

Daniel Nicolai Grønn

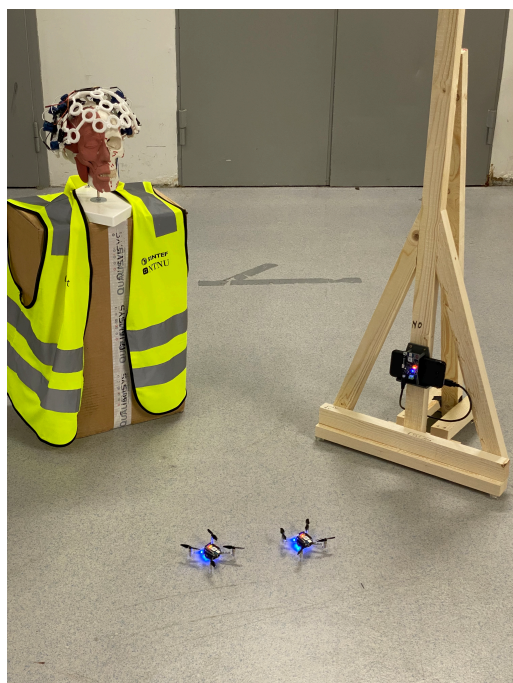
A Modular Brain-Computer Interface Framework for Autonomous Swarm Drone Navigation

Masteroppgave i Cybernetics and Robotics

Veileder: Marta Molinas

Medveileder: Luis Alfredo Moctezuma

Juli 2025



NTNU

Kunnskap for en bedre verden

Daniel Nicolai Grønn

A Modular Brain-Computer Interface Framework for Autonomous Swarm Drone Navigation

Masteroppgave i Cybernetics and Robotics

Veileder: Marta Molinas

Medveileder: Luis Alfredo Moctezuma

Juli 2025

Norges teknisk-naturvitenskapelige universitet



Kunnskap for en bedre verden

ABSTRACT

This thesis presents the development of a modular framework for controlling a swarm of drones using a brain-computer interface (BCI). The motivation lies in enabling intuitive, hands-free interaction with robotic systems, an approach with potential applications in assistive technologies, human-robot collaboration, and neuroergonomics. Controlling drones through thought alone offers a compelling, real-time visualization of brain activity and can be generalized to any actuated system.

Electroencephalography (EEG), a non-invasive method of recording electrical brain signals via scalp-mounted electrodes, is widely used in BCIs to capture neural activity linked to motor imagery, where users imagine limb movements to generate distinct brain patterns. The EEG signals will be acquired using the OpenBCI Ultracortex Mark IV headset and used to drive a system for drone swarm control, however, they are not recorded in this thesis.

The core contribution of this thesis is a flexible and extensible software framework for EEG-based control of multiple Crazyflie 2.1 drones developed in Python. Although full real-time classification is not implemented, the system supports EEG data acquisition and protocol-based recording for BCI experimentation. A graphical user interface (GUI) was developed to allow for intuitive path planning, swarm management, and offline collision detection. Drone positioning is achieved through a Time Difference of Arrival (TDoA) system, enabling localization within a defined 3D flight volume.

Technical highlights include the deployment of a reconfigurable and mobile positioning infrastructure using the Loco positioning system, achieving a spatial precision of 11 centimeters in radius and a general absolute accuracy of 17 centimeters within most of the flying area. A robust path planner with support for offline collision detection was implemented, enabling the successful simulation of swarm trajectories. The system was also tested in hardware-in-the-loop scenarios, with most tests succeeding. However, final tests involving the formation of letter shapes failed due to drone instability.

While real-time swarming and letter formation remain challenging, this work lays a strong foundation for future development. All core goals of the thesis have been met. By delivering an open, extensible, and modular framework, along with a compelling proof-of-concept, our project aims to encourage tomorrow's researchers to pursue the field of EEG and BCI technologies, ultimately contributing to assistive tools for individuals with mobility limitations.

SAMMENDRAG

Denne avhandlingen presenterer utviklingen av et modulært rammeverk for styring av en sværm med droner ved hjelp av et hjerne-datamaskin-grensesnitt (BCI). Motivasjonen er å muliggjøre intuitiv, håndfri interaksjon med robotsystemer, noe som har potensielle anvendelser innen hjelpemiddelteknologi, menneske-robot interaksjonssamarbeid og nevroergonomi. Droner styrt kun gjennom tankene blir så brukt som en engasjerende sanntidsvisualisering av bruksområdene for hjerneaktivitets-styrte applikasjoner og kan overføres til kontroll av alle slags aktuerte systemer.

Elektroencefalografi (EEG) er en ikke-invasiv metode for å registrere elektriske hjernesignaler via elektroder festet til hodebunnen, og brukes mye i BCI for å fange opp nevralt aktivitet knyttet til motorisk bevegelse-forestilling (MI). MI utføres når brukere forestiller seg bevegelse av lemmer for å generere hjernemønstre som likner hjernemønstre der bevegelsen ble utført. OpenBCI Ultracortex Mark IV-headsettet blir brukt som datainnsamlingsverktøy, og selv om fokuset til rapporten ikke inkluderer innhenting av data, er resten av systemet skreddersydd for sværmskontroll av droner ved bruk av disse signalene.

Hovedbidraget i avhandlingen er et fleksibelt og utvidbart rammeverk for EEG-basert kontroll av flere droner. Selv om full sanntidsklassifisering ikke er implementert, støtter systemet innsamling av EEG-data og protokollbasert opptak for BCI-eksperimenter. Det er utviklet grafiske brukergrensesnitt (GUI) som muliggjør intuitiv ruteplanlegging, sværm-administrasjon og offline kollisjonsdeteksjon. Dronenes posisjon bestemmes ved hjelp av et eksternt posisjoneringssystem, som gir lokalisering innen et definert tredimensjonalt flyvolum.

Tekniske fremskritt fra denne rapporten inkluderer implementering av en rekonfigurerbar og mobil posisjon infrastruktur, som oppnår en romlig presisjon på 11 cm i radius og en romlig absolutt nøyaktighet på 17 cm i størstedelen av flyområdet. En robust ruteplanlegger med støtte for offline kollisjonsdeteksjon er utviklet, og sværm-baner ble med simulert. Systemet er også testet i hardware-in-the-loop-scenarier, hvor de fleste av testene lyktes. Den siste testen, der dronene skulle forme bokstaver i luften, mislyktes imidlertid på grunn av ustabilitet i dronene ved tett flyvning.

Selv om dronesværm, særlig ved bokstavforming i plassbegrensede applikasjoner, slik som dette viser seg å være utfordrende, legger dette arbeidet et solid grunnlag for videre utvikling. Alle kjernemålene i rapporten er oppnådd. Ved å utvikle et åpen, utvidbar og modulært rammeverk, som er testet gjennom et proof-of-concept, er det dette projektet som mål å engasjere morgendagens forskere til å utforske feltet EEG- og BCI-teknologi, og dermed bidra til utviklin-

gen av hjelpemidler for personer med bevegelseshemninger.

PREFACE

This thesis marks the end of my Master’s degree in Cybernetics and Robotics at NTNU. Working on this project has been both technically challenging and deeply rewarding, allowing me to explore the intersection of the human brain and drone control through brain-computer interfaces (BCIs). This topic intrigued me from the moment I saw the project presented during the matriculation ceremony on my first day at NTNU almost two years ago. I find it both satisfying and poetic that I have now contributed to the very same project that first sparked my interest in Cybernetics and robotics as a whole, hopefully inspiring future researchers to pursue this field as well.

The demonstrational video of the swarm setup can be watched on youtube through this link: <https://youtu.be/DwOjBqX0rrw>

I want to express my sincere gratitude to my supervisor, Professor Marta Molinas, for her enthusiasm, support, and encouragement throughout the project. I am also thankful to my co-supervisor, Luis Alfredo Moctezuma, for his valuable technical input, ongoing help, and constructive feedback during the evaluation of deliverables. This thesis builds upon my specialization project from the fall of 2024 [1], as well as the earlier work of Bynestad and Vatsvåg [2], and Lønvik and Jonassen [3].

A special thanks to Mihir Vinay Kulkarni for granting me access to the drone testing lab, which made the practical aspects of this work possible. I am also grateful to the other lab members, especially Arvin Daas, for the assistance, good times, and collaboration during testing and development.

Finally, I would like to thank my family for their encouraging words during the more intense phases of writing and my wife Marte for her understanding and patience through long days and late nights. You are the best.

CONTENTS

Abstract	i
Preface	iii
Contents	ix
List of Figures	ix
List of Tables	xii
Abbreviations	xiii
1 Introduction	1
1.1 Background and Motivation	1
1.2 Previous Work	2
1.3 Scope	2
1.4 Objectives	3
1.5 Thesis structure overview	4
2 Theory	7
2.1 Brain-Computer Interfaces (BCIs)	7
2.1.1 The human brain	7
2.1.2 Invasive and non-invasive BCIs	7
2.1.3 The significance of the Motor Cortex	8
2.1.4 Motor Imagery	8
2.1.5 Why Motor Imagery is Challenging	8
2.2 Machine learning methods and evaluation of EEG BCI signals	8
2.2.1 Classifiers	9
2.2.2 Evaluation	10
2.3 Micro Quadcopters	11
2.3.1 PID Control in Embedded Flight Systems	11
2.3.2 Time Difference of Arrival (TDoA) Localization	12
2.3.3 The Kalman Filter and Extended Kalman Filter	13
2.3.4 Sensor Fusion in Embedded Robotics	15
2.4 Software Development	16
2.4.1 Object-Oriented Programming (OOP)	17
2.4.2 Version Control with Git	17

2.4.3	Multithreading	17
2.4.4	Graphical User Interfaces with Tkinter	17
3	Related Work	19
3.1	MI-Based BCI for Drone and Swarm Control	19
3.2	General Frameworks and BCI System Reviews	20
3.3	Drone-swarm control architectures and path planning	20
3.3.1	Centralized vs. distributed control	20
3.3.2	Formation control and swarm coordination	20
3.3.3	Path planning across swarms	21
3.3.4	Integration with MI-BCI user interface	21
3.4	Summary and chosen direction	21
4	Platform Architecture	23
4.1	System overview	23
4.1.1	OpenBCI EEG Headset	23
4.1.2	Path planning GUI	24
4.1.3	Crazyflie 2.1 Quadcopter	25
4.1.4	Loco Positioning System	26
4.2	Demonstration framework overview	28
4.3	Earlier project versions' process overviews	31
5	Command Input Systems	33
5.1	The dual input-source system	33
5.2	Command mapping	34
5.3	EEG BCI headset setup	35
5.3.1	Evaluation of Alternative BCI Systems	35
5.3.2	Recommendation for BCI solution going forward	36
5.3.3	Electrode configuration	37
5.4	Classification Pipeline	37
5.4.1	Preprocessing	37
5.4.2	Classification and Feature Extraction	38
5.4.3	Flat vs. Hierarchical Structure	38
5.4.4	Best Performing Pipelines	38
5.4.5	Using Data in real-time	39
5.5	Potential improvements to the BCI and input system	39
6	Drone Positioning System Setup	41
6.1	Choosing a positioning system	41
6.2	Physical Node Positioning	43
6.2.1	Stand Design and Node Mounting	43
6.2.2	Layout and Dimensions	47
6.2.3	Labeling and Setup Procedure	48
6.3	Setting up and using the positioning system	48
6.4	Position estimate accuracy and precision	49
6.4.1	The measurement setup	49
6.4.2	Inverse Distance Weighted Interpolation	51
6.4.3	Position accuracy	52
6.4.4	Position precision	53

6.4.5	Accuracy and Precision evaluation	55
6.5	To further increase the performance of the Loco positioning system	56
6.5.1	Adjustments to the physical node setup	56
6.5.2	Increasing Measurement Accuracy	56
6.5.3	Tuning the Controller	56
6.5.4	Tuning the Kalman Filter	57
6.5.5	Tuning the Outlier Filter	57
7	Path Planning	59
7.1	The reason to make a path planner	59
7.2	Path Planning Software Design	60
7.3	A Path Planner tailored to the Crazyflie 2.X quadcopter swarm . .	61
7.4	Path planner architecture	62
7.4.1	Drone Representation	63
7.4.2	Simulated Drone Extension	64
7.4.3	PointCanvas Interface	64
7.4.4	Drone Integration and Drawing	64
7.4.5	Data Structure and Storage	65
7.4.6	Graphical Path Definition	65
7.4.7	Editing and Reusability	65
7.4.8	Concurrency and Command System	66
7.4.9	CanvasManager: Coordinated Control and Synchronization of PointCanvas Instances	66
7.5	Simulation and collision danger detection	67
7.5.1	Simulation of drone paths	67
7.5.2	Comparison to Polynomial Interpolation	68
7.5.3	Collision danger detection	68
7.6	Starting position configuration	69
7.7	Setting up and using the drones	71
7.8	Application of the path planner - Path planning for writing letters with drone lights	73
7.8.1	Revisiting the Vision: Writing “NTNU” with Drones	73
7.8.2	Precision Requirements and Spatial Constraints	73
7.8.3	The Drone Font	73
7.8.4	Considerations for 3D Spatial Layout	75
7.8.5	Intermediate Transition Plane for Safety	76
7.8.6	Using the path planner to define paths forming letters	77
7.8.7	Possible improvements to the path planner	79
7.9	Requirements for Supporting New Drone Platforms	79
7.10	Creating an interface between the old and the new	80
8	System Integration and Testing	81
8.1	Integrating the swarm functionality with the old GUI framework . .	81
8.2	Developing for a future-proof framework	84
8.3	Implementation of safety measures	85
8.4	A test of using the framework	85
8.4.1	Basic test with one drone through the swarm controller	86

8.4.2	Dual path flying sequence with 2 drones with reduced collision danger due to separated paths	86
8.4.3	Dual path with 2 drones, flying above one another	87
8.4.4	Testing the use of random headset inputs with a dual path flying sequence using one drone	87
8.4.5	Trying to set up a dual path flying sequence with 8 drones in a confined space using keyboard inputs N -> K path with keyboard commands	89
8.4.6	Framework test conclusion	89
8.5	Multi-Pilot Control of Drone Swarms Using Dual BCIs	90
8.5.1	Motivation and Conceptual Benefit	90
8.5.2	Swarm Division Strategy	90
8.5.3	Collision Avoidance and Spatial Separation	91
8.5.4	System Architecture Modifications	91
8.5.5	Feasibility and Workload Evaluation	91
9	Discussion	93
9.1	Loco Positioning System Performance	93
9.2	Path Planning reflection	94
9.3	System Integration and Testing reflection	94
9.4	Usability and Practicality	95
10	Conclusion	97
10.1	Future work	97
	References	99
	Appendices	105
A	Project Github repository	106
B	User Guide	107
B.1	Contribution log	108
B.2	What you need to get started	108
B.3	Preparing Hardware	109
B.3.1	Charging the batteries	109
B.3.2	Crazyflie V2 Drone	110
B.3.3	Headset and electrode configuration	112
B.4	Preparing Software	114
B.4.1	Installing python	114
B.4.2	Coding IDE	114
B.4.3	Download Git	114
B.4.4	Download Github Desktop	114
B.4.5	Install the code repository from Github wavesresearch	115
B.4.6	Download OpenBCI GUI	117
B.5	Installing Drivers for Windows	117
B.5.1	Crazyradio USB dongle driver	117
B.5.2	Cyton USB dongle driver	118
B.6	Using the framework	118

B.6.1	Running the software	118
B.6.2	Using the software	121
B.6.3	Using the drone	124
B.6.4	Using the headset	124
B.7	Setting up for further development	126
B.8	Issues and Fixes	128
B.8.1	ModuleNotFoundError	128
B.8.2	BCI headset connection problems	128
B.8.3	Drone connection problems	129
B.8.4	Running downloaded scripts disabled error	129
B.8.5	Automatically detecting the port of the Cyton board	130
B.8.6	Export requirements to text file	130
C	Node stand - Technical drawings	132
D	AI declaration	136

LIST OF FIGURES

2.1	A visualization of the TDoA planes and the most likely location of the receiver.	13
2.2	A 2D example of how the Kalman filter takes both sensor input and model prediction into account when determining the most probable state of the drone. Figure from Bitcraze's blogpost about the KF [39]	14
2.3	An interface created using <code>tkinter</code>	18
4.1	The picture shows the OpenBCI Ultracortex IV headset frame with the cyton board and eight electrodes. In front lies the USB dongle used to transmit data.	24
4.2	One of two programs used for path planning. This is the path definition program, where on the canvas to the left one can define and edit points in a path and visualize in 3D on the right	25
4.3	The Drone used in this project is the Crazyflie 2.1, as shown here, but the earlier version 2.0 works just as well	26
4.4	The Loco positioning system node, used together with the Crazyflie platform to give a position estimate to enable swarming	27
4.5	The self-made node stands made for this thesis, to hold the nodes for positioning system. For production drawings see Appendix C . .	28
4.6	High-level demonstration system overview showing information flow, command inputs and the necessary parts to use the swarm demonstration program	30
4.7	A flowchart of what programs need to be run when, and where they are run from.	31
4.8	Process Overview from previous iteration of project with focus on recording and classification presented in [2]	31
4.9	Live demonstration process overview presented in [1]	32
5.1	An example class mapping for the final "NTNU" drone show. The keyboard and headset would in many cases also be operated by two different users.	34
5.2	The electrodes (highlighted green) which were used in testing. A1 and A2 are used as ref and noise cancel electrodes on the earlobes (highlighted black), and T3 and T4 (highlighted yellow) are occupied with support parts to hold the headset correctly on the head, not as electrodes. Image from [50]	37

6.1	Node stand hardware	44
6.2	The stand seen from the side	45
6.3	The stand has two hinges that allow it to fit both transportation and use. The stands are designed for portability.	46
6.4	The main technical drawing for the stand assembly, all drawing pages in full resolution are included as an appendix in Appendix C	46
6.5	The base markings on the stands assure correct stand configuration during setup.	48
6.6	Position estimate logging for calculation of measurement accuracy and precision	50
6.7	Position estimate precision (visualized by orange line) and accuracy (visualized by gray stipulated line)	51
6.8	The position accuracy measurement defined as the distance from the mean of logged data to the real position	52
6.9	This figure shows a heatmap of the positional estimate's accuracy	53
6.10	Point cloud and precision ellipsoids for two selected points of the volume. One is the theoretically best point, while the other is the worst corner in practice.	54
6.11	This figure shows a heatmap of the positional estimate's precision	54
6.12	Values for position estimate precision and accuracy alongside the node number and placing in space	55
7.1	The class diagram for the Path planner. It uses UML style as described in [57]. The attributes and methods which are colored, represent functionality which is connected to a class with the same color.	63
7.2	Drone position start configurations	70
7.3	The standard starting position	71
7.4	After configuring a drone ID have the last two characters of the ID been written on the drone. Drone 01, 02, 03 and 04.	72
7.5	The proposed drone font, balancing complexity and, in turn, intuitiveness, and practicality	74
7.6	The drone font is defined as shown earlier, but this figure shows a problem with both too complex and too simple letters	75
7.7	The inclined transition plane used as a safety measure for reduced drone path crossing. As shown by the last two subfigures, Figure 7.7c and Figure 7.7d, is the transition path from the start position to the inclined transition plane without crossings for 8 drones.	76
7.8	Two letter paths which should work to form the "N", which represents two out of four letters of "NTNU", and is the most complex out of them. Also being able to transform into a different letter, in this case "K"	78
7.9	The two letters formed by the simulated drones.	78
B.1	Running the main menu GUI program in Windows	107
B.2	Battery charging of drone battery (left), and headset battery (right)	109
B.3	Bent charger pins can lead to damaged battery and/or charger. Bend them back to avoid! The difference in the image was large enough to cause the issue.	110

B.4	Crazyflie 2.X drone	110
B.5	Symbol shows forward and up direction on the drone	111
B.6	The drone's on/off button	112
B.7	Electrode pin connections cyton board	113
B.8	Electrode configuration headset (Front is up)	113
B.9	Headset and headset dongle setting	114
B.10	Cloning the repository	116
B.11	Finding the repository location	116
B.12	Zadig installer settings	117
B.13	VCP driver download location	118
B.14	Run With PowerShell	119
B.15	The 4 steps to set up the shortcut	120
B.16	Main menu where all programs can be ran from	121
B.17	Recording protocol GUI	122
B.18	training parameter selection GUI	123
B.19	A confusion matrix	123
B.20	Control GUI echo	124
B.21	The cfclient	124
B.22	OpenBCI GUI Autoconnecting to headset	125
B.23	After adjusting the electrodes to get good impedance values	126
B.24	The function description for the reset_estimators() command . . .	129

INTRODUCTION

1.1 Background and Motivation

People with severe motor impairments often lose the ability to perform voluntary movements, yet retain cognitive function. For such individuals, invasive and non-invasive brain-computer interfaces (BCIs) based on electroencephalography (EEG) have emerged as effective tools to translate neural intent into actionable commands [4, 5].

A prominent paradigm in current research is motor imagery (MI), where users imagine movements—such as hand or foot—without physical execution. MI generates characteristic EEG patterns (e.g., sensorimotor rhythms) that can be decoded and classified in real time [5]. These signals have been successfully used to control assistive devices, from wheelchairs [4] to robotic exoskeletons, offering a pathway to increased autonomy and rehabilitation potential [6].

Despite the promise, deploying EEG-based BCIs in practical settings requires robust frameworks, including data processing pipelines, user-friendly interfaces, and clear paths for future researchers to build upon. While previous work addresses the first of these, the effect of having user-friendly interfaces and facilitating future researchers should not be underestimated. To address these demands, this work focuses on establishing a solid foundation through the use of replicable protocols, documented tools, modular software, and comprehensive evaluation methodologies.

To demonstrate the potential of MI-derived EEG signals in real-world scenarios, we present a visually stimulating application: the control of a drone swarm. Prior studies combining MI and visual imagery for drone swarm control have shown feasibility [7, 8]. Our goal is to extend this by integrating an EEG-driven, visually engaging interface that highlights both usability and inspiration for future researchers.

Combining EEG control with drone swarms offers two synergistic advantages: leveraging users' cognitive input to direct multiple agents, and exploring collaborative control paradigms such as dual-pilot arbitration. No prior work at NTNU has demonstrated even single-agent EEG control in a swarm context, let alone with two simultaneous users.

By delivering an open, extensible framework and a compelling proof-of-concept, our project aims to encourage tomorrow's researchers to pursue EEG and BCI

technologies, ultimately contributing to assistive tools for individuals with mobility limitations.

1.2 Previous Work

One of the earliest notable demonstrations of using EEG signals for drone control was conducted by LaFleur et al. in 2013, where a non-invasive motor imagery (MI)-based brain–computer interface was used to control a quadcopter in three-dimensional space [9]. This work marked a significant milestone by showing that imagined motor commands could be translated into real-time spatial control of a flying robot, paving the way for future exploration in assistive mobility and neuroadaptive systems.

Following this, other studies explored alternative control schemes. Ram and Zhang (2016), for instance, used eye blinks detected from EEG signals to trigger drone commands [10]. Their work coincided with the rise of consumer-grade EEG headsets, which increased accessibility for both hobbyists and academic institutions. However, limitations in form factor, signal quality, and software maturity presented challenges to widespread adoption.

In recent years, master’s students Kjersti Brynstad and Erlend Vatsvåg (2021) expanded on this line of research by implementing live EEG control of the Parrot AR Drone 2.0 platform using MI signals [2]. While their system achieved real-time interaction, the Parrot drone has since become outdated. As a result, this project transitions to using the open-source Crazyflie drone from Bitcraze, which offers a modular, extensible, and well-documented hardware ecosystem suited for research.

Building further on this foundation, the 2023 master’s thesis by Helene Tørle Lønvik and Pauline Mørch Jonassen explored several techniques for EEG-based motor imagery classification, with a particular focus on pipelines that support real-time control [3]. Their most promising pipeline, which included robust preprocessing steps and a carefully tuned convolutional neural network (CNN), demonstrated reliable performance on real-time classification tasks. The present work adopts this pipeline as a starting point for integrating motor imagery control into the new Crazyflie-based system, although the actual use of the pipeline is outside the scope of this thesis.

Together, these past efforts form the groundwork for this thesis. By combining lessons learned from previous implementations with newer hardware and modern classification techniques, this project aims to establish a sustainable platform for future research in EEG-based drone control.

1.3 Scope

Effective research in brain–computer interface (BCI) systems—particularly those involving hardware components such as drones—requires a well-defined and robust framework for testing and comparison. The nature of this project, centered around real-time drone demonstrations driven by EEG signals, introduces practical challenges related to hardware-in-the-loop execution, safety, and environmental constraints.

Previous researchers have, for good reason, placed their emphasis on the software pipeline and EEG signal classification, areas that can be developed and tested without reliance on physical setups. In contrast, this thesis focuses on laying the foundational infrastructure needed to enable and support future research that involves real-time interaction with physical drone swarms. This includes both software and hardware components designed with modularity, portability, and extensibility in mind.

Due to this focus, several areas have been deemed outside the feasible scope of this thesis. In-depth experimentation with EEG-based BCI paradigms, fine-tuning of control algorithms, and detailed analysis of positioning system accuracy are beyond the available time, resources, and access to testing facilities. The swarm configuration itself introduces additional constraints, including the need for larger, controlled spaces and stringent safety considerations, which limited testing opportunities during the project period.

Nonetheless, the contributions made here are intended to significantly lower the barrier to contributing to future development. Researchers who build on this work will find a platform that enables the exploration of advanced topics such as controller optimization, cooperative EEG control schemes, and real-time feedback integration. It is therefore encouraged to view this thesis not as an endpoint, but as a foundation that enables a framework for continued innovation, with specific recommendations for future work presented in the concluding sections.

To further simplify the setup process, a comprehensive User Guide for the framework has also been developed. This guide is meant to encompass as much of the setup process and use of the framework as possible, come with tips regarding common issues, and therefore help a user get quickly to demonstrating or continuing development. This User Guide is attached as an appendix and can be found in Appendix B.

1.4 Objectives

This thesis presents the development of a drone flight environment designed to support and advance future research in BCI technologies. By integrating real-time drone demonstrations, the system serves as a practical and flexible platform for evaluating BCI-based control strategies, enabling researchers to investigate neural signal interpretation, user interaction, and system responsiveness in a controlled yet dynamic setting.

The primary objective of this thesis is to develop a modular and extensible drone flight environment tailored for research on brain-computer interface (BCI) technologies. Through the integration of real-time drone demonstrations, the system aims to facilitate robust testing of real-time neural signal interpretation, control strategies, user interaction, and system responsiveness.

To achieve this, the following specific objectives have been defined:

1. **Construct the physical infrastructure** required to enable the Crazyflie drones to operate with the Loco Positioning System (LPS), including mounting hardware and anchor placement.
2. **Ensure system portability** by designing a mobile, low-overhead setup

that can be deployed across different environments with minimal calibration and setup time.

3. **Evaluate the accuracy of the positioning system** in various configurations to determine its reliability for real-time drone control and coordinated flight patterns.
4. **Develop a simulation and visualization framework** with an interactive graphical user interface (GUI) that supports path planning, collision detection, and multi-agent coordination.
5. **Design the system architecture to facilitate future development**, ensuring modularity, documentation, and integration points for additional control strategies and interfaces.
6. **Integrate the physical and simulated systems** into a cohesive, user-friendly environment that supports intuitive interaction and seamless transitions between components.
7. **Assess the feasibility of swarm-based expression** by testing the system's ability to spell out letters or shapes using coordinated drone movements.
8. **Explore dual-pilot control mechanisms** that allow two users to cooperatively interact with a single drone swarm, enabling richer forms of input and collaborative experimentation.

These objectives collectively aim to deliver a platform that not only enables the current research but also inspires and supports future work in EEG-based drone control and assistive BCI applications.

1.5 Thesis structure overview

The thesis will firstly discuss the existing work which forms crucial parts of the framework, ranging from the work done by Bitcraze in the controller, sensor fusion and positioning estimation techniques, the BCI which was developed by earlier master students, and the GUIs developed to make the framework user-friendly and easily set up.

These parts will then be presented in the broader context, along with the newly developed systems, in the Platform architecture chapter, Chapter 4. The command system, including both the EEG BCI headset and the keyboard, will be discussed in Chapter 5, although briefly. Then Chapter 6 focuses on how the drones can achieve an absolute position estimate and how accurate this estimate is. Chapter 7 explores the importance of a simulation for path planning and collision probability reduction. The following chapter, Chapter 8, covers how all these parts function together, presents the main framework testing results, and opens up for broader discussion on contributions and future perspectives. Reflections are then discussed in Chapter 9. Here, the objectives are discussed in detail in light of the results, also defining possible directions for future developers of the framework.

Lastly, a conclusion with suggestions for future work wraps up the thesis in Chapter 9. Since this thesis defines a base upon which other research is built, it is highly suggested to read this part. Again, emphasizing the importance of enabling continuation through documentation, fixing issues along the way, and maintaining user friendliness.

It should also be noted that a second edition of the lab's User Guide Appendix B has been developed and is available in the Appendix. This user guide is made to assist in the setup and use of the framework, now also with guides related to swarming. Access to the source code to continue developing this User Guide can be received from Marta Molinas.

2.1 Brain-Computer Interfaces (BCIs)

2.1.1 The human brain

The brain is made up of billions of neurons that communicate using small electrical signals. These signals are the result of voltage changes across neuron membranes, mainly from postsynaptic potentials [11].

The electrical activity of the brain can be categorized into different frequency bands, each associated with distinct states of mind and types of cognitive processes. The primary brain frequency bands include:

- Delta (0.5–4 Hz): Associated with deep sleep and restorative processes.
- Theta (4–8 Hz): Linked to light sleep, relaxation, and creativity.
- Alpha (8–12 Hz): Represents a state of relaxed wakefulness and is often observed during meditation.
- Beta (13–30 Hz): Associated with active thinking, problem-solving, and motor control.
- Gamma (30 Hz and above): Related to high-level information processing and cognitive functioning.

The mu (μ) band is usually categorized as 8-12 Hz, which is the same as alpha activity, but it is recorded from the motor or somatosensory cortex instead of the visual cortex.

2.1.2 Invasive and non-invasive BCIs

If an electrode is placed very close to these neurons, for example, during invasive brain surgery, you can record these tiny voltage changes directly.

In non-invasive systems like EEG (electroencephalography), electrodes are instead placed on the scalp. Since the electrodes are far away from the neurons, only large groups of neurons firing together produce a strong enough signal to be detected [12, 13]. This is one of the main challenges in using EEG for control purposes.

2.1.3 The significance of the Motor Cortex

One part of the brain that often produces strong and coordinated activity is the motor cortex. This area is responsible for controlling voluntary skeletal muscle movements. When we move a limb, the motor cortex becomes active and generates rhythmic patterns in the EEG signal, especially in the mu (8–12 Hz) and beta (13–30 Hz) frequency ranges. These patterns can be picked up by scalp electrodes [13].

2.1.4 Motor Imagery

Interestingly, the brain produces similar patterns as for motor execution when imagining said movement instead. This is called motor imagery (MI). While the signal is usually weaker than real movement, it still causes recognizable changes in the EEG, especially for some individuals [14, 15].

This makes MI an exciting method in BCIs, especially for people with movement disabilities. One could, in theory, use imagination alone to control external devices, such as wheelchairs, robotic arms, or even drones.

2.1.5 Why Motor Imagery is Challenging

Although MI-based BCIs work well for some individuals, they are not equally effective for everyone. There are significant differences between individuals, as well as differences in the same person over time. These differences are called inter-subject variability and intra-subject variability, respectively [16]. Factors like fatigue, focus, and emotional state also affect the quality of the signal.

To solve this, researchers have developed new algorithms that can adapt to each user. Deep learning and subject-specific calibration are promising methods to improve reliability [15, 14].

2.2 Machine learning methods and evaluation of EEG BCI signals

A major challenge when working with EEG-based Brain–Computer Interfaces (BCIs) is the high inter- and intra-subject variability in cortical activation patterns. The same user may exhibit different spatial distributions of sensorimotor rhythms across sessions or mental states [16]. Consequently, machine learning pipelines must be tailored to each individual and often recalibrated over time to maintain robust performance. To address these challenges, various feature extraction, classification, and evaluation strategies have been developed. Importantly, no single pipeline universally outperforms others; optimal choices depend on subject-specific neurophysiology and experimental conditions [2].

