "2021-04-29
        15:40:27.802400",
    "eyeAct": "neutral",
    "uAct": "neutral",
    "uPow": 0.0,
    "lAct": "neutral",
    "lPow": 0.0
}
```

Listing 4.8: Facial expression formatted *JSON* object example.

The *mental commands* are composed by 2 features as illustrated by table 4.4.

Table 4.4: Mental command data stream labels (adapted from [4]).

Label	**Type**	**Description**
act	string	A mental command action.
pow	number	The power of the action. It is a decimal number between 0 and 1, zero means "low power", 1 means "high power".

Code listing 4.9 is an example of a *pull* command and code listing 4.10 is the same command object after the formatting transformations.

```
{
    "com":[
            "pull",
            0.564
            ],
    "sid":...,
    "time":1559903099.348
}
```

Listing 4.9: Mental command raw *JSON* object example.

```
{
    "sid":...,
    "time":1559903099.348,
    "command":"pull",
    "confidence":0.564
}
```

Listing 4.10: Mental command formatted *JSON* object example.

Although crucial for the main goal of this thesis, this data stream is handled in a different way than the rest. The classification of a command not also does not add any value to the dataset but can worsen its accuracy. The number or category of commands should not influence the classification of the operator's mental state, meaning that, once these features are added to the dataset, the model

will rather learn that is more likely the operator is in a certain mood if he sends a certain amount of a command. For instance, if he only sends *left* commands to the drones when he is *focused*, then the model may never output a *focused* mental state at a time of a *right* command. So, this data stream is discarded from the training of the digital twin, but added during the live experiments.

4.2.1.2.2 Data Integration

As mentioned above, each request and response correspond to a single object of a certain data stream for a specific time instant. During data acquisition, these objects are appended to individual, categorized, temporary lists and, when ending the session, these lists are encoded to files as *python* objects. Additionally, each data stream has its own frequency rate, meaning that one type can receive five responses per second, and others thirty. The resulting datasets will significantly differ in number of observations and, since *JSON* objects are data oriented and not time oriented, i.e., each object matches a certain type of values as opposed to a single object containing all data streams for a specific point in time, observations may not match exactly to the same instant.

The main goal of this task is to integrate all data stream datasets to build one. It is divided into two sub-tasks: (1) an *internal* integration and (2) an *external* integration.

Initially, each data stream dataset is read and the *JSON* strings (converted when written to a file) are decoded to a *pandas* object, or dataframe, with the support of the widely used library *pandas*. *Pandas* dataframes and *python* dictionaries are similar structures, thus, for machine learning and data analysis, the first one has proved to be more useful and provides many data manipulation methods for this purpose.

The *internal* integration comprehends that, for recorded data streams (motion, facial expressions and band power streams) of a certain training session specific to each emotion, observations are matched according to their nearest point in time. For this purpose, it was performed a left-joint based on the *time* label, that is common to every observation type, and will select all features of each dataset and add to a single row. The creation of new variables is performed before the next integration and is described in section 4.2.1.2.3. Table 4.5 displays the head of the new dataframe after this first integration (most features are omitted).

Table 4.5: Head of the integrated dataframe for the *calm* mental state.

index	time	COUNTER_MEMS	emotion	...	arousal	valence
0	1619710827794	24	calm	...	0.310255	0.576670
1	1619710827825	25	calm	...	0.310255	0.576670
2	1619710827857	26	calm	...	0.310255	0.576670
3	1619710827888	27	calm	...	0.310255	0.576670
4	1619710827920	28	calm	...	0.370159	-0.253846

As described in section 4.2.1.1, the operator simulates four scenarios matching the four emotional states; therefore, after the *internal* integration, there will be four individual datasets, each

representing data collected during these simulations. At this phase, each dataset has a sum of 91 features: the *time* feature, seventy features from the *band power* data stream, five features from the *facial expressions* data stream, 12 features from the *motion* data stream, two features added during the integration (described in 4.2.1.2.3) and the target variable. The *external* integration aims to concatenate all previously integrated datasets to build a valid one for the training of the cognitive digital twin, resulting in a dataset with 78 400 observations. Furthermore, some features are binary encoded (as described in section 4.2.1.2.3), resulting in a dataset with 102 features (11 new features were added) and the resulting dataset is cleaned (as described in section 4.2.1.2.4), were six features were removed from the dataset, resulting in a sum of 96 features.

Considering that this concatenation is sequential and each dataset is appended at the tail of the last one, each class will not be equally distributed throughout the dataset and can lead to a poor performance from algorithms. Consequently, observations are shuffled at this phase.

4.2.1.2.3 Feature Engineering

There are multiple categorical, not ordinal, features that can have various values, so one-hot-encoding is performed to convert these multi-dimensional features into binary ones. Each category of the specific feature is converted to a new feature of the dataset, represented by a binary value (0 or 1). The features that were encoded were the *eyeAct*, *uAct* and *lAct*, resulting in the following ones: *x0_blink, x0_lookL, x0_lookR, x0_neutral, x0_winkL, x0_winkR, x1_frown, x1_neutral, x1_surprise, x2_neutral, x2_smile*.

Another step of feature engineering is creating new features from knowledge bases. Although the dataset already comprises large amounts of information regarding multiple perspectives, frequency waves recorded by the headset (i.e., *band power*) are in their raw values and perhaps are the features that are directly correlated with the brain-stimuli and activity, which represents high value to the dataset. According to study [40], *alpha* and *beta* waves can be useful to detect emotional states of subjects. *Arousal* (or excitement) (equation 4.1) is detailed by a high power level of beta waves with lower values of alpha waves, whereas *valence* (equation 4.2) is the definition of positive emotions or negative. These values are computed for each observation of the integrated dataset.

$$arousal = \frac{(F3/betaL + F4/betaL)}{(F3/alpha + F4/alpha)} \tag{4.1}$$

$$valence = \frac{F4/alpha}{F4/betaL} - \frac{F3/alpha}{F3/betaL} \tag{4.2}$$

4.2.1.2.4 Data Analysis and Data Cleaning

Considering that, at the integration task, observations are matched by the nearest recorded time and will fill all features accordingly, there will not be any missing values, thus, the dataset at this

point has multiple features with low value and even with unique-values, which does not bring any additional worth to it; therefore, the following features are eliminated, decreasing the size of the final dataset: original features that were previously binary-decoded (*eyeAct*, *uAct* and *lAct*); the *time* which no longer is needed for any operation forward; *COUNTER_MEMS* that represents a simple counter and with no related value for the measures of the brain activity and *INTERPOLATED_MEMS* which has a single value for all observations.

The final dataset has a shape of 78 400 observations, where 20 000 are part of the *distracted* class, 20 000 are part of the *stressed* class, 19 200 are part of the *calm* class and 19 200 are part of the *focused* class, and 96 features in total. From this dataset, the numerical, float, features are the following: motion and some facial expression data streams features (*Q0, Q1, Q2, Q3, ACCX, ACCY, ACCY, MAGX, MAGY, MAGZ, uPow, lPow*) and all band power data stream features (from *AF3/theta* to *AF4/gamma*). Other numerical, integer, features are: the remaining facial expression data stream features that were previously encoded from categorical to binary features (*x0_blink, x0_lookL, x0_lookR, x0_neutral, x0_winkL, x0_winkR, x1_frown, x1_neutral, x1_surprise, x2_neutral, x2_smile*). Finally, the categorical one, which is the target variable *emotion*. The whole dataset has a memory usage of about 58 MB. For more information regarding descriptive, statistical, data analysis, see appendix B. Since each class is represented closely by the same number of observations, the dataset is considered to be balanced.

4.2.1.3 Modeling

Regarding the concrete building of the operator's cognitive emotional profile, the modeling phase has several steps to be accomplished, as depicted by figure 4.3, and further explained in this section.

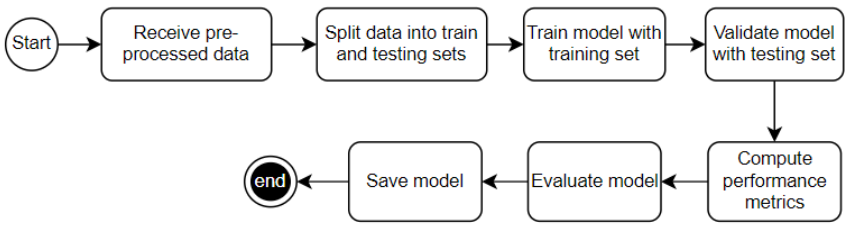


Figure 4.3: Activity diagram showing the workflow of the modeling task.

In this work, data was split into 70% for training the algorithms and 30% for testing. Eight machine learning algorithms (Decision Tree, Random Forest, Support Vector Machine, k-Nearest Neighbors (or k-NN), Naive Bayes, Linear Discriminant and Neural Networks) were evaluated in 4 performance metrics (accuracy, precision, recall and f1-score) and a confusion matrix:

$$accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \tag{4.3}$$

$$precision = \frac{TP}{(TP+FP)} \tag{4.4}$$

$$recall = \frac{TP}{(TP+FN)} \tag{4.5}$$

$$f1-score = 2 * \frac{(precision * recall)}{(precision + recall)} \tag{4.6}$$

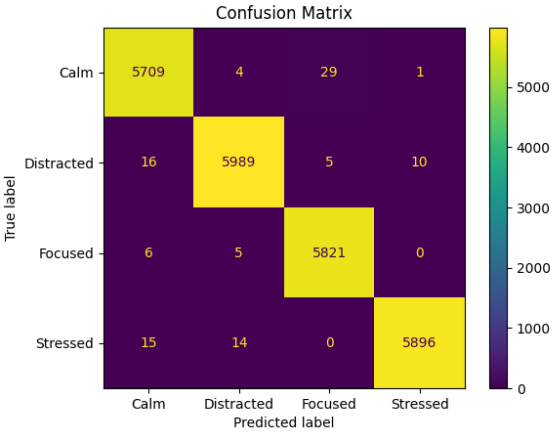
Where TP [2] are the true positives, TN [3] are the true negatives, FP [4] are the false positives and FN [5] are the false negatives. Accuracy represents the observations that were correctly classified (as part of the class and not part of the class) by the prediction model and valuable since the resulting dataset is balanced (i.e., each class has about the same number of observations). Precision is the measurement of how many observations were correctly identified as part of the class. Recall is the proportion of observations that were correctly identified as part of the class in the overall positives of the class and falsely classified ones. F1-score is the conjunction of these both last performance metrics. The confusion matrix is a disposition of the true and false positives and negatives, in other words, it will provide a way of measuring the level of confusion.

Hyper parameter optimization was conducted to choose the optimal or best suited parameter values, specific to each algorithm.

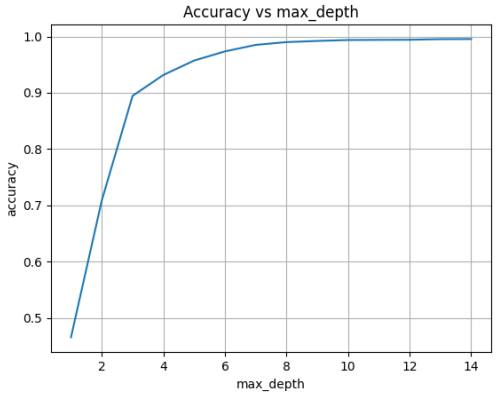
Figure 4.4 displays the resulting confusion matrix and a line chart for evaluating the relationship between the maximum depth of the decision tree and the given accuracy. The classifier identified mostly true positives (diagonal line on 4.4a), which indicates the number of predicted labels that belonged to the correct class. However, there were some observations that were wrongly classified. Inaccurate predictions between the two positive emotions (i.e., *calm* and *focused*) and between the two negative emotions (i.e., *distracted* and *stressed*), and even observations that belong to the positive class that are classified as negative emotions are not critical scenarios. What needs to be evaluated are the observations that belong to the negative classes but are classified as positive ones. 15 observations of the *stressed* class were mistaken as the *calm* class, 16 observations of the *distracted* class were mistaken as the *calm* class and 5 more were mistaken as the *focused* class. From the 23 520 split observations for testing (30% of the whole sample), these numbers of false negatives represents 0,15% of this sample. Figure 4.4b shows that the classifier reaches its maximum accuracy with max_depth parameter of 10, value used for the final modeling of this digital twin based on this algorithm.

Figure 4.5 displays the resulting confusion matrix and a line chart for evaluating the relationship between the maximum depth of the random forest and the given accuracy. Overall, as showed by figure 4.5a, the number of true positives almost reaches all observations. As explained above, what should also be measured is the number of observations that belong to the *negative* classes

[2] Observation where the prediction model inaccurately predicts the class it belongs [30].
[3] Observation where the prediction model inaccurately predicts the class it does not belong [30].
[4] Observation where the prediction model inaccurately predicts the class it belongs [30].
[5] Observation where the prediction model inaccurately predicts the class it does not belong [30].



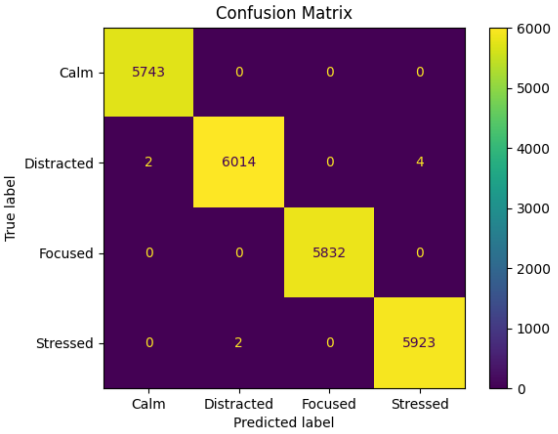
(a) Decision Tree confusion matrix.



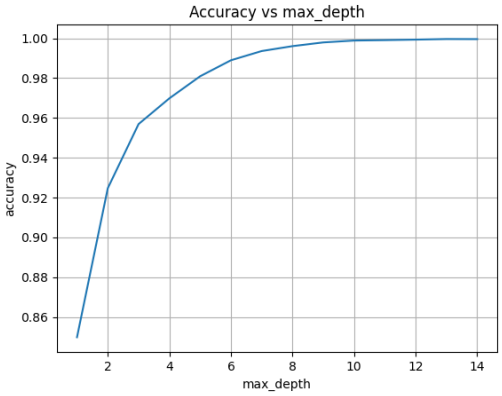
(b) Decision's Tree Line chart accuracy vs max_depth.

Figure 4.4: Decision Tree algorithm Evaluation.

that were classified as belonging to the *positive* class. Regarding the *stressed* class, there are no observations that were classified as part of the positive classes and only 2 observations from the true *distracted* class was classified as *calm*. From the 23 520-observation testing set, this error represents 0,009% of the sample. Similarly to the Decision Tree algorithm, the maximum accuracy is registered with max_depth of 10 (figure 4.5b).



(a) Random Forest confusion matrix.

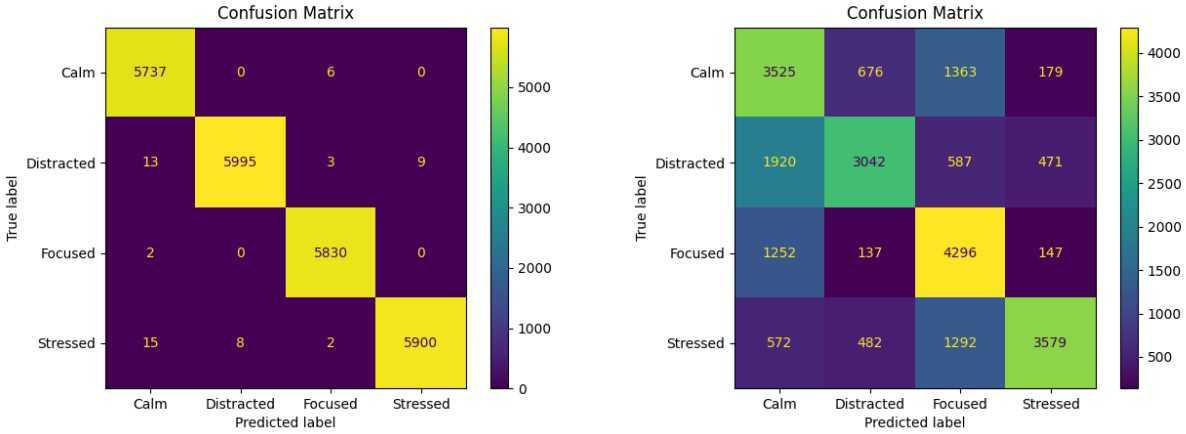


(b) Random Forest's Line chart accuracy vs max_depth.

Figure 4.5: Random Forest algorithm Evaluation.

Figure 4.6 displays the resulting confusion matrices of the testing of k-Nearest Neighbors and Naive Bayes algorithms. Regarding the first algorithm (figure 4.6a), overall the classifier can discriminate the four classes as the number of true positives is significantly high, thus, the *stressed* class has 15 observations mistaken by the *calm* class and 2 by the *focused* class and the *distracted* class has 13 observations mistaken by the *calm* class and other 3 by the *focused* class, which represents 0,14% of the testing sample. Regarding the modeling with the Naive Bayes

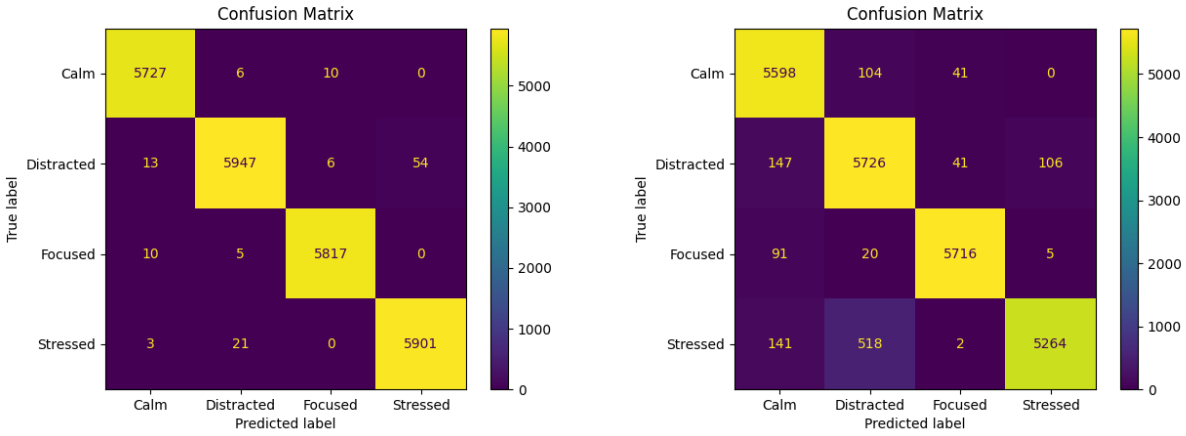
algorithm, figure 4.6b shows that the classifier cannot discriminate the four classes distinctly and has high level of error that, for the same sample, 18,6% of false negatives are from inaccurate classifications of the *negative* class.



(a) k-Nearest Neighbors confusion matrix.

(b) Naive Bayes confusion matrix.

Figure 4.6: k-nearest neighbors and Naive Bayes algorithm Evaluation.



(a) Support Vector Machine confusion matrix.

(b) Neural Networks confusion matrix.

Figure 4.7: Support Vector Machine and Neural Networks algorithm Evaluation.

Figure 4.7 displays the resulting confusion matrices of the modeling of Support Vector Machine and Neural Networks algorithms. Both of the classifiers can accurately discriminate the four classes, thus the Neural Network has a higher level of false negatives and positives than the SVM algorithm. Regarding the SVM (figure 4.7a), 0,09% of the false negatives and positives are observations that belong to the *negative* classes and are inaccurately classified as one of the *positive* classes. For this matter, Neural Networks (figure 4.7b) has a higher value of 1,14% of the testing sample. At this stage, this algorithm was modeled not by the optimizer, but with stable parameters of five hidden layers and 6000 of maximum iterations, because the increase of one of these

parameters can lead to model overfitting.

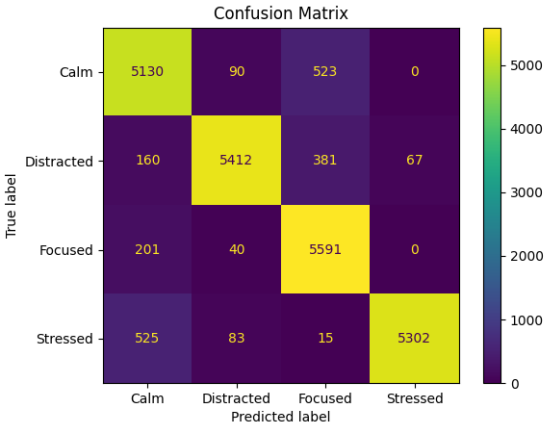


Figure 4.8: Linear Discriminant Analysis confusion matrix.

Figure 4.8 displays the resulting confusion matrix of the modeling of the Linear Discriminant Analysis algorithm. This classifier can discriminate the four classes, although not with the same accuracy than most of the algorithms previously presented. Concerning the number of false negatives for the negative classes, 4,6% of observations that were inaccurately classified as part of the positive classes.

Table 4.6 comprises the computation of the multiple performance metrics (equations 4.2.1.3) for the evaluation of each algorithm. These performance metrics are further used for the comparison between algorithms and selection of the best one for the modeling of the final digital twin. In addition, as mentioned above, confusion matrices are useful for analysing the percentage of false negatives regarding the classification of the *negative* classes. Since *negative* classes mean that, ultimately, the system will not allow drones to receive information, when analysing the false negatives registered under a *negative* classification (observations that belong to one of the negative classes but are classified as one of the positives), it is possible to rank each algorithm in terms of classification errors that have actual impact on the flight of the drone. For instance, algorithms with high percentage of these classification errors mean they can allow higher quantities of commands to be sent when the operator is under stress (or other degrading emotion). The algorithm with the lowest percentage of these classification errors should be considered. The goal is to select the model with the highest possible accuracy and the lowest percentage of classification errors.

Overall, the algorithms have high values of each performance metric. This phenomenon can be explained by the way the training of each emotional state was conducted by the operator. Each emotional state was followed by sets of specific scenarios to ease their replication by the operator. These scenarios comply with very distinct gestures, head rotation and even breathing (as described in section 4.2.1.1), reason why the algorithms can discriminate each class with distinction. In addition, each scenario was reproduced with caution and along with the scenario's guidelines, not allowing unexpected reactions from the operator. Observations resulted from unexpected reactions were deleted from the dataset during the simulation of each emotion.

Random Forest algorithm outperforms the remaining ones in every performance metric and has the lowest percentage of these false negatives with a value of 0,009%; therefore used to model the final cognitive digital twin.

Table 4.6: Evaluation of algorithms

Algorithms	**Performance Metrics**			
	Accuracy	***Precision***	***Recall***	***f1-Score***
Decision Tree	0.995	0.995	0.995	0.995
k-NN	0.997	0.997	0.997	0.997
LDA	0.911	0.916	0.911	0.912
Naive Bayes	0.614	0.645	0.614	0.617
Random Forest	0.999	0.999	0.999	0.999
SVM (linear kernel)	0.994	0.994	0.994	0.994
SVM (rbf kernel)	0.888	0.923	0.888	0.894
Neural Networks	0.948	0.949	0.948	0.948

4.2.2 The Visual Digital Twin

The second component is called the *visual digital twin* and its main goal is to provide complementary information to the decision component and boost the emotion recognition of the operator by adding an extra layer of software intended to analyze his visual appearance and facial expressions and include an additional input to the decision, validating or invalidating the classification from the cognitive digital twin and lowering the impact of its inaccurate classifications on the drone or swarm of drones.

The webcam will have access to the real-time image of the operator and the visual digital twin will output an emotion at each time instant. This component is adapted from an open-source project [29] that creates a prediction model with the Convolutional Neural Network algorithm, called the *mini-Xception*. This model was trained with the FER-2013 emotion dataset ([46]), which contains 35 887 observations (grey-scale images) matching the following classes: *angry*, *disgust*, *fear*, *happy*, *sad*, *surprised* and *neutral*. This model has an overall accuracy of 66% and figure 4.9 represents the confusion matrix for the emotion recognition.

According to the results [29], there are high levels of false negatives, for instance, 20% of observations belonging to the *fear* class are wrongly classified as *sad*, 26% of observations that belong to the *disgust* class are classified as *angry* and 19% of observations belonging to the *sad* class are classified as *neutral*. Similarly to the evaluation of the cognitive digital twin's algorithms based on their confusion matrices, *positive* classes that are confused with *negative* ones do not have significant impact on the flight of the drone; however, *negative*-class observations that are inaccurately classified as *positive* can have impact on the drones. The *positive* classes are the *happy* and *neutral* ones and the *negative* classes are the remaining. 14% of observations that belong to the *angry* class were classified as a positive class. 7% of observations from the *disgust* class were classified as a positive class. 16% of observations that belong to the *fear* class were classified

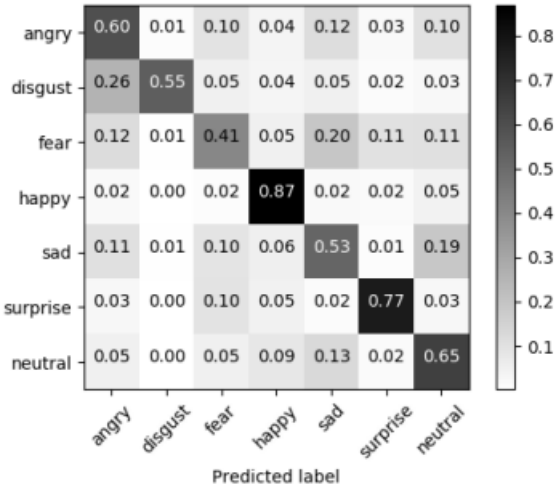


Figure 4.9: Confusion matrix resulted from the *mini-Xception* model (from [29]).

as a positive class. 25% of observations from the *sad* class were classified as a positive class (highest level). 8% of observations that belong to the *surprise* class were classified as a positive class. Overall, the *mini-Xception* model can discriminate the seven classes. The integration of both projects is explained on section 4.2.3.

4.2.3 The Decision Component

The third component of the Digital twin is the decision component. It aims for determine if the operator is stable by mentally and visually evaluate his state and decide whether the command formulated at the time should be sent to the drones. Previous sections (4.2 and 4.2.2) describe how the system classifies the emotional states of the operator by establishing two digital twins that are fed with cognitive and visual inputs, as both mental and visual classified emotions are required for the decision making. This procedure is further described in figure 4.10: (1) evaluation of the emotional state of the operator; (2) computation of the final mental command and (3) computation of coordinates.

Both digital twins should be outputting their predictions that concerns certain periods of time, so both models should be running concurrently. For this purpose, there will be two processes running in parallel by means of python multiprocessing [28]. These processes are spawned in this main decision component, where resources are allocated, including individualized memory regions, and will execute independently from each other. Considering that both components can run for unspecified periods of time and need to be recursively outputting predictions, it was established a shared memory region where all processes involved will have access to. Since the digital twin is built-in in this component, the shared variable is written to every time the visual digital twin has a new classification and the decision component will read it, avoiding data inconsistencies (i.e., only one component writes and another one reads).

This component comprises all procedures regarding the digital twin, for instance data acquisition, preparation and processing. Primarily, data acquisition is performed in sessions of two

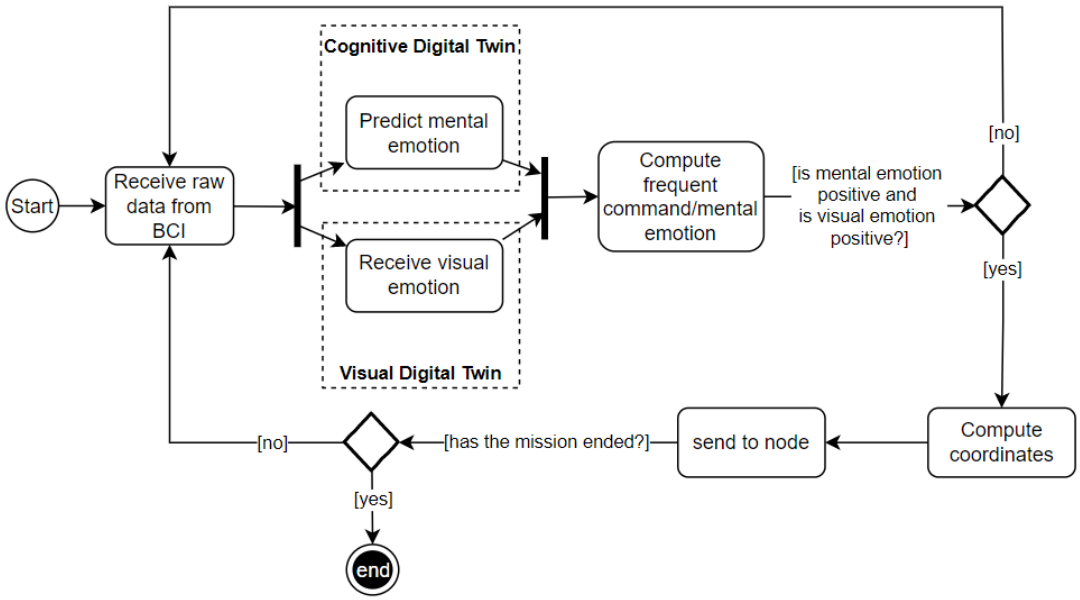


Figure 4.10: Activity diagram showing the workflow of decision making.

seconds, where all data streams are being recorded, followed by data preparation where the newly collected datasets are formatted and integrated. Since data subscriptions, as mentioned in section 4.2.1.2.2, have high frequency rates, a 2-second session can result in a 10-observation dataset, which, ultimately, will correspond to ten predictions by the classifier. Computation of the most frequent cognitive emotion and command and their confidences are conducted at this stage. The decision component will iterate the matrix of classified commands and mental emotions, count the occurrences of each class and output the most frequent one; then compute the average confidence of these observations.

Regarding the evaluation of the cognitive and visual states of the operator, the decision component retrieves the last written value of the visual digital twin from the shared variable and proceeds to the decision making task. At this stage, the decision component will comprehend that if the cognitive state of the operator is within the *positive* spectrum of classes, meaning that is either a *calm* or *focused* states, and the visual predicted emotion is either *neutral* or *happy* states, then the decision is *positive*, i.e., the command should be sent to the drones, as opposed to a *negative* decision, which means that no command should reach the drone or swarm of drones. Another possible scenario is that the predicted command is *neutral*, which is not a motion command and, therefore, these ones will not be sent as well.

4.2.3.1 Computation of Coordinates

Predicted command categories from the built-in classifier of the *EmotivBCI* application are not included in the drones' vocabulary, meaning that *left*, *right* and others are conceptual types of commands and drones do not have the intelligence to measure and interpret them. As the final task

of the decision component, these mental commands are translated into a compatible data scheme that can be acknowledged by the hardware: value of relative coordinates.

$$d = (c * i) * e \tag{4.7}$$

Equation 4.7 represents the distance to be travelled by the drone, where c is the command confidence of the classifier upon the prediction, varying between 0 and 1; e is the average emotion confidence of the cognitive digital twin upon the prediction, varying between 0 and 1 and i, a constant variable that represents the absolute increment and has value of 0,25. This increment value will set the maximum distance the drone can travel, that is, 100% of increment of position (both prediction confidences are 100%) means that the drone will shift 0,25 meters in a certain direction.

Each time the decision component checks all requirements for sending a command, it could settle an increment of position by 100%; however, if the cognitive and visual digital twins wrongly classify the emotional state of the operator as one of the *positive* classes, where he is affected by a *negative* emotion, the drone will receive a 100% of increment position and will move the whole 0,25 meters. In the context of execution of critical tasks, where the operator should be as meticulous as possible, 0,25 meters can have a significant impact on the hardware and the surrounding environment, so, to minimize those impacts, the decision component considers two more variables (confidences upon the predictions) for the computation of the distance. Consequently, considering that these confidences are based on the probability of the observation belonging to a certain class, uncertainties are included, to prevent those drones shift the whole 0,25 meters of distance when the classifiers are inaccurate.

$$t = \frac{i}{v} \tag{4.8}$$

The execution time of the command (*left* or *right*) is calculated (equation 4.8) according with the constant increment of position (i), of 0,25 meters, and the constant velocity established (v, of 0,25 meters per second. All command operations will be performed within one second. Regarding standard operations, such as landing and taking-off, the run time is set to two seconds for achieving a smooth execution.

The classified command will reveal the direction and sense of the operation. Figure 4.11 represents the two axis (x and y axis) in which the drone can be positioned. If the command is classified as *left*, the value increment of y will be the computed distance as positive and x will be 0, while if the command is classified as *right*, the value increment of y will be the computed distance as negative and x will be 0.

4.3 ROS2 Client Node

In previous sections, the subjects regarding the classification of the mental and visual emotional states of the operator are covered, and how both prediction models work together synchronously,

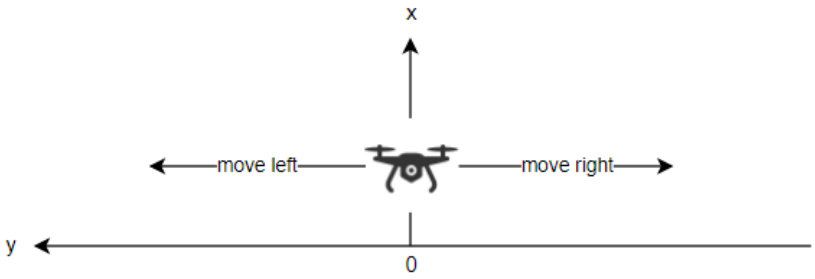


Figure 4.11: Above perspective of axis of the drone and corresponding command coordinates.

contributing to the decision of whether the system should send the reproduced command and compute its matching coordinates. Lastly, it was established a communication channel between the system and, ultimately, the operator, to the physical hardware, through the implementation of a ROS2 client-server architecture.

As described in chapter 3, *nodes* are components of software that represent a specific functionality and communication is achieved by *services*. In this context, the *client node* is implemented as a gateway of the decision component and will send synchronous requests to the *service node*. These requests should obey to the structure of the requested service, where each variable should be filled with its corresponding value (see appendix C for more information). The *service node* is called the *base station*, which is mainly the machine that manages one or more drones and that actually forwards the information to them. The implementation of this approach is divided in 3 tasks: (1) the creation of a ROS2 package, (2) creation of custom service and message files for the service-client and (3) creation of an RQT plugin interface.

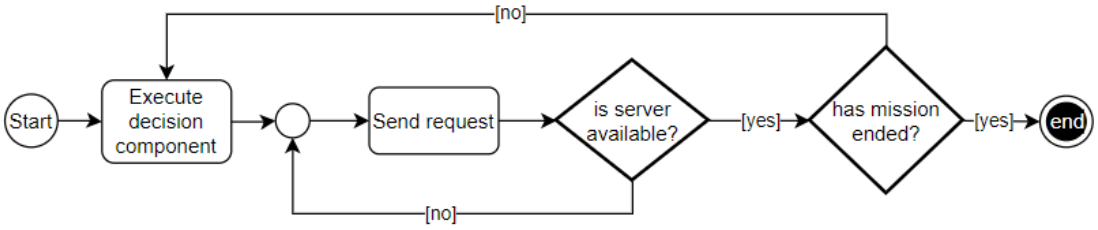


Figure 4.12: Activity diagram showing the workflow of sending commands.

First, a ROS2 package is created by sourcing the ROS2 installation module to access all ROS2 command options and choosing package creation one. All commands mentioned forward are described in appendix C.2 and C.3. Considering that the system creates custom services with specific schemes, those files need to be accessible by any ROS2 package of the project; therefore, two of these packages are created, one for the main code of the *client node* and necessary python files and the other for the creation of these services.

To achieve a basic flight, the necessary services are: the take-off (4.13), the landing (4.12) and a motion (4.11) command. Ultimately, the *client* is a class containing all requests for each service. For sending a request, it was implemented a loop to check if there is any response from the service. The decision component will compact all coordinates and information necessary to

send a *goto* request; thus, the remaining operations are directly sent by the RQT plugin, described in section 4.3.1.