2.2.1 Classifiers

Support Vector Machines (SVM)

Support Vector Machines (SVMs) are supervised learning models used for tasks like classification and regression. They work by finding the best possible boundary, called a hyperplane, that separates data from different classes. The idea is to choose the hyperplane that leaves the biggest margin between the closest points from each class. This margin helps the model generalize better to new data [17].

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis generates the best projection of the feature space that maximizes class separability by optimizing the ratio of between-class to within-class scatter [18]. In EEG BCIs, LDA has been adopted due to its computational efficiency and robustness on high-dimensional, low-sample-size data [19]. Under Gaussian assumptions, the decision boundary is linear, making LDA particularly suitable for real-time applications, if this assumption can be made.

Logistic Regression (LR)

Logistic Regression models the posterior probability of class membership via the logistic (sigmoid) function, fitting parameters by maximum likelihood estimation [20]. Unlike LDA, LR makes no assumption of equal covariance across classes, offering greater flexibility at the cost of potentially more complex model fitting. It is often used after projecting Riemannian covariance features into Euclidean space [21].

FGDA and MDRM

Fisher Geodesic Discriminant Analysis (FGDA) combines the principles of Fisher's criterion with Riemannian geometry by first mapping covariance matrices to the tangent space of the manifold. Performing classical LDA in that Euclidean tangent domain [22]. The Fisher Geodesic Minimum Distance to Riemannian Mean (FgMDRM) extends this by remapping the discriminative features back onto the manifold and classifying each trial by its minimum Riemannian distance to class centroids [22].

Riemannian Tangent Space Projection

Another Riemannian-based approach directly computes the Riemannian (affine-invariant) distance between symmetric positive-definite (SPD) covariance matrices:

$$\delta(P_1, P_2) = \|\log(P_1^{-1/2} P_2 P_1^{-1/2})\|_F.$$

Tangent space projection linearizes the manifold around a reference (e.g., the geometric mean of all training covariances), yielding vectorized features that can be classified with standard algorithms such as SVM or LR [21].

Common Spatial Patterns

Common Spatial Patterns (CSP) is a spatial filtering technique designed to maximize the variance difference between two classes of EEG signals, typically corresponding to different motor imagery tasks [19]. By computing spatial filters that project the multichannel EEG onto components with maximally differing variance, CSP effectively enhances discriminative information. The resulting spatially filtered signals serve as features for classification. CSP is widely used in BCI research due to its simplicity and effectiveness in extracting task-related oscillatory patterns, particularly in the μ and β bands.

Common Spatial Patterns with Support Vector Machines (CSPSVM)

CSPSVM combines the feature extraction power of CSP with the classification capabilities of Support Vector Machines (SVM). After extracting CSP features (typically log-variance of the filtered signals), an SVM is trained to find the optimal decision boundary between classes [23]. This hybrid approach leverages CSP’s ability to emphasize class-discriminative spatial patterns and SVM’s robustness in high-dimensional feature spaces, making it a popular choice for motor imagery classification in EEG BCIs.

Regularized Common Spatial Patterns (R-CSP)

Regularized CSP introduces shrinkage or sparsity constraints to the classical CSP algorithm to mitigate overfitting when few training trials are available [24]. By penalizing extreme spatial filters or encouraging smoothness, R-CSP improves generalization across sessions and subjects.

Discrete Wavelet Transform (DWT) Energy Features

Wavelet analysis decomposes EEG signals into time–frequency atoms, capturing transient oscillatory events. Energy features are computed from selected wavelet coefficients—often those corresponding to the μ (8–12 Hz) and β (13–30 Hz) bands—and serve as inputs to classifiers such as SVM [25, 26]. DWT features offer finer temporal resolution than fixed bandpass filters, at the expense of increased computational load.

2.2.2 Evaluation

Balanced Accuracy Score

Balanced accuracy addresses class imbalance by averaging per-class recall:

$$\text{Balanced Accuracy} = \frac{1}{2} \left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP} \right).$$

This metric ensures equal weight to both positive and negative classes, avoiding inflated performance estimates when one class dominates [27].

k-fold cross-validation

To estimate generalization performance, k -fold cross-validation partitions data into k subsets, iteratively training on $k - 1$ folds and testing on the held-out fold. Averaging results across folds yields a robust accuracy estimate and helps detect overfitting [28].

Confusion Matrix and Derived Metrics

The confusion matrix provides detailed insights into classifier mistakes by tabulating true vs. predicted labels. From this, precision, recall, and the F_1 -score can be computed to balance sensitivity and positive predictive value [29].

2.3 Micro Quadcopters

Micro quadcopters are compact unmanned aerial vehicles (UAVs) equipped with four rotors and minimal onboard payload, often weighing less than 100 grams. Due to their small size, low cost, and safety, they are widely used in indoor robotics research and education [30].

These quadcopters operate using differential thrust: by adjusting the speed of each rotor, they achieve control over roll, pitch, yaw, and vertical lift [31]. Their low inertia allows for agile flight but also makes them more sensitive to disturbances, requiring accurate control and stabilization [32].

The Crazyflie 2.X platform is a widely adopted platform in research, offering open-source firmware, modular expansion decks, and compatibility with localization and control systems, making it ideal for prototyping autonomous behaviors and conducting real-time experiments [30]. It is highly suggested to check out the previous research done with this quadcopter: <https://www.bitcraze.io/portals/research/>.

With the increased computational ability of small embedded platforms, it is possible to do advanced calculations on board. The Crazyflie handles PID-controller logic for stability, absolute position handling on-board through for example the use of Time Difference of Arrival (TDOA) measurements, and even state estimation and sensor fusion through the Extended Kalman Filter (EKF).

2.3.1 PID Control in Embedded Flight Systems

Proportional–Integral–Derivative (PID) controllers remain one of the most prevalent feedback control architectures in robotics and unmanned aerial vehicles (UAVs) due to their simplicity, effectiveness, and real-time applicability [33]. They adjust control efforts based on current, cumulative, and predicted error, making them suitable for stabilization and trajectory tracking under dynamic conditions.

Fundamental PID structure

A continuous-time PID controller has the form:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

where $e(t) = r(t) - y(t)$ represents the deviation between setpoint r and process variable y , and K_p , K_i , K_d denote respective gain coefficients [33]. Each term contributes as follows:

- *Proportional* $K_p e$ yields immediate error correction.
- *Integral* $K_i \int e$ eliminates residual steady-state bias.
- *Derivative* $K_d \dot{e}$ dampens oscillations by anticipating error trends.

Crazyflie Controller Architecture

Crazyflie uses a cascaded PID architecture implemented in the firmware:

- **Rate Loop (inner, 500 Hz):** Controls angular rates using gyro feedback.
- **Attitude Loop (500 Hz):** Controls orientation (roll, pitch, yaw).
- **Velocity Loop (100 Hz):** Controls linear velocity.
- **Position Loop (100 Hz):** Controls spatial position.

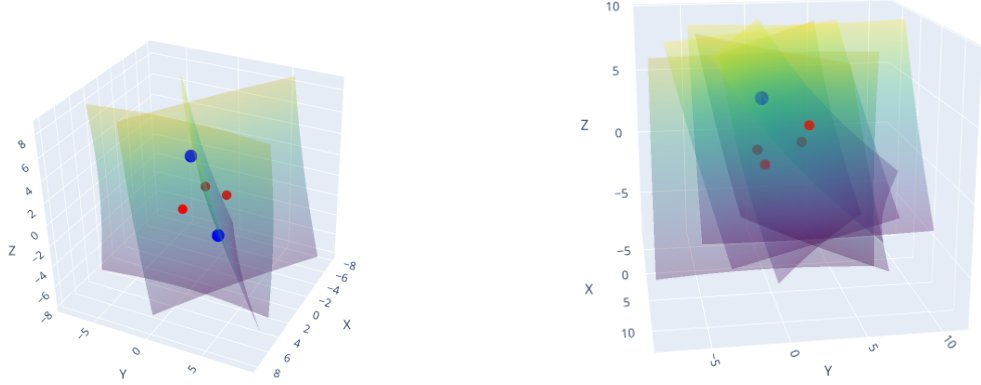
This layered structure ensures inner loops respond faster, providing stability and bandwidth to outer loops.

Key implementation points include derivative filtering, which prevents derivative kick by computing D-component from actual state change rather than raw error, anti-Windup, which helps halt integral accumulation when output saturates, and mode Cascade, which encourages stable convergence at each loop level.

2.3.2 Time Difference of Arrival (TDoA) Localization

Time Difference of Arrival (TDoA) is a localization technique that estimates the position of a signal source by measuring the differences in arrival times of the signal at multiple spatially separated receivers. Unlike Time of Arrival (ToA) methods, TDoA does not require knowledge of the exact transmission time, making it suitable for passive localization scenarios [34].

In a typical TDoA system, at least three receivers are required for two-dimensional positioning, and four for three-dimensional positioning. Each pair of receivers defines a hyperbolic curve on which the transmitter must lie; the intersection of multiple such curves yields the estimated position of the transmitter [35]. Precise time synchronization between receivers is crucial, often achieved using GPS timing or specialized protocols [36].



(a) With three nodes can the receiver be either above or below, there is no way to tell

(b) With four nodes can the receiver only be in one place. The precision of this measurement is something else, however.

Figure 2.1: A visualization of the TDoA planes and the most likely location of the receiver. There are two points which are initialized in both cases, at slightly different initial positions, and only for four nodes is the position unambiguous, which means the optimizer found the same point for both initial conditions.

Figure 2.1 is only for visualization in this thesis, but the program for visualizing was made as part of the framework to be able to visualize TDoA data later in the project. It can be found under *src/positioning_system/viz_hyper.py*. Using it ended up outside of the scope.

TDoA is employed in various applications, including real-time location systems (RTLS) in indoor environments, cellular network optimization, and interference detection in GNSS systems [37]. While TDoA offers high accuracy and scalability, challenges such as synchronization complexity and environmental sensitivity must be addressed to ensure reliable performance.

2.3.3 The Kalman Filter and Extended Kalman Filter

State estimation plays a critical role in autonomous systems, particularly when sensor noise and system uncertainties make it difficult to directly observe the true system state. The Kalman Filter (KF) and its nonlinear counterpart, the Extended Kalman Filter (EKF), offer a principled approach to estimate the hidden state of a dynamic system using probabilistic techniques. This section provides a theoretical overview of both filters, supported by insights from the implementation used in the Bitcraze Crazyflie firmware and the pedagogical explanations from [38].

Kalman Filter

The Kalman Filter is an optimal recursive estimator for linear systems with Gaussian noise. It assumes that the system evolves according to a linear process model

of the form:

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1}, \quad (2.1)$$

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k, \quad (2.2)$$

where $\mathbf{x}_k$ is the state vector at time step k , $\mathbf{u}_{k-1}$ the control input, $\mathbf{z}_k$ the measurement vector, and $\mathbf{F}_{k-1}$, $\mathbf{B}_{k-1}$, and $\mathbf{H}_k$ are the state transition, control, and observation matrices, respectively. The process noise $\mathbf{w}_{k-1}$ and measurement noise $\mathbf{v}_k$ are assumed to be zero-mean Gaussian with covariances $\mathbf{Q}$ and $\mathbf{R}$.

The Kalman Filter operates in two primary steps:

- **Prediction Step:** The filter projects the current state estimate and its uncertainty ahead in time.
- **Correction Step:** The projected estimate is updated using the new measurement and its associated uncertainty.

Mathematically, the prediction step is:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1}, \quad (2.3)$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^\top + \mathbf{Q}, \quad (2.4)$$

and the update step is:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top (\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R})^{-1}, \quad (2.5)$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}), \quad (2.6)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}. \quad (2.7)$$

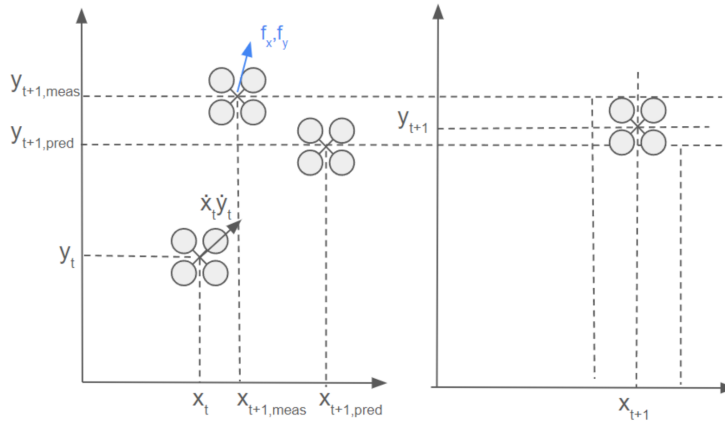


Figure 2.2: A 2D example of how the Kalman filter takes both sensor input and model prediction into account when determining the most probable state of the drone. Figure from Bitcraze’s blogpost about the KF [39]

The Kalman Filter is implemented in the Crazyflie firmware as a core module responsible for fusing IMU data with other sensor inputs. The ‘kalman_core.c’ file encapsulates the prediction and update routines through matrix operations and state handling that follow the principles outlined above.

Extended Kalman Filter

While the standard Kalman Filter assumes linearity, real-world systems such as drones often exhibit significant nonlinearities. The Extended Kalman Filter (EKF) generalizes the KF by using nonlinear functions, and then linearizing them around the current state estimate.

Given a nonlinear process model:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_{k-1}, \quad (2.8)$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k, \quad (2.9)$$

the EKF approximates the system using first-order Taylor expansions. The Jacobians of the process and measurement functions, $\mathbf{F}_k = \frac{\partial f}{\partial \mathbf{x}}$ and $\mathbf{H}_k = \frac{\partial h}{\partial \mathbf{x}}$, are computed at each time step.

The EKF follows similar steps to the KF, but incorporates these Jacobians:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}), \quad (2.10)$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1} \mathbf{P}_{k-1|k-1} \mathbf{F}_{k-1}^\top + \mathbf{Q}, \quad (2.11)$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top (\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R})^{-1}, \quad (2.12)$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k (\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})), \quad (2.13)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}. \quad (2.14)$$

In the Crazyflie ecosystem, the EKF is critical for combining high-rate IMU data with slower but more accurate sources such as optical flow, ultrasonic sensors, or UWB localization.

The conceptual clarity of the EKF, as presented in [38], emphasizes the intuition behind the filter—namely, that it performs a form of nonlinear smoothing by projecting the uncertainty forward through the nonlinear dynamics and correcting with noisy observations, all while maintaining a Gaussian approximation to the state belief.

2.3.4 Sensor Fusion in Embedded Robotics

Sensor fusion is the algorithmic integration of measurements from heterogeneous sensors. It is fundamental in robotics and autonomous systems for achieving accurate, robust, and comprehensive state estimation. It exploits complementary sensor characteristics, mitigates measurement noise, and yields more reliable outputs than any individual sensor could provide [40].

State-of-the-Art in Sensor Fusion

A rigorous literature survey, such as the one conducted by Yeong et al. [40], highlights numerous fusion strategies tailored to autonomous vehicles and robotic platforms. These include:

- **Kalman Filter Variants:** Linear (KF), Extended (EKF), and Unscented (UKF) Kalman Filters for combining inertial (IMU) and global sensors (e.g., GNSS, barometer).

- **Nonlinear and Graph-Based Methods:** Factor graph optimization, smoothing and mapping frameworks.
- **Modality Diversity:** LiDAR, camera, radar, and UWB fusion—supporting navigation, perception, and environment mapping.

Yeong et al. systematically compare these approaches, emphasizing that Kalman-based methods remain widely used in real-time embedded systems, while graph-based and unscented approaches are adopted when nonlinearity or global consistency is prioritized.

Crazyflie Sensor-Fusion Architecture

In the Crazyflie nano-quadcopter, sensor fusion is implemented via a modular EKF framework, which enables efficient integration of diverse sensors at high rates.

Benefits and Design Considerations

This implementation offers several practical advantages:

- *Sensor Modularity:* Each sensor type has its own update routine, enabling easy extension (e.g., magnetometer, barometer, mocap).
- *Computational Efficiency:* Matrix operations are optimized for a 500 Hz update cycle on a low-power microcontroller.
- *Stability Mechanisms:* Supervisor logic resets the filter when divergence is detected.
- *Adaptive Fusion:* Covariance tuning and robust cost functions automatically adjust information weighting from sensors.

2.4 Software Development

The software in this project is developed using a modular and iterative approach, prioritizing readability, reusability, and maintainability. Python is chosen as the primary programming language due to its extensive ecosystem, rapid development capabilities, and strong community support in scientific computing and machine learning [41].

Version control is managed using Git to track changes and facilitate collaboration. Development is guided by best practices such as separation of concerns, code documentation, and testing. Libraries and frameworks are selected based on reliability, performance, and integration with the project's goals.

The structure follows a layered design where interface, logic, and data handling components are kept distinct. This helps simplify debugging and future expansion, such as adapting the software for other devices or control inputs.

2.4.1 Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that organizes software around data structures called “objects,” which encapsulate both data and behaviors (methods) [42]. In Python, OOP enables clear structuring of complex systems through the use of classes, inheritance, and modular design. This makes the codebase easier to understand, reuse, and extend. In this project, OOP is used to represent components such as the GUI controller, signal processors, and classification pipelines, improving abstraction and separation of concerns.

2.4.2 Version Control with Git

Git is a distributed version control system that allows developers to track code changes, collaborate effectively, and manage multiple development branches without interfering with the main codebase [43]. By using commits, branches, and merges, Git ensures that all changes are reversible and well-documented. In this project, Git is used throughout development to manage revisions, experiment safely with new features, and maintain reproducibility. The use of the platform GitHub further facilitates collaboration and backup. Please see Appendix A

2.4.3 Multithreading

Multithreading is a programming technique that allows several tasks to run at the same time within a single program. Each task, or thread, can work independently, but all threads share the same memory. This makes multithreading useful for programs that need to do many things at once, like reading sensor data, controlling hardware, and updating a user interface.

Using multithreading can make a system more responsive and efficient, especially in real-time applications, but it also requires careful handling to avoid errors like race conditions [44].

2.4.4 Graphical User Interfaces with Tkinter

In this project, `tkinter` is used to create simple graphical user interfaces (GUIs) for user input and interaction. `Tkinter` is the standard GUI toolkit included with Python and provides a set of tools for building windows, buttons, labels, and input fields in a platform-independent manner [45]. It serves as a lightweight and accessible way to replace command-line input with interactive forms, improving usability in applications that require manual data entry or control.

The library is based on the Tk GUI toolkit and allows for rapid development of GUIs through an object-oriented programming interface. In this thesis, `tkinter` is used primarily to allow the user to enter subject information and control testing procedures in a structured and user-friendly manner.

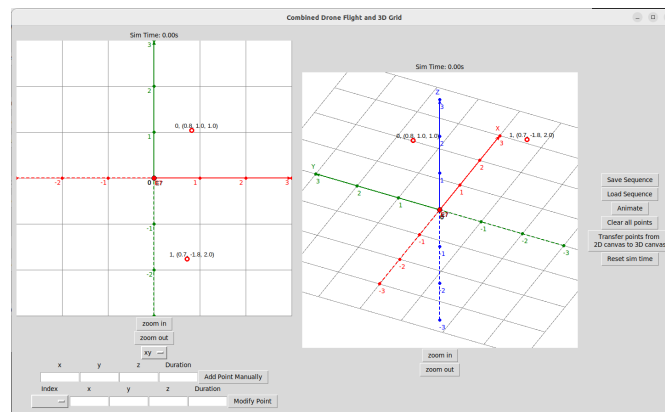


Figure 2.3: An interface created using tkinter.

RELATED WORK

The integration of brain-computer interfaces (BCIs) and unmanned aerial vehicles (UAVs) has evolved rapidly, especially within the domain of noninvasive motor imagery (MI)-based control. While many studies have focused on classification algorithms, decoding techniques, and paradigms, relatively few have emphasized the broader challenge of system integration—particularly concerning swarm control and user experience. This thesis builds upon previous work by prioritizing the user-centric design and system usability of drone swarms while also enabling live MI-based control. The following section surveys selected key studies that closely relate to this work and highlights how they inform and contrast with the approach taken in this thesis.

3.1 MI-Based BCI for Drone and Swarm Control

A significant body of literature explores the use of motor imagery EEG signals to control individual drones. One early and notable study by LaFleur et al. [9] demonstrated 3D quadcopter control using a noninvasive BCI. This study showcased the feasibility of real-time navigation but involved only a single drone and relied on a highly structured control interface, which limited practical usability. Nonetheless, it laid the groundwork for real-time aerial robotic control through EEG-based MI.

Building upon these foundations, Lee et al. [7] proposed a system for EEG-based swarm control using endogenous BCI paradigms. This is one of the few works explicitly targeting drone swarms rather than single drones. The authors introduced discrete control commands for multiple UAVs and emphasized the paradigm’s feasibility. However, their system required significant calibration and offered limited flexibility in command structure or task complexity, making it less suitable for iterative, long-term research workflows. Their work points to the need for more scalable and maintainable systems—an area directly addressed in this thesis by designing with modularity and lab usability in mind.

In a related continuation, Kim, Lee, and Lee [8] developed an EEGgram-based neural network aimed at improving decoding performance for visual imagery signals in swarm control contexts. Although they focus on visual rather than motor imagery, the core idea of robust signal decoding in a swarm scenario is highly

relevant. Their approach underscores the importance of advanced classification pipelines, which complements the ambition of this thesis to eventually support real-time MI classification for swarm command input.

3.2 General Frameworks and BCI System Reviews

Palumbo et al. [4] provide a broad systematic review of MI-based BCI systems applied to mobility contexts, primarily wheelchairs. While not directly drone-related, the underlying system architectures, performance metrics, and usability challenges are highly transferable to aerial swarm systems. The review also identifies the recurring challenge of poor user interface design in BCI systems—supporting the motivation for this thesis’ emphasis on user-friendly interfaces for drone swarm researchers.

The recent preprint by Lu et al. [5] introduces a geometric learning-based framework (LGL-BCI) that enhances MI classification through novel latent space representations. Although it does not focus on drone applications, it signals the ongoing evolution of classification techniques which could be integrated into future versions of the system presented in this thesis.

3.3 Drone-swarm control architectures and path planning

Drone swarms can be controlled via *centralized*, *distributed*, or *decentralized* schemes. This section reviews key approaches and links them to the goals of this thesis—enhancing usability, robustness, and real-time classification in swarm control.

3.3.1 Centralized vs. distributed control

Centralized architectures rely on a single “brain” (e.g., ground station or leader drone) issuing commands to the swarm (leader–follower, virtual structure). They are intuitive to design and guarantee consistent global behavior, but suffer from scalability issues and a single point of failure [46]. In contrast, distributed systems allocate decision-making to individual drones that collaborate via communication to achieve consensus—resulting in scalable, resilient, and adaptable swarms [46].

Hu et al. [47] empirically compare centralized and distributed swarms using 12 drones and a backend cluster, concluding that while centralized control yields higher decision quality under low load, it fails to scale gracefully and lacks reliability at scale. This trade-off is crucial when integrating BCI-based control, as a user-oriented interface (the BCI) must map to an architecture that supports flexible swarm interaction without central bottlenecks.

3.3.2 Formation control and swarm coordination

Recent surveys highlight conventional methods—leader–follower, virtual structure, behavior-based, consensus-based, artificial potential fields (APF)—and describe their alignment with centralized, decentralized, or distributed regimes [46]. Decentralized schemes exhibit resilience and simplicity by relying on local perception

and peer-to-peer communication, ideal for swarm systems with embedded MI-BCI commands [46]. It is exactly this which is implemented in the Crazyflie drone and which will be used in this thesis.

3.3.3 Path planning across swarms

Path planning defines how a swarm achieves coverage, navigation, and obstacle avoidance. Approaches range from graph/sampling-based methods to bio-inspired optimization (PSO, ACO, GA). An MDPI review outlines global UAV route-planning stages: mission objective, environment modeling, optimization of trajectories under constraints (energy, safety) [48]. A distributed PSO-based planner achieves collision-free full-area coverage without central control [49]. These methods offer promising integration points with MI-BCI user intents—for example, translating imagined commands into swarm-level waypoints processed by PSO.

3.3.4 Integration with MI-BCI user interface

The architecture of swarm control must align with BCI input: centralized control simplifies mapping MI commands to high-level swarm directives, but imposes latency and single-point-of-failure risks. Distributed planning with local autonomy more naturally accommodates user-friendly interfaces in the lab, permitting MUUC collaboration with real-time MI classification algorithms. This thesis will thus leverage distributed PSO-based swarm planners and set formation strategies to optimize responsiveness and flexibility in live experiments.

3.4 Summary and chosen direction

These studies illustrate both the progress and limitations in the field of MI-based drone control. While control of individual drones is well-explored, few studies tackle the added complexity of swarm dynamics, real-time classification, and practical usability for research environments. Especially with the usability in focus. This thesis therefore contributes a complementary perspective by laying the foundation for MI-based control of a drone swarm by designing a system architecture focused on modularity, transparency, and user friendliness. The goal is that this ensures long-term utility in a lab setting.

PLATFORM ARCHITECTURE

The developed platform is designed to enable users to define and execute drone swarm paths using a brain–computer interface (BCI) based on electroencephalography (EEG) signals. While the core of the project focuses on the graphical path planning interface, swarm execution infrastructure, and Loco Positioning System (LPS) integration, the system as a whole also includes an EEG signal acquisition and classification pipeline. This component, which translates motor imagery into control commands, was developed in earlier work and forms the upstream input to the control system described in this thesis.

4.1 System overview

At a high level, the system consists of four key components: the EEG interface, the path planning graphical user interface (GUI), the Crazyflie drone communication stack and the Loco Positioning infrastructure. Each of these elements communicates with the others through clearly defined interfaces, forming a modular and extensible system suitable for testing both single and multi-drone missions.

4.1.1 OpenBCI EEG Headset

The brain–computer interface used in this thesis is the OpenBCI Cyton board with the Ultracortex Mark IV headset. It provides 8–16 channels of EEG signal acquisition and streams data wirelessly to a host computer. This setup enables non-invasive monitoring of motor imagery-related brain activity and is compatible with a range of open-source signal processing libraries, making it an excellent choice for real-time BCI control experiments [50].

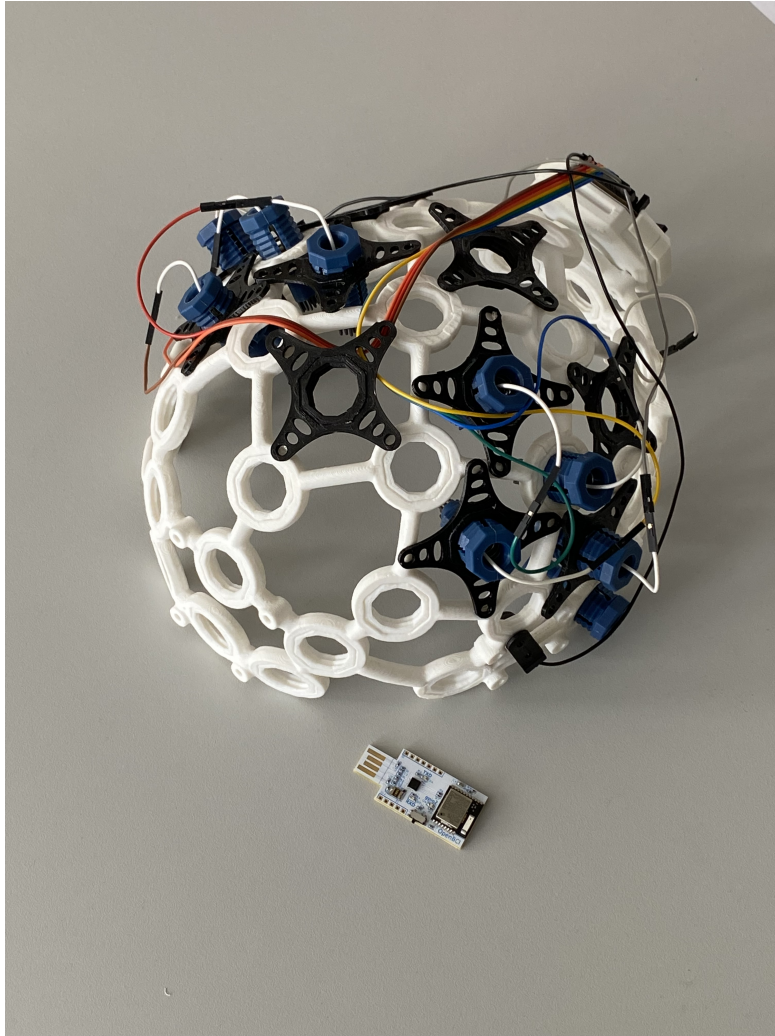


Figure 4.1: The picture shows the OpenBCI Ultracortex IV headset frame with the cyton board and eight electrodes. In front lies the USB dongle used to transmit data.

4.1.2 Path planning GUI

The Path Planner System is an interactive tool designed for planning, visualizing, and managing multi-drone flight paths within a 3D space. Built around a modular GUI framework, it allows users to define drone trajectories using a canvas-based interface, assign commands, and simulate movements in real time.

The system supports saving and loading flight sequences, synchronizing paths across multiple views, and ensures intuitive control over drone behaviors through features like drag-based rotation and animated playback. By combining visual feedback with file-based path management, it streamlines both experimentation and deployment for drone swarm coordination tasks.

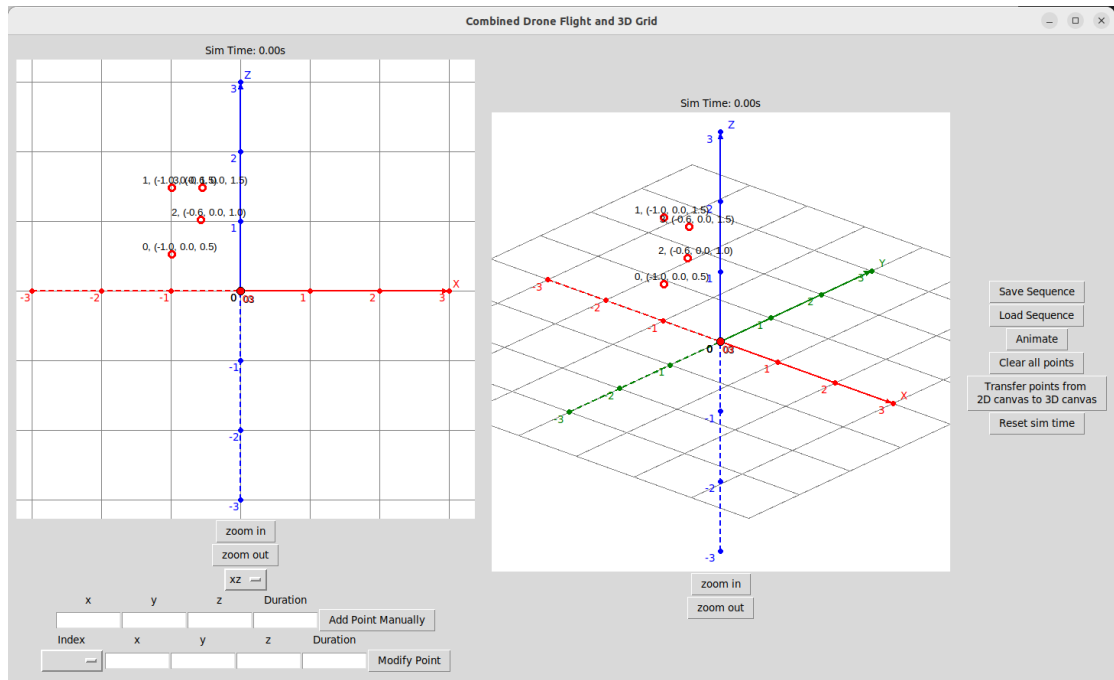


Figure 4.2: One of two programs used for path planning. This is the path definition program, where on the canvas to the left one can define and edit points in a path and visualize in 3D on the right

4.1.3 Crazyflie 2.1 Quadcopter

The Crazyflie 2.1 is a lightweight, modular micro-quadcopter developed by Bitcraze. It features onboard sensors such as an IMU, barometer, and a low-latency radio communication module for real-time control. The Crazyflie is programmable via the Crazyflie Python API and supports both autonomous and manual flight. Its small form factor and open-source software ecosystem make it well-suited for research in swarm robotics and indoor navigation [30].



Figure 4.3: The Drone used in this project is the Crazyflie 2.1, as shown here, but the earlier version 2.0 works just as well

4.1.4 Loco Positioning System

The Loco Positioning System (LPS) is an indoor localization solution also developed by Bitcraze. It uses Ultra-Wideband (UWB) communication and Time Difference of Arrival (TDoA) to estimate the drone's position in 3D space with sub-decimeter accuracy. The system consists of anchors fixed in the environment and a tag mounted on the Crazyflie. This setup allows for precise and scalable multi-agent tracking in environments without GPS [51].

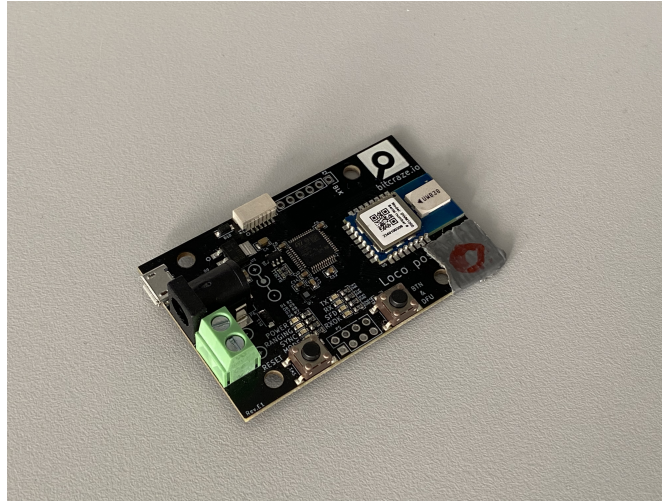


Figure 4.4: The Loco positioning system node, used together with the Crazyflie platform to give a position estimate to enable swarming

To mount the Loco Positioning System anchors securely and consistently throughout the test environment, custom node stands were constructed. These consist of wooden bases for structural support and 3D-printed holders specifically designed to fit the UWB anchors. The stands ensure precise and repeatable placement of each node, which is critical for accurate position estimation using TDoA. Their modular construction also allows for easy relocation and reconfiguration of the positioning system during experimental setup.



Figure 4.5: The self-made node stands made for this thesis, to hold the nodes for positioning system. For production drawings see Appendix C

4.2 Demonstration framework overview

Figure 4.6 illustrates the complete system and the interaction between its components. Even though the EEG classifier itself is not the focus of the implementation chapters, it is included here as an essential component of the system’s operational

logic and is necessary for a holistic understanding.

All the container boxes, like "hardware and user," represent a subsystem of the framework. Green resembles physical hardware, red is a thread which is started and joined (terminated) by the main program but has its own loop. It is affected by the blocks of the same color in the diagram. The orange container resembles a setup that needs to be done before the demonstration, like acquiring data. The blue container is the main program. Full lines resemble program information and live data flow, while stipulated lines resemble information or data that is used for setup.

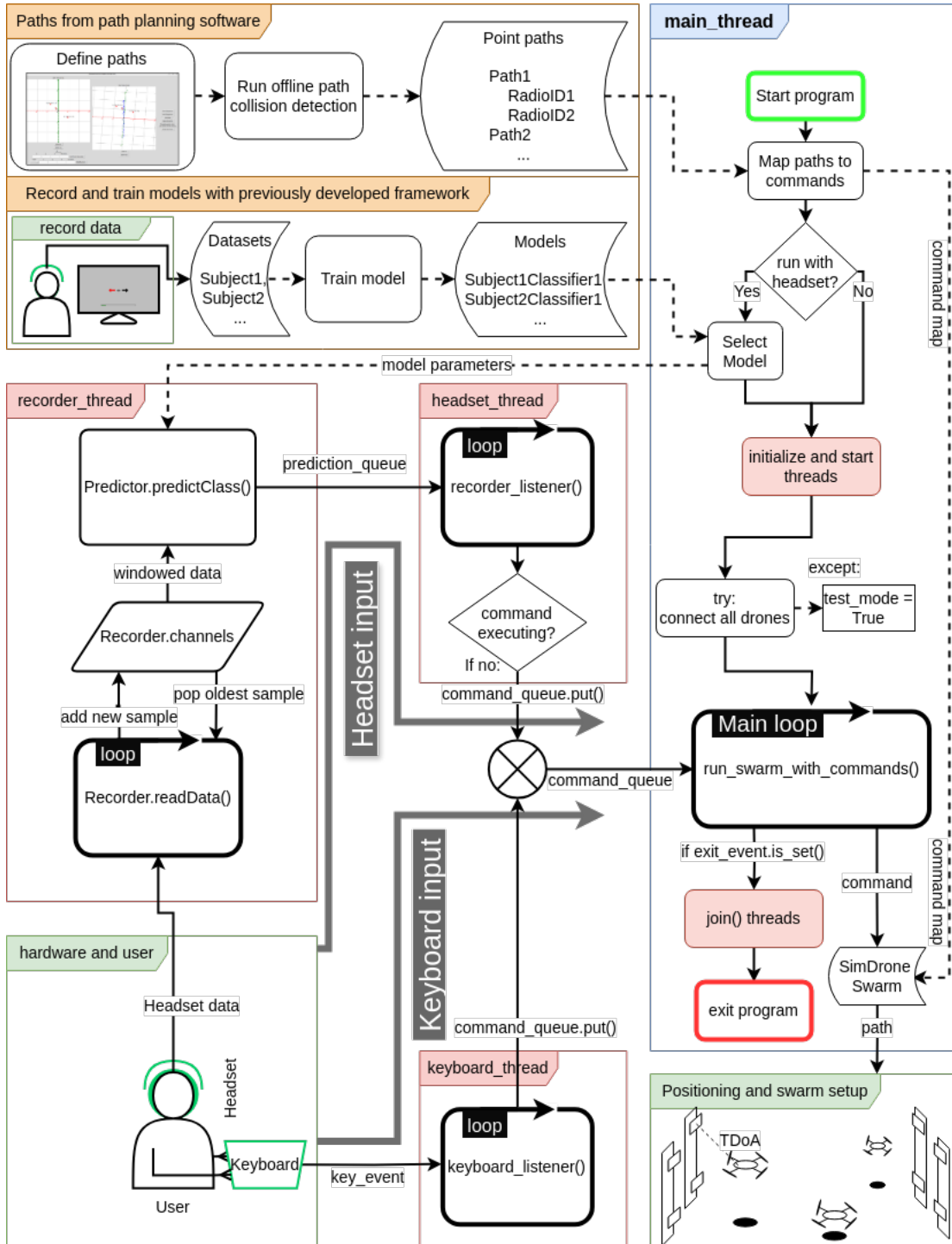


Figure 4.6: High-level demonstration system overview showing information flow, command inputs and the necessary parts to use the swarm demonstration program

As seen in Figure 4.6, the system overview of all the parts of the demonstration framework is connected. The order of execution is represented in a clearer way in Figure 4.7. Note that each block represents a program, and a block placed within another means that this program is run from within that first program.

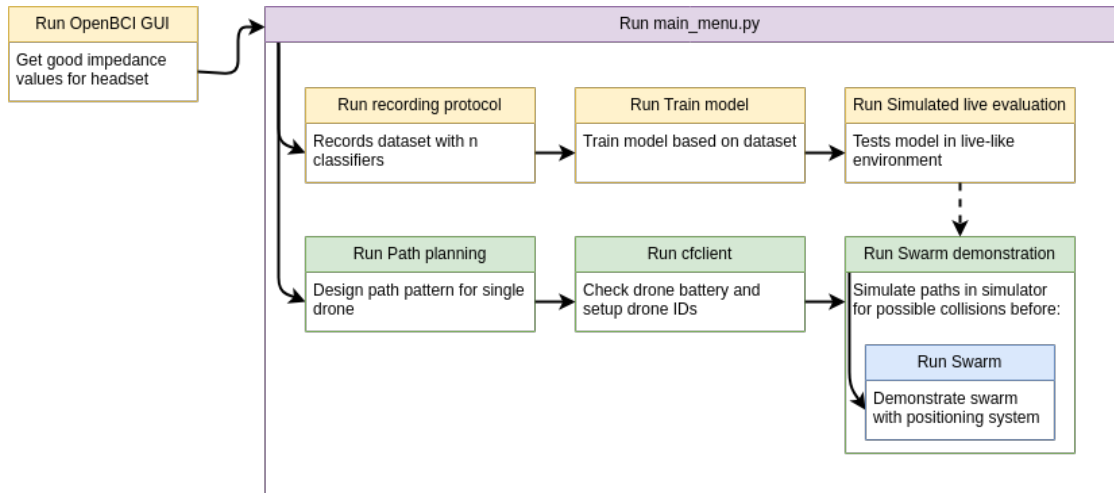


Figure 4.7: A flowchart of what programs need to be run when, and where they are run from. The program `main_menu.py` contains almost all the programs required, except the OpenBCI GUI, which must be run to calibrate the headset before recording BCI data or using it in any way. This is covered in more detail in [1].

4.3 Earlier project versions' process overviews

Since the thesis builds on previous work, are the earlier Process overviews included in Figure 4.8 and Figure 4.9. These show the already existing functionality and how the thesis implementation works as a glue to combine them into one framework.

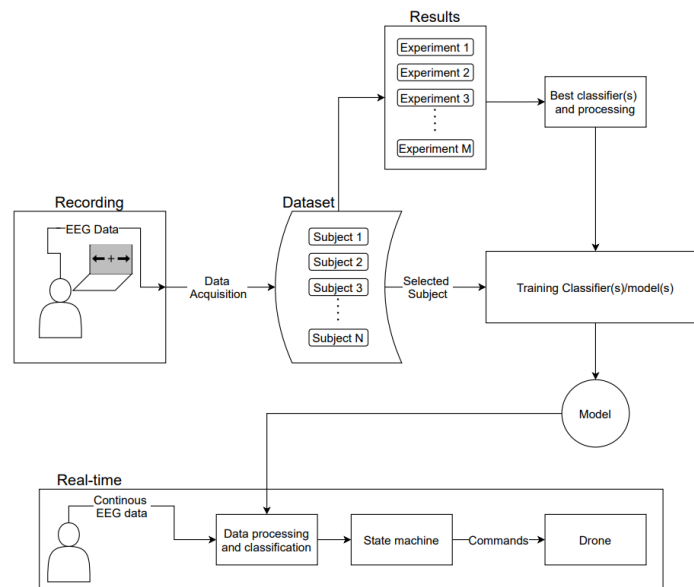


Figure 4.1: Overview of the process from data acquisition, subject and model selection, as well as the real-time system.

Figure 4.8: Process Overview from previous iteration of project with focus on recording and classification presented in [2]

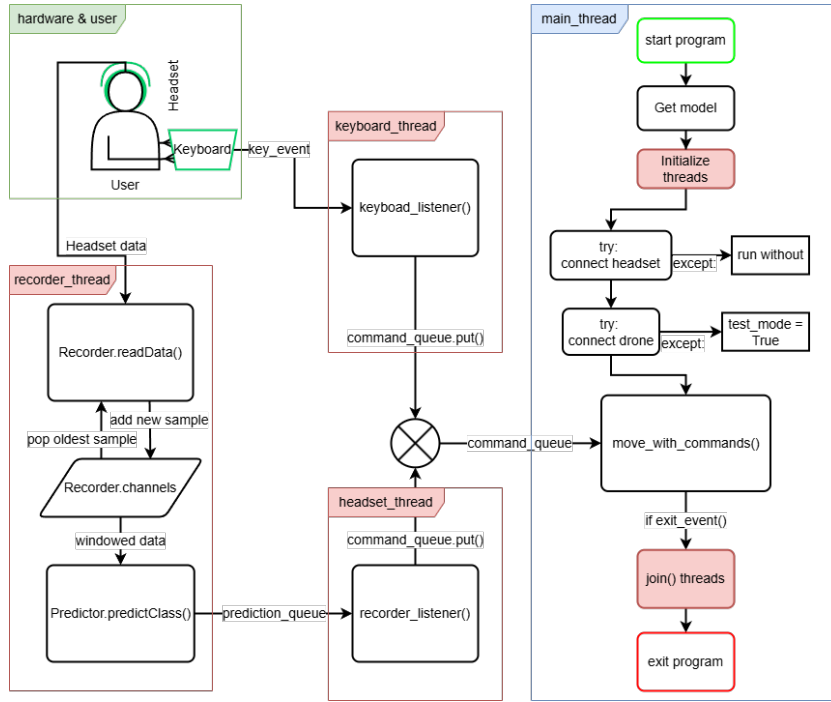


Figure 4.9: Live demonstration process overview presented in [1]

In Figure 4.9, the reason for using a separate recorder thread from the headset thread is that the input from the headset is translated into system-wide commands, which can be easily mapped to a set of combinable commands that are compatible with the keyboard. It also enables the possibility to implement a state machine for live inputs to a slow commanded system through system locking. The keyboard does not provide a constant flow of inputs in the same way as the headset, but locking the `command_queue` to reduce backlog could be implemented.

The subsequent chapters of this thesis will delve into the parts of the system that were designed and implemented during this thesis: the GUI for path planning, the setup and analysis of the Loco Positioning System, the integration with the drone swarm control software, and the evaluation of system performance under different test scenarios.

COMMAND INPUT SYSTEMS

This chapter provides an overview of the command system within the framework. Mostly, it will discuss the headset, as this is the most interesting component, and also the keyboard and how they interact on a surface level. Note that the work regarding EEG data processing and classification pipelines are implemented from the work of Kjersti Brynstad and Erlend Vatsvåg in their master thesis *“An Asynchronous Motor Imagery based Brain-Computer Interface for Two-dimensional Drone Control”* [2].

5.1 The dual input-source system

When beginning the implementation of earlier iterations of the project, it was clear that the software needed to be modularized. This issue was tackled by the preceding specialization project [1]. The framework was split into standalone parts, which could be tested without the rest of the system. This meant one needed to be able to classify headset data without the drone’s presence, the solution of which was to have a "test mode" where all commands were executed as "simulated commands". This is printed in the terminal. To then be able to test the drones dynamically without including the BCI, another source of input was included: The keyboard. This enables testing of drone flying stability and also later, path following in swarms, without having to include the BCI to be able to do so. A significant strength that previous systems have lacked.

This decision was also rooted in other benefits: It allows for finer control of the drone, which is separate from the classification during flight, and the classification, which can be uncontrollable if accuracy is low. This allows the BCI user to focus on the task at hand while a second operator ensures the drone remains within the designated flight volume. Additionally can it bring additional control which is needed but not possible with limited classification classes, in this case the "stop" command which lands the drone is only available though the keyboard, since the headset mapping is full. This implementation technique can be seen as both a strength and a limitation: By not having access to all commands through the headset, one could argue that complete control of the drone through BCI is not achieved. One could also argue that all flying-related inputs are available from the BCI user alone and that this is indeed a strength because setup commands

and visual appeal can be fine-tuned by the second operator behind the keyboard. In addition, the keyboard is crucial for executing the program in the first place and will always be present during the demonstration, thereby never becoming a limitation.

5.2 Command mapping

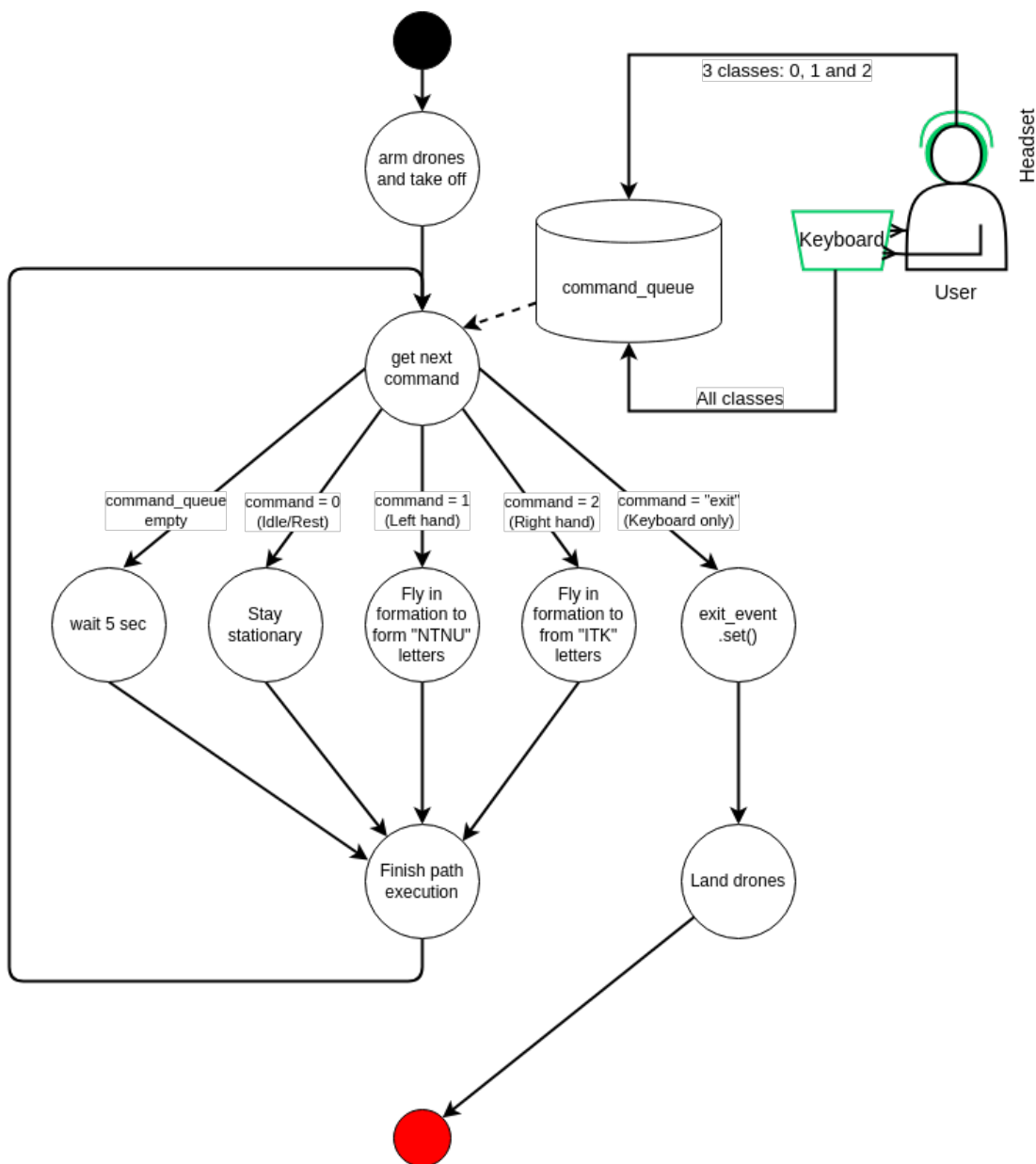


Figure 5.1: An example class mapping for the final "NTNU" drone show. The keyboard and headset would in many cases also be operated by two different users.

5.3 EEG BCI headset setup

5.3.1 Evaluation of Alternative BCI Systems

Since the start of the BCI-controlled drone project at NTNU in 2016, the landscape of EEG hardware has expanded significantly. The current implementation utilizes the OpenBCI Cyton board with the Ultracortex IV headset and OpenBCI electrodes. Given that the software developed in this thesis will be tightly coupled to the hardware platform, it is essential to reassess possible alternatives. Table 5.1 presents a technical comparison of selected commercially available EEG systems.

Table 5.1: Comparison of BCI Headset Options

System	Sampling Rate	Channels	Electrodes	Open-source	Strength
OpenBCI Cyton (+ Daisy)	250 Hz	8 (+3 aux) 16 w/ Daisy	CGZ wet/dry	Yes ([52], [53])	Easy to work with
Emotiv EPOC X	128/256 Hz (2048 Hz internal)	14	Saline-soaked felt	No ([54])	Lightweight, 9 h battery
g.Nautilus (g.tec)	Up to 500 Hz	Up to 64	Gel or hybrid active	Limited SDK	Medical-grade cap, more setup
Mentalab Explore+	Up to 1000 Hz (2 kHz offline), 24 bit	Up to 32	Wet or dry Ag/AgCl	Yes ([55], [56])	27 g, IMU included

Sampling Rate & Channel Count The Cyton board samples at 250 Hz (USB), or 125 Hz streaming when paired with Daisy; however, recording to SD card retains 250 Hz even at 16 channels [53]. Emotiv EPOC X streams at 128/256 Hz after internal downsampling from 2048 Hz [54]. g.Nautilus supports up to 500 Hz (250 Hz at 64 channels), while Mentalab Explore+ streams up to 1 kHz wirelessly and logs offline at 2 kHz with 24-bit resolution [55].

Electrode Type & Comfort OpenBCI offers flexible placement with wet or dry CGZ electrodes; fit and comfort vary depending on the headset. Emotiv uses saline-soaked felt pads—easy to place but potentially less stable. g.Nautilus employs gel or hybrid active electrodes embedded in a medical-grade cap. Mentalab uses lightweight Ag/AgCl electrodes (wet or dry) within customizable caps that include IMU sensors [55].

Open-source Ecosystem OpenBCI is fully open-source across hardware, firmware, and software, aligning well with academic transparency. Mentalab also provides fully open APIs (Python, MATLAB, C++) and supports LSL integration ([55], [56]). Emotiv and g.tec offer proprietary SDKs only, with limited freedom for customization or third-party integration.

Comfort & Usability Emotiv offers a comfortable fit with a 9-hour battery life. OpenBCI comfort depends on the chosen headset/cap and electrode type. Mentalab’s 27 g amplifier and soft neoprene caps combine mobility and comfort. g.Nautilus provides a robust research-grade experience at the cost of more involved setup.

Price Considerations Though pricing is secondary, it still informs feasibility. The Emotiv EPOC X, along with the OpenBCI Cyton ($\pm$ Daisy) and Mentalab Explore +, fall in a similar price range. g.Nautilus is enterprise-grade and significantly more expensive. In addition, the OpenBCI and Mentalab Explore + are already in storage.

5.3.2 Recommendation for BCI solution going forward

For open-source flexibility, moderate sampling rates, and flexible electrode placement, the OpenBCI Cyton remains a strong option. However, if higher channel counts and sampling rates are needed and budget allows, g.Nautilus becomes attractive but at the cost of openness and setup complexity. Emotiv EPOC X offers plug-and-play convenience but is limited by proprietary design and lower signal-grade electrodes, which falls outside the need in this project. Lastly, the Mentalab Explore+ strikes a balance with high resolution, open APIs, light form factor, and IMU integration and is, therefore, a very strong choice. In addition, there are existing experience with it within the lab, and the headset already is owned. It is therefore recommended to switch to the Mentalab Explore +, should the project need better quality data.

5.3.3 Electrode configuration

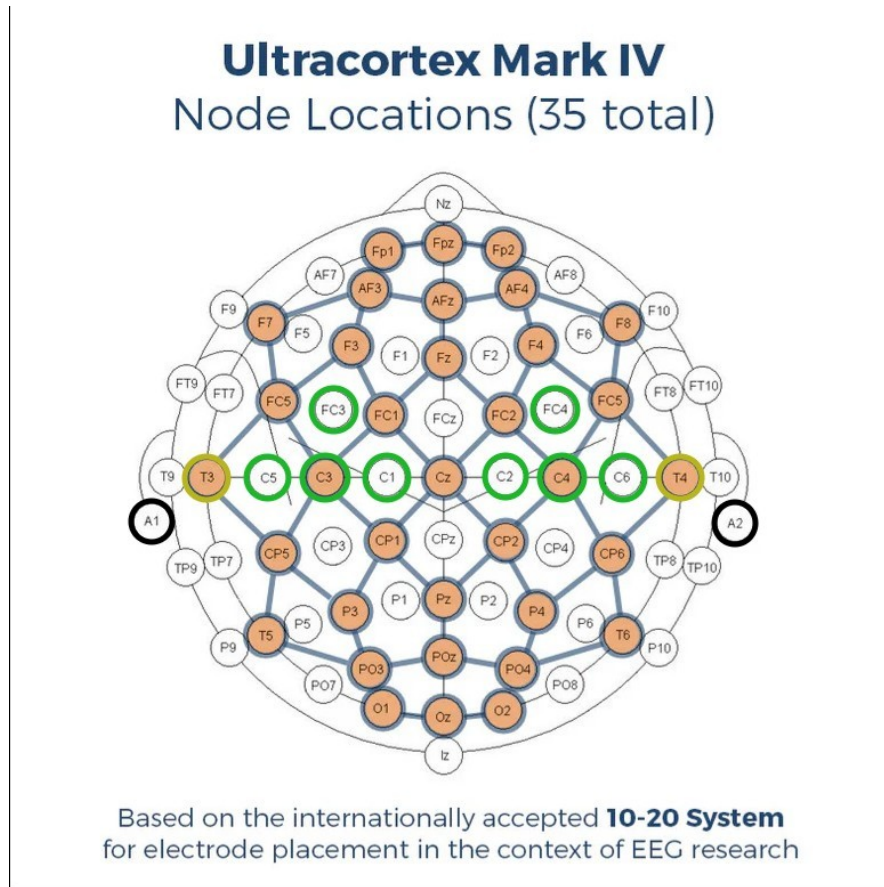


Figure 5.2: The electrodes (highlighted green) which were used in testing. A1 and A2 are used as ref and noise cancel electrodes on the earlobes (highlighted black), and T3 and T4 (highlighted yellow) are occupied with support parts to hold the headset correctly on the head, not as electrodes. Image from [50]

5.4 Classification Pipeline

The classification pipeline used in this BCI project is implemented as described in the thesis “*An Asynchronous Motor Imagery based Brain-Computer Interface for Two-dimensional Drone Control*” by Kjersti Brynstad and Erlend Vatsvåg [2]. Many of the key results are reiterated here for clarity on the inner workings of the framework. To be clear, the pipeline and the evaluations in this section are not the work of this thesis.

The pipeline used can be divided into two main stages: preprocessing and classification.

5.4.1 Preprocessing

The raw EEG data first undergoes basic filtering operations. Specifically, a notch filter at 50 Hz and a high-pass filter at 0.1 Hz are applied to remove power line noise and slow drifts, respectively. Optional re-referencing is then applied using

either the Common Average Reference (CAR), Laplacian re-referencing, or no re-referencing at all.

After re-referencing, the data is processed through either a bandpass filter or a Discrete Wavelet Transform (DWT) to extract the frequencies, which often encompass the useful information about MI tasks. For bandpass filtering, several frequency bands were tested: 8-12 Hz (μ), 12-30 Hz (β), a combined 8-30 Hz ($\mu + \beta$) band, and a selective 8-12 Hz + 15-19 Hz combination. For DWT, biorthogonal 2.8 wavelets with four decomposition levels were used, and either all levels or just the D3 and D4 coefficients (corresponding roughly to the μ and β bands) were selected for feature extraction [2].

5.4.2 Classification and Feature Extraction

Several combinations of feature extraction and classification techniques were tested and optimized. The techniques include:

- CSP (Common Spatial Patterns) with either SVM or LDA
- Covariance matrices projected via FGDA and classified by MDRM
- Riemannian covariance features with Tangent Space projection and classification using either Logistic Regression (LR) or SVM
- Covariance features followed by CSP (R-CSP) and LDA
- DWT energy features classified using SVM

5.4.3 Flat vs. Hierarchical Structure

Two structural models were evaluated: a flat model, where all classes (rest, MI tasks) are directly classified; and a hierarchical model, where the first level distinguishes between rest and MI tasks, and a second level classifies the specific MI task, should it correspond to a task.

Although theoretically promising, the hierarchical model did not outperform the flat structure in practice. Therefore, the flat structural model will be used exclusively in this thesis.

5.4.4 Best Performing Pipelines

When it comes to the performance of the different pipelines, no single pipeline works best for all subjects. It is highly individual which pipeline achieves the best accuracy, and so those implemented in the previous iteration will also be included in this project.

Some of the best performing pipelines with two-classes (plus rest) classification from the dataset were:

- Laplacian re-referencing, an 8–30 Hz bandpass filter, and a classification pipeline of covariance estimation, tangent space projection, and logistic regression

- No re-referencing, an 8–30 Hz bandpass filter, and CSP with seven components, followed by SVM classification.

It is worth mentioning that some of the pipelines, although represented in the GUI, are not yet implemented correctly. The second one mentioned above, the CSPSVM was used for the very limited testing done in this thesis, as this achieved one of the highest scores for some of the subjects, and it achieved the highest score out of those available in the preceding project, albeit with very limited data and only using ME [1].

5.4.5 Using Data in real-time

As it stands, the synchronization of the data stream is not sufficient for live control with a meaningful online accuracy. It is therefore not feasible to obtain live classification outside of the recording protocol, where tasks are queued and data is labeled. The framework can therefore not be tested with live data feed on a subject in this thesis.

5.5 Potential improvements to the BCI and input system

Working on live classification and improvement to be used with datastream in real time is strongly suggested. It is one of the main remaining things that needs to be done to achieve the goal of using EEG BCI to control a drone swarm.

DRONE POSITIONING SYSTEM SETUP

Earlier iterations of the EEG-controlled drone project have flown drones with the assumption that the drones will stay in the same place, and move relative to their position when they get a command from the command system [2, 1]. This is, of course, not always the case; the drone drifts slightly if no global truth is presented, and data inputs from sensors like IMUs are prone to drift over time. Implementing the drone control like this eliminates the necessity of external infrastructure for getting a position estimate. However, this was a simplification only possible since only one drone was flown at a time. When expanding the framework to more than one drone, it is crucial to know where all the drones are in space to reduce the probability of crashes in the air. To achieve this, a positioning system will be deployed.

The positioning system provides the drones with an absolute position estimate in a predefined coordinate system. It is key to the drones' safe flight, and crucial to be able to fly a whole swarm of drones in a defined and restricted volume. Bitcraze, the same company that delivers the Crazyflie drones, which have been used for this project over the last few years, has developed support for multiple absolute positioning solutions to obtain measurements for their drones.

The Crazyflie itself handles the positioning estimation on board and communicates directly with the positioning system. This reduces the amount of overhead needed to set up the positioning.

6.1 Choosing a positioning system

Bitcraze has three main supported schemes for providing their drones with an absolute positioning estimate. They have their own Loco positioning system (LPS), which is based on Ultra-wideband (UWB) radio frequencies. The Lighthouse positioning system is based on VR base stations from either HTC Vive (version 1) or Steam VR (version 2). The third is the Motion capture system, which uses an existing motion capture system and sends position information to the Crazyflies over the Crazyradio. All three are valid options for generating a positioning estimate.

To help guide the choice, are some design requirements defined by the use-case:

- The swarm flight need good positional accuracy to fly drones close to one another
- The system needs to be mobile enough to be taken outside the lab with as little work as possible for demonstration purposes
- It needs to be able to control a big (and scaleable) swarm
- It should be possible to use the framework outside (although not required)

A comparative analysis of available localization systems was conducted using a decision matrix, evaluating each option based on criteria such as accuracy, scalability, and setup complexity.

Table 6.1: Comparison of Loco, Lighthouse and Motion Capture Positioning Systems for Crazyflie, [51]

System	Loco (UWB)	Lighthouse (Optical)	Motion Capture (MoCap)
Max coverage area	Up to 50×50 m (not yet fully tested)	Approx. $5 \times 5 \times 5$ m	Unlimited (dependent on number of cameras)
Accuracy and precision	Decimeter level to Centimeter-level	Sub-centimeter to centimeter; relative precision millimeter	Sub-millimeter (<1 mm)
Line-of-sight requirement	Preferred but can work through obstacles	Required; no transparent/reflective surfaces	Required; IR cameras must see reflective markers
Swarm scalability	Good; requires 4+ anchors, can scale infinitely	Up to 4 base-stations supported in firmware (V2 supports up to 16)	Scales with number of cameras; swarm support via Crazyswarm2
Pose estimation	On-board via Kalman filter using UWB ranges	On-board via optical sweep/angle fusion	Off-board; external computer calculates pose and streams to CF
Ease of setup	Moderate (accurate placing of anchors)	Easy (auto-calculation of geometry via client)	Difficult (camera calibration, synchronization, markers)
Setup mobility	Anywhere, indoors and outdoors	Anywhere indoors	Indoors
Cost	Moderate (\$\$)	Moderate (\$\$)	High (professional motion-capture systems \$\$\$)

The motion capture positioning system offers the highest accuracy but was not considered viable for this project due to its complex setup and high cost. The decision, therefore, came down to two alternatives: the Loco positioning system and the Lighthouse positioning system.

There are valid arguments for both options. The Lighthouse system provides more accurate position estimates and has a slightly simpler setup, making it an attractive choice. On the other hand, the Loco system offers broader coverage, better scalability, and the potential for outdoor use, qualities that make it a more flexible and adaptable solution.