```
float32 x
float32 y
float32 z
float32 yaw
float32 duration
---
int8 ret
```

Listing 4.11: *GoTo* service scheme

```
float32 height
float32 duration
---
int8 ret
```

Listing 4.12: *Land* service scheme

```
float32 height
float32 duration
---
int8 ret
```

Listing 4.13: *Take-off* service scheme

4.3.1 RQT Plugin

The main goal of this thesis is to simulate a live environment for the control of a single or multiple drones, which requires the operator to be facing the hardware; however, for the purpose of facilitating the execution of experiments, described in chapter 5, and the experience of the operator managing operations, it was created a graphical user interface plugin using the RQT framework (as showed by figure 4.13).

This plugin is divided in two sections: (1) the execution of test levels and live control, including the standard operations of landing and taking-off the drone and (2) the training of the digital twin.

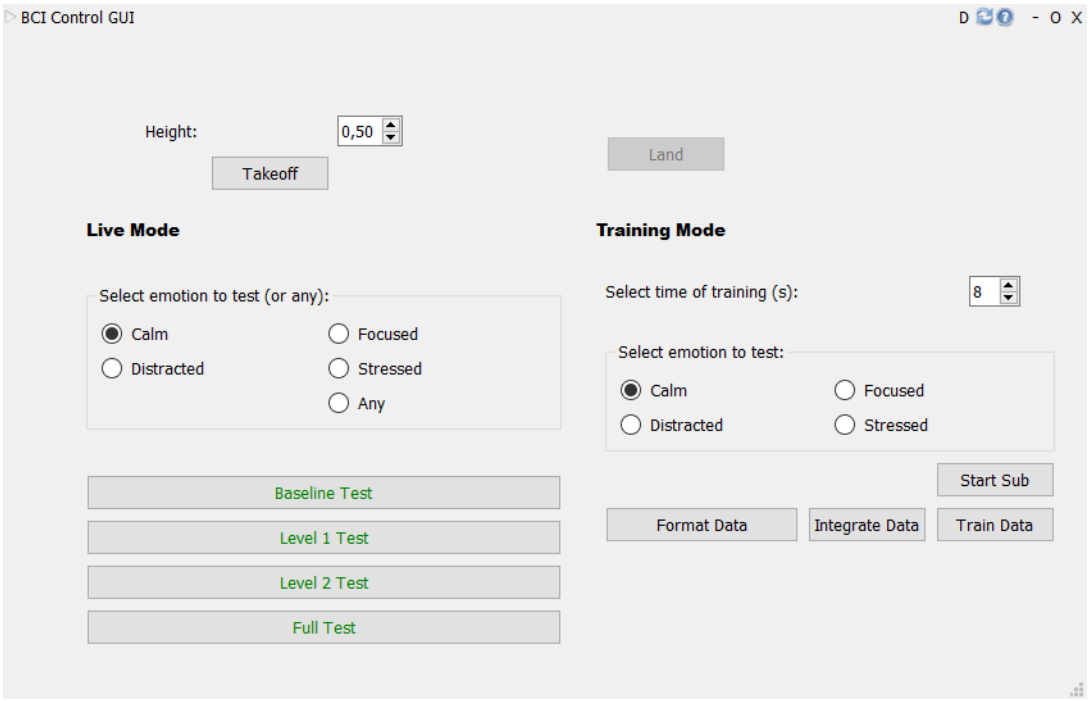


Figure 4.13: RQT plugin user interface.

The height of the take-off is defined *à priori* and has a default value of 0,5 meters. For executing any operation, it is mandatory that the operator takes-off the drone first. As mentioned above, for the purpose of performing the experiments, which imply the isolated simulation of each of the trained emotions, the user interface displays some functionalities for testing and, according to the type of test and simulation, results are written to specialized files. For an independent live drone control, the operator should click on *any* and on *full test*.

After the operator is done with the simulation, he can land the drone by pressing the *land* button, which sends a landing request by instantiating the land service with 0 of height.

Regarding the second part of the interface, the operator can refine the cognitive digital twin by sequentially selecting the button for data subscription, formatting, integration and training.

The underlying commands can also be reproduced through a command prompt by manually entering specific command lines. Considering that this system deals with the *take-off*, *landing* and motion (*goto*) operations, each one of these activities have their own command lines and variables, as depicted in the above service schemes. When one of these operations are performed, informative prints are written in the client (the one that runs the RQT plugin) and service consoles. All these command lines and prints are displayed in appendix C.3.

4.4 Summary

In this section, implementation details were explored and explained. For the development of this approach, multiple components were created to maintain a synchronized and sequential procedure for the resolution of a decision, according to the mental and visual emotional states of the operator.

First, the primary data source of this thesis is via a BCI, intended for the capture of multiple measurements, for instance, the rotation of the head. The connection with the *Emotiv Epoc+* headset (described in section 4.1), the selected device of this work, is achieved through the *Cortex* API by establishing web sockets and sending/receiving *JSON* requests and responses.

The first developed component of this system is the cognitive digital twin (section 4.2), the core solution of this approach, which is a digital representation of the cognitive profile of the operator and aims for predicting his emotional state. For the implementation of this component, machine learning techniques were applied. Three data streams provided by the headset are subscribed in sessions of eight seconds and in four different scenarios where the operator had to simulate the four mental states. Two of them are considered the *positive* states, i.e., suitable mental states for sending commands (*calm* and *focused*), and the remaining two are considered the *negative* states that, as opposed to the other classes, represent unsuitable states for sending commands (*distracted* and *stressed*). Multiple steps are performed in data preparation to compose a valid and single dataset input for the training of the prediction models, for instance, data is formatted, removing nested lists and matching the key-value pairs; the different datasets are integrated resulting in a single dataset for each simulated emotion, where each observation is matched with the corresponding feature values by their nearest point in time; new features are created based on knowledge of the brainwave frequencies as well as one-hot-encoding is performed to transform categorical features into

binary ones and data is cleaned from their worthless or unique-value features. Multiple algorithms were used for training a temporary digital twin and the resulting models were evaluated in four performance metrics (accuracy, precision, recall and f1-score). Random forest was the selected algorithm for training the final twin as it outperforms the remaining algorithms and has a low percentage of false negatives and positives regarding classifications belonging from the *negative* classes that are classified as the *positive* ones.

The second developed component was the visual digital twin (section 4.2.2), an additional software layer, that is considered to be a complementary component of the digital twin, aiming for classifying the visual facial expression of the operator and, in conjunction with the cognitive digital twin, minimize impacts consequent to inaccurate classifications. For this purpose, it was made a research and integration of an open-source project in python that created a prediction model based on a Convolutional Neural Network ([29]). This model can classify observations as *angry*, *disgust*, *fear*, *happy*, *sad*, *surprised* and *neutral* states with an overall accuracy of 66%.

The third and main developed component was the decision component (section 4.2.3) that comprises all procedures for the evaluation of the cognitive and visual states of the operator and resolution of the destination of formulated commands. The cognitive digital twin is run on the main process whereas the visual digital twin is executed in a different process to achieve independent parallelism. Data is recorded for 2-second sessions, pre-processed and fed to the both digital twins to output a set of classifications. The most frequent command, mental emotion and matching confidences are computed. The decision component will then evaluate whether the cognitive and visual states belong to the *positive* classes and if the command is a motion one. In these situations, relative coordinates are computed according to the confidences and direction of the command or the command is ignored.

The fourth and final component is the implementation of a ROS2 client node (section 4.3) for compacting the information and sending it to the drone. The node implemented on the project is called the *client node*, which will send request, and the node implemented on the machine that is linked to the drones (*service node*) is called the *base station*. These two nodes communicate over services, which are created to adapt to the drone's operations (take-off, landing and motion commands). The *client node* sends the services with the compacted information from the decision component to the *service node* and this one will forward the information to the drones. For the control of all functionalities of the project, it was created an RQT plugin.

Chapter 5

Results and Discussion

In this chapter, results and their discussion are presented. First, it is explained the set-up and how the experiments were conducted to validate the solution and to demonstrate its value for an isolated emotional environment and in a live environment. Secondly, one digital twin, trained with the Random Forest algorithm, is put under test in a multi-level experimental manner, discussed, evaluated in multiple performance metrics and then assessed in a free live session.

5.1 Experiments

To evaluate the different impacts of the solution, functionalities were split in an incremental, multi-level, manner that goes from the lowest, the baseline experiment, to the highest level and, therefore, the solution, to emphasize its value and impact on securing a stable control environment for critical systems as a drone. These experimental levels are described incrementally bellow:

1. ***Baseline test***: defines the current state of drone control without the support of emotion recognition, where the operator is directly connected with the drone and each reproduced command, being correctly or incorrectly classified by the BCI, is forwarded to the drone. As described in chapter 4, section 4.2.3.1, the computation of the coordinates is supported by the confidences of the resulting visual and mental classifications. Since machine learning is not incorporated in this test level, coordinates sent to the drone are solely computed depending on the direction of the command, at a 100% of increment. The goal at this level is to establish the lowest solution in terms of accuracy, security and reliability, giving a perspective of what the minimum valuable solution is.

2. ***Level 1 test***: represents the implementation of the core functionalities of the solution - the cognitive digital twin. This digital twin receives the real-time collected information from the BCI during the 2-second time period of data acquisition and will predict the operator's mental state and make decisions. As a first level experiment with a prediction model, the

computation of the coordinates is like the *baseline test*. This test level aims to demonstrate that the implementation of the solution, at its core, already increases the stability of the flight.

3. ***Level 2 test***: represents the implementation of the core functionalities with the addition of the computation of coordinates as described in chapter 4, section 4.2.3.1. At this test level, the goal is to validate that the power in which the subject sends a command has its effects on the increment of the position of the drone as unsure commands or even incorrect classifications have minor impact on the drone.

4. ***Full test***: represents the finalized solution, having all the same functionalities as the *level 2 test*, including the visual digital twin, supported by a camera. As the last level of the solution, the goal is to demonstrate that the addition of the visual digital twin increases the security of the flight as it prevents commands to be sent under incorrect classifications by the cognitive digital twin.

Except for the *baseline test*, which gives no importance to the mental state of the subject, each test covers the four mental states the models have been trained to classify (*focused*, *calm*, *distracted* and *stressed*) individually, each one with sessions of 8 seconds. The subject had to be put under the same circumstances in which he used to simulate the four emotions on the training phase, described in chapter 4.

Ultimately, and fulfilling the core goal of this thesis, a last experiment is conducted in a live setting where the subject has full and free drone control and space to feel whatever emotion. As a way of evaluating the long-term solution in a unpredictable environment, the subject is recorded for two consecutive minutes and is occasionally interfered with some diversion, i.e., a phone starts to ring, in a certain timestamp. The goal is to evaluate how both digital twins can classify rapid emotional switches of the subject.

All procedures of this thesis are conducted by one operator, meaning that the training of the commands, the training of the digital twin and validation of the solution is associated with a unique operator as the goal is to prove the adaptability of the models and gathered knowledge in accordance with the unique human profile of the operator.

5.2 Results and discussion

This section is related to the results and discussion of the validity of the proposed solution as well as the demonstration of the value it brings to not also the resolution of this thesis problems and the research questions but also the optimizations it brings. The model trained with the Random Forest, selected by its high accuracy, is used and discussed in this section. Multiple metrics were created to evaluate the solution, for instance, the success and error rates and sent commands by incorrect classifications on a negative emotion simulation.

The success rate represents the ratio of correctly classified observations from the total number of classifications of the sample and it measures how well the classifier can identify a certain emotion in an isolated environment. This metric can be computed according to equation 5.1, where E is the number of correct predictions of a certain emotion and S the total number of predictions of a certain session.

$$sr = \frac{E}{S} \tag{5.1}$$

As opposed to the success rate, the error rate represents the ratio of incorrectly classified observations from the total number of classifications of the sample. This metric can be computed according to equation 5.2, where A is the number of incorrect predictions of a certain emotion and S the total number of predictions of a certain session.

$$er = \frac{A}{S} \tag{5.2}$$

Regarding the negative emotional experiments, it is performed further result analysis, depicting the incorrect classifications, i.e., positive emotion detection during the simulation of either the *distracted* state or *stressed* state. For the purpose of this analysis, the collected observations are: all positive predictions detected during these sessions, the number of neutral commands from this fraction, the number of all sent commands from this same fraction and the number of positive emotions, predicted by the cognitive DT, where the visual DT classified the facial expression as a negative emotion, in conjunction with the non-neutral commands. Just a reminder that, as mentioned in chapter 4, there are multiple conditions to be checked to a command be sent to the drone, for instance, the both digital twins must predict an overall positive emotion and the most frequent formulated command must be a motion one, not neutral. In this section, when summing the number of sent commands, internally all these conditions are checked. The sum of sent commands represent moments in time where both classifiers were flawed. Unlike this metric, the last sum represents the number of commands that were prevented due to the addition of the extra layer of the visual DT.

5.2.1 Digital Twin

Given the environment set-up described in section 5.1, the number of observations per emotion and per experimental level are displayed in table 5.1.

5.2.1.1 Success and Error Rates

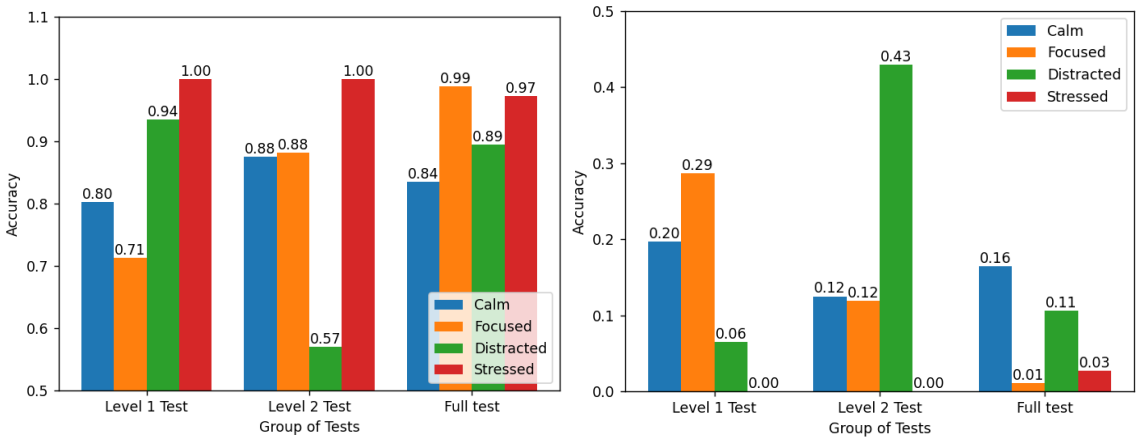
Given equation 5.1, for the *calm* state, the highest accuracy of the classifier was 87,5%, for the *focused* state a 98%, for the *distracted* state a 93,5% and for the *stressed* state a 100%. The lowest success rate of the cognitive digital twin, for the *calm* state was 80,3%, for the *focused* one, 71,3%,

Table 5.1: Number of Observations per Emotion and per Level

Emotions	**Group of Test**		
	Level 1 Test	***Level 2 Test***	***Full Test***
Calm	142	120	85
Focused	94	101	90
Distracted	124	135	104
Stressed	134	120	112

for the *distracted* state a 57% and *stressed* state a 97,3%. Figure 5.1a [1] displays all the values for each emotion and per experimental level [2].

Given equation 5.2, the highest and lowest error rates for the *calm*, *focused*, *distracted* and *stressed* states were 19% and 12,5%, 28,8% and 1%, 43% and 6,4% and 2,7% and 0% correspondingly. Figure 5.1b displays all the values for each emotion and per experimental level.



(a) Success Rate.

(b) Error rate.

Figure 5.1: Success and error rates per emotion and per test.

Even with a high average of 87,2% of success rate for detecting the subject's mental states, the most accurately classified emotion was the *stressed* state. The difference between them can be due to the distinct way the model is trained in this segment, which involves more physical movement to provoke agitation, rather than a low on motion condition on the remaining ones.

In addition, lower success rate depicted on *level 1* for the *focused* state classification can be explained by the different background noise and movement between the training and test phase, which cause the subject to deviate his attention, explaining the occurrences of *distracted* classifications during this period. Analyzing the next test levels, the success rate is no lower than 80%, which is explained by the calmer environment. As opposed to this situation, the lower success rate

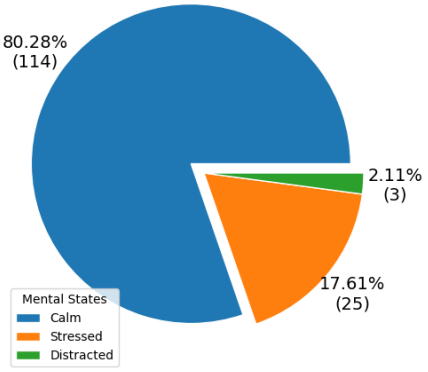
[1] Both bar charts have different scales.

[2] Implementation efforts in each experimental levels are not being evaluated. In this context, the experimental levels are different temporal sessions that allow the computation of accuracy on discriminating each emotion on different times.

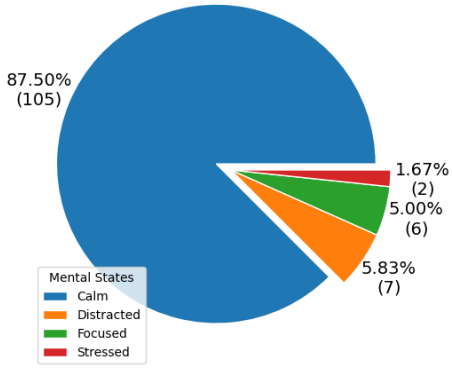
on *level 2* for the *distracted* state classification can be explained by the lower amount interference or other diversions derived from background movement which led to short occurrences of focus by the subject.

5.2.1.2 Overall Predictions

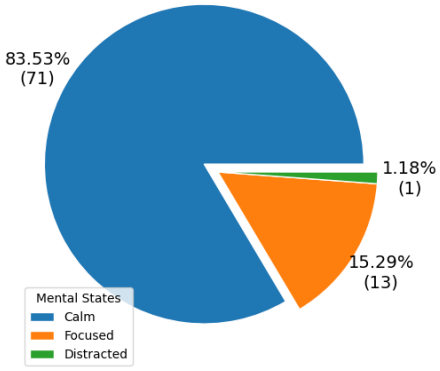
All predictions registered for the simulation of the *calm* state are displayed in figure 5.2. Regarding the first experimental level, from the 19,72% of error rate, 17,61% of observations were classified as *stressed* and the remaining 2,11% as *distracted*. For the second level (figure 5.2b), the 12,5% error rate corresponds to 5,83% of *distracted*, 5% of *focused* and 1,67% of *stressed* incorrect classifications. Finally, from the error rate of the last experimental level(16,47%) (figure 5.2c), the digital twin inaccurately classified 15,29% of observations as *focused* and 1,18% as *distracted* states.



(a) Level 1 Test.



(b) Level 2 Test.



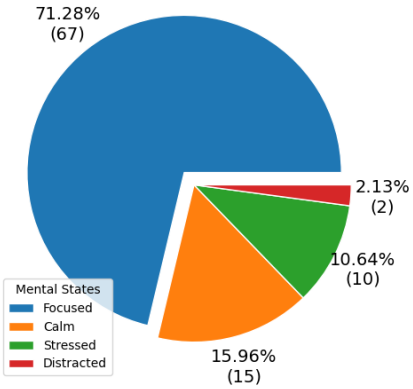
(c) Full Test.

Figure 5.2: *Calm* state overall predictions.

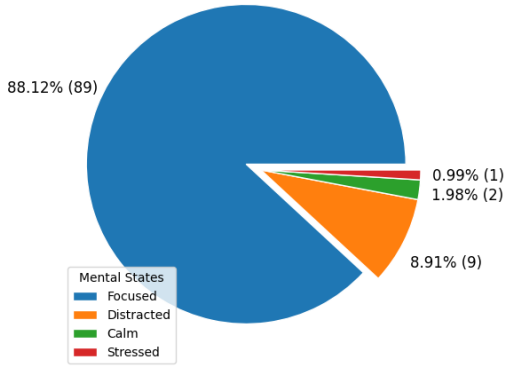
Although having a high average success rate of 84% and a low average error of 16%, in most experimental levels it was detected both the *stressed* and *distracted* cognitive states. If the operator is in a suitable cognitive condition and the cognitive DT incorrectly outputs a negative emotion,

the drone will simply not receive any coordinates as the final decision is not to send any commands at that time, which represents no associated risk to the control of the drone. The positive *focused* state was also detected in the last experimental level, which brings no instability to the drone because the command was to be sent regardless of which positive emotion was detected.

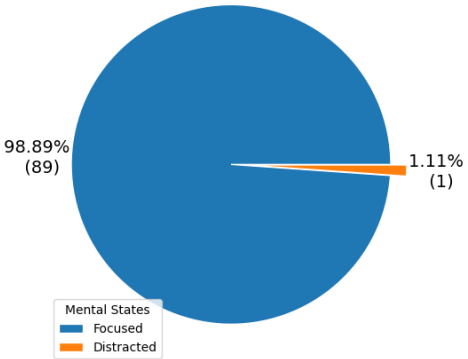
Regarding predictions registered in the simulation of the *focused* state, displayed in figure 5.3, from the 28,72% error rate of the first experimental level (figure 5.3a), 15,96% of observations were classified as *calm* state, 10,64% as *stressed* and 2,13% as *distracted*. As for the second level (figure 5.3b) with an error rate of 11,88%, 8,91% of observations were classified as *distracted* state, 1,98% as *calm* and 0,99% as *stressed*. In the final level (figure 5.3c), 1,11% of error rate corresponds solely to the detection of *distracted* state.



(a) Level 1 Test.



(b) Level 2 Test.

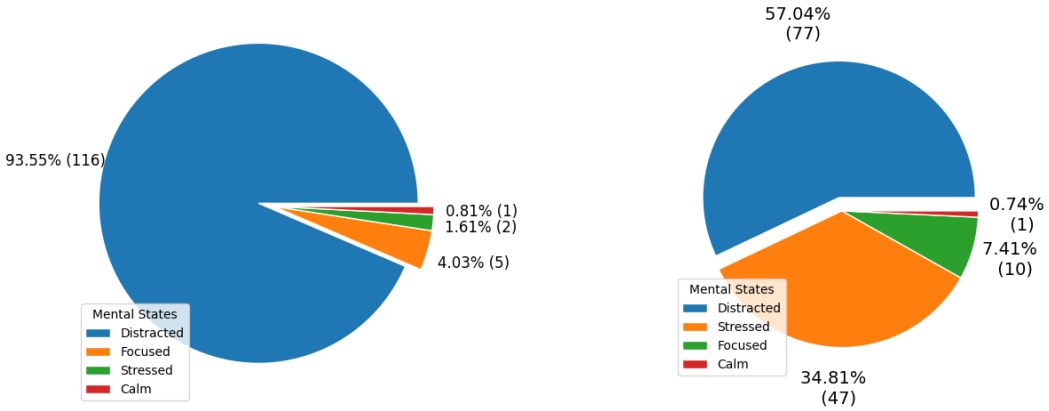


(c) Full Test.

Figure 5.3: *Focused* state overall predictions.

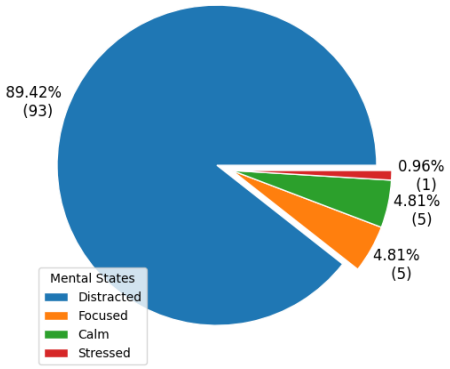
Similarly to the *calm* validation, due the fact the *focused* state is considered to be a positive emotional state, in the worst case scenario, meaning that the cognitive DT inaccurately classifies the operator's mental condition, the system will not simply allow the reproduced commands during this time to be sent to the drones, representing no risk to the operation. Confusions between positive emotions will not mark any differentiation because both represent suitable cognitive states of the operator.

Concerning the simulation of the *distracted* mental state, figure 5.4 shows all detections during each one of the performed experiments. For the first experimental level (figure 5.4a), from the 6,45% error rate, 4,03% were *focused* classifications, 1,61% were *stressed* and 0,81% were *calm* detections. With one of the highest error rates registered from all experiments (42,96%) at the second level experiment (figure 5.4b), 34,81% of observations were classified as *stressed*, 7,41% as *focused* and 0,74% as *calm* states. In the final experimental level (figure 5.4c), from the 10,58% of error rate, 4,81% of observations were classified as *calm*, 4,81% as *focused* and 0,96% as *stressed*.



(a) Level 1 Test.

(b) Level 2 Test.



(c) Full Test.

Figure 5.4: *Distracted* state overall predictions.

Having a negative emotional detection represents a turning point to the decision of the broadcasting of commands, so inaccuracies in negative scenarios from the cognitive DT can bring instabilities to the control of the drones. When the operator is in an inadequate condition, the system must prevent these commands to reach the drones as they can be misleading and it threatens the secure environment once created. In this simulation, the subject was doing some other tasks but the cognitive DT, although in a small ratio of observations, outputted positive emotions which represents a flaw and can lead to repercussions. In the *level 2 test*, with a high error rate, there was a confusion between the *distracted*, target value, and the *stressed*; however, as both values are

considered to be negative emotions, the reproduced commands will not be forwarded to the drones and, therefore, will not have any impact on the security of the environment established.

As for the simulation of the *stressed* state, with the highest success rates of all simulations, only at the last experimental level (figure 5.5), from the 2,68% error rate, 0,89% of observations were classified as *distracted*, 0,89% as *focused* and the other 0,89% as *calm*.

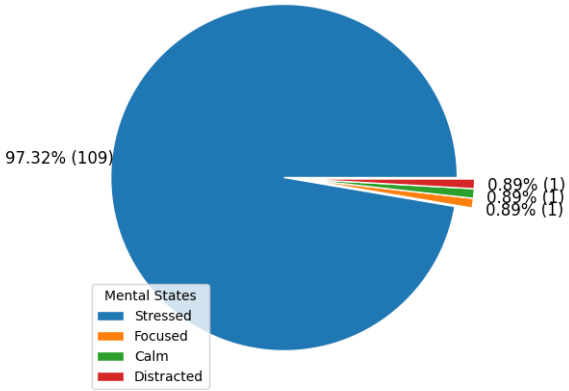


Figure 5.5: *Stressed* state overall predictions.

As mentioned above, although the cognitive DT can distinctly discriminate the *stressed* mental state, there are still some cases where the classifier is flawed and, because a negative emotion is being incorrectly classified as a positive one, these errors can reach the drone and provoke unexpected motions, disrupting the synchronization between operator and drone and destabilizing the flight.

5.2.1.3 Negative Spectrum Analysis

As referred in section 5.2.1.2, errors on the cognitive DT's classification when the operator is in a suitable cognitive state for sending commands has no impact or associated risk on the drones; however, the opposite scenario, for instance, when this DT should classify the mental state of the operator as one of the negative classes but instead classifies as one of the positives, this can allow commands to reach the drones contributing to a disruption of the stability of the flight. Taking this into account and, considering that previous the analysis concerns exclusively the cognitive DT, this section provides further analysis of the negative spectrum of the classification and the impact of the addition of the extra layer of the visual DT on the decision.

For the *distracted* and *stressed* states, table 5.2 and table 5.3 give some insight about the number of sent commands under an incorrect classification, described in the introduction of this section.

Regarding the *distracted* state, at *level 1* was detected six positive states, two of them sent; at *level 2* was detected 11 positive emotions in which four were sent and the at the full test ten positive emotions were detected, but only one command was sent to the drone due to the detection

Table 5.2: Distracted Emotion Recognition

Positive Detections	**Group of Test**		
	Level 1 Test	***Level 2 Test***	***Full Test***
Cognitive Digital Twin's Total number	6	11	10
Cognitive Digital Twin's number of neutral commands	4	7	6
Cognitive Digital Twin's number of sent commands	2	4	1
Cognitive Digital Twin's positive detection, Visual Digital Twin's negative detection	N/A	N/A	3

of a negative visual emotion by the visual DT, which prevented the subject to send three other commands.

Table 5.3: Stressed Emotion Recognition

Positive Detections	**Group of Test**		
	Level 1 Test	***Level 2 Test***	***Full Test***
Cognitive Digital Twin's total number	0	0	2
Cognitive Digital Twin's number of neutral commands	0	0	1
Cognitive Digital Twin's number of sent commands	0	0	0
Cognitive Digital Twin's positive detection, Visual Digital Twin's negative detection	N/A	N/A	1

For the *stressed* state, only at the *full test* was detected two positive emotions and none of them were sent by the subject because one was a neutral command and the other was associated with a negative visual emotion, detected by the visual emotion component.

Since the training of the mental commands is a task that requires some time to practice and refine, it is equally challenging to reproduce a command at a live setting and in an equivalent environment the subject trained (described in chapter 4). Even if the model has a classification error associated, most commands detected by the BCI are neutral ones, which have no implication on the drones; however, because the command classifier can incorrectly output a motion command, these ones can be sent to the drones. With the extra layer of the visual DT, these unique situations are assessed by it and some of those errors are prevented.

Considering that this is a 4-class classification problem, there is a probability of 25% that a baseline classifier correctly categorizes the subject emotion state and, in the *baseline test* characterized by the lack of machine learning, all commands are sent to the drones, regardless of the operator's emotional state, which could only be beneficial if the subject has perfect cognitive condition at all times, condition very unlikely to happen.

5.2.1.4 Real-time Mission with a Singular Drone

The previous sections of this chapter validates the solution in isolated simulations, where the operator explicitly is put under a certain condition to match a certain emotion and multiple experiments are conducted during these sessions; however, the purpose of this work is to develop a secure platform for an operator to control and manage a drone or swarm of drones within the context of a mission with a BCI. A mission is defined as a set of planned operations required for achieving a certain goal. As opposed to the individual experiments for the validation of each emotion that are supervised and controlled by the operator, the mission is conducted under a unpredictable live environment.

In this section, a real-time mission is conducted with one drone to analyze how the digital twin can identify multiple emotions at a longer execution time, as well as the switching of emotions is performed, in response to external events. For this purpose, a 2-minute experiment is conducted where the operator will be surprised at certain points in time by an alarm. The ideal outcome of these experiments is that when the operator is focused, the system detects a persistent cognitive *focused* state until the first alarm is triggered and the system detects the distraction or stress involved as well as the visual *surprised* or other *negative* emotion from the visual DT and stops sending commands. This is a cyclic event which should be happening by each triggered alarm. The second section is related to the validation of a 2-drone scenario, to simulate a swarm and the behavior of the system when sending information for more than one client node.

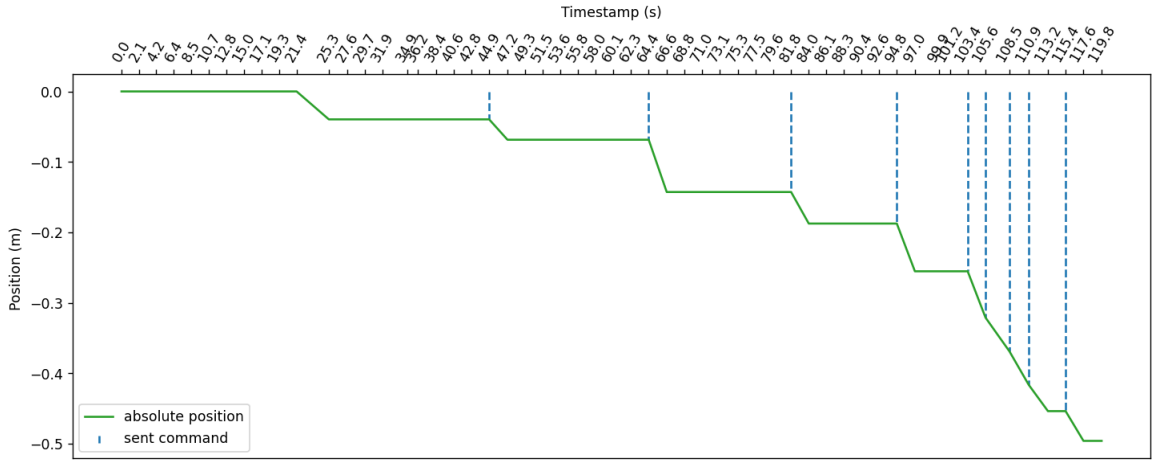
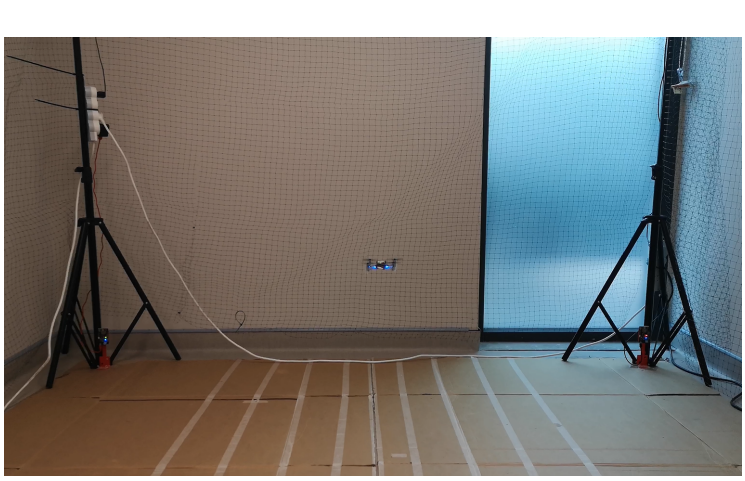


Figure 5.6: Real-time mission with distracting external events with a single drone.

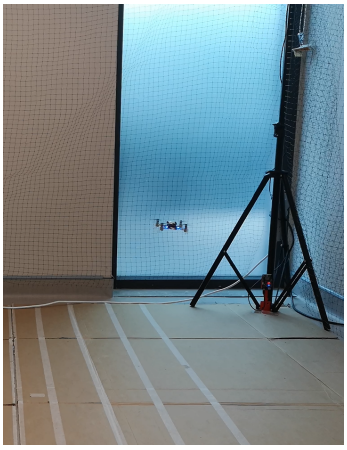
Figure 5.6 describes the trajectory the drone has made during the experiment. Considering that the valid commands are the *right* and *left* ones and they both operate only on the y axis, the y axis of this chart represents the absolute position of the drone. During this period, the operator reproduced commands to the *right* and *left*, being only valid and sent the ones to the *right*, reason why the absolute position has only negative numbers. The x axis represents the timestamps, in seconds, since the beginning of the experiment until the end of it.. Sent commands are indicated

by the vertical, dotted blue, line on the line chart, meaning the point in time/observation, where the drone has received a valid command and started moving to a different location on the *arena*.

In this case, two alarms were set for 27.6 seconds and one minute and twenty eight seconds after the experiment initiated. At the beginning of the experiment, the operator was distracted by the multiple events that were happening on the screen, considering it was initiating, and there was a period for relaxation and preparation to send commands, hence why the first 21,4 seconds the drone did not move. At this timestamp, the operator has been stabilized and focused on the drone and was able to send a *right* command with an increment of position of 0.04 meters. The next timestamp was detected a *focused* state, but a *neutral* command and would continue to send *right* commands if the first alarm would not sound at 27,6 seconds in. Here, the system immediately detected a *distracted* mental state and a *surprised* visual state, triggered by the vibration of the alarm. *Stressed* and *distracted* mental states were both detected during the next 17 seconds as the operator was turning of the alarm and redirecting his focus to the drone. At timestamp 44,9 seconds, the operator was able to send a set of commands for a period of forty seconds. During this time, where the operator was in a persistent mental state of *focused*, sending *right* commands and most neutral ones, it was detected a sum of 3 *negative* mental states. At the sound of the second alarm, at the exact timestamp denoted as 88,3 seconds, the digital twin resulted in a *focused* mental state but the visual DT resulted in a *fear* visual emotion, due to the significantly expressive facial expressions of the operator, a reaction to the unexpected vibration of the alarm. The operator did recover faster and, at timestamp 94,8 seconds, he began to send a set of commands under the mental state of *focused*, until the experiment ended. During this period, four *negative* emotions were detected: two *distracted* ones and two *stressed* ones at the end of the experiment.



(a) Drone after take-off.



(b) Drone after a *right* command.

Figure 5.7: Position of the drone during a flight.

Although the operator, in multiple periods of time during this experiment, has stabilized the *focused* mental state and was formulating motion commands, the cognitive DT has mistaken a few observations as one of the *negative* emotions; thus, it was not a persistent classification and did not have impact on the drone. With the alarms triggered at exact times, the visual DT detected

a disturbance on the operator before the cognitive one, not allowing sending the *right* command, as formulated by the operator, proving that this component is valuable to the system and provides extra security to the execution of the mission. In concern of the detection of humor switches, for instance from one *positive* emotion to a *negative* one and so on, the system is capable of discriminating each one at significant timestamps, meaning that on the sound of the alarms, the system was capable of detecting that the operator was no longer focused on the drone and rather distracted and surprised by the alarm and when the operator turned off the alarm and refocused on the drone, the system was also able to detect it. Even though the training of the emotional states was in short sessions of eight seconds, the operator was able to persist on a certain emotion in longer periods of time.

5.2.1.5 Swarm ROS2 Management Validation

Considering that the long-term goal is to apply this solution to a swarm of drones and, as mentioned in section 5.2.1.4, this system has been validated with a single drone with satisfactory results, this section demonstrates the inclusion of two drones on the experiments and the interaction between them and the system.

Since the *arena* has its own limitations and do not provide enough space for testing, this experiment was conducted with the support of a simulated server node, which will mimic the *base station* and does not have connection to a physical drone. The main goal of this experiment is to send commands through all client nodes involved and validating if the server node has received them. For this purpose, an additional client node was created as a second drone. The ROS2-related code was modified to comply with the current client nodes. Due to debugging purposes, messages that are printed in the consoles address their own drone to identify whether the system and the server node can receive requests from both client nodes. For this purpose, the operator executed one take-off, one motion command and a landing operation.