Ultimately, the Loco positioning system was selected. Although this decision was later met with some regret due to its lower positional accuracy, it was made with long-term goals in mind. The Loco system was viewed as a better investment for the lab, as it supports a wider range of future applications and demonstrations, including those yet to be defined. The Loco system’s performance can also potentially be improved through tuning of the outlier filter, controller, and Kalman filter estimator. It then becomes a state estimation problem, rather than a hardware problem, at least to a certain point.

6.2 Physical Node Positioning

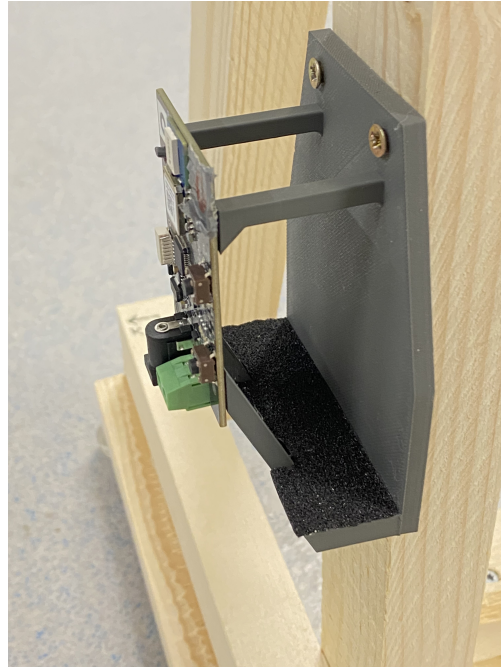
The Loco positioning system is built around a set of anchor nodes. This set must contain at least four nodes to make the position estimate unambiguous. In the best case, which Bitcraze also recommends [51], the set contains up to eight nodes. In this thesis’s application, all eight are used, distributed across four vertical stands.

6.2.1 Stand Design and Node Mounting

The stands themselves are made out of wood. This makes them easily produced, light, and cheap Figure 6.1a. The mounting components for the nodes, as seen in Figure 6.1b, are 3D-printed plastic. Nodes are hung onto these mounts and secured such that the orientation and position remain constant. Should different vertical spacing be required, can the mounts be repositioned or reprinted with other dimensions and screwed into the wood with screws. A power bank is placed behind each node and secured in the same mount to simplify wiring and maintain compactness. Both of these materials are chosen for two reasons: they are easy to work with, and they minimize multi-path interference, as reflective materials like metal are known to degrade UWB signal integrity.



(a) The node stand which erects node holders to correct height



(b) The node holder with a node and space with friction tape to hold a power bank



(c) The node connected to the power bank of choice

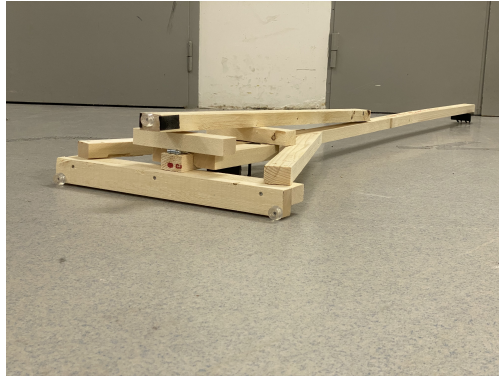
Figure 6.1: Node stand hardware

Each stand features a three-point leg configuration with rubber feet, maximizing stability and friction with the floor. A foldable back brace acts as a counterweight to the front-mounted nodes and power banks. It has this physical form because the flying volume should be unobstructed by the stands while also being as stable as possible as seen in Figure 6.2. For this same reason are the rubber feet placed at the very front of the lower support member.

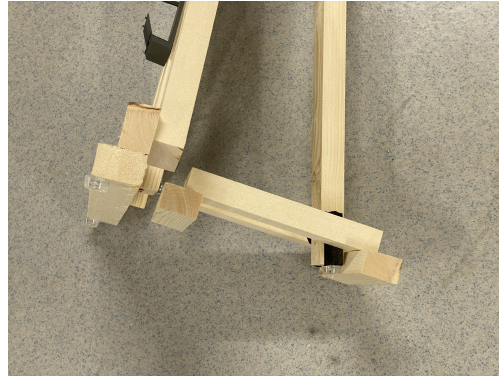


Figure 6.2: The stand seen from the side

The brace is held in place with high-friction tape, ensuring no slippage during operation. The 3D-printed mounts for the nodes and power banks are screwed into the wooden structure and use additional friction tape to prevent shifting during relocation or operation.



(a) The node stand folds together for easy transportation



(b) The nodestand has a back-brace for stability

Figure 6.3: The stand has two hinges that allow it to fit both transportation and use. The stands are designed for portability.

As shown in Figure 6.4, each stand holds two nodes: one positioned near the base at a height of 0.3 m and another mounted higher up at 2.1 m. These heights were chosen to maximize vertical separation and improve spatial resolution in the z-dimension, while being within reach of most people and fitting into a standard room height-wise. The lower node is slightly higher than at the ground to maximize the probability of line-of-sight, should the floor be covered in protection mats, tall grass, or something similar.

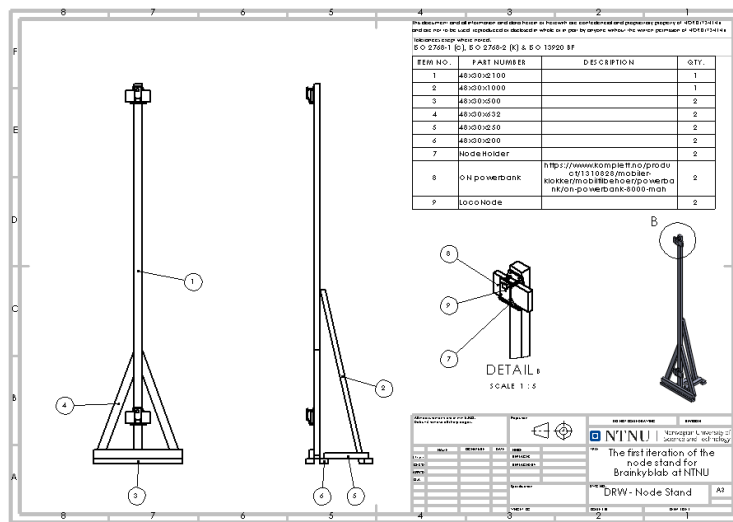


Figure 6.4: The main technical drawing for the stand assembly, all drawing pages in full resolution are included as an appendix in Appendix C

The stand was made in SolidWorks, and the CAD files are available as part of the code base at the location:


```
./assets/cad_stand/
```

SolidWorks is a Windows-only program and requires a license to use. Such a license is available for all students at NTNU through this page at the time of writing: <https://i.ntnu.no/wiki/-/wiki/English/Solidworks>

6.2.2 Layout and Dimensions

The four stands are arranged in a rectangular configuration aligned with the global coordinate system used by the drone's state estimator. In the current deployment environment (the NTNU drone lab), the stands are spaced 5.3 m apart in the x-direction and 4.1 m in the y-direction. This configuration results in a rectangular flying volume with diagonals carefully measured to ensure right angles using the Pythagorean theorem. The vertical spacing between nodes is 1.8 m. The origin is defined as on the floor in the center of the flying area.

In a new location, the rectangle is reconstructed manually using tape measures, floor markings, and diagonal validation. The flying area's dimensions and node positions are then defined in a `node_config.yaml` file. This file is uploaded to each node using the `cfclient` interface, which communicates via a Crazyflie drone. This is information which is stored in the nodes themselves, and is therefore not needed at runtime. There are however some programs which use this saved file as truth for the positioned nodes. If the layout is updated, make sure to update the file at

```
./data/node_config.yaml
```

If the physical layout is unchanged, the same configuration file can be reused without modification. The structure of the node config file is given as in Listing 1.

```
./data/node_config.yaml:  
0:  
  x: -2.65  
  y: -2.05  
  z: 0.30  
1:  
  x: -2.65  
  y: -2.05  
  z: 2.10  
...
```

Listing 1: Node config file structure

The drone loads this configuration and uses it for TDoA-based multilateration. It is essential that the file matches the physical layout exactly for optimal performance.

6.2.3 Labeling and Setup Procedure

All nodes are labeled with numbered pieces of tape corresponding to their identifiers in the configuration file (e.g., node "0"). Along with the stands having a corresponding number drawn on the stem, this ensures correct placement and allows fast verification during setup. Stands also feature coordinate axes drawn at their base to assist alignment with the global coordinate system and each other.

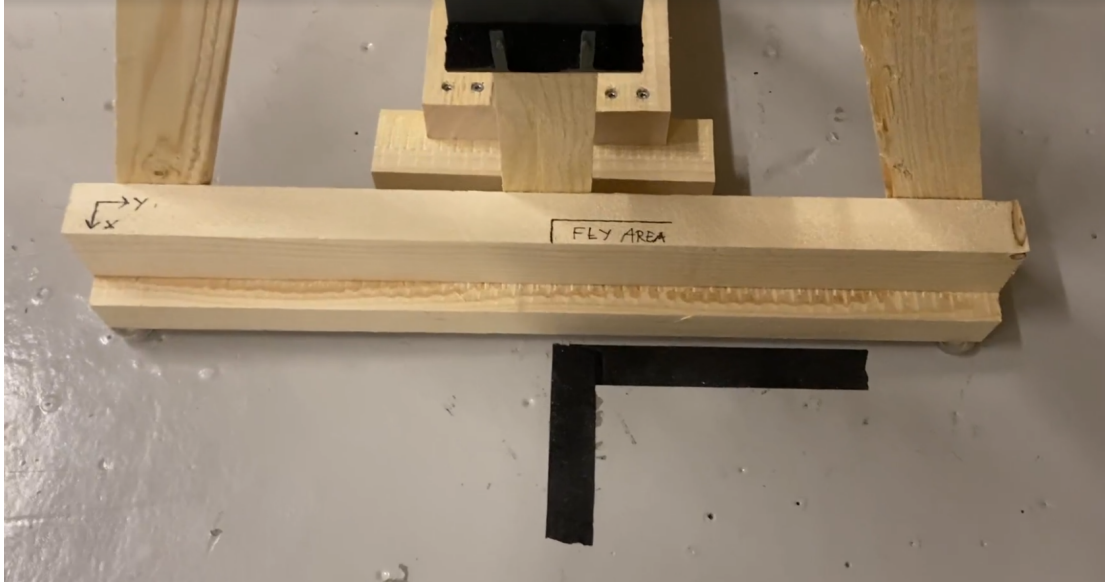


Figure 6.5: The base markings on the stands assure correct stand configuration during setup.

When re-deploying the system, setup typically takes approximately 15 minutes with two people, or up to 35 minutes when done alone. One person can complete the process, although measuring diagonals to validate right angles is significantly easier with two people: one person in each end of the tape measure. Once placed, the nodes' positions in software are set using the `cfclient` as described earlier.

6.3 Setting up and using the positioning system

The following procedure is meant to help guide the users, and will therefore also be included in the User Guide.

1. The first thing to do when arriving at a new location is to mark and measure the flying area, as well as mark the positions of the stands. It is suggested to use colored tape for indoor use and spray paint if done on grass to mark the corners. Be careful when flying in windy conditions, and make sure the stands are level if flying outside.
2. Make sure the marks, and by extension the stands, are placed correctly. It is advised to check the diagonals of the floor marks, ensuring the rectangle forms right angles. The ground needs to be as level as possible.
3. Each stand has two markings on it's base as shown in Figure 6.5: The coordinate system's x and y axis, and a corner of a square. Aligning the

corner of the flying area (here marked with black tape on the floor) with the square on the stand, and making sure to pick the correct stands so the axis marks align across all stands, will help placing the correct stands in the correct places.

4. Configure the nodes with correct coordinates in the defined space through the *cfclient*.
5. Putting the corresponding node into the correct holder bracket with power bank
6. Laying down a cushioning material and/or securing the flying area to protect the operator and potential audience from harm. This depends on the environment and application.
7. Placing drones down **facing positive x**, in the standard configuration as defined above in Section 7.6

The physical swarm system is now ready!

6.4 Position estimate accuracy and precision

Despite a stable and repeatable physical setup, the position estimate of the Crazyflie drone has shown spatial inconsistencies. Most notably, a tendency to get more unstable near the ground and dip lower on one side of the flying area. This behavior may originate from environmental reflections, inaccuracies in anchor positioning, or potential software configuration mismatches. All of which are very hard to rule out. Some of the issues might stem from the inherent system performance of the selected hardware.

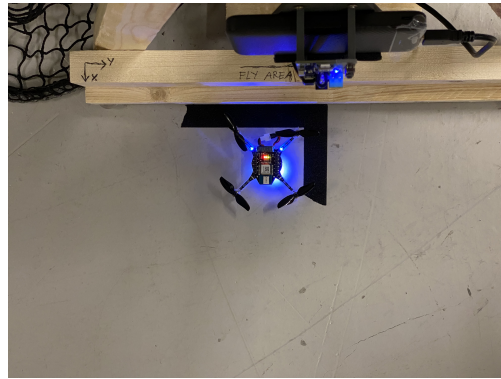
However, the effect it has on the position estimate is measurable. To understand where the error comes from, two tests were initiated: A mapping of the measurement precision and the absolute position accuracy. With these two tests, how close it is safe to fly the drones with regards to collision danger can be determined. Also, possibly achieving a better understanding of where the issue comes from.

6.4.1 The measurement setup

To evaluate these measurements of the Loco Positioning System, the Crazyflie drone was placed stationary on a tripod at various points throughout the flight volume as seen in Figure 6.6. Measurements were collected near the corners and at the center of the room to ensure spatial coverage. At each location, position data was logged for a duration of 10 seconds for a total of 100 measurements per position, while the drone remained still.



(a) Position logging higher up using a camerastand



(b) Position logging at the floor level

Figure 6.6: Position estimate logging for calculation of measurement accuracy and precision

From this data, two types of measurements were extracted: The mean of the data's distance to the true position of the drone, giving accuracy, and the spread of the data in 3D space, giving precision.

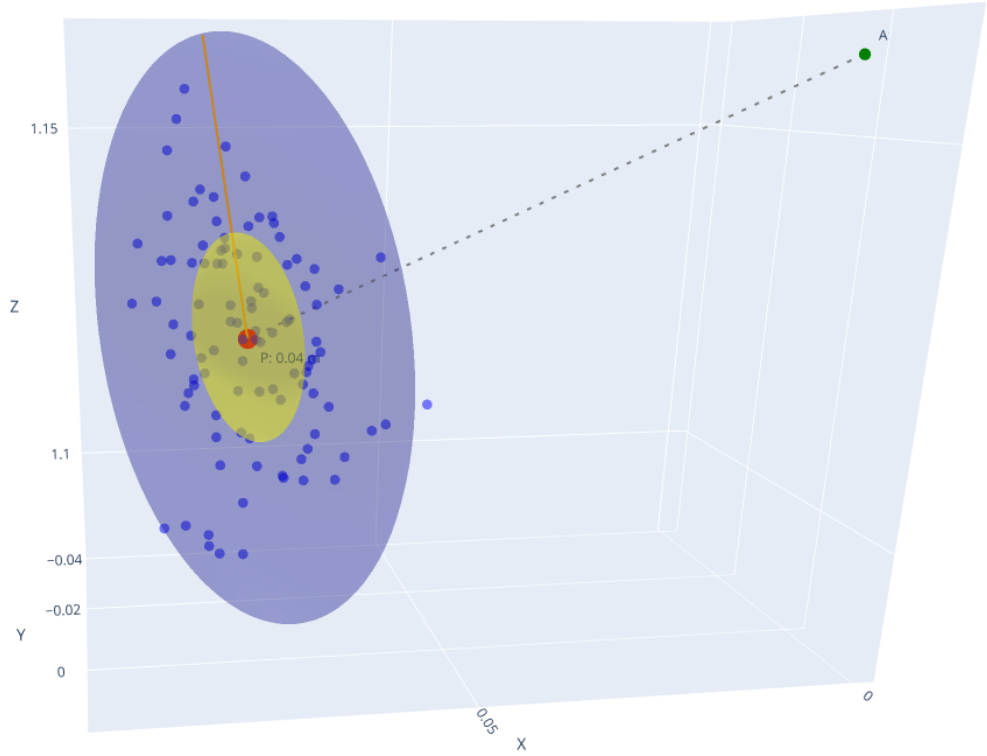


Figure 6.7: Position estimate precision (visualized by orange line) and accuracy (visualized by gray stipulated line)

These uncertainty values were then interpolated across the entire flight volume, producing a spatial map of the relevant parameters.

6.4.2 Inverse Distance Weighted Interpolation

To estimate these values throughout the 3D space, inverse distance weighted (IDW) interpolation was used. This method is used so that nearby points have more influence on the interpolated value than distant ones. For each point in the grid where interpolation is desired, the contribution from each known reference point is weighted inversely proportional to the distance.

Let $\mathbf{x} \in \mathbb{R}^3$ be a point in space where the value is to be interpolated, and let $\{\mathbf{x}_j\}_{j=1}^N$ be the set of known reference points (i.e., the mean positions where uncertainty or accuracy values are measured). Let v_j denote the known value (e.g., uncertainty or accuracy) at point $\mathbf{x}_j$.

1. Compute the Euclidean distances between the interpolation point and each reference point:

$$d_j = \|\mathbf{x} - \mathbf{x}_j\|$$

2. Compute the inverse distance weights:

$$w_j = \frac{1/d_j}{\sum_{k=1}^N 1/d_k}$$

If the sum of inverse distances is zero (i.e., all distances are infinite or undefined), a fallback weight normalization is applied to avoid division by zero.

3. The interpolated value $\hat{v}(\mathbf{x})$ at location $\mathbf{x}$ is then calculated as a weighted average of the known values:

$$\hat{v}(\mathbf{x}) = \sum_{j=1}^N w_j \cdot v_j$$

This method ensures a smooth and spatially consistent interpolation, giving higher influence to nearby points while still incorporating information from all available reference data.

6.4.3 Position accuracy

The position estimate accuracy is defined as the distance from the mean of the data to the actual position of the drone when logging the position. This distance is shown in Figure 6.8

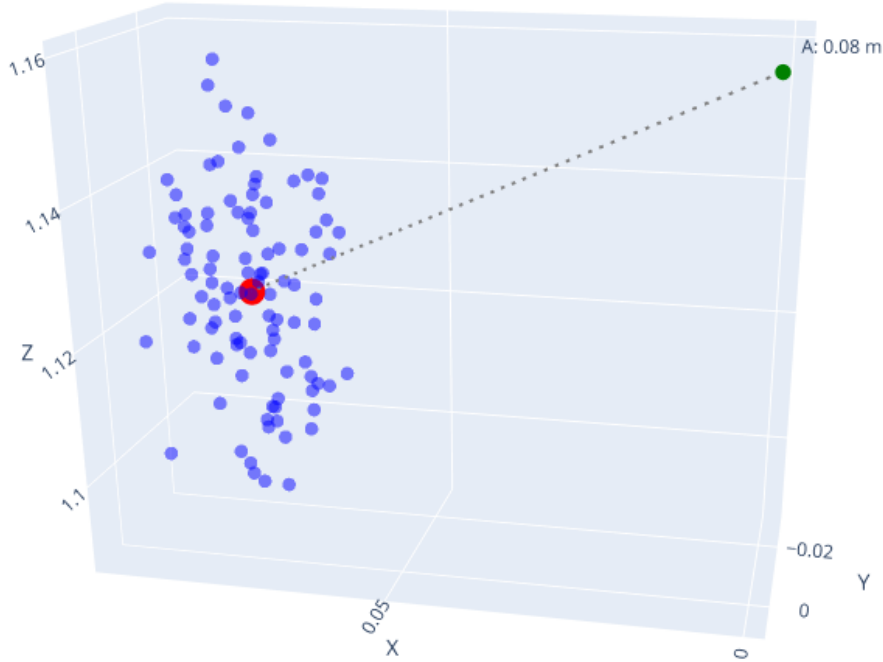
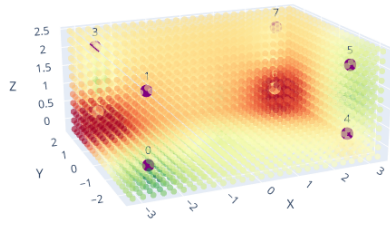


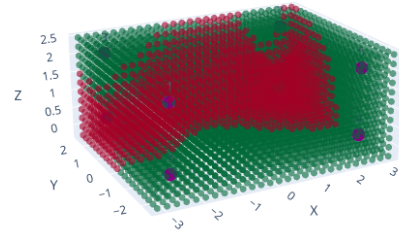
Figure 6.8: The position accuracy measurement defined as the distance from the mean of logged data to the real position

In Figure 6.9, the positional estimate's precision is visualized in space. In Figure 6.9a is the Red-Yellow-Green color spectrum corresponding to a precision

with deep red meaning a precision of 25 cm in radius, yellow meaning a radius of 12 cm, and green down to 0 cm. Figure 6.9b shows the exact same plot, but with values clipped to green when they are within the wished values of max a radius of 17 cm, and to red if they are worse than this. Which means that most of the flying volume has this precision or better. The areas which does not satisfy these requirements are near one long edge of the flying volume near the bottom. In practice was this a problematic area where the position of the drones dipped down, so it reflects the reality nicely.



(a) Positional precision uncertainty cloud

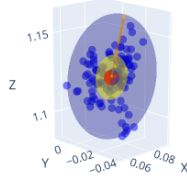


(b) Positional precision uncertainty cloud clipped

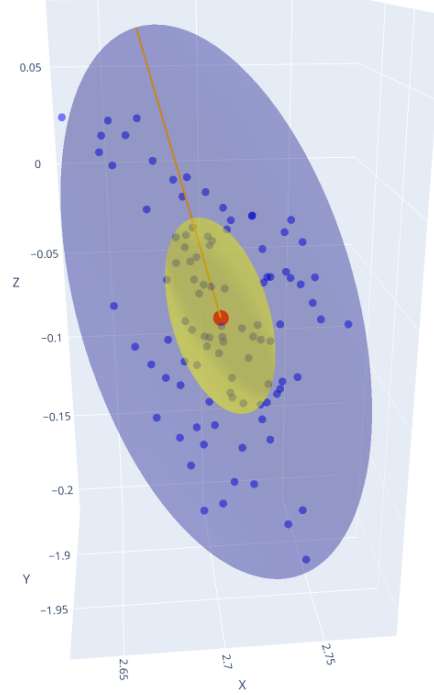
Figure 6.9: This figure shows a heatmap of the positional estimate's accuracy

6.4.4 Position precision

Two ellipsoids were generated to visualize the spatial precision in three dimensions: one representing the standard deviation of the position measurements, and another enclosing 95% of the recorded data points. These ellipsoids, shown in Figure 6.10, provide insight into the spread and orientation of localization errors. The longest axis of each 95th percentile ellipsoid, highlighted in orange in the Figure 6.10, was identified as the direction of greatest uncertainty. The length of this axis was used as a conservative estimate for the minimum recommended spacing between drones at that specific location. These values were then interpolated to fill the entire volume with the corresponding values. Bitcraze states that a precision in the decimeter range is to be expected by the loco positioning system [51].



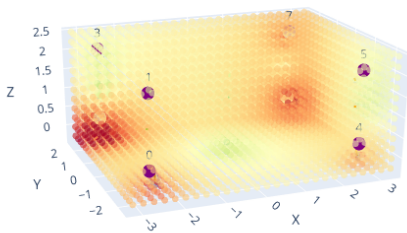
(a) Point cloud and precision ellipsoids for the center of the volume. 95th percentile worst axis: 3.9cm



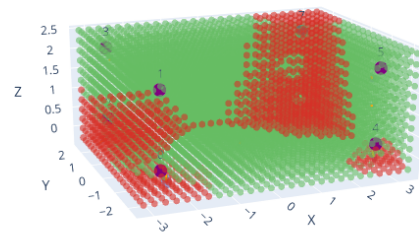
(b) Point cloud and precision ellipsoids for the worst corner of the volume. 95th percentile worst axis: 25cm

Figure 6.10: Point cloud and precision ellipsoids for two selected points of the volume. One is the theoretically best point, while the other is the worst corner in practice.

In Figure 6.11, the positional estimate's precision is visualized in space. In Figure 6.11a is the Red-Yellow-Green color spectrum corresponding to a precision with deep red meaning a precision of 15 cm in radius, yellow meaning a radius of 10 cm, and green down to 5 cm. Figure 6.11b shows the exact same plot, but with values clipped to green when they are within the wished values of max a radius of 11 cm, and to red if they are worse than this. Which means that most of the flying volume has this precision or better. The areas which does not satisfy these requirements are in some of the corners and especially those at the bottom.



(a) Positional precision uncertainty cloud



(b) Positional precision uncertainty cloud clipped

Figure 6.11: This figure shows a heatmap of the positional estimate's precision

6.4.5 Accuracy and Precision evaluation

The precision and accuracy of the Loco Positioning System were evaluated through systematic logging of stationary drone positions at multiple locations in the flying volume. Precision was quantified by calculating standard deviation ellipsoids and 95th percentile confidence regions around the recorded data points. Accuracy was assessed by comparing the mean estimated positions with the known ground-truth locations.

The results reveal that while large portions of the flying volume achieved precision within 11 cm or less, localized areas—particularly near the lower boundaries of the space—exhibited degraded performance, with uncertainties reaching up to 25 cm. Accuracy followed a similar trend: most of the flying volume aligned well with the physical placement, but slight spatial drift was observed near the corners and lower regions.

These findings were expected, since the measurements on the floor technically are outside of the flying volume, and it is known that positional measurements degrade rapidly outside of the flying volume. This means that while the system meets the general expectations for UWB-based positioning, it probably operates near the edge of feasibility for dense swarm applications without further filtering or spacing.

With tuning of filters and possible hardware layout refinements, both precision and accuracy could likely be improved, but the system in its current state already provides a usable platform for swarm control with defined spatial tolerances.

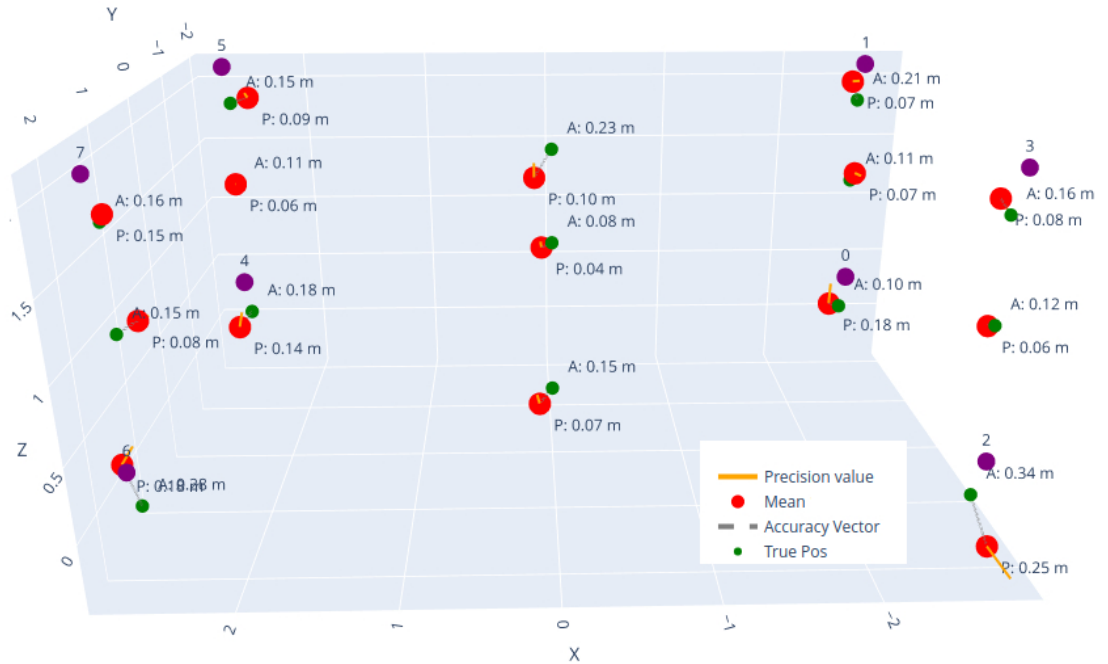


Figure 6.12: Values for position estimate precision and accuracy alongside the node number and placing in space

According to Bitcraze, the expected localization precision for the Loco Posi-

tioning System, which is based on the Decawave DWM1000 chip, lies around 10 cm [51]. This means that the result of 11 cm in most of the flying volume makes for a very comparable outcome with what is possible to achieve. Now, for using the suggested spacing of the drones, this measurement needs to be doubled, since a pair of drones could both be, in worst case, positioned 10 cm closer each than anticipated.

⇒ Purely based on the position estimate precision should the drones therefore be flown with a minimum planned distance of 20 cm.

6.5 To further increase the performance of the Loco positioning system

Many exciting opportunities open up as a result of this setup and the results from testing. There are undeniably many possible improvements to the positioning setup. These were not executed because they were out of scope, but some suggestions will be presented here.

6.5.1 Adjustments to the physical node setup

Trying different configurations for the node layout, possibly increasing the number of nodes. This could yield better results. To increase the number of nodes, the mode of the nodes must be switched to TDoA3 in the *cfclient*.

Exploring if the power bank being placed so close to the node decreases signal integrity. It is then a valid option to move the power banks to the base of the stand, where they would also increase the stand's stability, though lowering the center of mass.

6.5.2 Increasing Measurement Accuracy

Improving localization accuracy can be achieved by experimenting with node density and placement geometry to reduce geometric dilution of precision (GDOP). Using additional anchors increases the number of intersecting TDoA hyperbolas, improving spatial resolution.

6.5.3 Tuning the Controller

Fine-tuning the PID gains used in the Crazyflie's cascaded controller may lead to better response and stability. This can be done live through the *cfclient* interface, enabling iterative testing. Methods such as Ziegler–Nichols, step response tuning, or optimization-based tuning (e.g. minimizing trajectory error) can be explored. Testing with payload variations or at different speeds can reveal tuning trade-offs across operating conditions.

6.5.4 Tuning the Kalman Filter

The EKF used for state estimation relies on carefully chosen process and measurement noise covariances (Q and R). By logging flight data and comparing estimated versus actual positions, one can refine these parameters. A sensitivity analysis could be performed to evaluate how filter performance changes with different noise assumptions. Additionally, enabling or disabling sensor inputs (e.g., barometer or optical flow) allows for modular testing of their impact.

6.5.5 Tuning the Outlier Filter

The outlier rejection filter can be adjusted to be more or less aggressive in discarding anomalous data points from the UWB system. Parameters such as innovation thresholds or gating values can be tuned based on recorded measurement residuals. Adaptive or statistical filtering methods like Mahalanobis distance-based gating can also be evaluated to improve robustness to measurement spikes and environmental multipath effects.

PATH PLANNING

Flying a swarm of drones in formation, such as to spell out words or create figures, requires precise control and coordination. In this thesis, the chosen approach is to define discrete waypoints for each drone and interpolate between them, leveraging existing library functionality that allows position-based control. Since the Crazyflie 2.X drone is used, must the path planner adhere to the already established waypoint system for path usage, and this is also taken into account when designing the path planner.

7.1 The reason to make a path planner

Each waypoint must include spatial coordinates (x, y, z) and a time reference t , but manually entering these values quickly becomes cumbersome and impractical for complex trajectories. This might be apparent, but the issue goes deeper than it seems:

When defining a single path, it is key to get the coordinates correct. If the drone is supposed to fly in a specific shape, for example, in the shape of a star, the coordinates must be very accurate to be distinguishable as a star. Manually typing these out in a file somewhere is hard enough in itself, but when the paths are defined, all testing must be done with the physical hardware, which increases testing time. It is also hard to know exactly where the drone is, since the positioning system is as noisy as it is.

Now, increasing the number of drones in a swarm, the problem becomes even greater. Now all the drones need to form the correct paths, but in addition there is now the problem of collision. Since the only way to test out the paths is with the physical system, will this most likely end up with crashes in-air and many broken parts.

Since the drone application in this case will be using a BCI for input, will there be multiple inputs. Each of these require a different path for the audience to distinguish between the different MI tasks. The drones will therefore need to perform multiple different paths, in an unknown order. So not only is it necessary to know that the paths do not collide themselves, but the paths must also not collide when

doing transition between paths.

It is for these reasons near impossible to achieve usable and safe flight without a tool to handle some of the process automatically, like a path planner. Preferably also one which has some collision awareness in the planning stage this issue can be dealt with before the paths are executed with hardware.

7.2 Path Planning Software Design

To address these issues were a set of key requirements identified to guide the development of this software. This section is therefore, along with both development and implementation, modularized.

- First, the software must be implemented in a way where both the definition and visualization of paths is feasible while sharing much of the basic functionality. It should be implemented as a cohesive framework with one common codebase
- Defining points
 - The operator should be able to click directly on a 2D canvas to define waypoints.
 - A second input system using manual inputs should also be possible
 - An option to change the parameters of the existing points is also crucial.
- Visualization of a path
 - Needs to be 3D to get a proper idea of how the path looks
 - Simulation/animation
- Visualization and simulation of multiple paths at once
 - 3D canvas with possibility to load multiple paths at once
 - require all other parts of the software to be able to handle multiple instances of drones, animation, collision etc
 - Simulation functionality that mimics the interpolation behavior used by the drone in flight. Interpolating between current position and the waypoint
 - Collision detection between drones in the swarm should they come too close
- A path load/save manager
 - Beyond initial input and visualization, the software must support transferring of path data to and from the canvases
 - Saving paths
 - Loading one path
 - Loading many paths

- Loading starting positions
- Clearing canvases
- Handling simulation time
- Synchronize animation if more than one canvas is used at the same time

To address both usability and safety while enabling future expansion, this thesis introduces a custom-built 3D visualization GUI. This interface serves as a foundation for defining, testing, and executing drone paths, designed from scratch to meet the specific demands of swarm flight and collision-aware path planning.

7.3 A Path Planner tailored to the Crazyflie 2.X quadcopter swarm

To tailor the path planner for use with the Crazyflie drone, it is essential that the software interfaces directly with the drone’s existing waypoint functionality. The communication must follow the same structure and data types as those used in the official Crazyflie API. Specifically, each waypoint must be defined using three floating-point values for the spatial coordinates (x, y, z) and a fourth floating-point value representing the time allocated for the drone to transition to the next waypoint. This ensures that the paths planned and tested within the software can be seamlessly applied to real-world flights without additional conversion or formatting.

```
def run_sequence(scf: SyncCrazyflie, sequence):
    for arguments in sequence:
        commander = scf.cf.high_level_commander
        x, y, z = arguments[0], arguments[1], arguments[2]
        flight_time = arguments[3]
        print(f'Setting position {x, y, z} to cf
              ↪ {scf.cf.link_uri}')
        commander.go_to(x, y, z, 0, flight_time, relative=False)
        time.sleep(max(flight_time + 0.1, 5))
```

Listing 2: The `run_sequence()` method which takes in the point sequence from the planner and interfaces the drone.

The `go_to()` method in Listing 2 is the member function of the Bitcraze library which interfaces with the drone on a lower level. The function given is therefore the translator between these systems. In reality enables bitcraze for a yaw input in addition to x, y, z and time, but this was deemed unnecessary for the project needs. Since all drones are facing positive x , is it sufficient to always face the audience at the start to know that the drones will stay correctly oriented.

7.4 Path planner architecture

To accomplish this integration efficiently and cleanly, the software is developed using object-oriented programming (OOP) principles. A central class, called *PointCanvas* is implemented to encapsulate the core functionality of the path planning system. This includes methods for visualizing paths, simulation, coordinate transformation, projection of points, plotting of lines and so much more. Inheriting from this class will two new classes be defined, one for 2D point-definition, and one for 3D visualisation. The extended functionality of these correspond to what interaction is possible and if there are any other extra functionality, like the 2D having manual input fields aswell.

In addition is there both a class for a single *Path*, a class for a *Drone* which contains multiple paths, the drone position, radioID etc, and also a class which inherits from this drone class called *SimDrone*. It inherits from and extends the *Drone* class to also store and use information related to simulation like canvas information and the `draw_drone()` method.

Lastly, is there the *CanvasManager* class, which is responsible for all external communication. It handles saving and loading paths to external files and synchronization of simulation animation, simulation time and canvas updates between canvases if there are multiple.

By encapsulating these features in a class-based architecture, the design allows for modular expansion, improved code readability, and reusability in future projects. The full class diagram can be seen in Figure 7.1, and each part will be discussed in more detail afterwards.

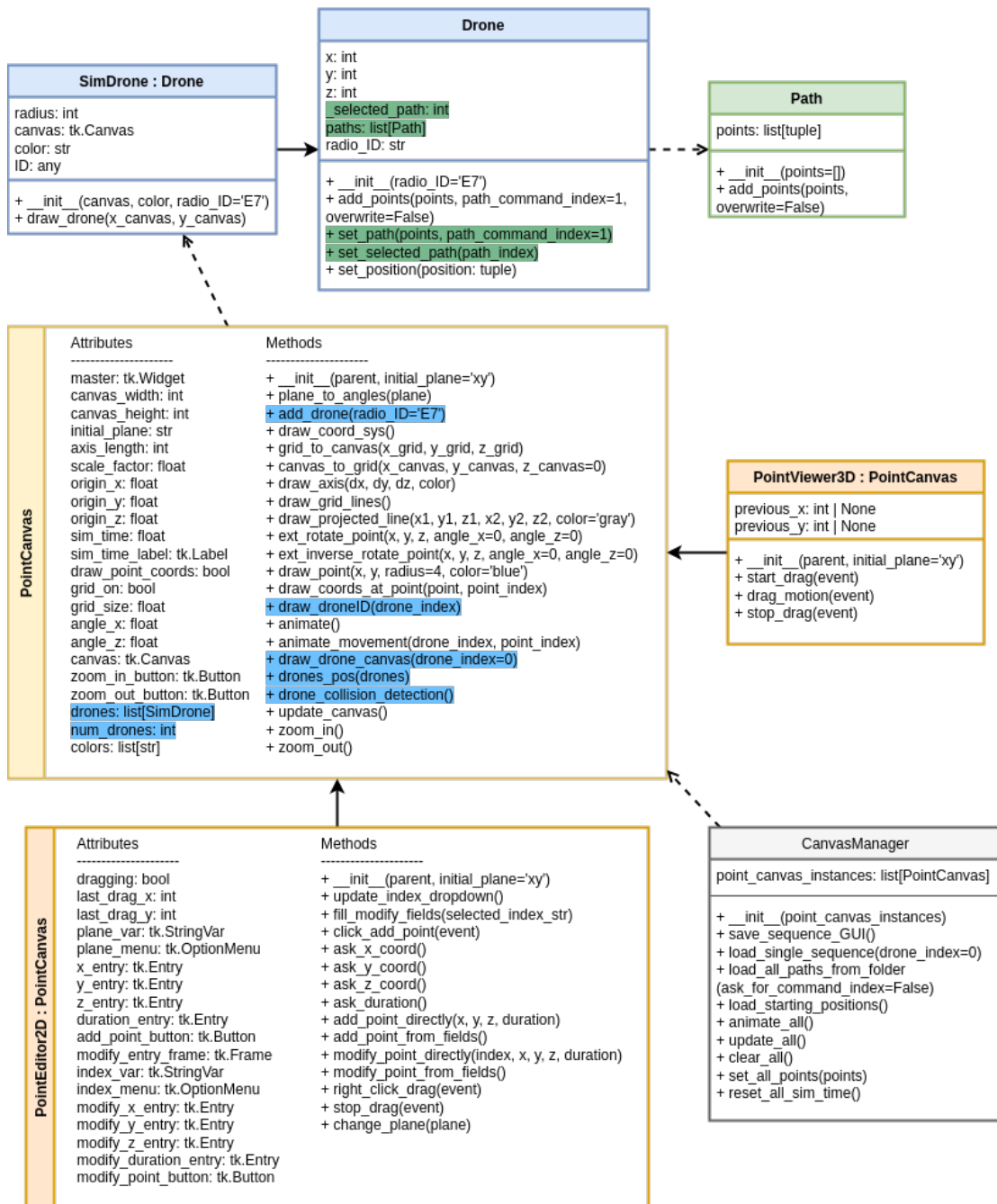


Figure 7.1: The class diagram for the Path planner. It uses UML style as described in [57]. The attributes and methods which are colored, represent functionality which is connected to a class with the same color.

7.4.1 Drone Representation

The **Drone** class stores information about individual drones, including position data and assigned flight paths. Each drone can maintain multiple distinct paths via a list of **Path** instances, allowing for the definition of command-specific flight trajectories. An example of path initialization is shown below:

```
self.paths = [Path() for _ in range(4)]
```

The `Path` class is a simple data structure that maintains an ordered list of 3D points, allowing for addition and optional overwriting of waypoints. This structure enables flexible modification and extension of drone behavior during runtime.

7.4.2 Simulated Drone Extension

To integrate drones into the graphical canvas, the `SimDrone` class extends the `Drone` class by adding graphical attributes such as color, drawing radius, and a canvas-specific object ID. It also defines the `draw_drone` method, which updates or renders the drone on a Tkinter canvas given canvas coordinates:

```
self.ID = self.canvas.create_oval(
    x_canvas - self.radius, y_canvas - self.radius,
    x_canvas + self.radius, y_canvas + self.radius,
    fill=self.color)
```

This allows the drone's physical state to be tightly coupled with its visual representation.

7.4.3 PointCanvas Interface

The `PointCanvas` class creates a resizable, zoomable graphical interface to visualize and interact with the drone environment. It supports multiple planes of projection (e.g., `xy`, `xz`, `yz`) using transformation matrices defined via angle parameters `angle_x` and `angle_z`. These parameters are initialized according to the selected projection using the `plane_to_angles` method.

The coordinate system is scaled and centered within a Tkinter canvas, allowing conversion between world coordinates and screen coordinates using methods such as `grid_to_canvas` and `canvas_to_grid`. These allow for consistent rendering of grid lines, axis lines, and annotations.

Grid rendering is controlled via the `draw_grid_lines` method, which iterates through calculated grid boundaries and renders projective lines based on the selected coordinate plane. Similarly, the `draw_axis` method visually indicates the coordinate system origin and directions using color-coded lines and axis labels (X: red, Y: green, Z: blue).

7.4.4 Drone Integration and Drawing

Drones are instantiated through the `add_drone` method, which assigns a unique color and radio ID to each `SimDrone` object before appending it to the internal drone list. Upon each canvas update, drones are re-rendered at their current positions using the `draw_drone_canvas` method, ensuring synchronization between simulated state and visualization.

The system is built with extensibility in mind, allowing future support for animation, collision detection (via external modules), and time-stepped simulation using the internal `sim_time` counter.

7.4.5 Data Structure and Storage

Each individual drone is represented in code by an instance of the `Drone` class. Each `Drone` contains a list of `Path` objects, with each `Path` corresponding to a particular command index. Each `Path` object contains a list of waypoints stored as 4-tuples (x, y, z, t) where x, y, z are coordinates in meters, and t is the duration in seconds the drone should take to fly from the previous waypoint to the current one. A more GUI-oriented extension of `Drone` is implemented in the `SimDrone` class, which adds canvas-rendering capabilities using Tkinter.

Paths are stored on disk using the following folder hierarchy:

```
./data/flight_patterns/<swarm_path_name>/<radioID>_sequence.JSON
```

Each swarm path has its own folder, and each file within that folder contains the waypoint sequence for a specific drone in that path, identified by its radio ID. This file organization supports flexible mapping between drones and flight commands and allows arbitrary reuse and swapping of swarm behaviors. Within the program, paths are handled via a central `CanvasManager` class instance, which loads and saves all drone paths to and from disk. (Note: this structure assumes that drone hardware is shared across swarm sequences, which should be verified and documented as a design constraint.)

7.4.6 Graphical Path Definition

The graphical user interface supports both visual and manual entry of flight paths. Users define paths through a 2D point-definition canvas, which supports plane selection (e.g., XY, XZ, YZ) and uses popup dialogs for entering the remaining coordinate and the time value for each point. Alternatively, users may input waypoints manually via a dedicated form located below the canvas, which allows exact specification for high-precision requirements.

This dual-mode input ensures that the software remains accessible to both novice users and experienced operators requiring tight control over flight paths. In either case, the path data is stored internally within the `Drone` instances and synchronized to disk via the `CanvasManager`.

7.4.7 Editing and Reusability

The GUI also allows users to load existing paths as templates, edit specific points (including position and duration), and re-save paths under new swarm names. This is particularly valuable in iterative testing or in cases where collisions or timing issues are discovered post hoc. Points may be edited individually without redefining the entire path.

Although there is currently no built-in versioning or undo system, users are encouraged to maintain versioned filenames or folders manually as a part of their workflow. (Note: a naming convention for this purpose should be documented in the user guide.)

7.4.8 Concurrency and Command System

Each drone receives its waypoint list via a dedicated thread in the main program, ensuring that communication delays in the main loop do not affect command transmission. Path execution is mapped to integer-based commands, with each command corresponding to a specific path index in the `Drone` object. These commands can be triggered via GUI buttons or external inputs such as keyboard commands or classifier models. This mapping system provides a robust and extensible interface between high-level control signals and low-level path definitions.

7.4.9 CanvasManager: Coordinated Control and Synchronization of PointCanvas Instances

The `CanvasManager` class acts as an overarching controller for coordinating multiple instances of `PointCanvas`. This manager is essential in multi-canvas environments where synchronized operations such as saving, loading, animating, and resetting drone path data are required across several visual canvases. It supports functionalities such as persistence of sequences, GUI-assisted file operations, and runtime synchronization of path updates.

`CanvasManager` is initialized with a list of `PointCanvas` instances:

```
def __init__(self, point_canvas_instances: list[PointCanvas]):
    self.point_canvas_instances = point_canvas_instances
```

Listing 3: Initialization of the Canvas manager class requires reference to all canvas instances in the current program

Each `PointCanvas` contains `Drone` objects, which in turn hold `Path` instances. This nested structure allows the manager to access and manipulate trajectory data for each drone via its associated canvas.

Saving Sequences

The `save_sequence_GUI` method allows the user to persist the trajectory of the first drone in the first `PointCanvas` instance. It uses a GUI prompt to gather a folder path and drone ID, then serializes the current path's points using `json.dump()` into a file named `<ID>_sequence.json`. This ensures compatibility with the rest of the system which loads paths based on the drone ID.

Loading and Distributing Sequences

To support flexibility in testing or operation, the manager provides both `load_single_sequence()` and `load_all_paths_from_folder()`. The former loads a sequence for a specific drone index across all canvases, while the latter imports all JSON files from a directory, matching them by drone ID and appending them to newly created or existing `Drone` instances.

In `load_all_paths_from_folder()`, the user may be prompted to choose a `command_index` (e.g., for left/right hand classification), after which the corresponding points are added to the drone paths. This method integrates with the `set_all_selected_paths` function (from the swarm module) to ensure the same command index is applied across all drones, ensuring alignment during classification or synchronized control.

Canvas-Level Control and Synchronization

The methods `animate_all()`, `update_all()`, and `clear_all()` allow for holistic canvas management. For instance, `clear_all()` resets all drone instances and repopulates each canvas with a single default drone to avoid downstream index errors. Similarly, `set_all_points(points)` pushes a uniform point list to the first path of the first drone on each canvas instance, ideal for broadcasting a shared motion plan.

Finally, `reset_all_sim_time()` zeroes out simulation time across all canvases. This is useful before new path animations or when re-running sequences to ensure consistency in time-based visualization.

Integration with PointCanvas and Drone Abstractions

The `CanvasManager` complements and extends the design of the `PointCanvas` class by centralizing batch operations. While `PointCanvas` is responsible for rendering and internal logic of a single visual space, the manager operates across multiple canvases, facilitating GUI interaction, batch updates, and synchronization of drone data.

This modular separation allows for high flexibility and scalability. For example, different canvases could represent different stages of a motion plan, user classifications, or test conditions, all while being managed uniformly through the `CanvasManager` interface.

7.5 Simulation and collision danger detection

To prepare for both individual and multi-drone simulations, the animation system must support concurrent visualization of multiple drone paths. This is particularly important in swarm flight scenarios, where formation coherence and collision avoidance are vital. Given that the Crazyflie’s position estimates are known to be somewhat noisy, an issue discussed in the earlier Chapter, Section 6.4 on positioning, there must also be a system in place to monitor the proximity of drones during simulation and highlight potential collision risks. This adds an extra layer of safety and planning support.

7.5.1 Simulation of drone paths

In this thesis, the simulation of drone trajectories is achieved using an animated time-stepped linear interpolation method. The algorithm animates the movement of drones between discrete waypoints defined in a selected path. The Python

implementation executes this interpolation through the `animate()` and `animate_movement()` functions.

The method emulates the behavior of the `go_to(x, y, z, yaw, duration)` command from the Crazyflie Motion Commander API, which directs a drone to move from its current position to a specified target over a given time duration. Similarly, in the simulation, each drone proceeds from a start position $\mathbf{p}_0 = (x_0, y_0, z_0)$ to a target position $\mathbf{p}_1 = (x_1, y_1, z_1)$ using linear interpolation over a fixed number of steps.

Given a total movement duration T , the movement is divided into N equal steps. At each step $i \in [0, N]$, the position of the drone is updated as:

$$\mathbf{p}_i = \mathbf{p}_0 + \frac{i}{N}(\mathbf{p}_1 - \mathbf{p}_0) \quad (7.1)$$

This approach results in a piecewise linear trajectory, ensuring a smooth visual transition from one waypoint to the next. Each animation step is executed after a fixed delay $\Delta t = \frac{T}{N}$ using the Tkinter event scheduler via `after(delay, ...)`. The simulation time is concurrently updated, maintaining temporal consistency.

7.5.2 Comparison to Polynomial Interpolation

While linear interpolation is computationally simple and guarantees predictable motion, it may result in discontinuous velocities or accelerations at path junctions. In contrast, polynomial interpolation—particularly cubic or quintic—can produce smoother trajectories with continuous first and second derivatives. A polynomial trajectory for position $\mathbf{p}(t)$ is typically of the form:

$$\mathbf{p}(t) = a_0 + a_1t + a_2t^2 + \dots + a_nt^n \quad (7.2)$$

such that it satisfies boundary conditions for position, velocity, and optionally acceleration at both ends of the trajectory. This is particularly advantageous in real-time control for physical drones where abrupt changes in acceleration are undesirable due to physical constraints or control bandwidth.

However, for the purposes of simulation visualization—where simplicity, clarity, and real-time responsiveness are priorities—the linear interpolation approach provides a practical and efficient solution. This differs from the Crazyflie’s internal use of polynomial interpolation. While this introduces a visual mismatch, it is acceptable for user validation purposes, especially considering that real-world inaccuracies are often dominated by noise in the drone’s position estimation system. Future improvements may include integrating polynomial visualization and real-time estimation overlays to more accurately reflect onboard drone behavior.

7.5.3 Collision danger detection

The precision of the position measurements were found to be approximately ± 10 cm. Combined with the drone’s size of approx 12 cm diagonally would the drones be in danger of colliding if two spheres, each at the desired drone’s location, with a radius of 16 cm were to intersect.

In Listing 4 is pseudocode for the collision detection method which is implemented. It uses a KDTree to search for neighbors within a certain radius. It is set as 0.16 m. The collision pairs function is given as an optional input if the detection should be continued between function runs in the future.

```
# Usage of the function
col_pairs = collision_danger(pos_drones, radius=0.16)

...

# Definition
def collision_danger(points: np.ndarray, collision_pairs=None,
    ↪ radius = 0.1):
    """Detect potential collisions between points within a
    ↪ specified radius given in meters."""
    for i, query_point in enumerate(points):
        tree = KDTree(new_points)
        neighbors = tree.query_ball_point(query_point, radius)
        for neighbor in neighbors:
            if i != neighbor_original_idx:
                col_pair = [query_point, neighbor]
                if col_pair not in collision_pairs:
                    collision_pairs.append(col_pair)
    return collision_pairs
```

Listing 4: Pseudo code for the collision danger detection method

These potential collisions are flagged and printed to the terminal but are not enforced by the GUI logic, allowing the user to choose whether to act on them. The function `collision_danger()` (see `src/positioning_system/collision_detection.py`) implements this functionality efficiently using `scipy.spatial.KDTree`.

7.6 Starting position configuration

The starting positions of the drones are very important to have well defined when the swarm starts to become large. After some testing does it seem like 0.5 meters space between the drones are smart when starting the drones, and 0.2 when flying. The reason for this is that when starting the drone, the drones are affected by something called "the ground effect" which means that the thrust of the drone is positively affected by there being a plane close beneath it. This can make the drone less stable when taking off or when landing.

To simplify the flying experience and reduce the chance of user error, is a standard starting position convention a possible improving factor.

During brainstorming were many options evaluated, but out of these stood two out for having both a practical pattern and centralized positions: "The snake" and "The quadrants"

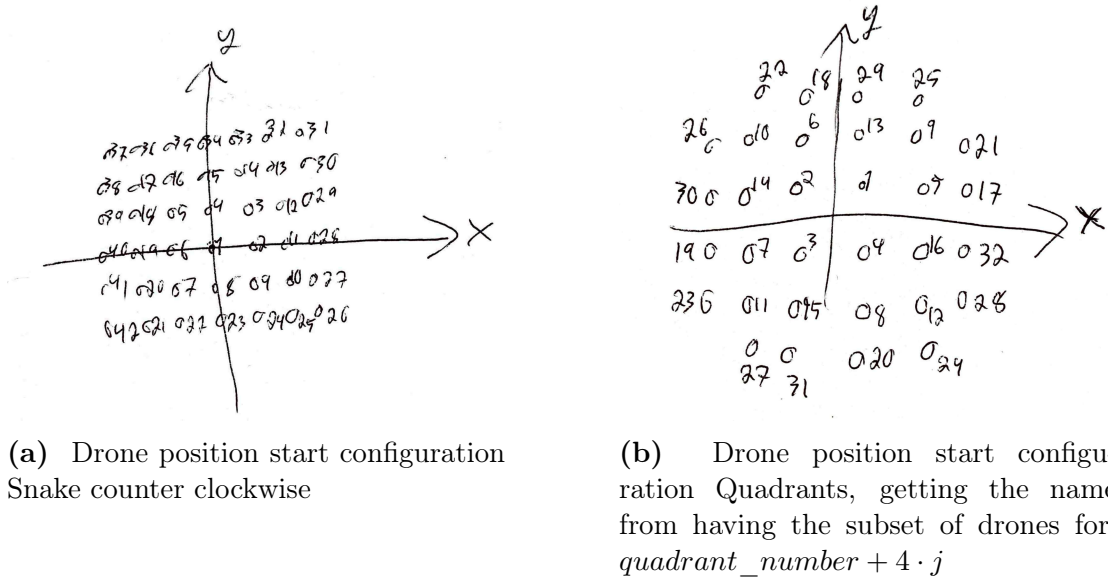


Figure 7.2: Drone position start configurations

The thought behind the quadrant one is that the "center of drones" remains as close to the origin no matter how many drones are flown with. This means that it will always make sense to use the same drone set for a part of a given path even if you add two more drones. In the case where you expand an existing pattern, and want it to look even, and therefore you add one drone to each side, will the same drones be used for the old positions and the new ones will be added distributed across the space.

However, the snake configuration is deemed to have the benefit of being very easy to remember the pattern of, and is therefore easier to use in practice. You also then get a drone positioned in the origin, which makes sense in the case of $n = 1$ drones.

Since there is no limitation of using drones which comes directly after each other in the radioID order, can whichever set of drones which is needed, be used. For example will the set 02, 03, 05, 06, 07 and 09 shape out two lines in the y direction, and work completely fine. The positioning conventions intuitiveness is therefore more important in larger swarms, and in niche cases.

The standard configuration for starting positions, which are included in the code base by default, is therefore chosen to be the counter clockwise snake configuration.

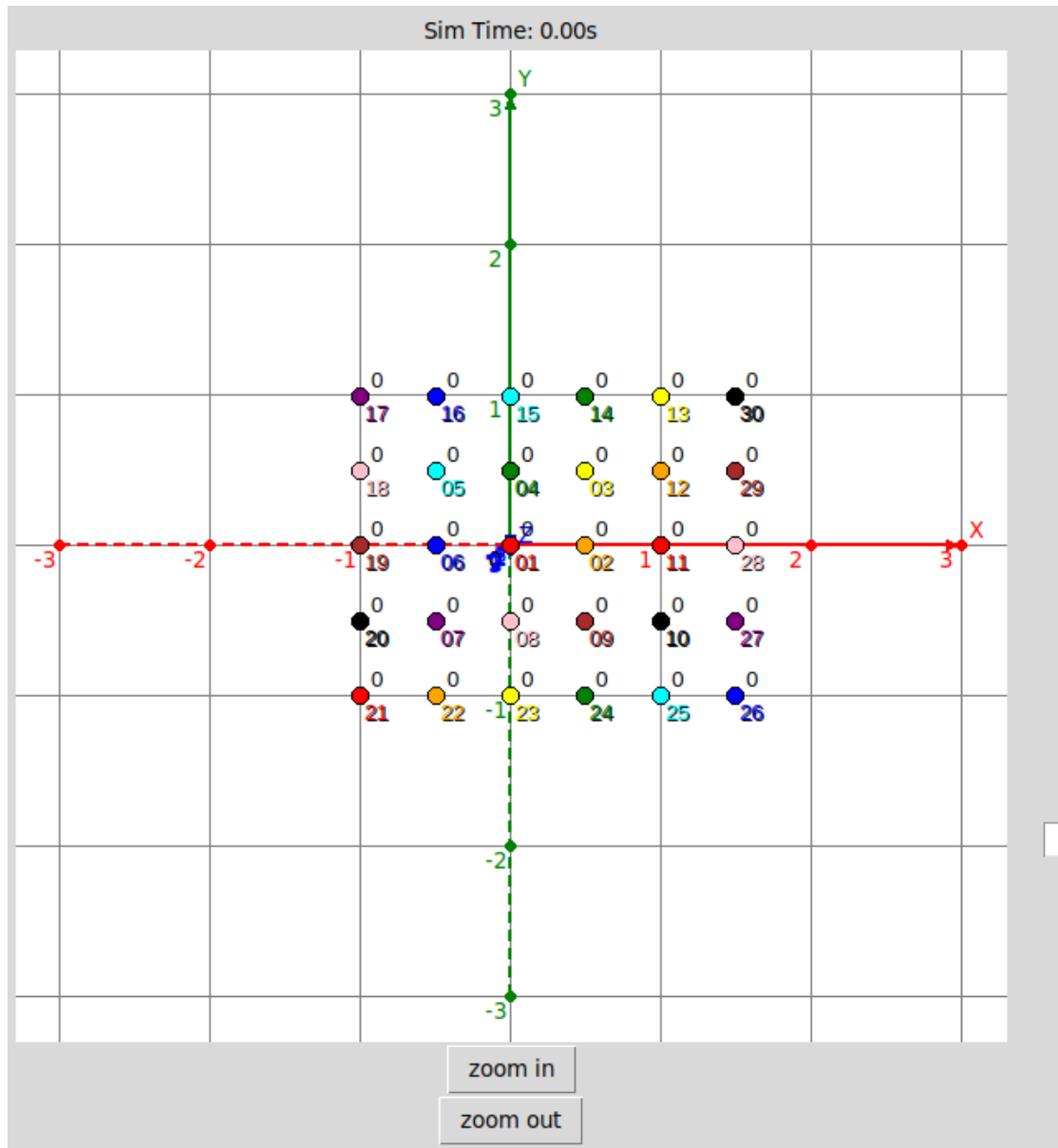


Figure 7.3: The standard starting position

7.7 Setting up and using the drones

The following procedure is meant to help guide the users, and will therefore also be included in the User Guide.

1. Setting a custom URI (Uniform Resource Identifier) is crucial for the flying of the drones in a swarm. The program needs to be able to communicate with the drones and know the difference between them. To do this connect the drone to the *cfclient*. The default radio URI is: `'radio://0/80/2M/E7E7E7E7E7'`.
2. Set the last two characters to define the number on the drone. This is done through the menu at the top: `connect->Configure 2.X`.

3. Write the radio ID number (last two digits) on the drone as seen in Figure 7.4 To differentiate between them. The total URI of the first drone (with drone number 01) would be: `'radio://0/80/2M/E7E7E7E701'`. The radio ID is the last part which is given in hexadecimal. It is strongly suggested to go straight from 09 to 10 (skipping 0A, 0B, 0C etc) to make it easier to use the correct individual and amount of drones.
4. Make sure the battery level of the drones are good, as this can result in the drones crashing if not.

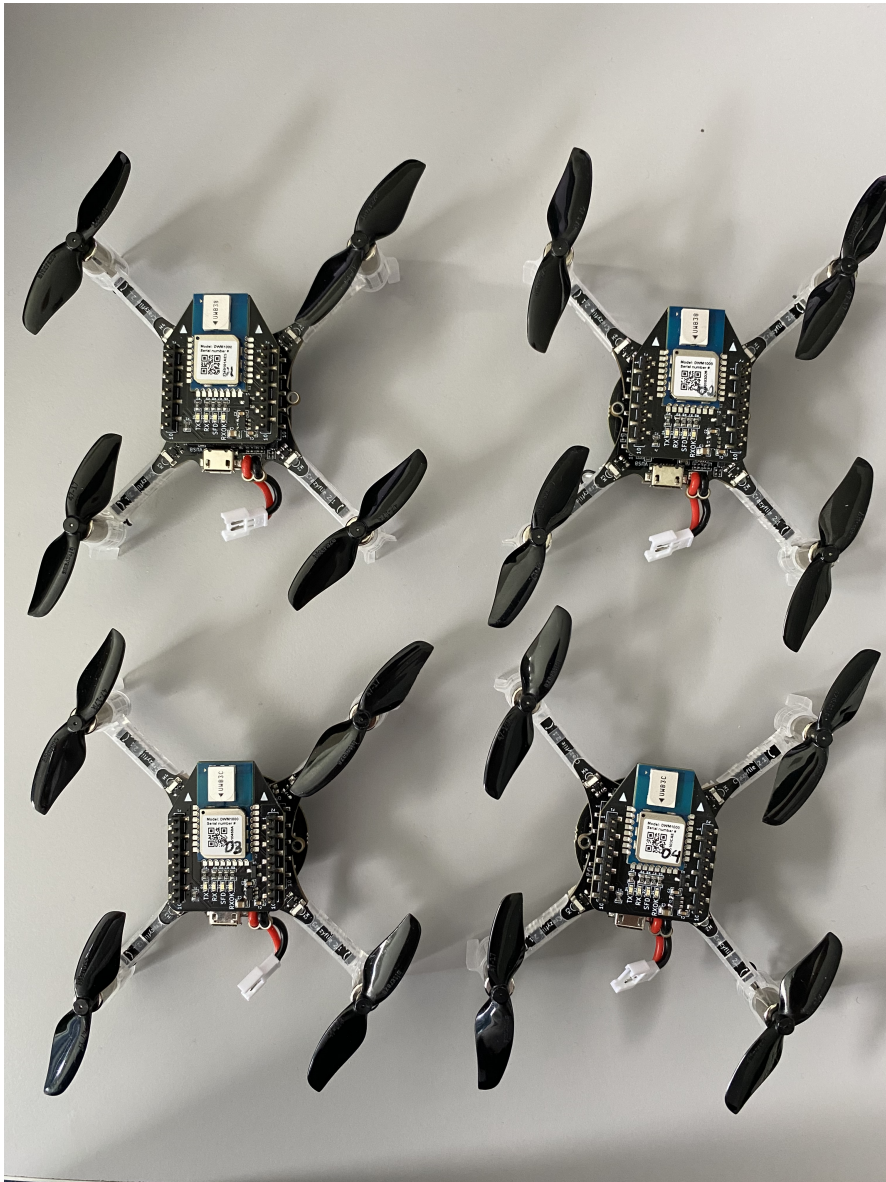


Figure 7.4: After configuring a drone ID have the last two characters of the ID been written on the drone. Drone 01, 02, 03 and 04.

7.8 Application of the path planner - Path planning for writing letters with drone lights

To test the application of the path planner, multiple paths will be defined. The primary goal is to use the planner to generate letter-shaped trajectories. Before this can be achieved, however, several standards must be established. Such standards include how the letters are represented as paths, and how the results are visualized in a clear and meaningful way.

7.8.1 Revisiting the Vision: Writing “NTNU” with Drones

A long-standing aspiration for this project has been the ability to orchestrate a drone swarm to visually spell out the letters “NTNU” in three-dimensional space. While this goal initially appeared overly ambitious, advancements in positioning accuracy, path planning, and visual layout design have brought this vision closer to reality. The challenge now lies in translating this ambition into a safe, reliable, and visually effective implementation.