```
1 [INFO] [1623235179.701766700] [minimal_service]: Take off incoming request
2  height: 0.500000
3  duration: 2.000000
4  response: 1
5
6 [INFO] [1623235179.704947100] [minimal_service]: Take off incoming request
7  height: 0.500000
8  duration: 2.000000
9  response: 1
```

Listing 5.1: *Server node take-off* message.

```
1 [INFO] [1623235180.412227300] [drone1]: Sending information to server: height:
     0.500000
2  duration: 2.000000
```

```
 response: 1

[INFO] [1623235180.413246332] [drone2]: Sending information to server: height:
    0.500000
 duration: 2.000000
 response: 1
```

Listing 5.2: *Client nodes take-off* messages.

```
[INFO] [1623235924.239647700] [minimal_service]: Go to incoming request
 x: 0.000000 y: 0.041117 z: 0.000000 yaw: 0.000000 duration: 1.000000

[INFO] [1623235924.244461100] [minimal_service]: Go to incoming request
 x: 0.000000 y: 0.041117 z: 0.000000 yaw: 0.000000 duration: 1.000000
```

Listing 5.3: *Server node go to* message.

```
[INFO] [1623235925.104762600] [drone1]: Sending information to server: x coordinate
    : 0.000000
 y coordinate: 0.041117
 z coordinate: 0.000000
 rotation: 0.000000
 duration: 1.000000
 response: 1

[INFO] [1623235925.106783200] [drone2]: Sending information to server: x coordinate
    : 0.000000
 y coordinate: 0.041117
 z coordinate: 0.000000
 rotation: 0.000000
 duration: 1.000000
 response: 1
```

Listing 5.4: *Client nodes go to* messages.

```
[INFO] [1623231818.296960000] [minimal_service]: Land incoming request
 height: 0.000000
 duration: 2.000000
 response: 1

[INFO] [1623231818.407826700] [minimal_service]: Land incoming request
 height: 0.000000
 duration: 2.000000
 response: 1
```

Listing 5.5: *Server node land* message.

```
[INFO] [1623235180.412227300] [drone1]: Sending information to server: height:
    0.000000
 duration: 2.000000
 response: 1

[INFO] [1623235180.413246332] [drone2]: Sending information to server: height:
    0.000000
 duration: 2.000000
 response: 1
```

Listing 5.6: *Client nodes land* messages.

By analyzing these console logs, it is possible to conclude that the system can support more than one client node, meaning that it allows the sending of commands to more than one drone. In this experiment, it was added a second client node with few code changes so, increasing the number of drones that operate in the swarm does not imply huge efforts and modifications to the code base.

5.3 Summary

In this section, a random forest-based digital twin is validated under four experimental levels that represent an increment of security: (1) the *baseline test*, (2) the *level 1 test*, (3) the *level 2 test* and finally (3) the *full test*, where the cognitive DT is boosted by the addition of the visual DT.

The *baseline test* has no consideration for the emotional cognitive and visual states of the operator and, therefore, all commands reproduced are immediately broadcast to the drone. Since humans are prone to fatigue, distractions and even irritations, due to the unexpected behavior of the drone or to external events, it is very likely that negative emotions are very frequent, and these situations can create instability to the flight.

As for the *level 1 test*, that incorporates the digital twin of the operator, success rates were measured between 71% and 100%, being the *stressed* mental state the best discriminated one. *Level 2 test* also measures its success rate from 57% to 100%, thus, although it is noticeable that the minimum success rate dropped significantly, classification errors can be propagated to the drone, but their impact is minimized due to the computation of coordinates based on the confidences of the predictions, commonly lower in these situations, meaning that drones will not probably move enough to disrupt the stability of their flight. Regarding the *full test*, success rates are measured between 84% and 99%.

In a positive case scenario, where the operator is in a mental emotional state considered to be suitable to send commands (*calm* and *focused*), if the cognitive DT outputs a negative emotion,

the command will simply not be sent to the drone, having no impact whatsoever on the flight. However, in a negative case scenario, meaning that the operator is not in a suitable state to send commands, if one of the cognitive DT outputs a positive emotion, the commands can be sent and destabilize the flight, so, this solution has proven to minimize the impact on the drones under these incorrect classifications when added the extra layer of the visual DT. Results show that some, if not all, observations where the cognitive DT outputs a positive emotion and the reproduced command is a motion one, the visual DT outputs a negative emotion, preventing them to be sent to the drones.

In a free, live, simulation, switches between emotions, more importantly from positive to negative emotions and so on, are correctly identified as well as continuous emotions that persists for longer periods of time.

This solution has proven to not also accurately identify the operator's visual and mental states and to handle, with efficiency, the commands as far as computing coordinates and providing a communication channel to the hardware, but also minimizes the impact of flaws of the system on the flight of the drone. In addition, comprehending that a ROS2 client node is associated with one drone through a client-server architecture, a swarm of drones is easily composed and managed by adding as many client nodes as needed.

Chapter 6

Conclusion

In this chapter, overall conclusions about each part of the development of this work are presented. The research questions mentioned in chapter 1 are answered according with the results and gathered knowledge obtained by the implemented solution and what it represents in the context of the problem. In addition, it is mentioned how this solution could be optimized or enhanced and how can it fit in a bigger plan than given in this thesis.

6.1 Conclusions

In this work, I analyzed data captured by the BCI *Emotive Epoc+* of a drone operator and, using machine learning techniques, I was able to build a digital twin of the operator capable of predicting his emotional (cognitive and visual) state and decide whether the commands formulated should be sent to the drone. The classification of the emotional state is mainly supported by BCI recorded data and by the analysis of the visual facial expressions, for higher levels of precision for the decision. In addition, the communication between the system and the drone is done through a ROS2 client node.

To implement the proposed solution, I divided the work into the following tasks (see section 1.4): (1) getting acknowledged with the *EmotivBCI* headset and aggregated software as well as building a mental profile of commands; (2) perform data analysis and data pre-processing on recordings from the BCI; (3) build a cognitive digital twin of the operator with best fitted machine learning algorithm; (4) search and include a visual digital twin in the project; (5) develop a decision component capable of determine the destination of commands and (6) build a communication channel between the system and the drone based on a client-server architecture with ROS2.

First, the work started with the exploration of the *Emotiv Epoc+* device which was associated with multiple software, most of them requiring a license for access. *EmotivBCI* application did not need any special license and it had most core functionalities I considered to be crucial, meaning that it provided a platform for training the mental commands and to monitor our cognitive

metrics. Right from the beginning, I noticed that the training of these mental commands was such a time-consuming and complex task, because this whole concept of formulating commands with thoughts was not also very new to me but having to learn this process from scratch was a little difficult. Establishing a strategy to formulate these commands, one that could be easily reproduced whenever I wanted, had to be well-though. To validate that the commands formulated were uniquely based on thoughts, I could not move any facial muscle. In addition, different days mean different states of mind and consciousness, making the task even more challenging when trying to reproduce equivalent thoughts. Beyond the fact that this training was based on trial and error, the number of commands was also a great concern. As many commands I add to my profile, the more confusing it gets, meaning that training one command and having confidence that the model can discriminate it well enough and adding a new one is sufficient to alter the whole profile and to create confusions for the model. Both commands need to have very different formulation strategies to the model accurately discriminate each other. So, the challenge lays not also in teaching our brain to formulate commands and to practice them, but also to ensure that each are well discriminated and separated in this profile. The operator has total responsibility for creating strategies to formulate commands to accomplish them whenever he wants. For example, if the training occurs when the operator has high levels of stress, then the model will learn that the command he is training is correlated with these levels and, presumably, will only output the command when the operator is in an equivalent state. Classification errors occurs, thus as many training the operator completes, the better the classifier will perform on distinguishing the commands. In this thesis, the operator is focused on formulating commands, but occasionally the classifier outputs commands when the operator is with higher levels of stress or distracted.

With a suitable and well-trained profile in the *EmotivBCI* application, the next step was to perform some data analysis. Since all data from the headset is sent to the server, the system needs to send requests to an API and receive the data streams in *JSON* format. Considering that each request is unique and is received multiple times per second, pre-processing the information was a time-consuming task to get everything in a single dataset, where each point in time is matched with the correct values of each data stream.

Training data was collected with as much precision as possible, meaning that I used small sessions of data acquisition so that it would be easier to concentrate on replicating a certain emotion. If, for whatever reason, I was not able to replicate equivalent scenarios for each simulation, the training would be then discarded and this session would be replaced, resulting in a task based on trial and error. Multiple machine learning algorithms were validated with this training set, for instance Decision Tree, K-Nearest Neighbors, Linear Discriminant Analysis, Naive Bayes, Random Forest, Support Vector Machine and Neural Networks (see accuracies in table 4.6). Random Forest was the algorithm with the highest values on each performance metric and therefore used for training the cognitive digital twin. Results from the validation experiments show that this component, the cognitive digital twin has a high success rate of 84%.

Regarding the visual digital twin, I searched multiple repositories for a suitable project, in python, that would have a model trained with visual input for classifying the operator's facial

expression. I found a project with a publication concerning their visual recognizer that was trained with the FER-2013 dataset, a generic one that has thousands of images with people's faces and even of cartoons. Although the classifier has an accuracy of 66%, the real-time run of this solution is rather surprising. In one hand, a video appears on the screen from the webcam, and the resulting classification appears on the face of the operator with specific colors for each detected emotion. This model can detect switches from one emotion to another and the classification is usually accurate. On the other hand, these face switches are detected with a little delay, is not an immediate reaction from the model and it can more easily detect the *neutral* and *happy* states. Considering this, the model's classification is correlated with the rotation of the head of the operator, meaning that if the operator lowers his face, the classifier will output a *sad* state. Lowering the face can be a standard behavior of someone who is sad, thus, in this work, the goal is having an operator managing a drone and looking at it in real-time, so it is normal that the operator's eyes and head follow the motions of the drone. In these situations, it surfaces an inconvenience as the position of the camera is a variable to consider. In this way, the classification can output a wrong classification but it just translates into an inconvenience and not an actual negative impact to the drone (because the decision component will not send the commands if this visual DT outputs a negative emotion).

In the decision component it lays the responsibility of evaluating both the visual and cognitive states of the operator and decide whether the command should be forwarded to the drone. To minimize the impact of classification errors on the drone, the coordinates are computed according to the confidences of the classified command and the classified mental emotion by the models. All procedures for data cleaning and processing are reused for the newly collected data. One recurrent concern when running the solution in a real-time setting was its processing time. Considering that the raw *JSON* data is crossed by multiple transformations to be handled by the digital twin, these events could have a significant run-time, enough to disrupt the flow of a real-time control. Thus, the solution is efficient enough to not spend more than 0,2 seconds to run all these procedures and send the command to the drone.

Lastly, the final component of this system is the ROS2 client node. The addition of this component boosts the value of this system significantly because not also it is easier later to add more drones as I wish and manage them with ease, but also it proves that it is feasible to control drones in a live, free will, environment. In this work, it is explicit that I developed the client node, but it is hidden the real complexity of building the whole ROS2 framework. It requires other modules to be built to connect the hardware to the system and the level of synchronization needed between all these modules is imperative. Having a virtual, simulated, device is a simpler validation procedure but does not have the same impact and value when applied to a real hardware in real scenarios. Thus, these connections are more unstable and unpredictable because I am handling critical and under-development systems and it represents a frequent challenge. Furthermore, I demonstrate that I can manage drones through ROS2 via any client node without the limitation of different development environments (operating systems), as this thesis' solution is implemented in python and the *server node* in C++. Overall, the ROS2 client node is a secure and stable option to communicate with the drones.

In summary, the digital twin can accurately discriminate the operator's emotional states at a live setting and the combination of classification models improved the security and reliability of the system to decide upon the broadcasting of the formulated commands by the operator.

6.2 Response to Research Questions

In response to **RQ1**, **TS1** has proven its veracity. The cognitive digital twin is built according to the operator's specific natural reactions and inherent emotions. Since it learns from the operator and finds patterns on data recorded from a BCI, it can accurately discriminate each emotional state.

Concerning **RQ2**, **TS2** has proven its veracity. By adopting a digital twin of the operator to detect both cognitive and visual emotional states and by computing coordinates proportionally to the confidences of the predictions, commands formulated in a negative state are discarded and classification errors have a minimized impact on the drones. The cognitive digital twin delivers conclusive information for the decision upon the formulated commands; thus, the prediction model behind this DT has its uncertainties and can output an inaccurate classification that leads to destabilization of the drone or swarm of drones. Particularly when the operator is stressed or distracted, but the cognitive DT classifies his emotional condition as one of the positive classes. As part of this thesis' validation, multiple levels of experiments were established to ascertain the value of each added component for handling commands formulated on these specific situations. Results show that the last experimental level, which incorporates all mentioned components, delivers the highest level of robustness of the solution.

6.3 Final Appreciations

The resolution of this thesis' problem was one of the most satisfactory ones I had in my academic years. This thesis was proposed in a completely different context I was used to and handled with so many technologies that I have never learnt, since the programming language to the BCI headsets and software. There was an exponential learning path where I became acknowledged with so many areas and could build such a complete solution, one that would ultimately surpass any expectations and more. Overall, this solution has reached all goals established in the beginning of the project, proved to be very reliable and stable, reflects the amount of efforts and teamwork that were necessary and it has showed its relevance as this is the beginning of a bigger, broader, project for the future.

It was also submitted and accepted a short paper (four pages) on the tenth International Conference on Intelligent Systems and Applications (or INTELLI 2021 [6]) summarizing all work done in this thesis and with preliminary results, in the context of case studies of intelligent solutions, intelligent data analysis, intelligent human-computer interaction systems and real-time intelligence. In addition, it is expected the submission of a full-length (six pages) paper, containing the final results of this thesis, on the twentieth International Conference on Advanced Robotics (or ICAR

[1]). Additionally, this solution is integrated in the context of an industrial use case, being currently demonstrated to associated clients of Capgemini Engineering and it opens the possibility of using this *decision making* solution to other safety-critical systems.

6.4 Future Work

As future work, I would like to better the operator's experience regarding the training of the mental commands. Since the procedure was to rely on the *EmotivBCI* application to formulate and practice the commands, when providing a new platform without the dependency from external applications, I could control and link all training in a unique application and adapt the interface and functionalities to better fit the control and flight of drones, meaning that, for instance, instead of a generic object appearing (cube), it could appear a drone and add as much drones as the operator would like.

Regarding the digital twin, the core functionality of this system, I would like to collect and add more data for its training. Additionally, because different days can make a difference in the operator's state of mind, I would like to add a complementary recording of a baseline each day the operator accesses to this system, so that the digital twin can adapt itself according to these values, as the *EmotivBCI* application establishes. Furthermore, data cleaning and processing can be optimized so that larger datasets in the training phase can be handled with more efficiency and consume less time. Since this solution is not generic and depends on each operator's profile, I would like to have a larger number of subjects and with different demographics (i.e., genders, ages, etc.) to evaluate this solution in a broader level, which was not possible in this thesis due to *Covid-19*.

The chosen BCI for the purpose of this thesis was the *Emotiv Epoc+*; thus, there are multiple other options to take into consideration, for instance the *OpenBCI* that has the advantage of having open-source software. Data collected from this BCI will not need to be sent to a server and can output raw EEG, unlike the *Emotiv Epoc+* without the proper license. I would like to explore these others BCIs and assess them in the context of this problem.

Appendix A

Cortex Auxiliary Information

In this appendix, additional information regarding the *Cortex* API is displayed. Since this API provides a gateway to access the *Emotiv* services and data, the first section shows all method calls required for the implementation of this thesis proposed solution. The second section explains the reproduced *Cortex* examples to demonstrate this API's functionalities.

A.1 Method Calls

Considering that the *Cortex* examples' repository ([23]) was the foundation of the code base, the *Cortex* library synchronous method calls was adapted for this project's needs.

First of all, concerning the connection to the headset described in section 4.1 and showcased by figure A.1, the procedure is composed by 5 steps: (1) query the available headsets, (2) connect to the desired headset, (3) request access for accessing to the user's application, (4) authorize the headset's user and (5) create the session. These method calls are displayed above with the corresponding request bodies and parameters.

```
query_headset_request = {
        "jsonrpc": "2.0",
        "id": g.QUERY_HEADSET_ID,
        "method": "queryHeadsets",
        "params": {}
}
```

Listing A.1: *queryHeadsets* request body.

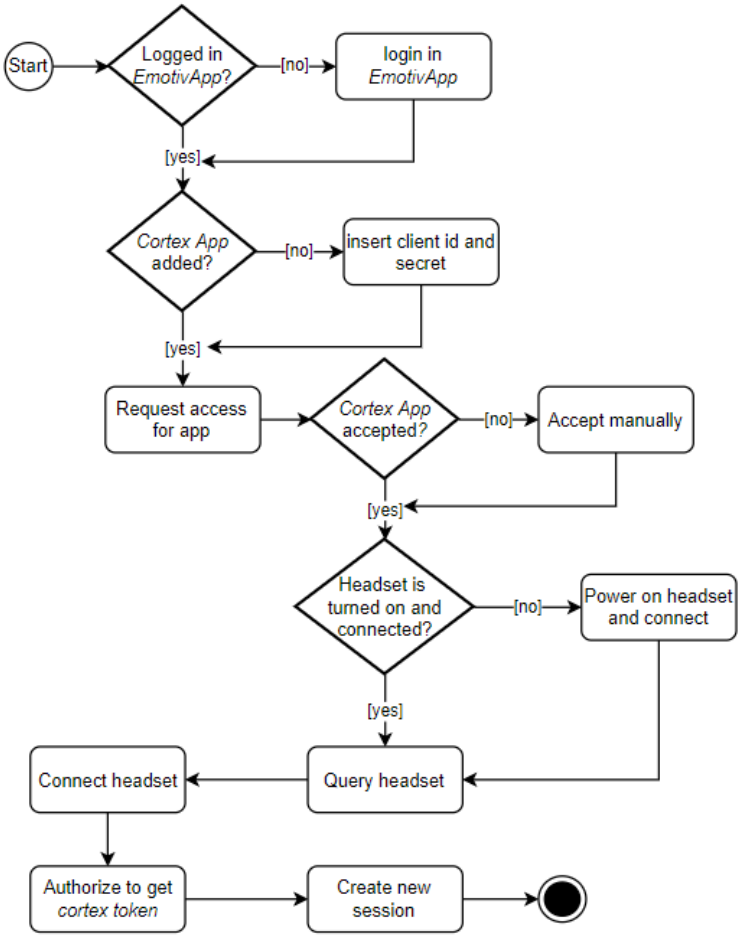


Figure A.1: Activity diagram showing the workflow of the connection of the *Emotiv Epoc+* headset (from [26]).

```
connect_headset_request = {
        "jsonrpc": "2.0",
        "id": g.CONNECT_HEADSET_ID,
        "method": "controlDevice",
        "params": {
            "command": "connect",
            "headset": <headset_id>
        }
}
```

Listing A.2: *controlDevice* request body for connecting the headset.

```
connect_headset_request = {
        "jsonrpc": "2.0",
        "id": g.CONNECT_HEADSET_ID,
        "method": "controlDevice",
        "params": {
            "command": "disconnect",
            "headset": <headset_id>
        }
}
```

Listing A.3: *controlDevice* request body for disconnecting the headset.

Table A.1: Control device method call parameters (adapted from [3]).

Name	Type	Required	Description
command	string	yes	The command must be "connect", "disconnect" or "refresh".
headset	string	no	The id of the headset that you want to connect or disconnect. The headset id is returned by *queryHeadsets*. If the command is "refresh", then you should omit this parameter.

```
request_access_request = {
        "jsonrpc": "2.0",
        "method": "requestAccess",
        "params": {
            "clientId":<client_id>,
            "clientSecret": <client_secret>
        },
        "id": g.REQUEST_ACCESS_ID
}
```

Listing A.4: *requestAccess* request body.

Table A.2: *requestAccess* method call parameters (adapted from [3]).

Name	Type	Required	Description
clientId	string	yes	The client id of your Cortex application.
clientSecret	string	yes	The client secret of your Cortex application.

```
authorize_request = {
        "jsonrpc": "2.0",
        "method": "authorize",
        "params": {
            "clientId": <client_id>,
            "clientSecret": <client_secret>,
            "license": <license>,
            "debit": <debit>
        },
        "id": g.AUTHORIZE_ID
}
```

Listing A.5: *authorize* request body.

Table A.3: *authorize* method call parameters (adapted from [3]).

Name	**Type**	**Required**	**Description**
clientId	string	yes	The client id of your Cortex application.
clientSecret	string	yes	The client secret of your Cortex application.
license	string	no	A license id. In most cases, you don't need to specify the license id. Cortex will find the appropriate license based on the client id.
debit	number	no	Number of sessions to debit from the license, so that it can be spent locally without having to authorize again. You need to debit the license only if you want to activate a session. The default value is zero.

Regarding the data subscription

```
sub_request_json = {
        "jsonrpc": "2.0",
        "method": "subscribe",
        "params": {
            "cortexToken": <auth>,
            "session": <session_id>,
            "streams": stream
        },
        "id": g.SUB_REQUEST_ID
}
```

Listing A.6: *subscribe* request body.

Table A.4: *subscribe* method call parameters (adapted from [3]).

Name	Type	Required	Description
cortexToken	string	yes	A token returned by *authorize*.
session	string	yes	A session id returned by *createSession*.
streams	array of string	yes	The data streams you want to subscribe to.

A.2 Reproduced Examples

As mentioned above, *Cortex* API provides multiple demonstrations that were run at the time of the headset exploration and research; thus, this section only contains the demonstrations that were useful for the understanding and implementation of this thesis proposed solution, starting with the subscription of data streams. In addition, the library, the subscription method has encapsulated a writing method that fetches the data and writes it to a separated file.

```
import numpy as np
import pandas as pd
from cortex import Cortex

class Subcribe():
    count = 5

    def __init__(self):
        self.c = Cortex(user, debug_mode=True)
        self.c.do_prepare_steps()

    def sub(self, streams):
        self.c.sub_request(streams)

user = {
    "license": "",
    "client_id": <client_id>,
    "client_secret": <client_secret>,
    "debit": 100
}

s = Subcribe()
streams = ['com']
s.sub(streams)
```

Listing A.7: Subscribe method call example python file.

```
{"id":6,"jsonrpc":"2.0","result":{"failure":[],"success":[{"cols":["act","pow"],"
    sid": <sid>,"streamName":"com"}]}},{"com":["neutral",0.0],"sid":<sid>,"time"
    :1622641319.5343},{"com":["neutral",0.0],"sid":<sid>,"time":1622641319.6593}
```

Listing A.8: Data streams read from the written file.

Regarding the training of the mental commands, the following code listings displays the procedure of the tutorial and execution.

```
from cortex import Cortex

class Train():
  def __init__(self):
    self.c = Cortex(user, debug_mode=True)
    self.c.do_prepare_steps()

  def train(self,
      profile_name,
      training_action,
      number_of_train):

    stream = ['sys']
    self.c.sub_request(stream)

    profiles = self.c.query_profile()

    if profile_name not in profiles:
      status = 'create'
      self.c.setup_profile(profile_name, status)

    status = 'load'
    self.c.setup_profile(profile_name, status)

    print('begin train ----------------------------------')
    num_train = 0
    while num_train < number_of_train:
      num_train = num_train + 1

      print('start training {0} time {1} ---------------'.format(training_action,
          num_train))
      print('\n')
      status='start'
      self.c.train_request(detection='mentalCommand',
                action=training_action,
                status=status)

      print('accept {0} time {1} ---------------'.format(training_action, num_train
          ))
```

```
38        print('\n')
39        status='accept'
40        self.c.train_request(detection='mentalCommand',
41                  action=training_action,
42                  status=status)
43
44      print('save trained action')
45      status = "save"
46      self.c.setup_profile(profile_name, status)
47
48      status = 'unload'
49      self.c.setup_profile(profile_name, status)
50
51
52    def live(self, profile_name):
53      print('begin live mode ----------------------------------')
54      # load profile
55      status = 'load'
56      self.c.setup_profile(profile_name, status)
57
58      stream = ['com']
59      self.c.sub_request(stream)
60
61 user = {
62   "license" : "",
63   "client_id" : <client_id>,
64   "client_secret" : <client_secret>,
65   "debit" : 100
66 }
67
68 t=Train()
69 profile_name = 'Train_test'
70 number_of_train = 1
71
72 training_action = 'neutral'
73 t.train(profile_name,
74      training_action,
75      number_of_train)
76
77 training_action = 'push'
78 t.train(profile_name,
79      training_action,
80      number_of_train)
81
82 t.live(profile_name)
```

Listing A.9: Complete procedure for the training of the mental commands.

```
subscribe request --------------------------------
"{\"id\":6,\"jsonrpc\":\"2.0\",\"result\":{\"failure\":[],\"success\":[{\"cols
    \":[\"event\",\"msg\"],\"sid\":\<sid>,\"streamName\":\"sys\"}]}}"


NEW DATA -----------------------
{"id":6,"jsonrpc":"2.0","result":{"failure":[],"success":[{"cols":["event","msg"],"
    sid":<sid>,"streamName":"sys"}]}}
query profile --------------------------------
query profile request
 {
    "jsonrpc": "2.0",
    "method": "queryProfile",
    "params": {
        "cortexToken": <cortex_token>
    },
    "id": 8
}


query profile result
 {'id': 8, 'jsonrpc': '2.0', 'result': [{'meta': {'creation_time': '2021-02-12T10
     :51:57.223+00:00'}, 'name': 'Diana'}, {'meta': {'creation_time': '2021-06-02
     T14:23:07.510+01:00'}, 'name': 'Test'}, {'meta': {'creation_time': '2021-02-12
     T15:45:24.858+00:00'}, 'name': 'Diana 2'}, {'meta': {'creation_time': '
     2021-03-24T09:25:36.479-01:00'}, 'name': 'Diana 3'}, {'meta': {'creation_time'
     : '2021-06-02T14:27:52.185+01:00'}, 'name': 'Train_test'}]}


extract profiles name only
['Diana', 'Test', 'Diana 2', 'Diana 3', 'Train_test']


setup profile --------------------------------
setup profile json:
 {
    "jsonrpc": "2.0",
    "method": "setupProfile",
    "params": {
        "cortexToken": <cortex_token>,
        "headset": <headset_id>,
        "profile": "Train_test",
        "status": "load"
    },
    "id": 7
}


result
```

```
 {
    "id": 7,
    "jsonrpc": "2.0",
    "result": {
        "action": "load",
        "message": "The profile is loaded successfully",
        "name": "Train_test"
    }
}

begin train ----------------------------------
start training neutral time 1 ---------------

 YOU HAVE 8 SECONDS FOR TRAIN ACTION NEUTRAL

{
    "id": 9,
    "jsonrpc": "2.0",
    "result": {
        "action": "neutral",
        "message": "Set up training successfully",
        "status": "start"
    }
}
{
    "sid": <sid>,
    "sys": [
        "mentalCommand",
        "MC_Started"
    ],
    "time": 1622641819.284287
}
{
    "sid": <sid>,
    "sys": [
        "mentalCommand",
        "MC_Succeeded"
    ],
    "time": 1622641828.159287
}
accept neutral time 1 ---------------


{
    "id": 9,
    "jsonrpc": "2.0",
    "result": {
        "action": "neutral",
        "message": "Set up training successfully",
```

```
        "status": "accept"
    }
}
{
    "sid": <sid>,
    "sys": [
        "mentalCommand",
        "MC_Completed"
    ],
    "time": 1622641828.284287
}
save trained action
setup profile --------------------------------
setup profile json:
 {
    "jsonrpc": "2.0",
    "method": "setupProfile",
    "params": {
        "cortexToken": <cortex_token>,
        "headset": <headset_id>,
        "profile": "Train_test",
        "status": "save"
    },
    "id": 7
}


result
 {
    "id": 7,
    "jsonrpc": "2.0",
    "result": {
        "action": "save",
        "message": "The profile is saved successfully",
        "name": "Train_test"
    }
}


setup profile --------------------------------
setup profile json:
 {
    "jsonrpc": "2.0",
    "method": "setupProfile",
    "params": {
        "cortexToken": <cortex_token>,
        "headset": <headset_id>,
        "profile": "Train_test",
        "status": "unload"
```

```
    },
    "id": 7
}


result
 {
    "id": 7,
    "jsonrpc": "2.0",
    "result": {
        "action": "unload",
        "headsetId": <headset_id>,
        "message": "Profile unloaded successfully for headset <headset_id>"
    }
}


subscribe request --------------------------------
"{\"id\":6,\"jsonrpc\":\"2.0\",\"result\":{\"failure\":[],\"success\":[{\"cols
    \":[\"event\",\"msg\"],\"sid\":\<sid>",\"streamName\":\"sys\"}]}}"


NEW DATA -----------------------
{"id":6,"jsonrpc":"2.0","result":{"failure":[],"success":[{"cols":["event","msg"],"
    sid":<sid>,"streamName":"sys"}]}}
query profile --------------------------------
query profile request
 {
    "jsonrpc": "2.0",
    "method": "queryProfile",
    "params": {
        "cortexToken": <cortex_token>
    },
    "id": 8
}


query profile result
 {'id': 8, 'jsonrpc': '2.0', 'result': [{'meta': {'creation_time': '2021-02-12T10
     :51:57.223+00:00'}, 'name': 'Diana'}, {'meta': {'creation_time': '2021-06-02
     T14:23:07.510+01:00'}, 'name': 'Test'}, {'meta': {'creation_time': '2021-02-12
     T15:45:24.858+00:00'}, 'name': 'Diana 2'}, {'meta': {'creation_time': '
     2021-03-24T09:25:36.479-01:00'}, 'name': 'Diana 3'}, {'meta': {'creation_time'
     : '2021-06-02T14:27:52.185+01:00'}, 'name': 'Train_test'}]}


extract profiles name only
['Diana', 'Test', 'Diana 2', 'Diana 3', 'Train_test']

```

```
setup profile --------------------------------
setup profile json:
 {
    "jsonrpc": "2.0",
    "method": "setupProfile",
    "params": {
        "cortexToken": <cortex_token>,
        "headset": <headset_id>,
        "profile": "Train_test",
        "status": "load"
    },
    "id": 7
}


result
 {
    "id": 7,
    "jsonrpc": "2.0",
    "result": {
        "action": "load",
        "message": "The profile is loaded successfully",
        "name": "Train_test"
    }
}


begin train ----------------------------------
start training push time 1 ---------------



 YOU HAVE 8 SECONDS FOR TRAIN ACTION PUSH

{
    "id": 9,
    "jsonrpc": "2.0",
    "result": {
        "action": "push",
        "message": "Set up training successfully",
        "status": "start"
    }
}
{
    "sid": <sid>,
    "sys": [
        "mentalCommand",
        "MC_Started"
    ],
```

```
    "time": 1622641828.409287
}
{
    "sid": <sid>,
    "sys": [
        "mentalCommand",
        "MC_Succeeded"
    ],
    "time": 1622641837.284287
}
accept push time 1 ---------------


{
    "id": 9,
    "jsonrpc": "2.0",
    "result": {
        "action": "push",
        "message": "Set up training successfully",
        "status": "accept"
    }
}
{
    "sid": <sid>,
    "sys": [
        "mentalCommand",
        "MC_Completed"
    ],
    "time": 1622641837.409287
}
save trained action
setup profile --------------------------------
setup profile json:
 {
    "jsonrpc": "2.0",
    "method": "setupProfile",
    "params": {
        "cortexToken": <cortex_token>,
        "headset": <headset_id>,
        "profile": "Train_test",
        "status": "save"
    },
    "id": 7
}


result
 {
    "id": 7,
```

```
    "jsonrpc": "2.0",
    "result": {
        "action": "save",
        "message": "The profile is saved successfully",
        "name": "Train_test"
    }
}


setup profile --------------------------------
setup profile json:
 {
    "jsonrpc": "2.0",
    "method": "setupProfile",
    "params": {
        "cortexToken": <cortex_token>,
        "headset": <headset_id>,
        "profile": "Train_test",
        "status": "unload"
    },
    "id": 7
}


result
 {
    "id": 7,
    "jsonrpc": "2.0",
    "result": {
        "action": "unload",
        "headsetId": <headset_id>,
        "message": "Profile unloaded successfully for headset <headset_id>"
    }
}

begin live mode ----------------------------------
setup profile --------------------------------
setup profile json:
 {
    "jsonrpc": "2.0",
    "method": "setupProfile",
    "params": {
        "cortexToken": <cortex_token>",
        "headset": <headset_id>,
        "profile": "Train_test",
        "status": "load"
    },
    "id": 7
}
```