7.8.2 Precision Requirements and Spatial Constraints

Tests performed using the optical positioning system revealed that each drone’s position estimate is associated with an uncertainty ellipse, with a maximum observed radius of approximately 10 cm. To ensure safety and minimize the risk of mid-air collisions due to position estimation noise, a conservative minimum separation distance of 20 cm between any two drones has been established. This safety buffer becomes a critical parameter in both the visual legibility of letter formations and the dynamics of drone navigation.

7.8.3 The Drone Font

A custom font, referred to as the *drone font*, was designed specifically to address the question: *What is the minimum number of drones required to form each letter of the alphabet in a distinguishable manner?* The font needs to define unique spatial configurations of drones for each uppercase letter (A–Z) and the exclamation mark. Figure 7.5 illustrates the proposed letters from this font.

Through iterative design and testing, it was determined that a maximum of 8 drones is sufficient to uniquely and recognizably represent every character in the alphabet. This efficiently reduces drone density and the number of drones needed, while still ensuring visual clarity.

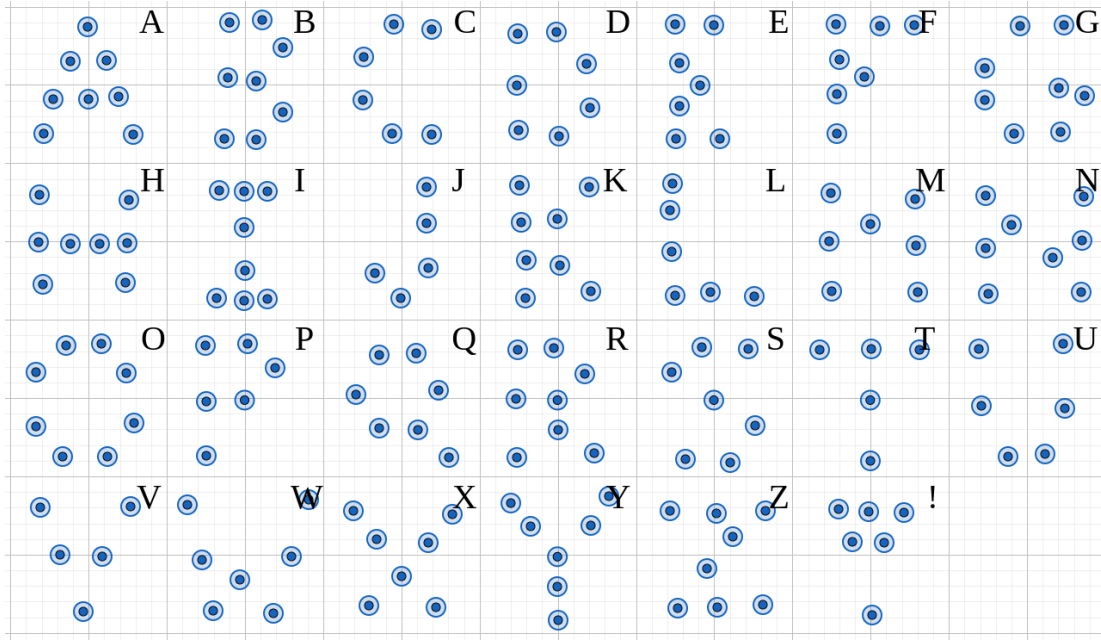


Figure 7.5: The proposed drone font, balancing complexity and, in turn, intuitiveness, and practicality

Many of the letters require fewer than eight drones to represent, such as "T", "J", etc. In these cases, an evaluation of whether or not the framework should use the same number of drones for all letters would result in very easy spelling, and a set of drones would go exclusively to a certain character. The downsides of this were, however, that some of the letters made no sense, including more drones, even if they use fewer drones, like "V" for example. A discrepancy of a few drones was therefore allowed. This further decreases the number of required drones to visualize a string of text.

Another consideration that was made was that for letters like "I", when we are allowed to use fewer drones for each letter, could be represented with just three drones. In cases like this, not only was the individual character clarity taken into account, but the letter was also considered in context with other letters as well. If just three drones were used to define the "I", could the sequence "IJ" maybe be misunderstood for a "W" or a "U", see Figure 7.6a. The noise of the position estimate, at least for the current state of the framework, will at times reduce the clarity significantly, so considerations like these are necessary. Letters like "I" are, therefore, to the greatest degree possible, given distinctive features that make them unique and thus distinguishable.

Letters like "M", "B", "W", and "Q" have the opposite problem. They have complex enough shapes where more than eight drones would be preferable. However, including more than eight drones would rapidly increase complexity and drone density, and so the limit was set at eight. The letters are still distinguishable, but "B" and "Q" especially suffer from this. "B" has some reduced clarity, and "Q" differs slightly from standard lettering. Alternatives which use nine drones for these letters are given in Figure 7.6b.

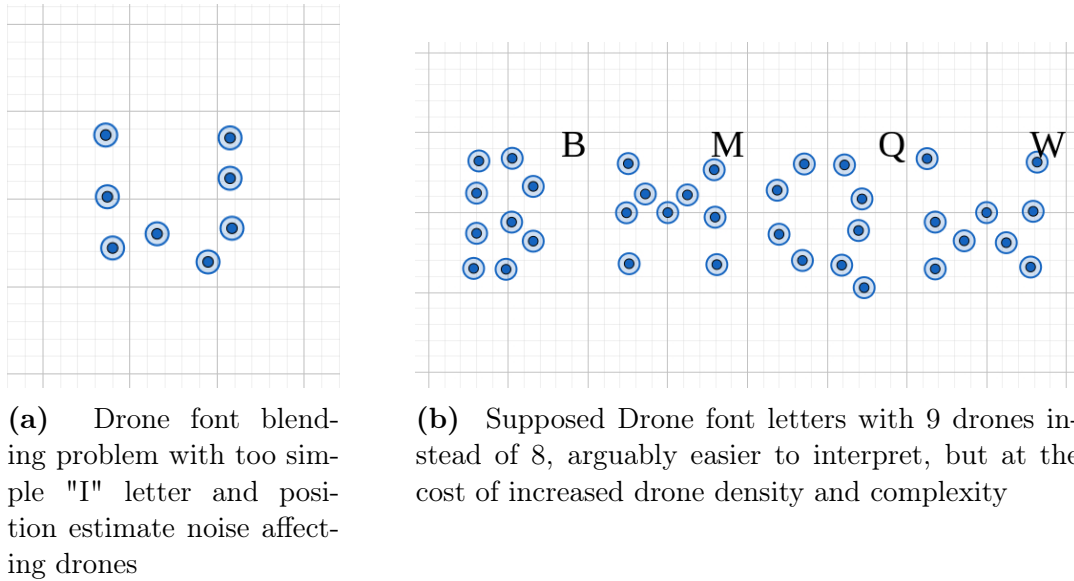


Figure 7.6: The drone font is defined as shown earlier, but this figure shows a problem with both too complex and too simple letters

7.8.4 Considerations for 3D Spatial Layout

Operating in three dimensions introduces unique challenges, particularly related to vertical positioning. A key concern arises from aerodynamic interference: when drones fly directly above one another, the downward airflow known as downwash, from the upper drone creates instability for the drone below. This issue was confirmed by experimental testing later described in this report, where it was found that drones positioned vertically in line experience significant disturbances, leading to erratic behavior and compromised flight stability.

To address this, are all drones placed on a sloped visualization plane. By angling the formation plane, drones can occupy visually distinct positions without overlapping vertically. This ensures that no drone is directly underneath another, thereby reducing the impact of downwash while preserving the intended visual effect from the observer's point of view.

In addition to the sloped plane, an intermediate transition step was incorporated into the path planning process. Before moving into their final positions in the visualization plane, all drones first move to predefined positions in a separate intermediate plane. This intermediate plane is spatially offset such that drones can reach it without crossing above or below others. Only after all drones have reached this staging area do they proceed to their final target positions. This two-phase movement strategy minimizes the risk of unsafe vertical overlaps during dynamic transitions and contributes to a safer overall system.

These design decisions were necessary responses to the following critical observation:

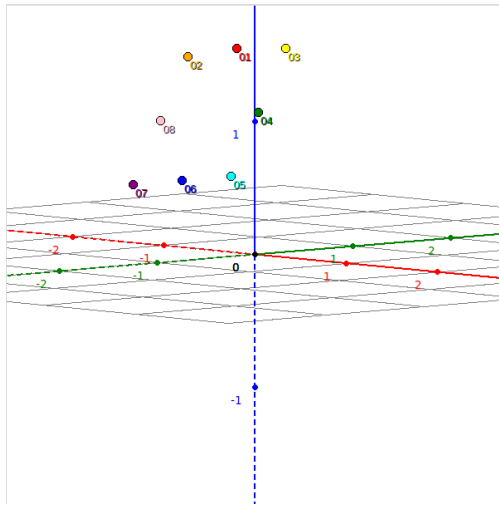
- **Drones flying above others introduce severe instability due to aerodynamic downwash.**

The combination of a sloped visualization plane and a carefully planned intermediate transition layer provides a robust solution to this issue, enabling safe and

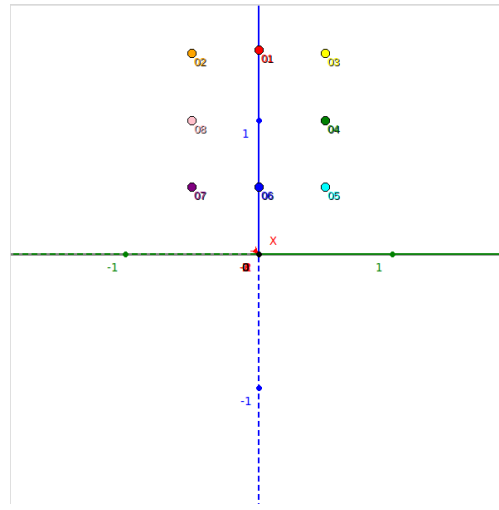
stable flight formations while maintaining the desired visual clarity of the letter formations.

7.8.5 Intermediate Transition Plane for Safety

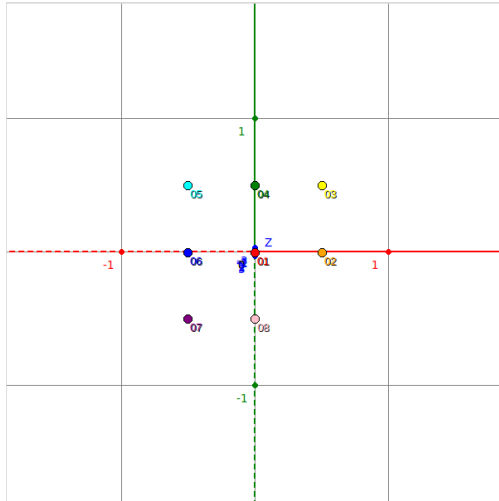
To enhance safety during transitions between letters, an intermediate step is introduced before each new letter is formed. This involves moving all drones to predefined intermediate positions located on a secondary plane behind the main visualization plane. By directing drones to this transition plane prior to entering their final visual positions, we reduce the likelihood of intersecting flight paths and collisions.



(a) Side view of the inclined transition plane

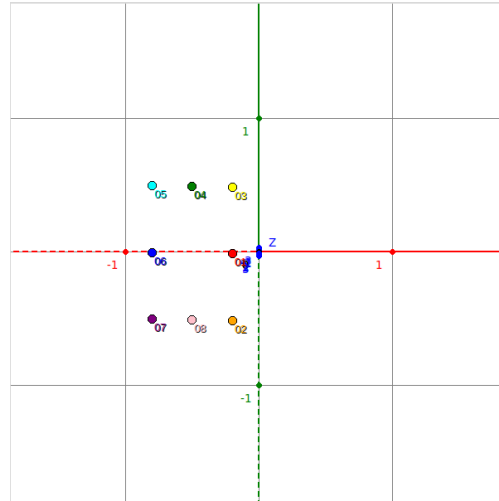


(b) Front view of the inclined transition plane



(c) Top view of the standard start position

$\Rightarrow$



(d) Top view of the inclined transition plane.

Figure 7.7: The inclined transition plane used as a safety measure for reduced drone path crossing. As shown by the last two subfigures, Figure 7.7c and Figure 7.7d, is the transition path from the start position to the inclined transition plane without crossings for 8 drones.

This intermediate step also introduces flexibility: drones can arrive at the intermediate plane with slight delays or follow diverse trajectories, as long as they reach their staging point within an acceptable time window. This spatial decoupling not only improves safety but also creates a moment of suspense in the choreography, as the letter is not revealed instantaneously but emerges progressively.

For all drones to be in the same plane, must the top and bottom nodes' offset in the x-direction not be dependent on the number of nodes above each other in the vertical direction. If the downwash is to be approximated by going straight down, would the same margins apply for this offset as if the drones were side by side. It is this assumption which will be used. Then, each layer of drones in the vertical direction needs to be offset a minimum of 20 cm. The maximum amount of node layers in the vertical direction is four Figure 7.5, and such to have all nodes in the same plane, must the top node be offset 0.6 m from the bottom most drone, regardless of the amount of drones in the specific letter.

7.8.6 Using the path planner to define paths forming letters

The framework will now be tested on performing part of the big long-term goal: To shape out letters with the simulated drone flight.

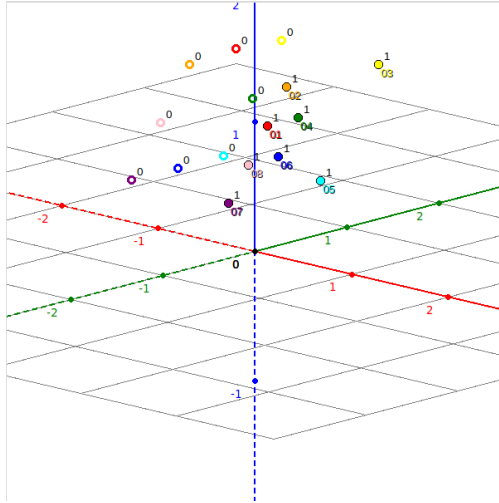
Designing the letters require some manual labor: Finding the coordinates of each drone, now also satisfying the sloped visualization plane, require some calculations. These were done by hand, but follow the same pattern: Find the size of the letter, then use interpolation to find each drone-waypoint's location. In the case of the letter "N" are the positions as follows:

```
DroneID: list(coordinates)
01: [0.6, -0.166, 1.166, 5.0]
02: [0.8, -0.5, 1.5, 5.0]
03: [0.8, 0.5, 1.49, 5.0]
04: [0.5, 0.5, 0.99, 5.0]
05: [0.2, 0.5, 0.49, 5.0]
06: [0.4, 0.166, 0.833, 5.0]
07: [0.2, -0.5, 0.5, 5.0]
08: [0.5, -0.5, 1.0, 5.0]
```

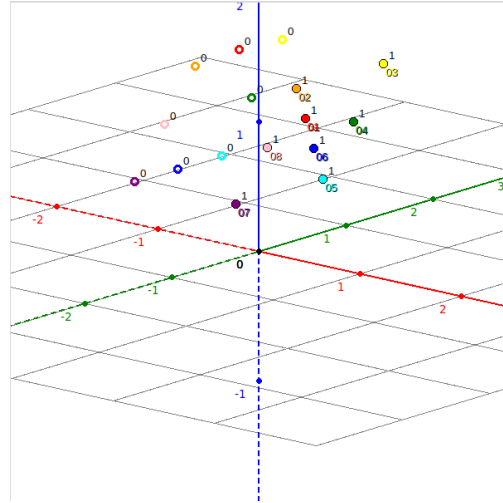
Listing 5: The drone coordinates for the letter "N" taking into account correct spacing in all directions

In the file structure, the "N" position data is saved separately in files named after the droneID, but is collected here for demonstration purposes.

When the points are collected, do they form points as shown in Figure 7.8



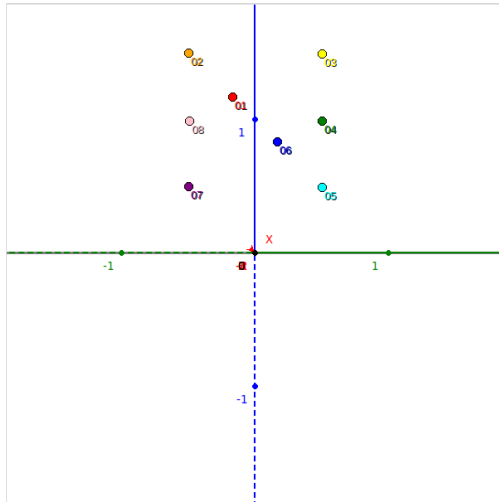
(a) "K" letter path with intermediate transition step



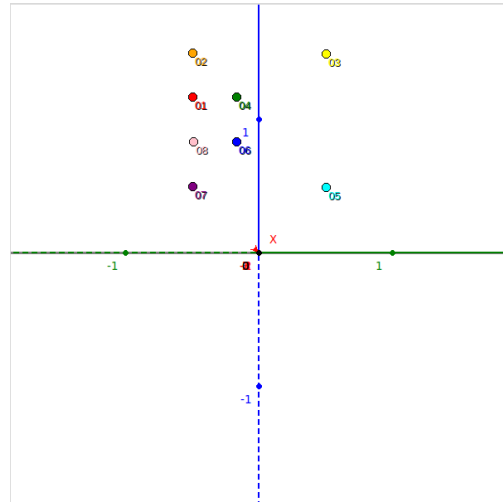
(b) "N" letter path with intermediate transition step

Figure 7.8: Two letter paths which should work to form the "N", which represents two out of four letters of "NTNU", and is the most complex out of them. Also being able to transform into a different letter, in this case "K"

As the audience won't see the waypoints, and for clarifying the look of the lettering, are the final positions of each letter also included in Figure 7.9.



(a) The Simulation representation of the letter "N" configuration, which is sent to the drone swarm directly



(b) The Simulation representation of the letter "K" configuration, which is sent to the drone swarm directly

Figure 7.9: The two letters formed by the simulated drones.

With this, the path planning component of the framework is complete and functional, allowing for the writing of letters with the newly created Drone font. The letters were spelled out without the collision danger detection going off, meaning the drones should be safe from collision, as long as there are no additional problems, such as positional estimation inaccuracy. Tweaks can now be done to improve the path planner, but regarding the applied functionality, this part is complete.

7.8.7 Possible improvements to the path planner

Although spelling out the letters, with collision detection in the simulation, is now possible thanks to the functionality of the path planner, it is still quite time-consuming to define complex paths like letters. A possible upgrade to the functionality toolbelt could be the ability to specify the n -th waypoint for all drones in a definition session, rather than being forced to define the entire path for each drone individually. This would drastically increase usability and intuitiveness, as it would enable one to see the whole drawn shape while working with it. To do this, the point definition canvas could be included in the swarm simulator GUI, which would make it similar to the other GUIs, having one for defining and one for visualization and simulation.

Another piece of functionality which could be nice, but which require much work, is the ability to abstract the definition even further: The user specifies the letters which is wanted, and the drone figures out where the letter needs to go in the space to account for the other letters, which drones correspond to which letters etc. As the creator of the path planner, I would recommend against making this. The reason for this is that there is probably not enough usage of either solution to be able to justify spending countless hours making this abstraction.

7.9 Requirements for Supporting New Drone Platforms

While the current software is tailored to the Crazyflie platform, it is designed to be extensible to support other drones by integrating appropriate interface layers. An example of a much used and highly precise manufacturer are the DJI camera drones. DJI offers robust waypoint-based mission control through its Onboard Software Development Kit (SDK) and Mobile SDK, which support waypoint missions with advanced features such as curvature flight, speed control, and embedded actions like photo capture [58].

Although DJI's waypoint paradigm aligns conceptually with the (x, y, z, t) model used here, their APIs introduce additional parameters such as heading, gimbal orientation, and action sequences. To integrate DJI drones, our software should include a translation module that converts generic waypoint entries into DJI-specific mission commands using the available SDKs. This allows the rest of the software, including the GUI and visualization engine, to remain unchanged but this might leave performance on the table. Since these drones are very advanced, might an approach using Robot Operating System (ROS) or a complete software be the better approach. This is however out of scope for this thesis to discuss.

Communication methods also differ. DJI platforms use proprietary radio protocols, Lightbridge, or Wi-Fi, and identify devices through serial numbers or network-based methods. Supporting DJI drones would therefore also require modifying the communication layer of the planner to accommodate these protocols [59].

Other drones such as the DJI drones can be integrated by:

- Developing an adapter that maps internal waypoints to DJI-compatible commands.
- Supporting alternate communication protocols.
- Accounting for SDK availability and file-based mission formats.
- Change of localisation system, or integrating it with the old. It is hard to determine how much work that would be.

This approach enables our system to remain modular and extensible to both consumer and enterprise-level drone platforms.

7.10 Creating an interface between the old and the new

To ensure reusability and scalability of the path planner, it is recommended that any new drone platform be interfaced in such a way that the same internal waypoint model can be preserved. This means developing a translation layer or adapter module that converts the generic waypoint data into the specific command format required by the new drone. By doing so, the existing software architecture can remain largely unchanged, maintaining its core logic, user interface, and visualization system.

In most realistic scenarios, the waypoint-based trajectory model will still be conceptually valid. The primary requirement will be implementing the appropriate logic to send these commands in a form compatible with the new drone. Therefore, the waypoint functionality does not need to be replaced, but rather extended to accommodate translation to new command formats.

Another area likely to require modification is the communication system. Each drone platform may use a different protocol or identification system. For example, the framework identifies the drones with a two-character radio ID which it fills into a hard-coded URI for swarm communication. If a new platform uses a different method such as Wi-Fi, Bluetooth, or another RF protocol, then the identification and addressing system will need to be updated. This again calls for a well-structured interface module that abstracts communication details from the rest of the software, allowing the core planner to remain agnostic to how the messages are delivered.

SYSTEM INTEGRATION AND TESTING

System integration of all of these parts is crucial to the usefulness of the framework. It is together that they do something spectacular. Although the parts were designed to function together, they were made individually. System integration is always a hard task, especially when integrating multiple pieces of hardware. I do, however, want to take a moment to drive this point home: It is the fact that the framework is so subdividable into parts that makes it possible to work with and test out. It will also allow for future development.

8.1 Integrating the swarm functionality with the old GUI framework

To be able to integrate the old framework had the drone controller to be updated with the swarm based control, and integration with the new path definition needed to be sorted out.

The script which handled the demonstration framework had to be updated to take in the paths, and as a consequence had this program to be run through the simulator. The added bonus of which was that the operator had incentive to hit "animate" to test the path planning before using it. The alternative of which would be to run the script, and select a file directly. This however would not enable mapping of the paths to the right commands, which is also necessary for use with the headset, and for paths which may change depending on the desired outcome.

```
def run_single_drone_demonstration():  
  
    # was changed to  
  
    def run_swarm_drone_demonstration(drones: list[Drone],  
    ↪   clf_filename: str, log_folder_name: str = ""):
```

Listing 6: The two different versions which start the demonstration, old and new. The new takes in parameters for path, EEG data classifier, and logging

The swarm controller was used from Bitcraze's API, and works very similar to the previous system. The only difference is that it uses a command to parallelize the commands, which opens a separate working thread for each drone, to send separate commands to each drone.

```
def move_with_commands(scf, command_queue: queue.Queue,
    ↪ exit_event: threading.Event, test_mode: bool):
    ...
    with MotionCommander(scf, default_height=DEFAULT_HEIGHT)
    ↪ as mc:
        print("MotionCommander started with drone connected.
        ↪ Awaiting commands...")

        while not exit_event.is_set():
            try:
                command = command_queue.get(timeout=0.1)
                print(f"Received command: {command}")

                # Execute commands based on input
                if command == COMMAND_FORWARD:
                    mc.forward(0.2)
            ...
```

Listing 7: The old way to handle commands where there is only one drone [1]

```
def run_swarm_with_commands(drones: list[Drone], command_queue:
    ↪ queue.Queue, exit_event: threading.Event, test_mode: bool =
    ↪ False, log_folder_name: str = ""):
    """
    Run the swarm continuously with the given drones and command
    ↪ queue.
    Args:
        drones (List[Drone]): A list of Drone class instances
        ↪ representing the drones in the swarm.
        command_queue (queue.Queue): A thread-safe queue for
        ↪ passing commands to the swarm.
        exit_event (threading.Event): An event used to signal
        ↪ when the swarm should stop running.
```



```

    test_mode (bool, optional): A flag to indicate whether
    ↪ the function is running in test mode. Defaults to
    ↪ False.
Returns:
    None
"""
if drones is None or test_mode:
    while not exit_event.is_set():
        try:
            command = command_queue.get(timeout=0.1)
            print(f"Simulated command: {command}")
            if command == "exit":
                exit_event.set()
        except queue.Empty:
            continue
else:
    uris = get_radio_uris(drones)
    cflib.crtp.init_drivers()
    swarm = connect_swarm_with_cache_check(uris)
    if swarm is not None:
        with swarm as swarm: # Do this to trigger enter and
            ↪ exit methods
            # Activate logging for position and TDoA
            log_files = {}
            if log_folder_name != "":
                today = datetime.now().strftime("%Y%m%d")
                full_folder = os.path.join("./data/logs",
                    ↪ f"{today}_{log_folder_name}")
                os.makedirs(full_folder, exist_ok=True)

            for scf in swarm:
                uri_safe = scf.cf.link_uri.replace("/",
                    ↪ "_")
                log_path = os.path.join(full_folder,
                    ↪ f"log_{uri_safe}.csv")
                file_handle = open(log_path, "w")
                log_files[scf.cf.link_uri] = file_handle
                setup_logging(scf, file_handle)
            print(f"Logging enabled for {len(log_files)}
                ↪ drones in {full_folder}")

    # Arm and take off
    swarm.reset_estimators()
    swarm.parallel_safe(disable_light_ring) #
    ↪ <- Always off
    swarm.parallel_safe(deactivate_led_bit_mask) #
    ↪ <- Turn off front LEDs before takeoff

```

```

swarm.parallel_safe(arm)
swarm.parallel_safe(take_off)
print('Takeoff done, entering command loop')
while not exit_event.is_set():
    try:
        command = command_queue.get(timeout=0.1)
        print(f"Received command: {command}")

        ↪ swarm.parallel_safe(deactivate_led_bit_mask)
        ↪ # turn off lights
    if command.isdigit():

        set_all_selected_paths(drones,
            ↪ int(command))
        seq_args =
            ↪ get_radio_uris_and_points(drones)

        ↪ swarm.parallel_safe(deactivate_led_bit_mask)
        ↪ # Ensure lights are off before
        ↪ sequence
        swarm.parallel_safe(run_sequence,
            ↪ args_dict=seq_args)

        ↪ swarm.parallel_safe(activate_led_bit_mask)
        ↪ # Turn on lights after path
        ↪ following

```

Listing 8: The new way to handle commands with logic to send paths based on the incoming command

Other than that, the system integration was made simple because the framework has been developed to work seamlessly from the start. This is, in turn, one of the reasons why the framework was developed from building blocks instead of using a commercially available product.

8.2 Developing for a future-proof framework

From the get-go, the framework has been developed to work with a specific set of hardware: the Crazyflie, the Loco positioning system, and the OpenBCI headset. This means that the solution has been specifically developed for this use case. This is simultaneously the framework's greatest strength and its greatest potential weakness. It has, therefore, been in focus throughout the working period and also discussed in all parts of the thesis regarding the next steps for the hardware and how the software is affected by this. In all parts, the conclusion has been that this future-proofing is something which has been successfully done:

- The BCI can be switched out without affecting the usability of the rest of

the system because the keyboard remains a backup input solution.

- Changing the BCI requires work on:
 - A new datastream interface, taking into account the data structure, frequency, etc.
 - Adjusting the recording protocol and data storage
- Changing the positioning system requires the drone to be able to interact with it, and a new stand holder part out of 3D print, or drilled directly into the wooden stand.
- Changing the drone would prove to be the biggest challenge, since the positioning system needs to work with it. In most cases, a change in the drone would also require a change in the positioning system, as the existing positioning system is custom-made for the Crazyflie drone. On the software side, it is relatively straightforward, requiring an interface for the new queuing system.

That being said, as a modular framework divided into individually functioning parts, most of the framework itself can probably be reused with minimal effort.

8.3 Implementation of safety measures

In a complex framework, it is increasingly important to have safety systems in place to identify as many issues as possible before launch. The systems need to be implemented, balancing increasing complexity and runtime, while ensuring safety. Some safety measures taken in this framework include: safe code with checks, an emergency stop built into the commander, nets around the flying area, and the use of a backup controller, such as a keyboard. This has been a big focus of the previous project [1]. Here, the second controller was introduced, and the `exit_event()` was implemented as an emergency stop. However, with the increase in drone numbers, and therefore also risks, there should be a safety protocol in place to keep both the user and the potential audience safe. This protocol has been added as an appendix to the thesis and should be followed while using the framework for demonstrational and experimental purposes.

8.4 A test of using the framework

All the framework components have now been implemented and integrated. In the following trials, the goal is to test various aspects of the framework's functionality, culminating in the final tests where all components are integrated and everything must work together for the system to function correctly. Each step of the way will help understand a piece of the framework and has been designed to test specific things. Ultimately, the swarm will hopefully be able to fly in formation and form letters. Here, the drone font, which was developed in Section 7.8.3, will be used for the first time ever.

It should also be noted that the tests presented here document some of the first tests conducted with the framework, enabling the identification of the steps required to implement the framework itself. Some of the resulting ideas have therefore been discussed in earlier chapters in the individual parts.

All the tests presented here were recorded, and is a part of the demonstrational youtube video at this link:

<https://youtu.be/DwOjBqX0rrw>

8.4.1 Basic test with one drone through the swarm controller

Since the earlier version of the framework developed the code needed to fly a drone, this test is tailored to testing the capabilities of flying, still one drone, but through the swarm framework. Although the apparent functionality would seem the same, many things have happened in the background. It is now using the swarm controller and is in reality a swarm of $n=1$ drones.

The first test will see if the new drone interface to the swarm architecture behaves like the simulated path planning architecture, and if the controller is still thread-safe, meaning the program is not locked up when the drone is flying. This was a problem in earlier iterations of the project when implementing new hardware. In addition, this test will be the first where multiple paths are executed in series, so it will also test if this functionality works as designed.

8.4.1.1 The result of the test: Success!

The drone flew as expected with the new swarm controller. It functions as before, still being able to perform all the commands from the previous iteration [1], but now also with the ability to follow paths. The drones were able to follow both paths in sequence with no issue.

The GUI is also still operational, and the program shuts down as expected, indicating that the program remains thread-safe. This was also a common problem with earlier iterations. The program finished with no errors.

8.4.2 Dual path flying sequence with 2 drones with reduced collision danger due to separated paths

Now, this is the first test including more than one drone. To approach this slowly, the test was conducted with theoretically completely independent paths, where the drones were always 1 m apart, overshooting the recommended distance many times over. In practice, there should be no real collision danger as long as the drones are placed correctly in their starting positions and the program works as planned.

From this test, the first results on how much the drones' aerodynamic turbulence affects each other will be revealed, and the controller will be tested to see if

it can operate multiple drones while still being thread-safe.

8.4.2.1 The result of the test: Success!

Both drones flew without issue. It seems that having the drones as far as 1 m apart eliminates any problems caused by turbulence. The program remains thread-safe, even when controlling multiple entities.

8.4.3 Dual path with 2 drones, flying above one another

The drones affect each other quite little while being apart horizontally, but vertically, the situation is quite different. The downfall of air coming from the thrust of a drone above might affect the stability of a drone flying below it. This first test will test with a far distance of 1 m apart in the vertical direction.

8.4.3.1 The result of the test: Fail

The air vortexes from the top drone change the nature of the air around the bottom drone to such a degree that the lower drone becomes completely unstable and crashes.

8.4.3.2 Suggested improvement: Introduce a slight offset in x-direction

To be able to fly drones above each other seems unlikely. But it is not completely necessary for our use case to have the drones exactly above each other. The plan is to have all the drones facing in the positive x-direction, and the audience, as a consequence, facing in the negative x-direction. Introducing an offset in the x direction, meaning that the shapes are "drawn" on a plane that is slightly tilted towards or away from the audience, seems plausible and would not significantly affect the visual appearance of the shapes. It is therefore suggested to rerun the test with this change.