```
330
331
332 result
333  {
334     "id": 7,
335     "jsonrpc": "2.0",
336     "result": {
337         "action": "load",
338         "message": "The profile is loaded successfully",
339         "name": "Train_test"
340     }
341 }
```

Listing A.10: Mental Command training example execution.

Appendix B

Descriptive Data Analysis

B.0.1 Motion Data Stream

Table B.1: Overall Motion data stream statistics.

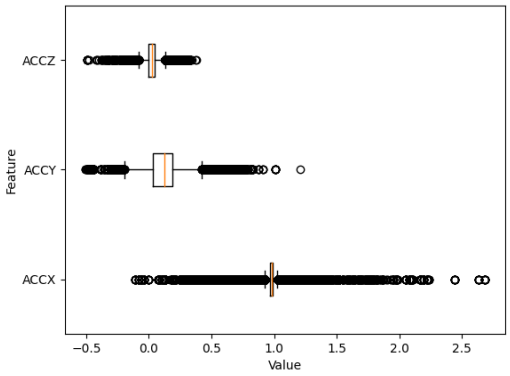
Feature	Count	Mean	Standard Deviation	Minimum	Maximum
ACCX	78 400	0.973062	0.101278	-0.108888	2.685588
ACCY	78 400	0.122431	0.035157	-0.509773	1.208026
ACCZ	78 400	0.023528	0.053613	0.048829	0.377447
MAGX	78 400	-98.237158	1.788835	-108.415351	-85.535973
MAGY	78 400	-137.213464	8.684884	-170.174716	-107.667659
MAGZ	78 400	41.764094	7.266094	24.374762	63.254749
Q0	78 400	0.382023	0.212553	0.000000	0.743819
Q1	78 400	-0.047733	0.534911	-0.770569	0.827820
Q2	78 400	-0.386831	0.226918	-0.763611	0.326538
Q3	78 400	-0.110115	0.554278	-0.780518	0.794556

B.0.2 Facial Expression Data Stream

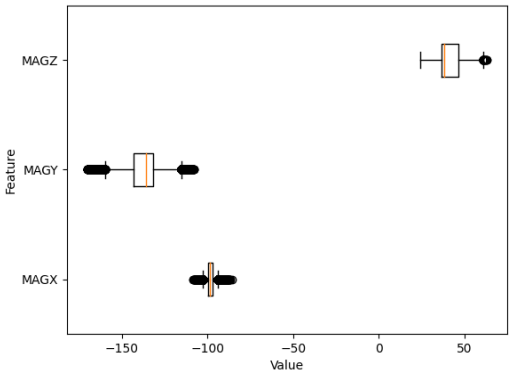
Table B.2: Overall Facial Expressions categorical data stream statistics.

Feature	Count	Most Frequent Category
eyeAct	78 400	*Neutral*
uAct	78 400	*Neutral*
lAct	78 400	*Neutral*

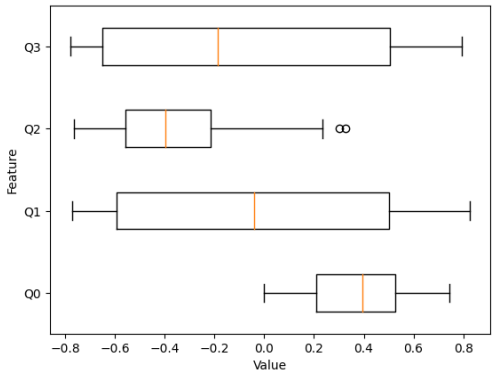
B.0.3 Band Power Data Stream



(a) *ACC* boxplot.



(b) *MAG* boxplot.



(c) *Q* boxplot.

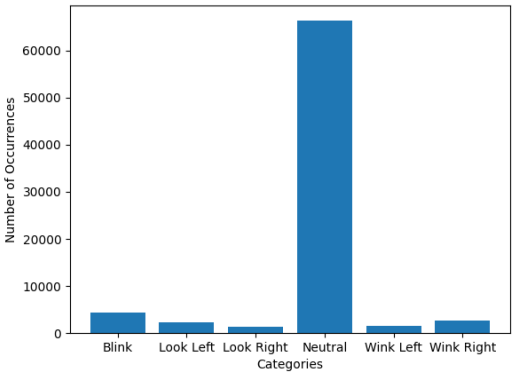
Figure B.1: Overall data distribution for the motion streams.

Table B.3: Overall Facial expressions numerical data stream statistics.

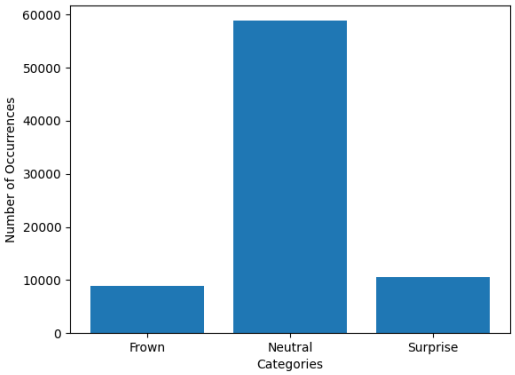
Feature	Count	Mean	Standard Deviation	Minimum	Maximum
uPow	78 400	0.097225	0.219010	0.000000	1.000000
lPow	78 400	0.012644	0.087591	0.000000	1.000000

Table B.4: *AF4* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

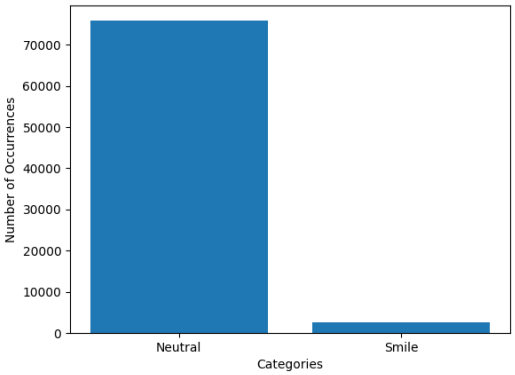
Feature	Count	Mean	Standard Deviation	Minimum	Maximum
AF4/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
AF4/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
AF4/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
AF4/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
AF4/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556



(a) *eyeAct* boxplot.



(b) *uAct* boxplot.



(c) *lAct* boxplot.

Figure B.2: Overall data distribution for the facial expressions categorical streams (before one-hot-encoding).

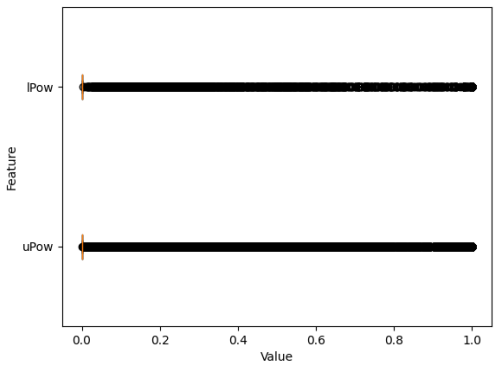


Figure B.3: Facial expressions numerical data stream box plot.

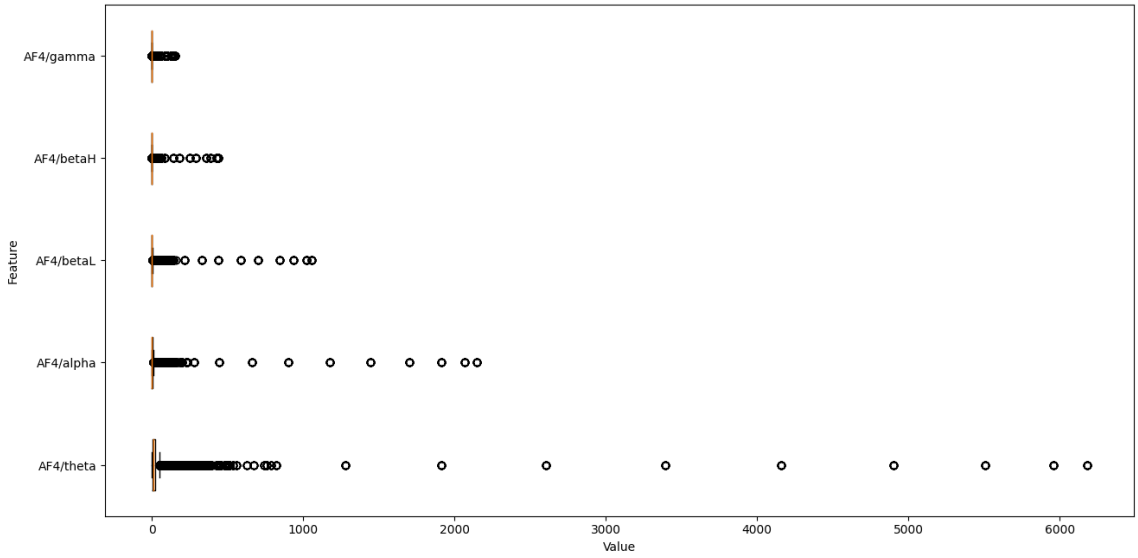


Figure B.4: *AF4 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.5: *F8* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	**Count**	**Mean**	**Standard Deviation**	**Minimum**	**Maximum**
F8/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
F8/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
F8/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
F8/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
F8/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

Table B.6: *F4* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	**Count**	**Mean**	**Standard Deviation**	**Minimum**	**Maximum**
F4/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
F4/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
F4/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
F4/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
F4/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

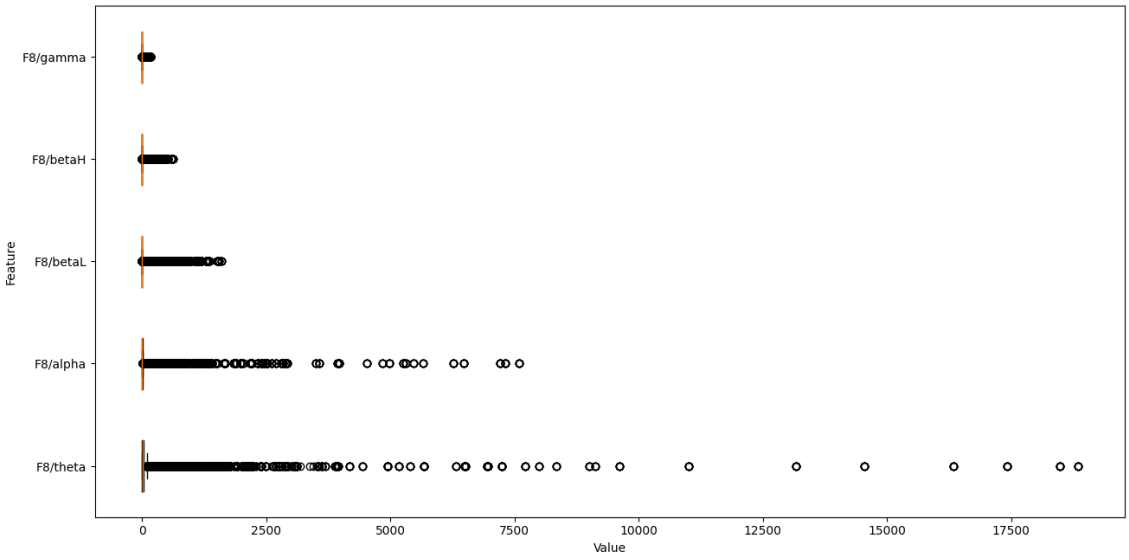


Figure B.5: *F8 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

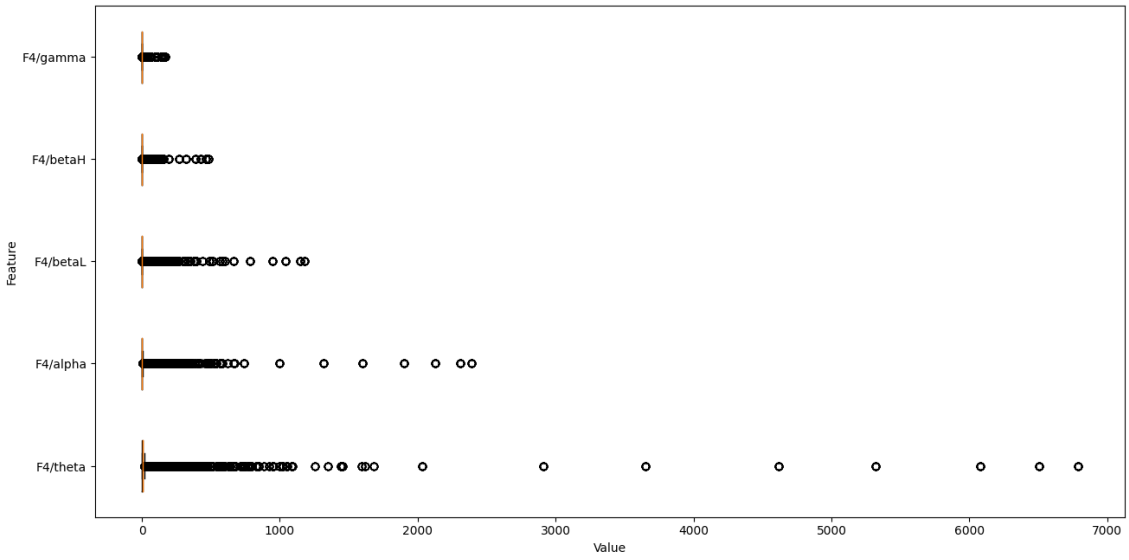


Figure B.6: *F4 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.7: *FC6* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
FC6/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
FC6/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
FC6/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
FC6/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
FC6/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

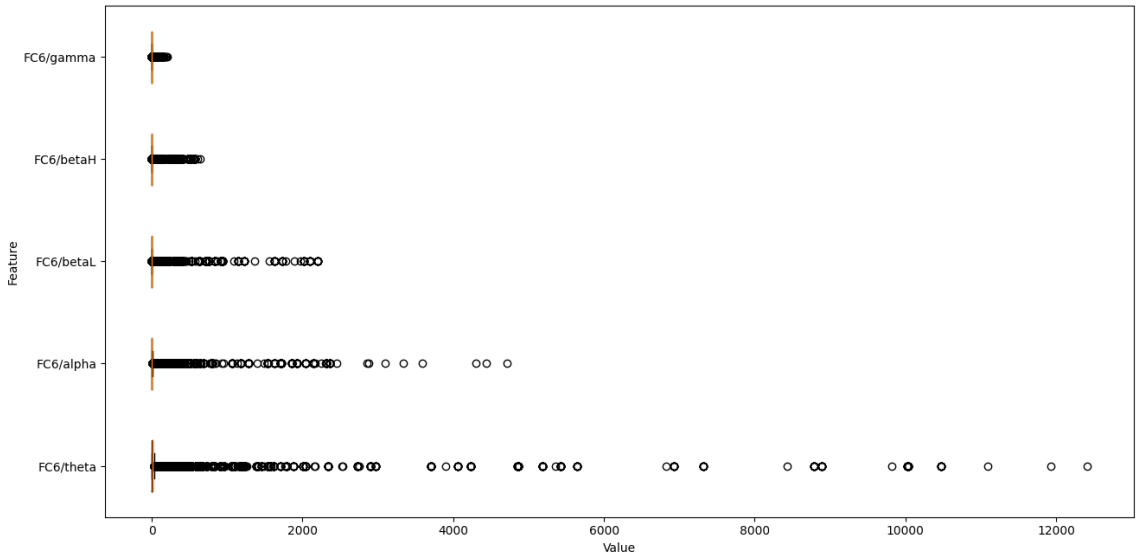


Figure B.7: *FC6 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.8: *T8* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
T8/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
T8/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
T8/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
T8/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
T8/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

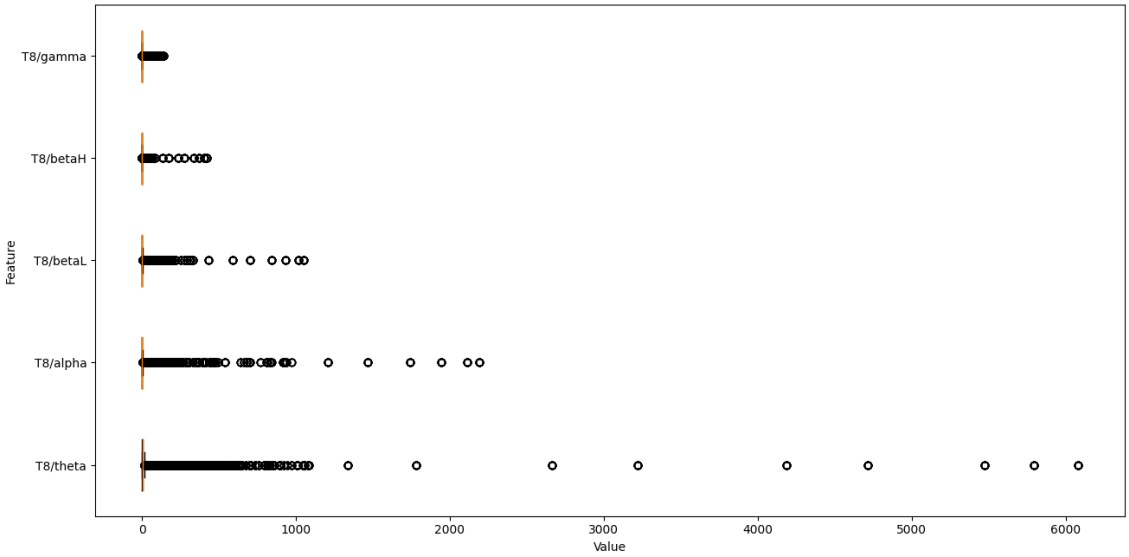


Figure B.8: *T8 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.9: *P8* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	**Count**	**Mean**	**Standard Deviation**	**Minimum**	**Maximum**
P8/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
P8/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
P8/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
P8/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
P8/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

Table B.10: *O2* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	**Count**	**Mean**	**Standard Deviation**	**Minimum**	**Maximum**
O2/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
O2/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
O2/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
O2/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
O2/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556
P8/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