8.4.3.3 The result of the test: Success!

This worked much better, although from the video, it is possible to see and hear when the drones align vertically, from the unstable nature of the drones, which causes vibrations. The applied offset was 20 cm, which is recommended for all drones. Additionally, the plane is designed to have the least adverse effect on the visual appearance if the top of the drone is sloped towards the audience. A higher drone will therefore be farther along in the positive x direction. When the drones are in the intermediate-passive plane, their positions should be sloped in the same direction.

8.4.4 Testing the use of random headset inputs with a dual path flying sequence using one drone

Since the live headset classification is not yet working correctly, testing with the headset will unfortunately be impossible. However, using the headset as a command source is far from impossible; to the contrary, it is a crucial step in testing

the effectiveness of the system integration. To clarify, this test will be conducted using completely random inputs to the drone swarm, which will be generated by the headset.

Since the headset commands are what needs to be tested, this test will use the two drones using the first dual-path drone case with completely separated paths, to limit the risk of crashing should something happen. Everything else remains the same.

This test will therefore verify that the recording and headset command thread functions correctly and that they are thread-safe.

8.4.4.1 The result of the test: Fail

In the sense that the flying was unsuccessful. However, the swarm receives commands from the headset and translates them to paths, executing different paths depending on the radio ID. The only issue came from the drone with ID 02 flipping early into the path.

The drone was then replaced to determine if there was a hardware issue, and the test was rerun; however, it still failed.

After some trial and error, the issue ended up being that to capture this on film, the author walked inside the net, and the line of sight to some of the positioning nodes was therefore obstructed. After becoming aware of the issue, the swarm flew 10 commands in a row without any hiccups!

However, the command queue quickly became backlogged. This means nothing for this test, since the only requirement was to receive any input from the headset at all. However, for a user, it would be both unintuitive and impractical for any application to have a command queue backlog like this. This is because the current case is that all commands executed within the first minute are collected within the first few seconds, introducing a progressively larger time delay to the system, depending on the execution times of the path. This, in turn, makes the command system less and less "real-time" controllable. Having real-time control is crucial to the framework's long-term goals, so a solution was implemented: a state machine that allows the headset command from the prediction algorithm to enter the command queue only if the command queue is empty. This implementation is illustrated in Figure 5.1.

The backloging issue also tested an essential safety system of the framework: the `exit_event()` was triggered even when the command queue was backlogged, which is the intended functionality of the `exit_event()`. It does not, however, work mid-path. This might be for the better, however, since the drones may crash if trying to land mid-path due to this landing timing not being simulated in the GUI beforehand. An example of this is when drones are passing above each other or very close to each other, when projected onto the horizontal plane. When the drones are landing, they could then crash. A true emergency stop, which shuts

off all propellers of all drones, would be advisable, and should be implemented as a complementary safety mechanism. This is outside the scope of the current project, as it requires direct connection with the drones' firmware.

8.4.5 Trying to set up a dual path flying sequence with 8 drones in a confined space using keyboard inputs N -> K path with keyboard commands

For the final test, it is time to see if multiple drones are able to fly in a very close proximity to each other. The test here will have eight drones spell out two different letters: an "N" which transforms into a "K". This will be the ultimate test to see if the setup works for spelling out words like "NTNU", or if some improvements need to be made. This test will test the stability of the drone swarm when flying many drones at once.

8.4.5.1 The result of the test: Fail

The drones are unstable early in the path sequence. Then some drones crashed, and the rest became much more stable. It therefore seems to be an issue of proximity, and that the drones destabilize the ones around themselves because of aerodynamic turbulence caused by having so many drones in such a small space.

The first path worked for almost many of the drones forming something which could remind of a letter. When trying to move to the next command, some of the lower drones lagged behind, which resulted in the drones above flying directly above them, which resulted in many of the lower drones crashing.

8.4.5.2 Suggested improvement: Increase travel time to reduce the likelihood of accidental downfall unsability

The test will be rerun, but now with double as long travel time between waypoints.

8.4.5.3 The result of the test was: Fail

Although this might have helped slightly, individual randomness between runs will matter more for the observed differences than the actual improvement. The improvement is within the margin of error, so determining if this helped is not possible without much more data.

Nevertheless, the problem lies in the turbulence created around the drones. It might not be feasible to spell out letters as proposed without significant improvements to the stability and precision of the drone positioning estimate.

8.4.6 Framework test conclusion

Although not all the tests were successful, even after tweaking, the framework as a whole is definitely a success. It handles all the system integration flawlessly; the issues lie in the drone's stability from the positioning measurements and controller. These are issues that would be fixed as the next step in the development

process, but are out of scope for this thesis. To increase the performance of the entire framework, it is therefore strongly recommended to prioritize improving the drone’s stability and the position estimate in the upcoming projects.

Then, if it is determined that the hardware is incapable of achieving the performance that is necessary, it is recommended to switch the hardware. This goes for both the positioning system and the BCI, should the achievable classification accuracy be the issue.

8.5 Multi-Pilot Control of Drone Swarms Using Dual BCIs

The following section will discuss the potential for implementing dual pilot control of the swarm. This is an exciting research opportunity, and would also increase the intuitiveness of the difficulty, and give the audience something to compare the task to, because the pilots would be compared to each other. It has not been within the scope of this thesis to develop this functionality, but it has nonetheless been taken into account that it should be made possible in the future. Exactly how will be discussed here.

8.5.1 Motivation and Conceptual Benefit

An experimental extension of the existing BCI-controlled swarm system involves introducing **two human pilots**, each with their own EEG-based BCI interface. This approach opens up opportunities for more nuanced motor imagery (MI) classification and enhanced interactivity, while also introducing considerable architectural and safety challenges.

In a demonstrational setting, there is not much that assures the audience that the BCIs are actually used to control the swarm movements. Therefore, the use of two pilots to control a drone swarm could improve the believability of MI classification. Observers might find the system more intuitive and credible if they can visually detect differences in control between the two pilots, especially if one pilot performs better than the other. This division also adds an element of competitive or cooperative dynamics, potentially increasing engagement and research interest.

8.5.2 Swarm Division Strategy

To implement dual control, the swarm must be divided into two distinct and independently controlled groups. This would, for example, be the left and right halves of the full formation. Each pilot would then control one half of the swarm. In the software architecture, this could be handled by paths that issue commands selectively to either half of the swarm. Drones in the non-targeted half would simply idle until they receive a relevant command. Since inactive drones remain stationary and responsive, the system remains modular and scalable. This functionality was considered when creating the path planner, and it works out of the gate.

8.5.3 Collision Avoidance and Spatial Separation

One major safety concern arises from the asynchronous behavior of the two drone groups. If each half of the swarm operates independently, the drones may be at different stages in their path sequences, creating a risk of collision. To mitigate this, it is proposed that the spatial flying volumes of the two half-swarms be separated. For example, by assigning non-overlapping left and right halves of the physical airspace. This ensures that drones from one half do not interfere with those from the other, even under asynchronous operation, which increases operational safety.

8.5.4 System Architecture Modifications

Implementing dual control would require several changes to the existing system architecture:

- **Two EEG Recording Threads:** One for each pilot's headset, running in parallel to capture real-time brain activity independently.
- **Two BCI Classifier Pipelines:** Each pilot's signals must be processed and classified separately.
- **Dual Command Queues:** Separate input queues ensure isolated control logic and prevent race conditions. Mutual exclusion mechanisms (locks) must be applied to each queue independently.
- **Keyboard Interface Adaptation:** The manual keyboard input system could be extended to send distinct commands to either half-swarm. Shared commands such as `STOP` or `RESET` may still be broadcast to both simultaneously.

8.5.5 Feasibility and Workload Evaluation

The concept of dual BCI control is feasible and may offer new experimental insights. The necessary modifications span software architecture, command routing, and safety evaluation. Furthermore, ensuring robust synchronization and spatial isolation between the two swarms will require careful path planning and testing.

Overall, the approach is promising but demands substantial development work. If resources and time allow, this is an interesting direction to take the project down the line. The modularity of the current system, especially in terms of command dispatch and swarm partitioning, does support such an expansion with minimal rewrites of the core logic.

DISCUSSION

- **Reflections on system usability and practicality**
The custom GUI is flexible but requires user training; EEG mapping works reliably for two classes but struggles with more complex commands.
- **EEG limitations and feasibility for real-world swarm control**
Motor-imagery speed and accuracy trade-offs limit rapid command sequences.
- **Loco Positioning limitations: latency, noise**
Achieved mean localization error of X cm (see Results chapter??????); anchor tilt and environmental interference remain challenges.
- **Using it for demonstrational purposes**
A polished GUI and prerecorded flight patterns serve as robust demos, even under variable lab conditions.
- **Potential improvements:**
 - Real-time collision avoidance via onboard sensors
 - More intelligent EEG parsing (deep learning classifiers)

9.1 Loco Positioning System Performance

The development and deployment of the Loco Positioning System in this project highlight both the potential and the inherent limitations of UWB-based localization for drone swarm control. The decision to use the Loco system was driven not only by accuracy considerations, but also by its scalability, cost-effectiveness, and suitability for mobile setups. While UWB cannot match the sub-millimeter precision of optical systems, it aligns with the long-term goal of enabling demonstrations and research beyond fixed laboratory environments.

Although the achieved positional accuracy fell slightly short of the specifications stated by Bitcraze, it proved sufficient for coordinated multi-drone flight when paired with appropriate safety margins and outlier rejection techniques. A systematic evaluation of spatial precision—using stationary logging, 3D interpolation, and visual analysis—demonstrated the reliability of the framework and revealed areas where improvements are most impactful. For example, degraded

accuracy near the floor was consistently observed and confirmed through visualization, while the majority of the operational flight volume remained within expected tolerances. These results can serve as guidance for drone placement, path design, and potential use in collision avoidance strategies.

One of the system’s major strengths is its modular and reconfigurable infrastructure. The use of wooden node stands and 3D-printed mounts enabled rapid and repeatable deployment in varied environments, supporting flexible experimentation and scalability.

Despite these successes, several areas for improvement remain. Enhancing the performance could involve further tuning of the outlier and Kalman filters, as well as the controller gains to better adapt to the noise characteristics of each environment. Exploring alternative anchor configurations or expanding to TDoA3 with more than eight anchors may also yield improved accuracy. Ultimately, this highlights the importance of a holistic approach where hardware layout, filtering, and control robustness evolve together to ensure a stable and safe swarm system.

9.2 Path Planning reflection

The path planning module developed for the Crazyflie swarm demonstration system provides a graphical interface for defining, editing, and managing 3D waypoint-based trajectories. It leverages the Crazyflie’s native waypoint interface, which allows the user to specify 3D coordinates alongside timing information that determines how long each drone should take to reach the next point. These waypoints are interpolated onboard the drone for smooth transitions.

The architecture of the path planner was built using object-oriented programming principles. This structure proved advantageous throughout development, especially since both graphical user interfaces (GUIs) shared a common base class. This made it easy to extend features across both canvases and utilize the canvas manager for more complex interactions.

Informed by the measured positioning precision, the planner enforces spacing standards between drones to avoid collisions. Several standards were defined: the starting position layout, a custom drone-shaped font for spelling out letters, and transitional buffer points for safe maneuvering. These conventions contributed significantly to minimizing crashes during testing.

9.3 System Integration and Testing reflection

Once individual components were functioning independently, integrating the full system proved straightforward. Replacing the previous command interface was the most significant task. However, the existing single-drone control system provided a reliable fallback during the transition, allowing for incremental testing of communication protocols and hardware interaction. This approach ensured a smooth integration process and resulted in a cohesive final system.

9.4 Usability and Practicality

The practicality and usability of the system were continuously evaluated throughout the project. While the path definition interface performed well, it is not yet intuitive enough to be used independently without prior planning or assistance. A key limitation is the inability to define new paths while viewing those of other drones simultaneously. As a result, a planning phase—where paths are sketched out and coordinated beforehand—is essential for safe and effective usage.

Nevertheless, once the system is understood, it performs well in practice. The framework is relatively quick to set up and operate, and its modularity supports flexible experimentation and adaptation. These qualities make it a promising platform for further research and educational demonstrations.

CONCLUSION

The thesis has focused on making a good foundation for future research within EEG BCI-controlled drones here at NTNU. To do this, the OpenBCI Ultracortex headset was used along with the Crazyflie 2.1 drone and the Loco positioning system, both supplied by Bitcraze. From there a physical infrastructure was created that enabled providing a positional estimate to the drones. The setup is lightweight, easy to move with its foldable back-brace, and with the Loco positioning system, delivers precise enough to position estimation to achieve near producer-specified performance. Even so, the positioning system had too low precision to perform as well as hoped with the dense drone swarm. Still, the physical infrastructure is seen as a success. This, along with the nodes being supplied with power from power banks located on the stands, and the rest of the hardware being wireless, is the framework mobile and agile. The framework has succeeded in creating a very easily modified and moveable setup which will hopefully work in a variety of environment.

Two Graphical User Interfaces were created to enable the definition, visualization, and detection of collision dangers in the path planning stage. It is easy to use, and integrates seamlessly with the previously made main menu program.

The positioning setup, with its swarm control interface, and the path planner, is made with great modularity in mind. Most parts are evaluated to be reusable, either in the case of switching some hardware out or if tweaks are performed to improve the framework. The cases of switching to different hardware has been evaluated to be feasible, with many of the components still usable with minimal changes. This, along with providing a User Guide, enables the future of research to work from the current state of the framework easily.

Introducing two pilot control through two separate EEG BCIs, introduces development of the codebase and facilitate long-term progress.

10.1 Future work

Throughout the thesis has multiple areas of possible improvement and possibility of future development been mentioned. Here are some interesting possibilities for

continuation of the work:

Implement the ability to see live position data in the simulator with online collision danger detection and/or with headset actuation for a drone-out-of-loop online testing possibility with visual feedback.

Tune the controller and Kalman filter estimation to improve flying stability in the swarm.

Explore techniques to reduce the effect of downwash and in turn increase stability. For example downwash aware control/modelling or more sophisticated downwash avoidance.

Implement classification and use of EEG data in real-time control of the headset through proper task synchronization.

Implement a dual-pilot control scheme where the two pilots need to cooperate to achieve the correct drone patterns.

Explore different drone visualization possibilities like

REFERENCES

- [1] Daniel Nicolai Grønn. *Imagine and Fly: A Brain-Computer Interface for Motor Imagery Drone Control*. Specialization project in Cybernetics, Norwegian University of Science and Technology. Supervisor: Marta Molinas, Co-supervisor: Luis Alfredo Moctezuma Pascual. Dec. 2024.
- [2] Kjersti Brynstad and Erlend Vatsvåg. “An Asynchronous Motor Imagery-based Brain-Computer Interface for Two-Dimensional Drone Control”. Master’s Thesis, Department of Engineering Cybernetics. MA thesis. Norwegian University of Science and Technology, 2021.
- [3] Helene Tørle Lønvik and Pauline Mørch Jonassen. “An Exploration of Techniques for Electroencephalography-Based Motor Imagery Classification for Real-Time Drone Control”. Master’s Thesis, Department of Engineering Cybernetics. MA thesis. Norwegian University of Science and Technology, 2023.
- [4] Arrigo Palumbo et al. “Motor-Imagery EEG-Based BCIs in Wheelchair Movement and Control: A Systematic Literature Review”. In: *Sensors* 21.18 (2021), p. 6285. DOI: 10.3390/s21186285.
- [5] Jianchao Lu et al. “LGL-BCI: A Motor-Imagery-Based Brain-Computer Interface with Geometric Learning”. In: *arXiv preprint* (2025). arXiv:2501.05589.
- [6] Jianan Chen et al. “fNIRS-EEG BCIs for motor rehabilitation: A review”. In: *Bioengineering* 10.12 (2023), p. 1393. DOI: 10.3390/bioengineering10121393.
- [7] Dae-Hyeok Lee et al. “Design of an EEG-based Drone Swarm Control System using Endogenous BCI Paradigms”. In: *Proceedings of the 9th IEEE International Winter Conference on BCI* (2021). DOI: 10.1109/BCI51272.2021.9385356.
- [8] Sung Jin Kim, Dae Hyeok Lee, and Seong Whan Lee. “An EEGgram-based Neural Network Enhancing the Decoding Performance of Visual Imagery EEG Signals to Control the Drone Swarm”. In: *2022 IEEE International Conference on Systems, Man, and Cybernetics*. 2022, pp. 2281–2286. DOI: 10.1109/SMC53654.2022.9945460.
- [9] K. LaFleur et al. “Quadcopter Control in Three-Dimensional Space Using a Noninvasive Motor Imagery-Based Brain-Computer Interface”. In: *Journal of Neural Engineering* 10.4 (2013), p. 046003. DOI: 10.1088/1741-2560/10/4/046003.

- [10] Sumit Kumar Ram. *Experience designing a BCI for flying a drone based on the use of Empirical Mode Decomposition*. Tech. rep. Project report, Faculty of Information Technology, Mathematics and Electrical Engineering, NTNU. Trondheim, Norway: Norwegian University of Science and Technology, Department of Engineering Cybernetics, Nov. 2016. DOI: 10.13140/RG.2.2.16858.13761.
- [11] Kobe Christensen. “Postsynaptic Potentials (PSPs): The Source of EEG Signal”. In: *Advanced Neuroscience (Pressbooks)* (2025). EEG signals are generated by a complex series of synchronized postsynaptic potentials.
- [12] Various. “Introduction – Electroencephalography (EEG): An Introductory Review”. In: *NCBI Bookshelf* (2018). EEG records synchronous postsynaptic potentials from large cortical pyramidal neuron populations.
- [13] S
ndor Beniczky and Donald L. Schomer. “Electroencephalography: basic biophysical and technological aspects important for clinical applications”. In: *Epileptic Disorders* 22.6 (2020), pp. 697–715. DOI: 10.1684/epd.2020.1217.
- [14] Zahra Khademi, Farideh Ebrahimi, and Hussain Montazery Kordy. “A review of critical challenges in MI-BCI: From conventional to deep learning methods”. In: *Journal of Neuroscience Methods* 383 (2023), p. 109736. DOI: 10.1016/j.jneumeth.2022.109736.
- [15] Sergio Perez-Velasco, Eduardo Santamaría-Vázquez, et al. “EEGSym: overcoming inter-subject variability in motor imagery based BCIs with deep learning”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* (2021). Introduces a DL architecture (EEGSym) to address inter-subject variability in MI-based BCI.
- [16] Simanto Saha and Mathias Baumert. “Intra- and inter-subject variability in EEG-based sensorimotor brain–computer interfaces: A review”. In: *Frontiers in Computational Neuroscience* 13 (2020), p. 87. DOI: 10.3389/fncom.2019.00087.
- [17] Christopher J.C. Burges. “A Tutorial on Support Vector Machines for Pattern Recognition”. In: *Data Mining and Knowledge Discovery* 2.2 (1998), pp. 121–167. DOI: 10.1023/A:1009715923555. URL: <http://dx.doi.org/10.1023/A:1009715923555>.
- [18] R. A. Fisher. “The use of multiple measurements in taxonomic problems”. In: *Annals of Eugenics* 7 (1936), pp. 179–188.
- [19] H. Ramoser, J. Müller-Gerking, and G. Pfurtscheller. “Optimal spatial filtering of single trial EEG during imagined hand movement”. In: *IEEE Transactions on Rehabilitation Engineering*. Vol. 8. 4. 2000, pp. 441–446.
- [20] D. R. Cox. “The regression analysis of binary sequences”. In: *Journal of the Royal Statistical Society: Series B* 20.2 (1958), pp. 215–242.
- [21] A. Barachant et al. “Classification of covariance matrices using a Riemannian-based kernel for BCI applications”. In: *Neurocomputing* 112 (2013), pp. 172–178.

- [22] A. Barachant et al. “Riemannian geometry applied to BCI classification”. In: *International Conference on Latent Variable Analysis and Signal Separation*. 2010, pp. 629–636.
- [23] Kai Keng Ang et al. “A framework of common spatial patterns based on support vector decomposition machine”. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 3575–3580. DOI: 10.1109/IJCNN.2008.4620998. URL: <https://ieeexplore.ieee.org/document/4620998>.
- [24] F. Lotte and C. Guan. “Regularizing common spatial patterns to improve BCI designs: unified theory and new algorithms”. In: *IEEE Transactions on Biomedical Engineering*. Vol. 58. 2. 2011, pp. 355–362.
- [25] A. Subasi. “EEG signal classification using wavelet feature extraction and a mixture of expert model”. In: *Expert Systems with Applications* 32.4 (2005), pp. 1084–1093.
- [26] P. S. Addison. “Wavelet transforms and the ECG: a review”. In: *Physiological Measurement* 26.5 (2005), R155–R199.
- [27] Takaya Saito and Marc Rehmsmeier. “Index of Balanced Accuracy: A Performance Measure for Skewed Class Distributions”. In: *Proceedings of the 2015 European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 312–325. DOI: 10.1007/978-3-642-02172-5_57. URL: https://link.springer.com/chapter/10.1007/978-3-642-02172-5_57.
- [28] R. Kohavi. “A study of cross-validation and bootstrap for accuracy estimation and model selection”. In: *International Joint Conference on Artificial Intelligence*. 1995, pp. 1137–1143.
- [29] D. M. W. Powers. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”. In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63.
- [30] Bitcraze. *Crazyflie 2.X - A versatile open source flying development platform*. <https://www.bitcraze.io/products/crazyflie-2-1/>. Accessed: 2025-07-02. 2024.
- [31] Gabriel M Hoffmann et al. “Quadrotor helicopter flight dynamics and control: Theory and experiment”. In: *AIAA Guidance, Navigation and Control Conference* (2007), pp. 1–20.
- [32] Robert Mahony, Vijay Kumar, and Peter Corke. “Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor”. In: *IEEE Robotics & Automation Magazine* 19.3 (2012), pp. 20–32.
- [33] K.H. Ang, G.C.Y. Chong, and Y. Li. “PID Control System Analysis, Design, and Technology”. In: *IEEE Transactions on Control Systems Technology* 13.4 (2005), pp. 559–576.
- [34] H. W. L. Naus and Y. Toh. “Time difference of arrival localisation exploiting all available time differences”. In: *IET Radar, Sonar & Navigation* 14.6 (2020), pp. 847–853. DOI: 10.1049/iet-rsn.2019.0338. URL: <https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-rsn.2019.0338>.

- [35] everything RF. *What is Time Difference of Arrival (TDoA)?* [Online; accessed May 26, 2025]. 2023. URL: <https://www.everythingrf.com/community/what-is-time-difference-of-arrival-tdoa>.
- [36] Inpixon. *Time Difference of Arrival (TDoA) Multilateration*. [Online; accessed May 26, 2025]. 2023. URL: <https://www.inpixon.com/technology/standards/time-difference-of-arrival>.
- [37] Daniel Neunteufel, Stefan Grebien, and Holger Arthaber. “Indoor Positioning of Low-Cost Narrowband IoT Nodes: Evaluation of a TDoA Approach in a Retail Environment”. In: *Sensors* 22.7 (2022), p. 2663. DOI: 10.3390/s22072663. URL: <https://www.mdpi.com/1424-8220/22/7/2663>.
- [38] Simon D. Levy. *Extended Kalman Filter Tutorial*. <https://simondlevy.github.io/ekf-tutorial/>. 2023.
- [39] Bitcraze AB. *Explaining Kalman Filters with the Crazyflie*. <https://www.bitcraze.io/2024/01/explaining-kalman-filters-with-the-crazyflie/>. 2024.
- [40] D.J. Yeong et al. “Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review”. In: *Sensors* 21.6 (2021), p. 2140. DOI: 10.3390/s21062140.
- [41] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. 2009. URL: <https://docs.python.org/3/>.
- [42] Robert W. Sebesta. *Concepts of Programming Languages*. 10th. Addison-Wesley, 2012. ISBN: 9780131395312.
- [43] Scott Chacon and Ben Straub. *Pro Git*. 2nd. Apress, 2014. ISBN: 9781484200773. URL: <https://git-scm.com/book/en/v2>.
- [44] Krishna Kavi. “Multithreading Implementations”. In: *ResearchGate* (1998). URL: https://www.researchgate.net/publication/2734966_Multithreading_Implementations.
- [45] J. E. Grayson. *Python and Tkinter Programming*. Manning Publications, 2000. ISBN: 9781884777813.
- [46] Yajun Bu, Ye Yan, and Yueneng Yang. “Advancement challenges in UAV swarm formation control: A comprehensive review”. In: *Drones* 8.7 (2024), p. 320. DOI: 10.3390/drones8070320.
- [47] Justin Hu et al. “To Centralize or Not to Centralize: A Tale of Swarm Coordination”. In: *arXiv preprint* (2018). arXiv:1805.01786.
- [48] Wei Meng et al. “Advances in UAV Path Planning: A Comprehensive Review of Methods, Challenges, and Future Directions”. In: *Drones* 9.5 (2025), p. 376. DOI: 10.3390/drones9050376.
- [49] Nafis Ahmed, Chaitali J. Pawase, and Kyung Hi Chang. “Distributed 3-D Path Planning for Multi-UAVs with Full Area Surveillance Based on Particle Swarm Optimization”. In: *Applied Sciences* 11.8 (2021), p. 3417. DOI: 10.3390/app11083417.
- [50] OpenBCI LLC. *Ultracortex “Mark IV” EEG Headset*. <https://shop.openbci.com/products/ultracortex-mark-iv>. Accessed: 2025-07-03. 2025.

- [51] Bitcraze AB. *Loco Positioning System*. <https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/>. Accessed: 2025-06-29. 2024.
- [52] OpenBCI Documentation. *How OpenBCI Products Go Together*. 2025. URL: <https://docs.openbci.com/FAQ/HowProductsGoTogether/>.
- [53] OpenBCI Documentation. *Using SD Card with Cyton and CytonDaisy Board*. 2024. URL: <https://docs.openbci.com/Cyton/CytonSDCard/>.
- [54] Emotiv. *Technical specifications – Emotiv EPOC X*. 2025. URL: <https://emotiv.gitbook.io/epoc-x-user-manual/introduction/technical-specifications>.
- [55] Mentalab (MagstimEGI). *Mentalab Explore+ Low Density EEG*. 2025. URL: <https://www.magstim.com/product/mentalab-low-density-eeeg/>.
- [56] Mentalab. *Open-Source EEG Software Solutions for Mentalab Explore*. 2025. URL: <https://mentalab.com/eeeg-software/>.
- [57] Visual Paradigm. *UML Class Diagram Tutorial*. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>. Accessed: 2025-07-06.
- [58] DJI. *Motion Planning – Onboard SDK Documentation*. Accessed: 2025-07-06. 2022. URL: <https://developer.dji.com/onboard-sdk/documentation/tutorial/motion-planning.html>.
- [59] DJI Enterprise. *SDK Ecosystem for Enterprise Drones*. Accessed: 2025-07-06. 2024. URL: <https://enterprise-insights.dji.com/blog/dji-sdk-guide>.
- [60] Rudy Mens. *How to fix "Running Scripts is Disabled on this System"*. Accessed: 2025-01-05. 2022. URL: <https://lazyadmin.nl/powershell/running-scripts-is-disabled-on-this-system/>.

APPENDICES

PROJECT GITHUB REPOSITORY

All code used in the framework described in this report are included in the Github repository linked below. Access is gained from an admin of the wavesresearch GitHub organisation.

It has been a goal with this project to start a focus on creating a collected GitHub repository for continued development of the same codebase, to hopefully facilitate the project to progress well in the coming years. This means, as a developer of this framework, you should use the same repository, in different branches instead of making a new repository. If the functionality that is created can be merged into the main branch, then this is done at the end of that development period.

Github repository

- https://github.com/wavesresearch/drone_demo

USER GUIDE

This is the user guide to get all the way from novice to demonstrating the EEG-drone project. Hopefully it can be a good resource for experienced users as well. You will be shown how to set up all the hardware, how to use it, download the software you need, and then using the software. Read through each step carefully before using the framework, as mistakes can result in unstable drone behavior and crashes which usually damages the drone and may hurt people.

For more details about procedure and framework architecture see [1], and for EEG collection and processing see [2, 3].

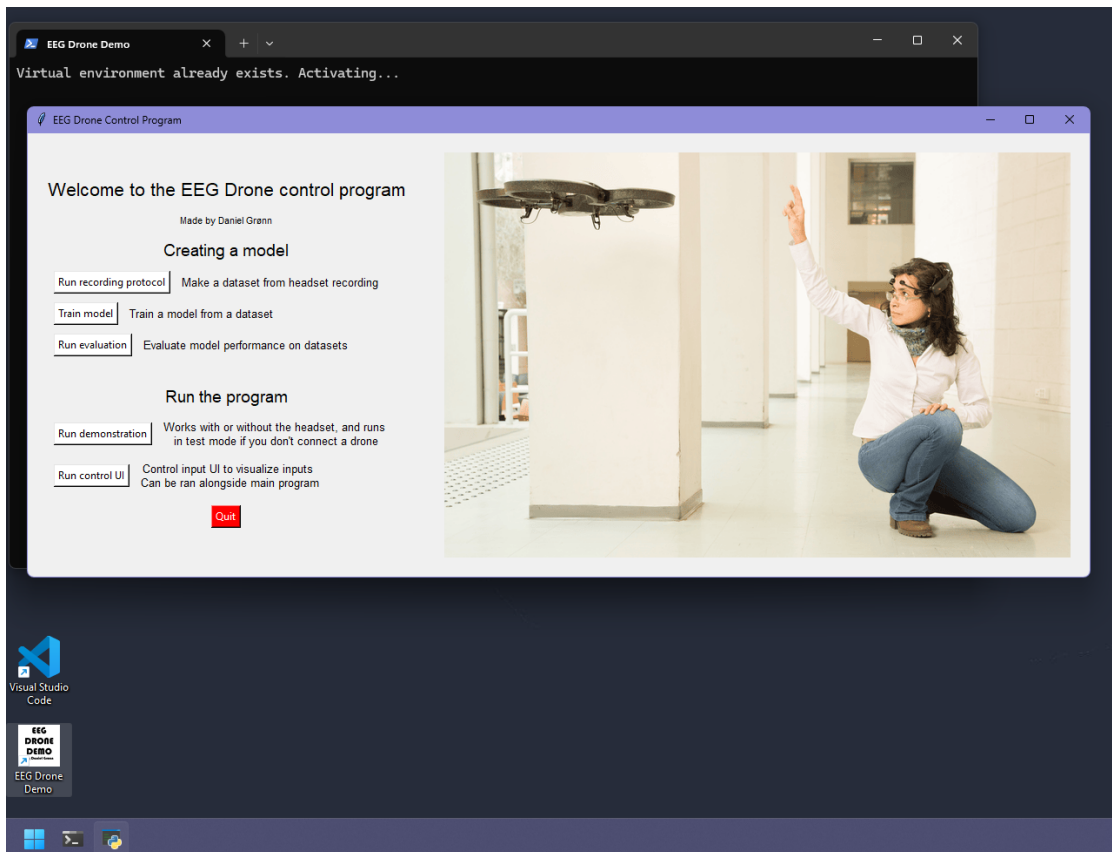


Figure B.1: Running the main menu GUI program in Windows

B.1 Contribution log

The source code for this User Guide available as an Overleaf project. If you want to contribute, please contact Marta Molinas for access.

- **V1.0** Doc made, describing setup in detail for windows - *D. Grønn* Fall 2024
- **V2.0** Expanded to swarm control and linux setup - *D. Grønn* Spring 2025

B.2 What you need to get started

Hardware

- Crazyflie 2.X drone
- Crazyflie USB dongle
- OpenBCI ultracortex IV with 8 electrodes
- OpenBCI USB dongle
- A Windows OR Linux PC with a keyboard and two open USB A ports
- minimum 4 GB of disk space
 - 3 GB for the repository with libraries
 - 1 GB for datasets and models

Software

- python 3
- VS code or similar IDE
- Git
- (GitHub Desktop app)
- Access to wavesresearch GitHub organisation or otherwise access to the drone_demo code repository
- Crazyradio USB driver (using Zadig)
- Cyton USB driver (called VCP from FDTI)
- OpenBCI GUI
- Administrator access (Linux only)

B.3 Preparing Hardware

B.3.1 Charging the batteries

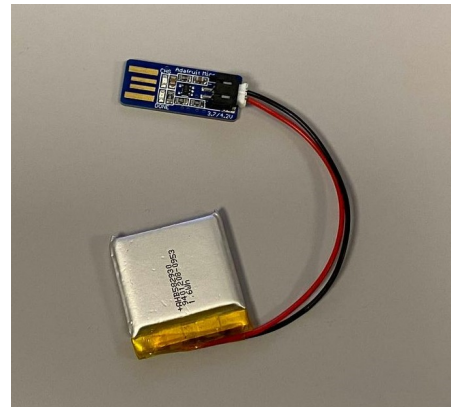
Make sure both drone battery and headset battery has enough charge to do a run. Approximated battery life:

- Crazyflie 2.0: 7 minutes
- OpenBCI Ultracortex: 4 hours
- Big OpenBCI compatible aftermarket battery: 10 hours

If the performance of either drone or headset is diminishing, it could be the battery being nearly empty. Use the provided charging parts as shown in Figure B.2.