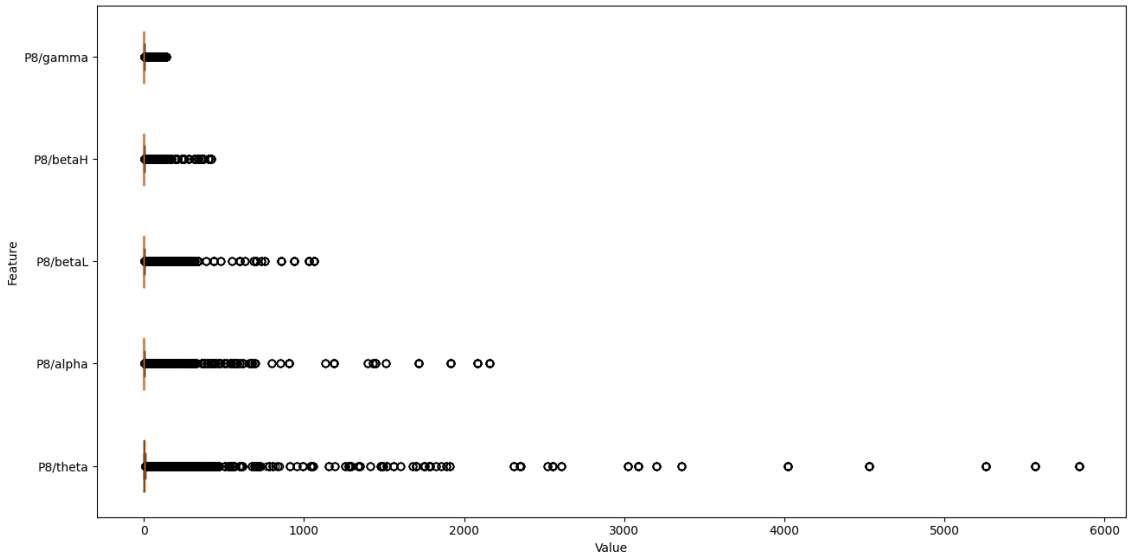


Figure B.9: *P8 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

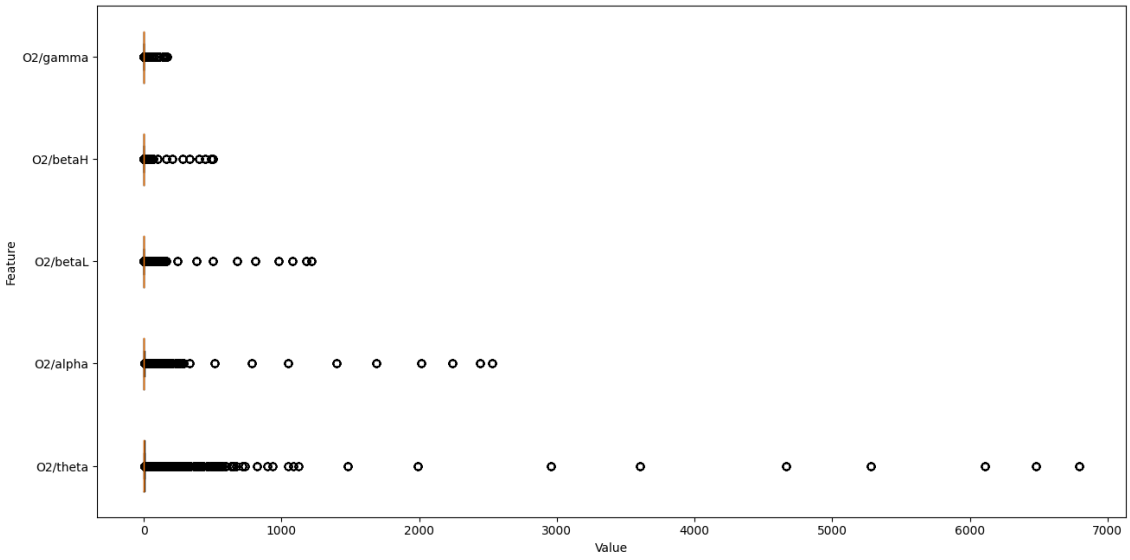


Figure B.10: *O2 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.11: *O1* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
O1/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
O1/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
O1/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
O1/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
O1/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

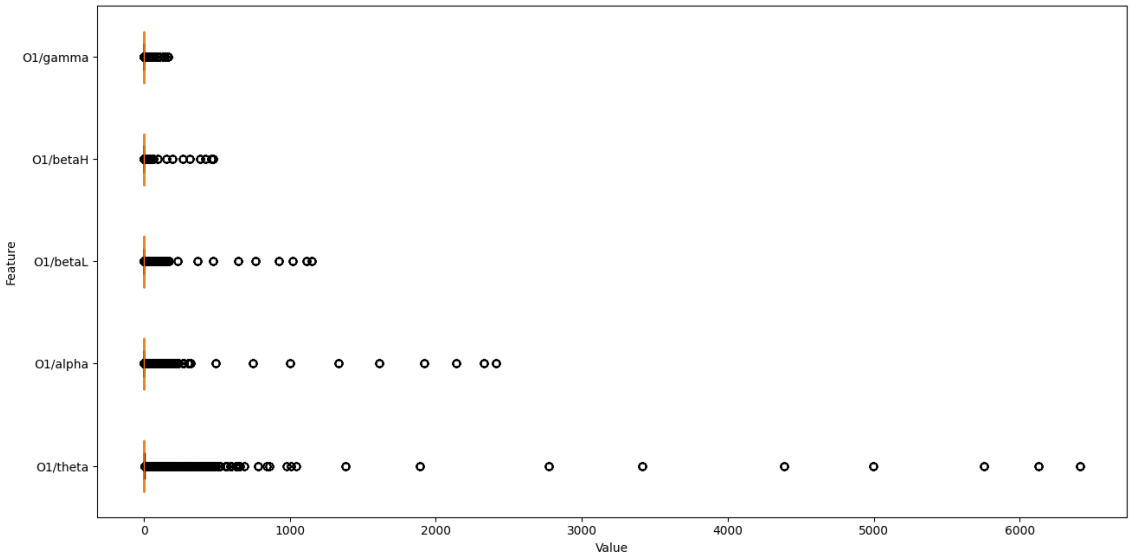


Figure B.11: *O1 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.12: *P7* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
P7/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
P7/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
P7/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
P7/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
P7/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

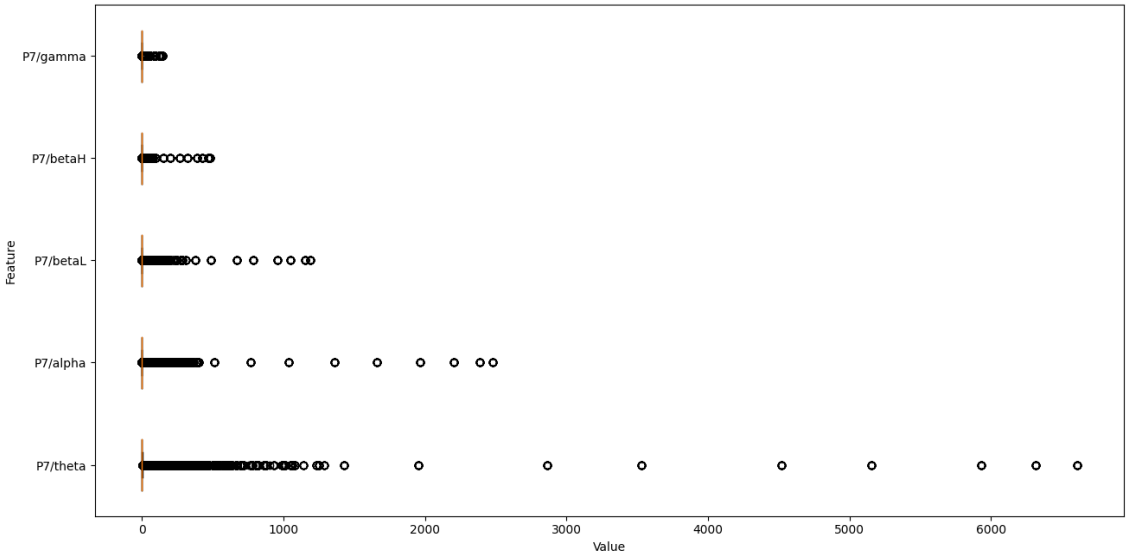


Figure B.12: *P7 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.13: *T7* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
T7/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
T7/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
T7/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
T7/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
T7/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

Table B.14: *FC5* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
FC5/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
FC5/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
FC5/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
FC5/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
FC5/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

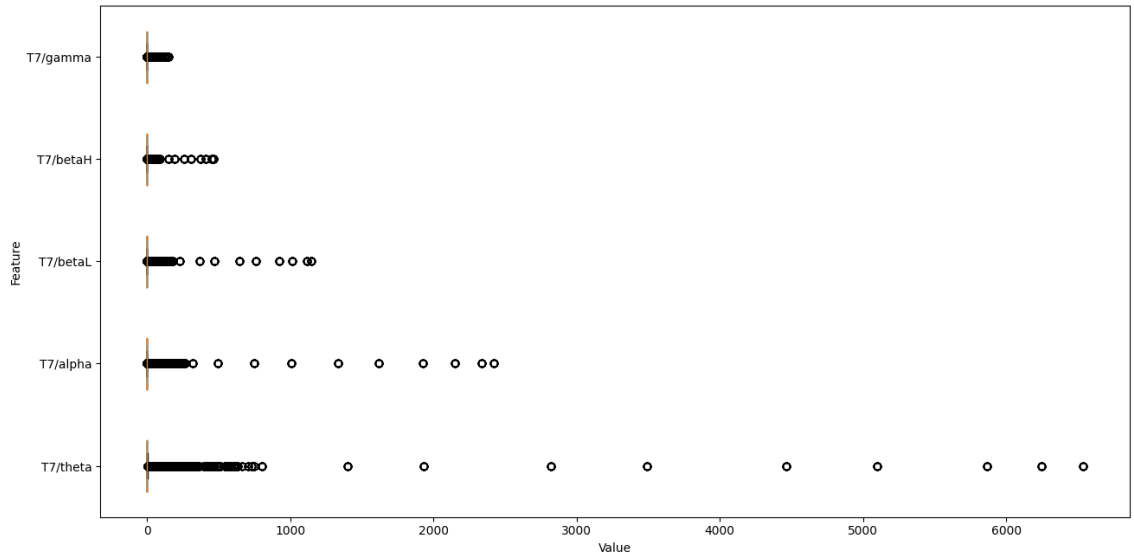


Figure B.13: *T7 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

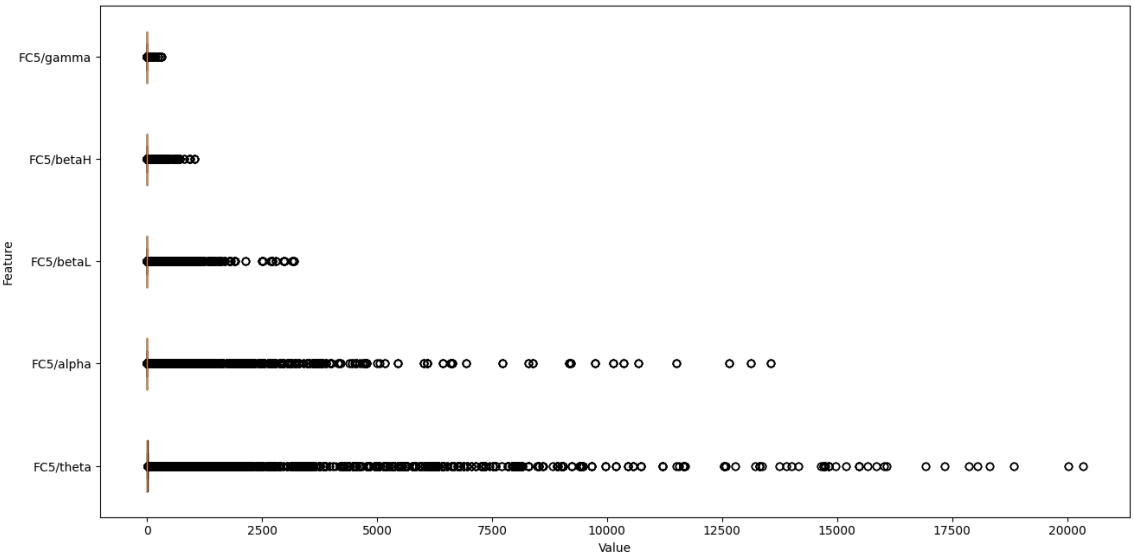


Figure B.14: *FC5 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.15: *F3* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
F3/theta	78 400	-0.386831	0.226918	-0.763611	0.326538
F3/alpha	78 400	-0.110115	0.554278	-0.780518	0.794556
F3/betaL	78 400	-0.386831	0.226918	-0.763611	0.326538
F3/betaH	78 400	-0.110115	0.554278	-0.780518	0.794556
F3/gamma	78 400	-0.110115	0.554278	-0.780518	0.794556

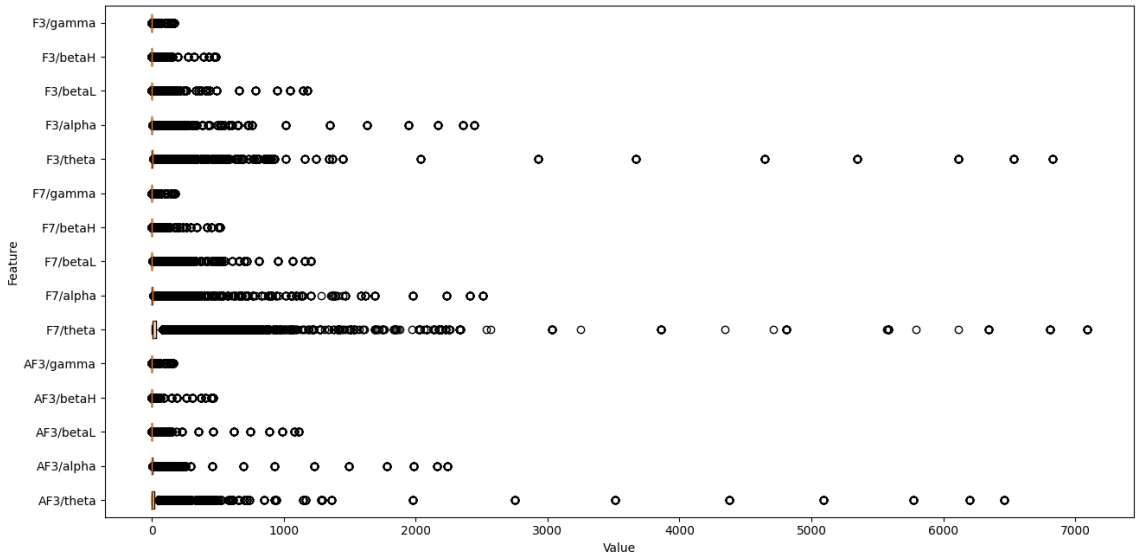


Figure B.15: *F3 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.16: *F7* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	Count	Mean	Standard Deviation	Minimum	Maximum
F7/theta	78 400	-98.237158	1.788835	-108.415351	-85.535973
F7/alpha"	78 400	-137.213464	8.684884	-170.174716	-107.667659
F7/betaL	78 400	41.764094	7.266094	24.374762	63.254749
F7/betaH	78 400	0.382023	0.212553	0.000000	0.743819
F7/gamma	78 400	-0.047733	0.534911	-0.770569	0.827820

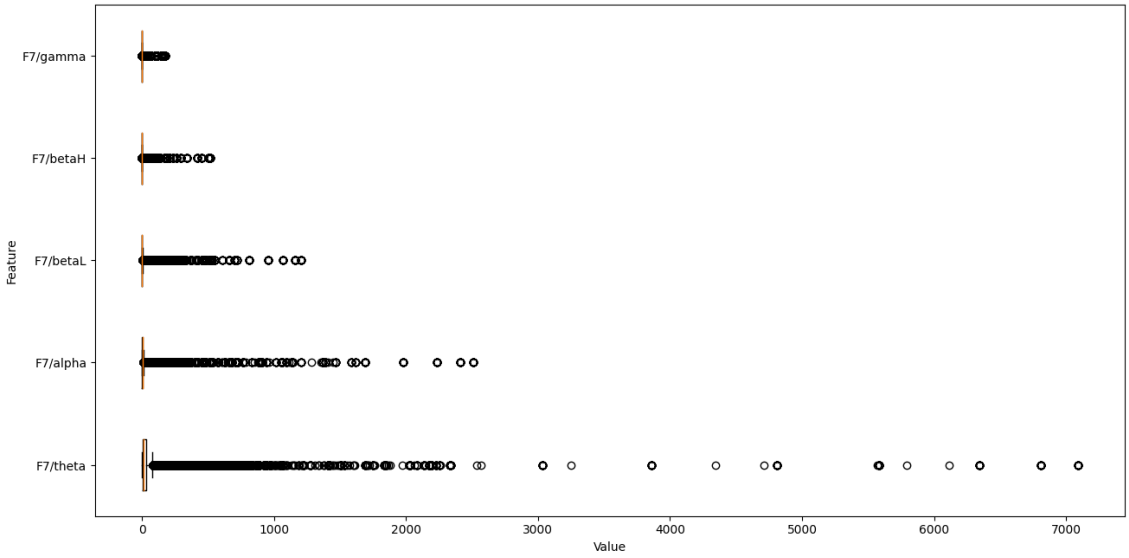


Figure B.16: *F7 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Table B.17: *AF3* sensor and *theta, alpha, betaL, betaH* and *gamma* bands data distribution.

Feature	**Count**	**Mean**	**Standard Deviation**	**Minimum**	**Maximum**
AF3/theta	78 400	0.382023	0.212553	0.000000	0.743819
AF3/alpha	78 400	1	1	1	1
AF3/betaL	78 400	0.973062	0.101278	-0.108888	2.685588
AF3/betaH	78 400	0.122431	0.035157	-0.509773	1.208026
AF3/gamma	78 400	0.023528	0.053613	0.048829	0.377447

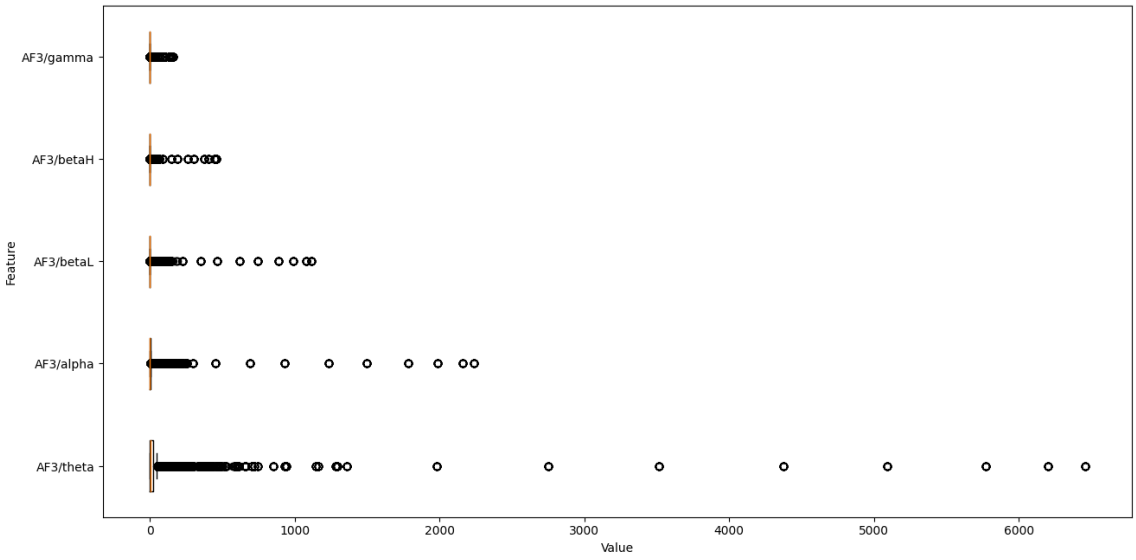


Figure B.17: *AF3 sensor and theta, alpha, betaL, betaH and gamma bands box plot.*

Appendix C

ROS2 Auxiliary Information

For the integration of the ROS2 client node on the project, research and tutorials were consulted and followed to build a general understanding of ROS concepts and implementation schemes. For this purpose, this section explains the reproduced tutorials that are available in the official ROS2 documentation ([45]).

C.1 ROS2 Service Architecture

Since the existing system already implements a service node (described in section 3.1.2.1), it was defined that the service-client architecture needed to persist and the new node, the client one (as described in section 4.3), was to be created. Ultimately, a node is a software component that is responsible for some specific task. For 2 nodes to communicate, ROS2 provides multiple ways: by topics, services, actions or parameters.

As described in figure C.1, a service is an additional software layer with a specific scheme for request and responses, where the client calls the specific service, instantiates its values matching its scheme and composes the request message. Considering that the message is successfully received by the server node, a response is sent following the same procedure. In addition, multiple nodes can be implemented to the system.

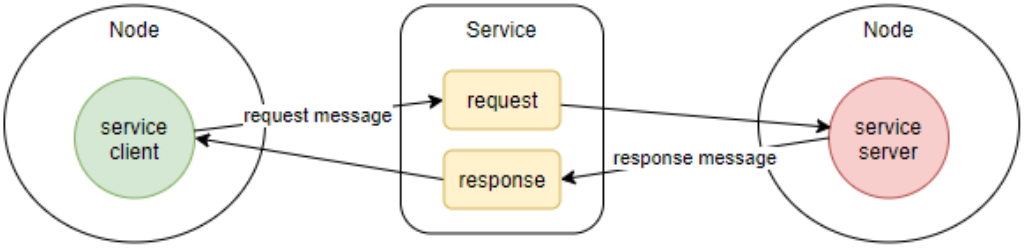


Figure C.1: Service-client ROS2 architecture (adapted from [44]).

C.2 Tutorials

This section presents reproduced tutorials of interest that aid on the implementation of the client node of this thesis. First, it was followed a tutorial for implementing a simple service and client nodes in python ([45]), on a Windows environment, that will fetch a pre-defined service file from an example's package (*AddTwoInts*). The purpose of this tutorial is to provide a server node that receives two integers from the client node's request message, computes their sum and sends the final value in the response message. Every step and code below is from [45]:

1. Source ROS2 installation:

```
call <path to installation>\local_setup.bat
```

2. Create a ROS2 package:

```
ros2 pkg create --build-type ament_python py_srvcli --dependencies rclpy
    example_interfaces

int64 a
int64 b
---
int64 sum
```

3. Create a ROS2 package:

```
ros2 pkg create --build-type ament_python py_srvcli --dependencies rclpy
    example_interfaces
```

4. Update *package.xml*:

```
<description>Python client server tutorial</description>
<maintainer email="you@email.com">Your Name</maintainer>
<license>Apache License 2.0</license>
```

5. Update *setup.py*:

```
maintainer='Your Name',
maintainer_email='you@email.com',
description='Python client server tutorial',
license='Apache License 2.0',
```

6. Write the service node:

```
from example_interfaces.srv import AddTwoInts

import rclpy
from rclpy.node import Node


class MinimalService(Node):

    def __init__(self):
        super().__init__('minimal_service')
        self.srv = self.create_service(AddTwoInts, 'add_two_ints', self.
            add_two_ints_callback)

    def add_two_ints_callback(self, request, response):
        response.sum = request.a + request.b
        self.get_logger().info('Incoming request\na: %d b: %d' % (request
            .a, request.b))

        return response


def main(args=None):
    rclpy.init(args=args)

    minimal_service = MinimalService()

    rclpy.spin(minimal_service)

    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

7. Write the client node:

```
import sys

from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node


class MinimalClientAsync(Node):

```

```
10          def __init__(self):
11              super().__init__('minimal_client_async')
12              self.cli = self.create_client(AddTwoInts, 'add_two_ints')
13              while not self.cli.wait_for_service(timeout_sec=1.0):
14                  self.get_logger().info('service not available, waiting again
                        ...')
15              self.req = AddTwoInts.Request()
16
17          def send_request(self):
18              self.req.a = int(sys.argv[1])
19              self.req.b = int(sys.argv[2])
20              self.future = self.cli.call_async(self.req)
21
22
23      def main(args=None):
24          rclpy.init(args=args)
25
26          minimal_client = MinimalClientAsync()
27          minimal_client.send_request()
28
29          while rclpy.ok():
30              rclpy.spin_once(minimal_client)
31              if minimal_client.future.done():
32                  try:
33                      response = minimal_client.future.result()
34                  except Exception as e:
35                      minimal_client.get_logger().info(
36                          'Service call failed %r' % (e,))
37                  else:
38                      minimal_client.get_logger().info(
39                          'Result of add_two_ints: for %d + %d = %d' %
40                          (minimal_client.req.a, minimal_client.req.b, response
                                .sum))
41                  break
42
43          minimal_client.destroy_node()
44          rclpy.shutdown()
45
46
47      if __name__ == '__main__':
48          main()
```

8. Add entry point to the *setup.py* on the *console_scripts*:

```
1       entry_points={
2       'console_scripts': [
3           'service = py_srvcli.service_member_function:main',
4           'client = py_srvcli.client_member_function:main',
```

```
    ],
    },
```

9. In a new terminal, source the setup files and run the service node:

```
    cd <path to project>
    call install/setup.bat
    ros2 run py_srvcli service
```

10. In a another new terminal, source the setup files and run the client node:

```
    cd <path to project>
    call install/setup.bat
    ros2 run py_srvcli client 2 3
```

11. Get request message on the server node:

```
    [INFO] [minimal_service]: Incoming request
    a: 2 b: 3
```

12. Get response message on the client node:

```
    [INFO] [minimal_client_async]: Result of add_two_ints: for 2 + 3 = 5
```

Secondly, it was followed a tutorial that defines its own service and message files, without the dependency of external packages, as opposed to the above tutorial. The purpose of this tutorial is to provide a server node that receives three integers from the client node's request message, computes their sum and sends the final value in the response message. Every step and code below is from [41]:

1. Source ROS2 installation:

```
    call <path to installation>\local_setup.bat
```

2. Create a ROS2 package:

```
    ros2 pkg create --build-type ament_cmake tutorial_interfaces
```

3. Create the message and services directories:

```
cd <path to source>/tutorial\_interfaces
mkdir msg
mkdir srv
```

4. Create the service *AddThreeInts.srv*:

```
int64 a
int64 b
int64 c
---
int64 sum
```

5. Update *CMakeLists.txt*:

```
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Num.msg"
  "srv/AddThreeInts.srv"
 )
```

6. Build the *tutorial_interfaces* package:

```
colcon build --merge-install --packages-select tutorial_interfaces
```

7. In a new terminal, source the files:

```
cd <path to workspace>
call install/setup.bat
```

8. Write the service node:

```
from tutorial_interfaces.srv import AddThreeInts

import rclpy
from rclpy.node import Node


```

```
7   class MinimalService(Node):
8
9       def __init__(self):
10          super().__init__('minimal_service')
11          self.srv = self.create_service(AddThreeInts, 'add_three_ints',
                self.add_three_ints_callback)
12
13      def add_three_ints_callback(self, request, response):
14          response.sum = request.a + request.b + request.c
15          self.get_logger().info('Incoming request\na: %d b: %d c: %d' % (
                request.a, request.b, request.c))
16
17          return response
18
19  def main(args=None):
20      rclpy.init(args=args)
21
22      minimal_service = MinimalService()
23
24      rclpy.spin(minimal_service)
25
26      rclpy.shutdown()
27
28  if __name__ == '__main__':
29      main()
```

9. Write the client node:

```
1   from tutorial_interfaces.srv import AddThreeInts
2   import sys
3   import rclpy
4   from rclpy.node import Node
5
6
7   class MinimalClientAsync(Node):
8
9       def __init__(self):
10          super().__init__('minimal_client_async')
11          self.cli = self.create_client(AddThreeInts, 'add_three_ints')
12          while not self.cli.wait_for_service(timeout_sec=1.0):
13              self.get_logger().info('service not available, waiting again
                    ...')
14          self.req = AddThreeInts.Request()
15
16      def send_request(self):
17          self.req.a = int(sys.argv[1])
18          self.req.b = int(sys.argv[2])
19          self.req.c = int(sys.argv[3])
```

```
20            self.future = self.cli.call_async(self.req)
21
22
23    def main(args=None):
24        rclpy.init(args=args)
25
26        minimal_client = MinimalClientAsync()
27        minimal_client.send_request()
28
29        while rclpy.ok():
30            rclpy.spin_once(minimal_client)
31            if minimal_client.future.done():
32                try:
33                    response = minimal_client.future.result()
34                except Exception as e:
35                    minimal_client.get_logger().info(
36                        'Service call failed %r' % (e,))
37                else:
38                    minimal_client.get_logger().info(
39                        'Result of add_three_ints: for %d + %d + %d = %d' %
                                                        # CHANGE
40                        (minimal_client.req.a, minimal_client.req.b,
                            minimal_client.req.c, response.sum)) # CHANGE
41                break
42
43        minimal_client.destroy_node()
44        rclpy.shutdown()
45
46
47    if __name__ == '__main__':
48        main()
```

10. Add the following line to *package.xml*:

```
1    <exec_depend>tutorial_interfaces</exec_depend>
```

11. Build the package:

```
1    colcon build --merge-install --packages-select py_srvcli
```

12. In a new terminal, source the setup files and run the server node:

```
1    ros2 run py_srvcli service
```

13. In a another new terminal, source the setup files and run the client node:

```
ros2 run py_srvcli client 2 3 1
```

14. Get request message on the server node:

```
[INFO] [minimal_service]: Incoming request
a: 2 b: 3 c: 1
```

15. Get response message on the client node:

```
[INFO] [minimal_client_async]: Result of add_two_ints: for 2 + 3 + 1 = 6
```

Lastly, it was followed a tutorial that introduces the graphical user interface for ROS2 (RQT), another technology that was used in this thesis. The purpose of this tutorial was to start RQT and explore its built-in functionalities with the aid of the *turtlesim* demonstration, for instance, the *service caller* for verifying the *turtlesim* available services, the *spawn* service and *set_pen* service. *Turtlesim* is a simulator of a turtle that appears in the middle of the screen and executes whatever movement triggered by certain keys of the keyboard. Steps below are from [43]:

1. Source ROS2 installation:

```
call <path to installation>\local_setup.bat
```

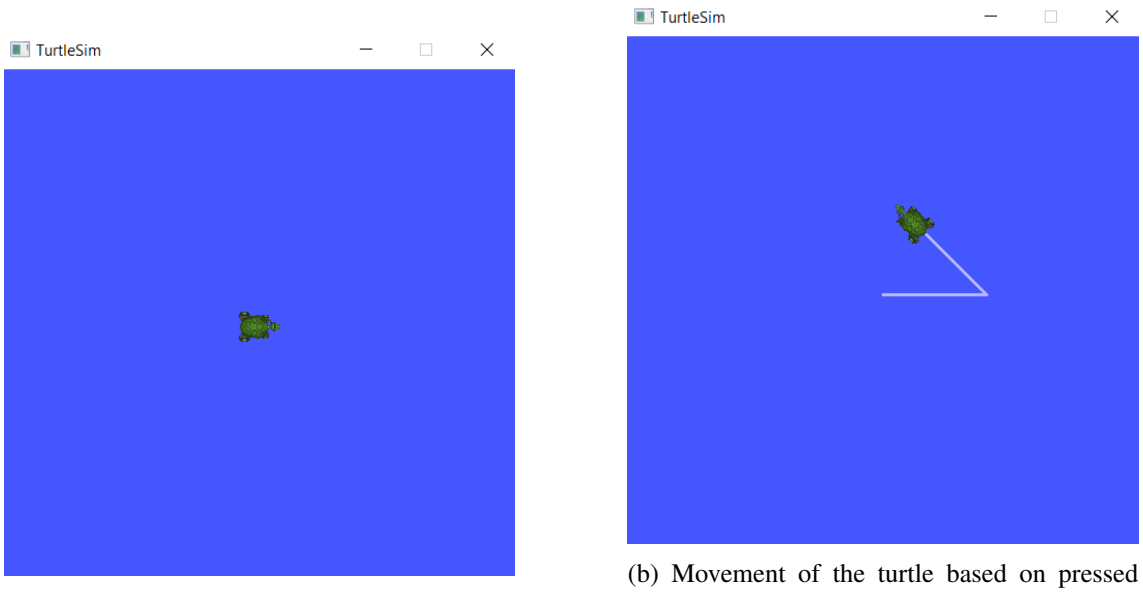
2. Start *turtlesim* (figure C.2a).

```
ros2 run turtlesim turtlesim_node
```

3. Check feedback on the terminal:

```
[INFO] [1622624716.108938100] [turtlesim]: Starting turtlesim with node
    name /turtlesim
[INFO] [1622624716.187215900] [turtlesim]: Spawning turtle [turtle1] at x
    =[5.544445], y=[5.544445], theta=[0.000000]
```

4. Open a new terminal, source the files and run the controller node:



(a) *Turtlesim* execution.

(b) Movement of the turtle based on pressed keys.

Figure C.2: Turtles trajectories.

```
    call <path to installation>\local_setup.bat
    ros2 run turtlesim turtle_teleop_key
```

5. Click on forward arrow followed by the *e* key (figure C.2b).

6. Check feedback on both terminals:

```
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to
     absolute orientations. 'F' to
    cancel a rotation.
'Q' to quit.
```

Listing C.1: Key controller node (terminal 2).

```
[INFO] [1622624948.634001200] [
    turtlesim]: Rotation goal
    completed successfully
```

Listing C.2: Turtle simulator (terminal 1).

7. Start rqt:

```
    rqt
```

8. Spawn a new turtle with the *turtlesim spawn* service (figure C.3).

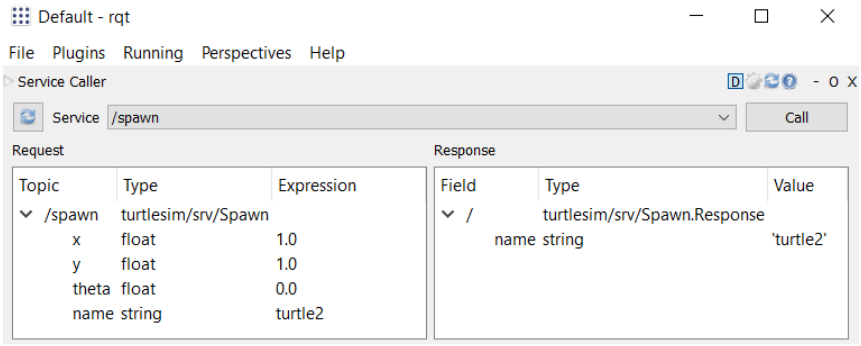


Figure C.3: *Turtlesim* execution.

9. Check feedback on the *turtlesim* node (terminal 1):

```
[INFO] [1622626285.540052200] [turtlesim]: Spawning turtle [turtle2] at x
    =[1.000000], y=[1.000000], theta=[0.000000]
```

10. Draw a trajectory line with the *turtlesim set_pen* service (figure C.4).

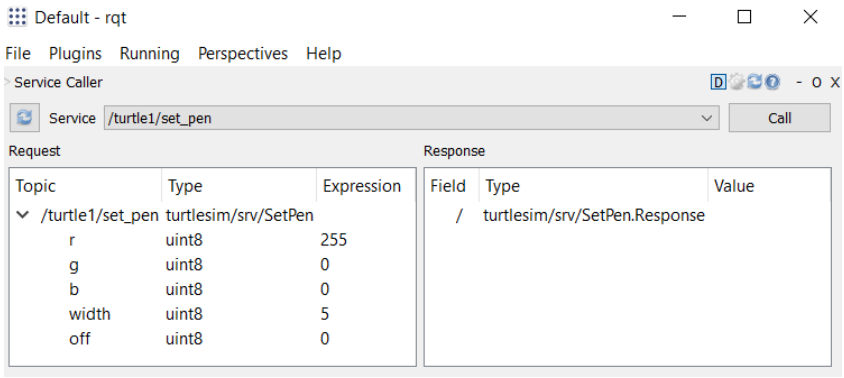


Figure C.4: *Turtlesim* execution.

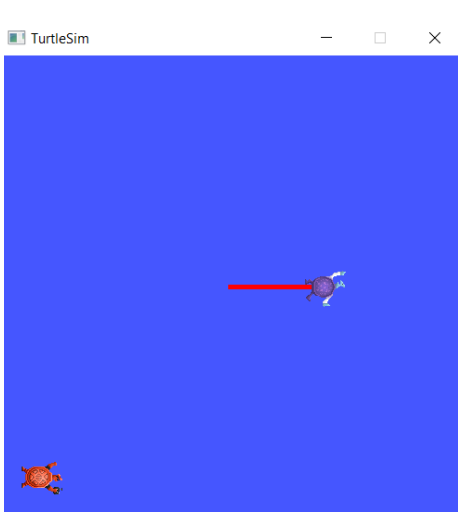
11. Open new terminal, source files and remap the controller node:

```
ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=
    turtle2/cmd_vel
```

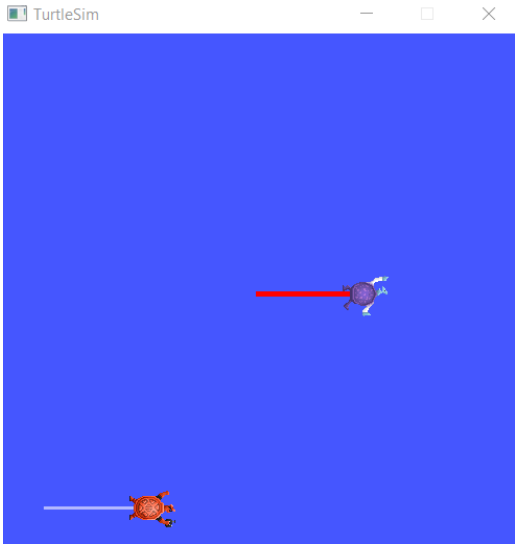
12. Check turtle positions (figure C.5).:

C.3 Command Line Operations

```
call <path to installation>\setup.bat
```



(a) *Turtle1* in a forward movement with custom trajectory line.



(b) *Turtle2* in a forward movement.

Figure C.5: Turtles trajectories.

```
call install\setup.bat
ros2 run rqt_mypkg service
```

Listing C.3: Run the *server node*.

```
[INFO] [1622643702.357147400] [minimal_service]: Take off incoming request
height: 0.500000
duration: 2.000000
response: 1
```

Listing C.4: *Server node* receives a request.

```
ros2 service call /drone1/takeoff cf_messages/srv/TakeOff "{height: 0.5, duration:
    2}"
```

Listing C.5: *Client node take-off* command line.

```
Preparing for takeoff...
[INFO] [1622643703.314919500] [minimal_client]: Sending information to server:
    height: 0.500000
duration: 2.000000
response: 1
```

Listing C.6: *Client node take-off* message.

```
ros2 service call /drone1/gotorel cf_messages/srv/GoTo "{x: 0.0, y: 0.5, z: 0.0,
    yaw: 0.0, duration: 2}"
```

Listing C.7: *Client node goto* command line.

```
Preparing for takeoff...
[INFO] [1622643703.314919500] [minimal_client]: Sending information to server: x:
    0.0
y: 0.5
z: 0.0
yaw: 0.0
duration: 2
response: 1
```

Listing C.8: *Client node goto* message.

```
ros2 service call /drone1/land cf_messages/srv/Land "{height: 0.5, duration: 2}"
```

Listing C.9: *Client node land* command line.

```
Preparing for takeoff...
[INFO] [1622643703.314919500] [minimal_client]: Sending information to server:
    height: 0.500000
duration: 2.000000
response: 1
```

Listing C.10: *Client node land* message.

```
Preparing for takeoff...
[INFO] [1622644217.633422100] [minimal_client]: service not available, waiting
    again...
```

Listing C.11: *Service not available* message.

References

[1] 2021 international conference on advanced robotics. Available at http://icar-2021.org/. Accessed on 21.06.2021.

[2] Aeronaves não tripuladas (uas/drones). Available at https://www.anac.pt/vPT/Generico/drones/Paginas/AeronavesCivisPilotadasRemotamente.aspx. Accessed on 25.06.2021.

[3] Authentication. Available at https://emotiv.gitbook.io/cortex-api/authentication/getuserlogin. Accessed on 24.05.2021.

[4] Data sample object. Available at https://emotiv.gitbook.io/cortex-api/data-subscription/data-sample-object. Accessed on 29.06.2021.

[5] Drone. Available at https://www.merriam-webster.com/dictionary/drone. Accessed on 25.06.2021.

[6] Intelli 2021. Available at https://www.iaria.org/conferences2021/INTELLI21.html. Accessed on 21.06.2021.

[7] Qingsong Ai, Quan Liu, Wei Meng, and Sheng Quan Xie. Chapter 6 - eeg-based brain intention recognition. In Qingsong Ai, Quan Liu, Wei Meng, and Sheng Quan Xie, editors, *Advanced Rehabilitative Technology*, pages 135 – 166. Academic Press, 2018.

[8] Airbus. Urban air mobility. Available at https://www.airbus.com/innovation/zero-emission/urban-air-mobility.html. Accessed on 11.06.2021.

[9] Pooja Akhtar, Sujata Yardi, and Murtaza Akhtar. Effects of yoga on functional capacity and well being. *International journal of yoga*, 6:76–9, 02 2013.

[10] Amazon. Amazon prime air. Available at https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011. Accessed on 11.06.20121.

[11] Microsoft Azure. O que é paas? plataforma como serviço: Microsoft azure. Available at https://azure.microsoft.com/pt-pt/overview/what-is-paas/. Accessed on 28.06.2021.

[12] Alekhyo Banerjee. Computational complexity of svm. Available at https://alekhyo.medium.com/computational-complexity-of-svm-4d3cacf2f952, Aug 2020. Accessed on 30.11.2020.

[13] Federica Bazzano, Paolo Montuschi, F. Lamberti, Gianluca Paravati, Silvia Casola, Gabriel Cerón Viveros, Jaime Londoño, and Flavio Tanese. Mental workload assessment for uav traffic control using consumer-grade bci equipment. pages 60–72, 12 2017.

[14] Bitcraze. Crazyflie 2.1. Available at https://store.bitcraze.io/products/crazyflie-2-1. Accessed on 30.06.2021.

[15] Bitcraze. Crazyflie 2.1. Available at https://www.bitcraze.io/products/crazyflie-2-1/. Accessed on 30.06.2021.

[16] Bitcraze. Loco positioning system. Available at https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/. Accessed on 30.06.2021.

[17] Brad Cain. A review of the mental workload literature. *English*, page 35, 07 2007.

[18] European Commission. Urban air mobility (uam). Available at https://smart-cities-marketplace.ec.europa.eu/action-clusters-and-initiatives/action-clusters/sustainable-urban-mobility/urban-air-mobility-uam. Accessed on 20.06.2021.

[19] The Qt Company. Embedded software development tools: Cross platform ide: Qt creator. Available at https://www.qt.io/product/development-tools. Accessed on 16.06.2021.

[20] EMOTIV. Connecting to the cortex api. Available at https://emotiv.gitbook.io/cortex-api/connecting-to-the-cortex-api. Accessed on 24.05.2021.

[21] EMOTIV. Emotivbci. Available at https://www.emotiv.com/emotiv-bci/. Accessed on 28.06.2021.

[22] EMOTIV. Getting started. Available at https://emotiv.gitbook.io/cortex-api/. Accessed on 17.06.2021.

[23] EMOTIV. Python example. Available at https://github.com/Emotiv/cortex-v2-example/tree/master/python. Accessed on 28.06.2021.

[24] EMOTIV. Developers - join our worldwide community. Available at https://www.emotiv.com/developer/, Sep 2020. Accessed on 25.05.2021.

[25] EMOTIV. Emotiv epoc 14-channel wireless eeg headset. Available at https://www.emotiv.com/epoc/, Sep 2020. Accessed on 13.07.2021.

[26] EMOTIV. Overview of api flow. Available at https://emotiv.gitbook.io/cortex-api/overview-of-api-flow, 2020. Accessed on 15.06.2021.

[27] DDS Foundation. What is dds? Available at https://www.dds-foundation.org/what-is-dds-3/. Accessed on 16.06.2021.

[28] Python Software Foundation. multiprocessing - process-based parallelism. Available at https://docs.python.org/3/library/multiprocessing.html. Accessed on 28.06.2021.

[29] Uttara Gogate, Alap Parate, Shubham Sah, and Sagar Narayanan. Real time emotion recognition and gender classification. In *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)*, pages 138–143, 2020.

[30] Google. Classification: True vs. false and positive vs. negative. Available at https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative. Accessed on 17.06.2021.

[31] Michael Grieves. Origins of the digital twin concept. 08 2016.

[32] N. Grigoropoulos and S. Lalis. Simulation and digital twin support for managed drone applications. In *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–8, 2020.

[33] Werner Grohmann. *Von der Software zum Service: ASP-Software on Demand- Software-as-a-Service- Cloud Computing ; Neue Formen der Software-Nutzung*. H.K.P. Consulting, 2009.

[34] M. Hinchey and L. Coyle. Evolving critical systems: A research agenda for computer-based systems. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pages 430–435, 2010.

[35] Hee Yong Jeon, Cedric Justin, and Dimitri Mavris. Improving prediction capability of quadcopter through digital twin. 01 2019.

[36] Wil Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3, 04 2018.

[37] Alex Kreilinger, Vera Kaiser, Christian Breitwieser, John Williamson, Christa Neuper, and Gernot Müller-Putz. Switching between manual control and brain-computer interface using long term and short term quality measures. *Frontiers in neuroscience*, 5:147, 01 2011.

[38] Andrea Kübler. The history of bci: From a vision for the future to real support for personhood in people with locked-in syndrome. *Neuroethics*, 13(2):163–180, 2020.

[39] Chris Manna. An intro to the crisp-dm methodology. Available at https://medium.com/@chrismanna/an-intro-to-the-crisp-dm-methodology-c58cbe0371a3, Jun 2019. Accessed on 21.03.2021.

[40] Rafael Ramirez and Zacharias Vamvakousis. Detecting emotion from eeg signals using the emotive epoc device. In Fabio Massimo Zanzotto, Shusaku Tsumoto, Niels Taatgen, and Yiyu Yao, editors, *Brain Informatics*, pages 175–184, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[41] Open Robotics. Creating custom ros 2 msg and srv files. Available at https://docs.ros.org/en/foxy/Tutorials/Custom-ROS2-Interfaces.html. Accessed on 13.06.2021.

[42] Open Robotics. Installing ros 2 on windows. Available at https://docs.ros.org/en/foxy/Installation/Windows-Install-Binary.html. Accessed on 14.06.2021.

[43] Open Robotics. Introducing turtlesim and rqt. Available at https://docs.ros.org/en/foxy/Tutorials/Turtlesim/Introducing-Turtlesim.html. Accessed on 30.06.2021.

[44] Open Robotics. Understanding ros 2 services. Available at https://docs.ros.org/en/foxy/Tutorials/Services/Understanding-ROS2-Services.html. Accessed on 30.06.2021.

[45] Open Robotics. Writing a simple service and client (python). Available at https://docs.ros.org/en/foxy/Tutorials/Writing-A-Simple-Py-Service-And-Client.html. Accessed on 30.06.2021.

[46] Manas Sambare. Fer-2013. Available at https://www.kaggle.com/msambare/fer2013. Accessed on 20.06.2021.

[47] Tianwei Shi, Hong Wang, and Chi Zhang. Brain computer interface system based on indoor semi-autonomous navigation and motor imagery for unmanned aerial vehicle control. *Expert Systems with Applications*, 42(9):4196 – 4206, 2015.

[48] Amit Raj Singh. Drone and smart cities- how are uavs crucial for smart cities? Available at https://www.geospatialworld.net/blogs/how-drones-are-crucial-for-smart-cities/, Apr 2018. Accessed on 30.06.2021.

[49] Dirk Thomas. Rqt - package summary. Available at http://wiki.ros.org/rqt. Accessed on 16.06.2021.

[50] Ravi M. Vishwanath, Saumya Kumaar Saksena, and S. N. Omkar. A real-time control approach for unmanned aerial vehicles using brain-computer interface. *CoRR*, abs/1809.00346, 2018.