(a) Crazyflie battery charging: with micro-USB cable



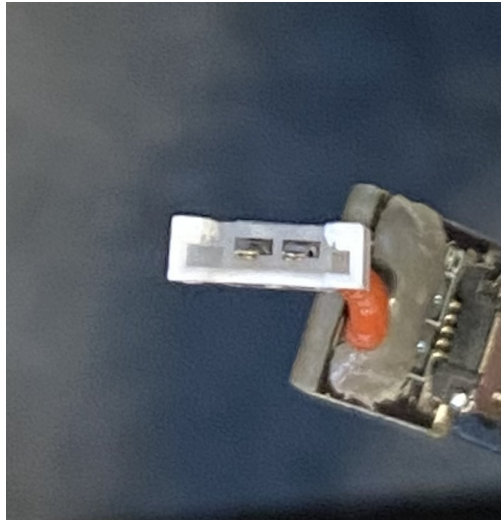
(b) OpenBCI battery charging: directly into USB port

Figure B.2: Battery charging of drone battery (left), and headset battery (right)

- **Drone battery charger:** green light when charging, no light when finished
- **Headset battery charger:** red light when charging, green light when finished
- **How to check the drone's battery:** Run cfclient through the main menu and it will open the client. You can now connect to the drone and you will get motion data displayed and information about how much battery is left.
- **How to check the headset battery:** There is not a known way to check for headset battery level in a similar way. It is recommended to charge the battery regularly.

NB: The pins in the drone charger part sometimes bend downwards, which increases resistance when putting the battery to charge. This resistance comes from the pin being forced backward into the charger, and ruins the part if it happens!

This force required to break it this way, which breaks either the charger or battery (often both!) is very small, so be careful. If the resistance increases beyond "very easy" to "why is it not going in?" try bending the pins upwards as shown in Figure B.3.



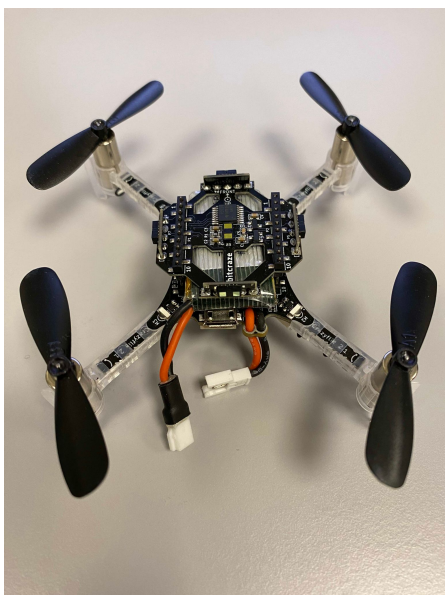
(a) Charger with bent pins (before fixing)



(b) Charger with straight pins (after fixing)

Figure B.3: Bent charger pins can lead to damaged battery and/or charger. Bend them back to avoid! The difference in the image was large enough to cause the issue.

B.3.2 Crazyflie V2 Drone



(a) Crazyflie off



(b) Crazyflie on

Figure B.4: Crazyflie 2.X drone

Figure B.4 shows what the Crazyflie 2.0 drone looks like when fully mounted. It has two different propellers which are made to spin in different directions. The A1 propeller spins clockwise, and the B1 propeller spins counterclockwise. Each propeller is mounted on a diagonal. There are symbols on the drone which show which direction the propellers will spin. Make sure the propellers are mounted correctly before use.

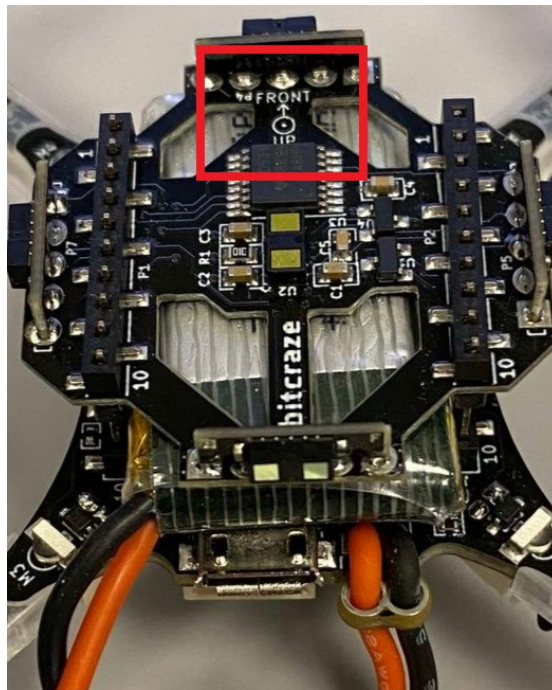


Figure B.5: Symbol shows forward and up direction on the drone

The forward direction of the drone is defined by the symbol on the drone shown in Figure B.5. Another way to observe the orientation of the drone is that the battery connection cable is in the rear.

When swapping batteries should the empty battery be slid rearwards out. To do this easily should the top board not be squeezed to hold the battery still, but instead allow the battery some room. It is secure from the cable being connected.

The drone's power button is located at the front as shown in Figure B.6. This button can be held to put the drone in bootloader mode.

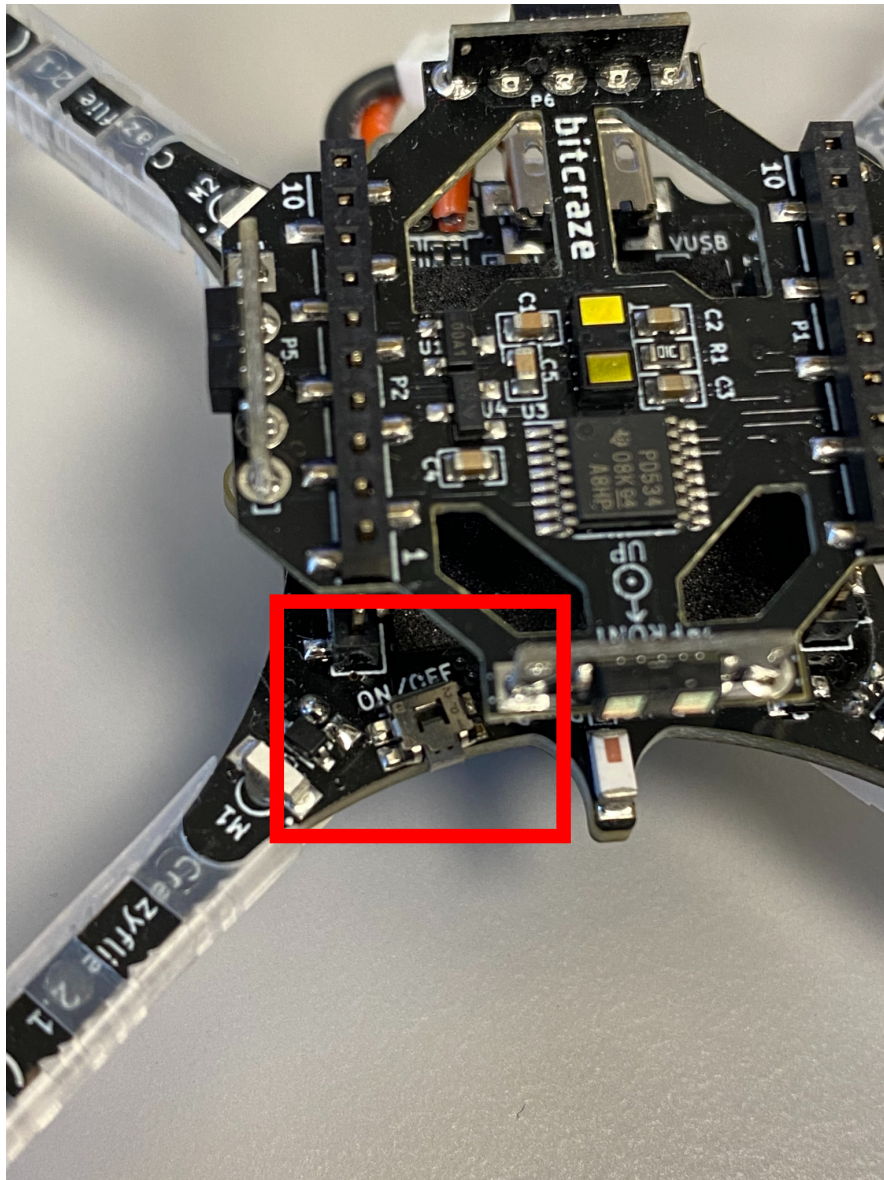


Figure B.6: The drone's on/off button

B.3.3 Headset and electrode configuration

How the electrodes are connected and placed will affect the performance of the models greatly. Since this project focuses on moving the drone with ME or MI, will the electrodes be placed around the motor cortex. The headset electrode configuration used by the author is shown below:

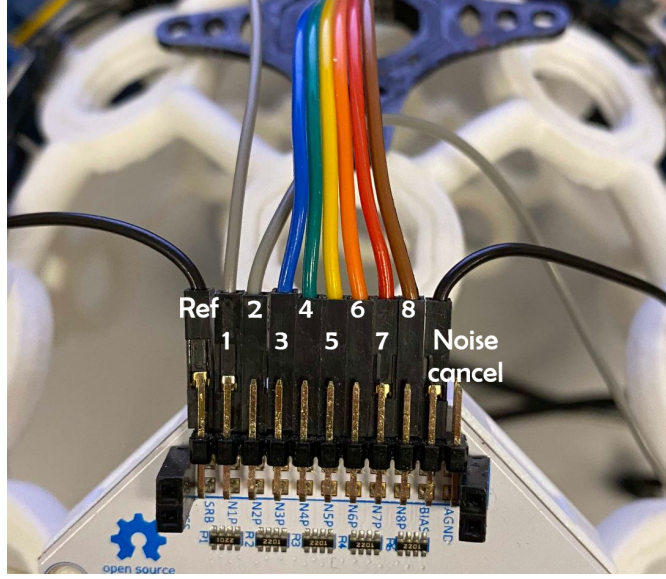


Figure B.7: Electrode pin connections cyton board

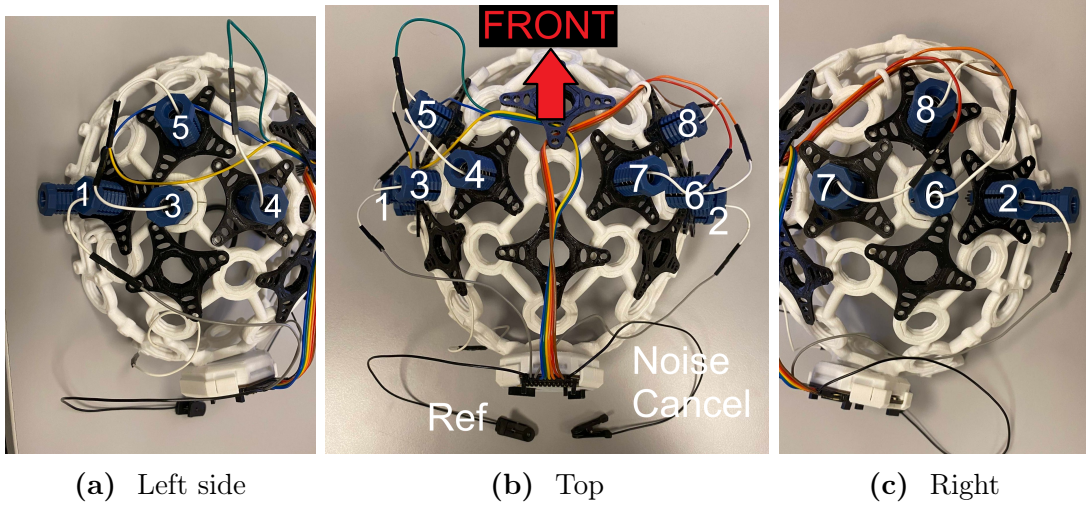
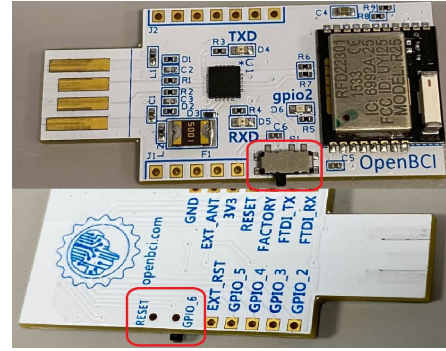
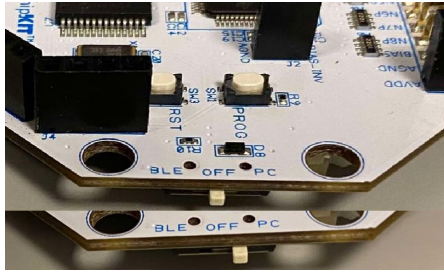


Figure B.8: Electrode configuration headset (Front is up)

Table B.1: Wire/electrode mapping

Wire/pin	10-10	Pin(Upper)	color in image
Ref	Reference	SRB	Black_left
1	C5	N1P	Gray_1
2	C6	N2P	Gray_2
3	C3	N3P	Blue
4	C1	N4P	Green
5	FC3	N5P	Yellow
6	C4	N6P	Orange
7	C2	N7P	Red
8	FC4	N8P	Brown
Noise Cancel	Noise Cancel	BIAS	Black_right



(a) The setting to use is "pc" which is up (b) The mode used for the headset dongle when the board is in the cap is this one: GPIO 6 (and not reset)

Figure B.9: Headset and headset dongle setting

Both the dongle and the headset board (cyton board) has a switch which needs to be in the correct mode. The cyton board needs to be in "PC" mode when on, which is upwards when the helmet is worn. The dongle should be in GPIO6 mode, which is towards the computer.

Note: In some cases can connection issues with the headset be solved by using the "reset" setting on the cyton dongle.

B.4 Preparing Software

B.4.1 Installing python

Author has used python 3.11. The newest release and many more are found here: <https://www.python.org/downloads/>

B.4.2 Coding IDE

It is strongly recommended to install and use an Integrated Development Environment (IDE) when using the code base. One such IDE is Visual Studio Code (VS Code). It can be installed from this website: <https://code.visualstudio.com/download>.

B.4.3 Download Git

Git is a commonly used version control solution. It allows for collaboration by tracking and sending/receiving changes in project files. It is downloaded from their website: <https://git-scm.com/downloads>

B.4.4 Download Github Desktop

Github Desktop will make it much easier to clone the code repository to your computer, but it is possible to use Git through the git bash terminal if wanted. That will not be covered here. The install can be found here: <https://desktop.github.com/download/>

After downloading, login to a GitHub desktop account or make a new account if you would rather do that. It is recommended to use a private GitHub account to not have to set up two-factor authentication for multiple ones, as this is required for some functionality. GitHub handles organization of the projects and access in such a way where it is not a problem to be a part of multiple organizations. It is of course totally up to the user.

B.4.5 Install the code repository from Github wavesresearch

To get the easiest access to the repository and further develop it, it is recommended to do so by cloning the repository through GitHub. Access to the wavesresearch organisation, and therefore the drone_demo repository has to be given by an administrator. Two such people are Marta Molinas, and Luis Alfredo Moctezuma at the time of writing.

When access is given, all that is needed is to clone the repository to the working computer is to follow the procedure:

1. Search for "drone_demo"
2. Select the correct repository
3. Select where you want the code to be **stored locally on your computer**
4. Click "Clone" at the bottom

Note: Do not install the repository on Onedrive as this can cause a multitude of issues, instead navigate through "This PC" in the file explorer.
--

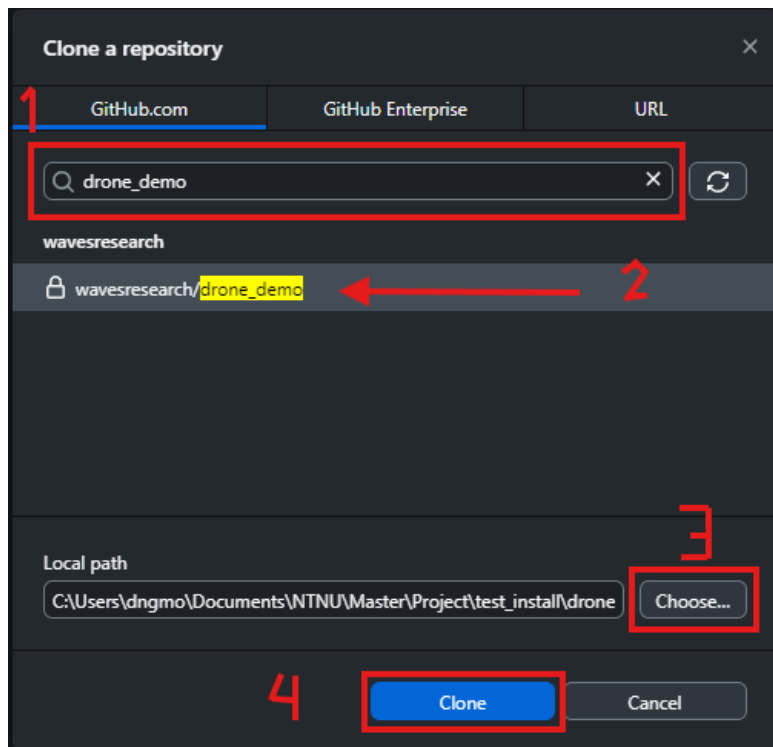


Figure B.10: Cloning the repository

Note: The interface may look different depending on version and if you have used GitHub Desktop before. To find "Clone repository" option either click "Ctrl+Shift+o" or click top left on File -> Clone Repository. If it is your first time opening GitHub Desktop should a search field for cloning repositories appear after logging in if there is no existing repository on the computer.

If you are unable to find the files after installing it, is it possible to right click the top left and choose show in explorer, as shown in Figure B.11

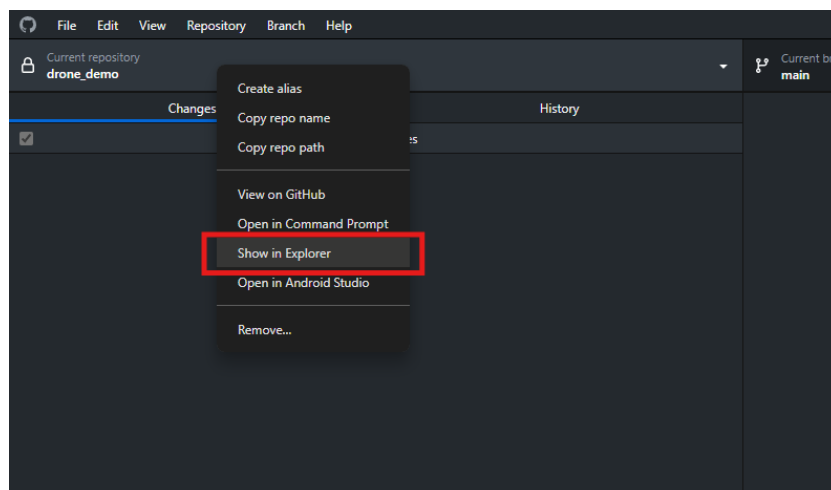


Figure B.11: Finding the repository location

B.4.6 Download OpenBCI GUI

Download OpenBCI GUI from their website: <https://openbci.com/downloads>

When the install is finished, should you make a shortcut and put it on your desktop or somewhere which it is accessible to you.

B.5 Installing Drivers for Windows

To install the mandatory drivers for connecting to the hardware on windows, we will need a program called Zadig, aswell as a driver for the headset.

B.5.1 Crazyradio USB dongle driver

This part of the manual is copied in part from this tutorial:

<https://www.bitcraze.io/documentation/repository/crazyradio-firmware/master/building/usbwindows/>.

For the Crazyflie (Crazyradio) USB dongle to work properly it is necessary to get the driver for it. To get the required driver does bitcraze (maker of the crazyflie drone) recommend to use a tool called "Zadig". It is a USB driver installation tool. It is downloaded from their website: <https://zadig.akeo.ie/> when the tool is installed, launch it and plug in your Crazyradio dongle. When the dongle is plugged in should it automatically show up. Then install the USB driver with the settings which are shown here (version may change):

Note: The driver should be downloaded with libusb. To find this option use the up and down arrows to the right of the driver option.

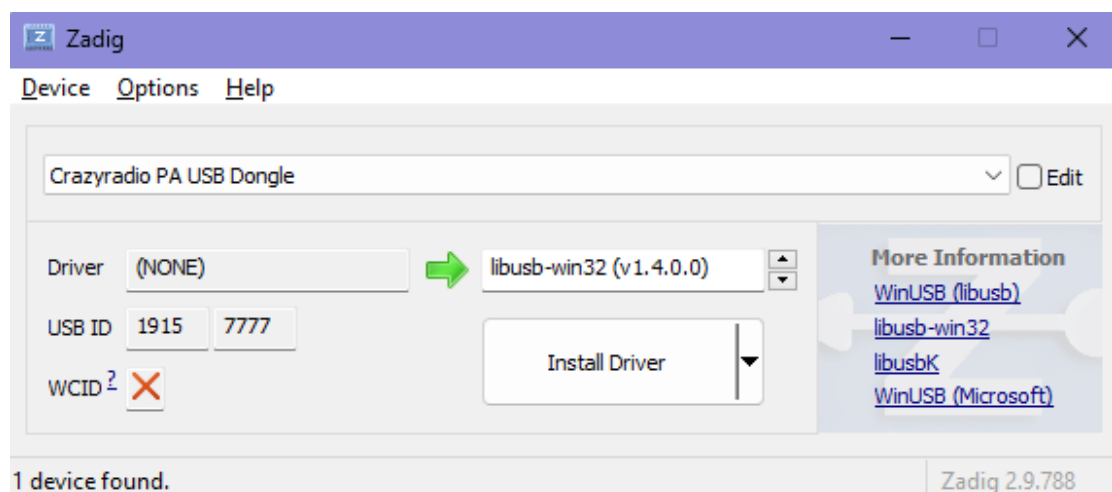


Figure B.12: Zadig installer settings

Note: If you need to reinstall the driver, click options-> List all devices then select the Crazyradio dongle. You can now reinstall. For other problems see the tutorial linked at the start.

B.5.2 Cyton USB dongle driver

Cyton is the board for the OpenBCI headset. The driver for it can be installed from OpenBCI's trusted third party supplier, FTDI. We need the VCP (Virtual COM Port) driver to recognize the dongle. It is a general driver which is used by multiple boards. Here's a link to the site where the newest VCP driver can be installed: <https://ftdichip.com/drivers/vcp-drivers/>

Install the driver provided by the link shown in Figure B.13

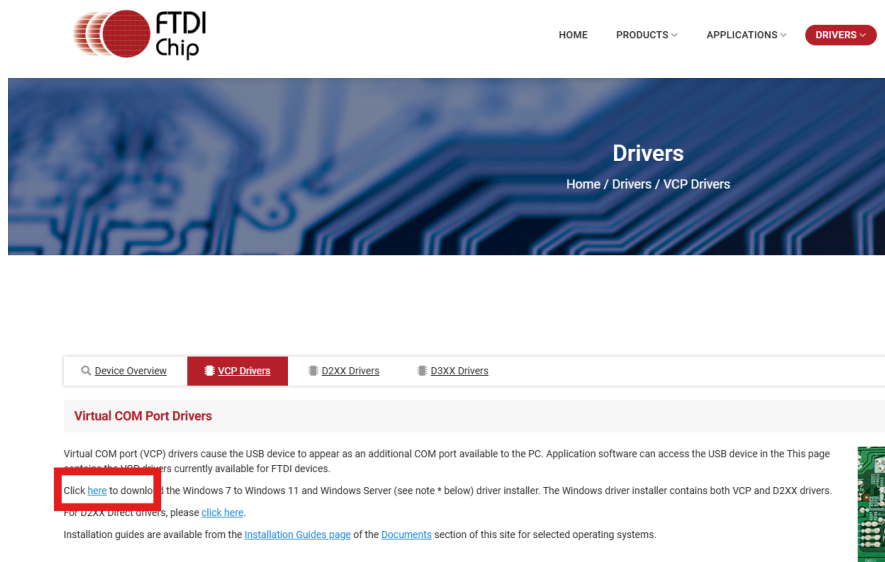


Figure B.13: VCP driver download location

B.6 Using the framework

With the hardware ready and software installed we now need to prepare before using the framework.

It is quite painful to sit with the headset on for long periods of time, it is therefore recommended to prepare as much as possible before putting it on. Any subject who is going to have the headset on should be informed about this.

B.6.1 Running the software

You should now have all the required software and hardware set up to run the demonstration. All that is left is to run the program. As a heads up, might an issue with drone connection problems appear as discussed in Section B.8.3 appear. If the program closes immediately, might this also be the cause.

The files can be found in the directory which was chosen during cloning of the repository in Section B.4.5.

B.6.1.1 Run with PowerShell

To run the program is the first option to run *Run_Demo.ps1*. To do this right click on the file, then select run with PowerShell.

Note: OpenBCI GUI needs to be closed to run the program with the headset. This is because the serial data stream can only have one recipient in its current state.

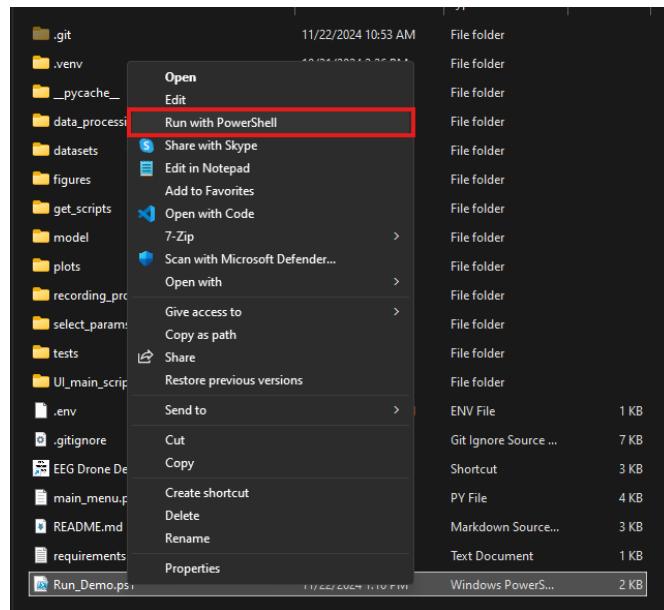


Figure B.14: Run With PowerShell

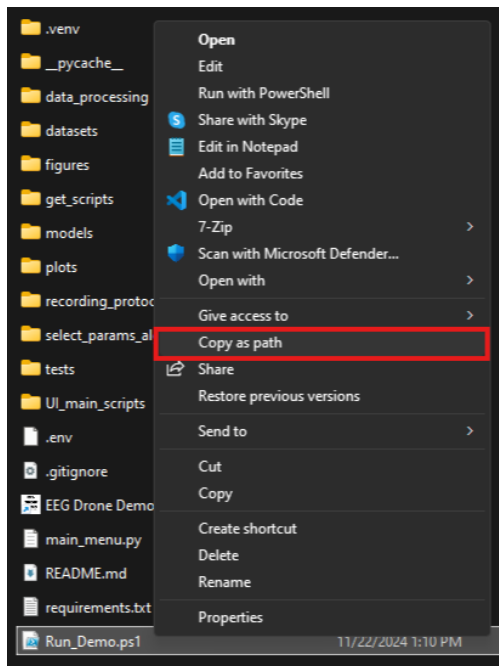
Note: This will, as the terminal echo is also reporting, create a `.venv` environment to store all the required packages if one is not present. It will also install the requirements for the project from the *requirements.txt* file, and activate the virtual environment. It then runs the *main_menu.py* script. Because it has to install all the packages, will this take alot longer to run the first time if the environment is not already present. It does not create nor try to install packages if the `.venv` environment is present without packages or with an incomplete set of packages.

The installation then has to be done manually, but that will not be covered here. Visual Studio Code has built in functionality to set this up.

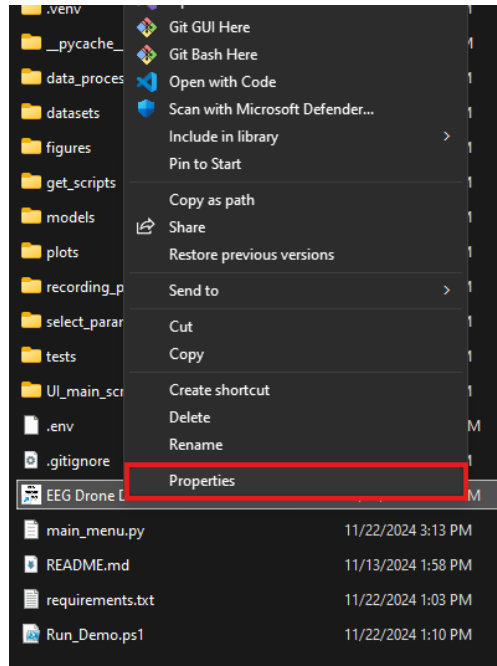
B.6.1.2 Run through the created shortcut

You can also make a shortcut which runs this program directly, and which can be placed anywhere. You can also rename the shortcut and change icon if you want.

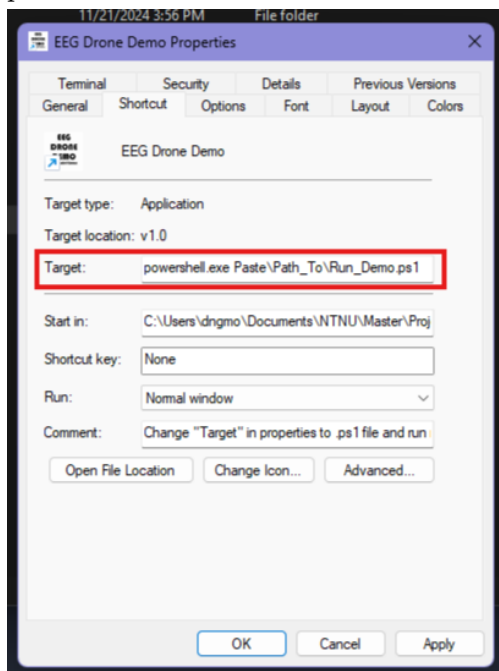
A shortcut like this is included in the project folder, but needs the right path to the *Run_Demo.ps1* file. To set this correctly do this:



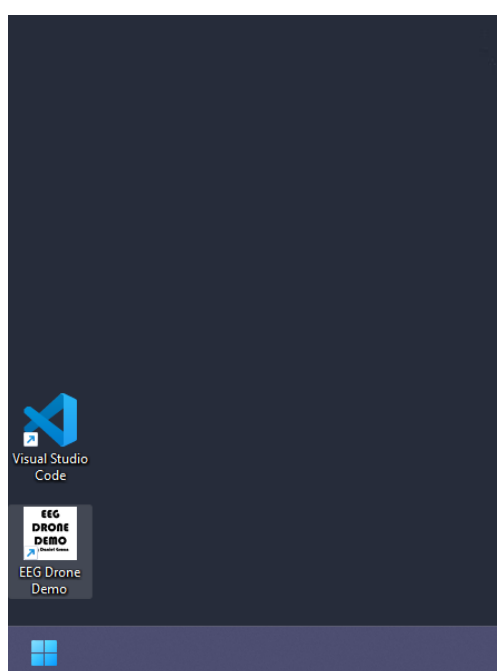
(a) Right click on the PowerShell file *Run_Demo.ps1* and select "Copy as path"



(b) Right click on the shortcut and navigate to properties



(c) Change "Target" to "powershell.exe" followed by the path to the PowerShell file, Select "OK" at the bottom



(d) You can now move and run the shortcut (and the project files) from anywhere within your machine!

Figure B.15: The 4 steps to set up the shortcut

Some benefits from creating and using the Shortcut and the PowerShell script

instead of running the *main_menu.py* file directly:

- It fixes an issue with the terminal automatically, the issue is discussed in Section B.7
- It is not necessary to select "run with PowerShell" since having the "powershell.exe" before the path does this for us automatically.
- It automatically sets up the environment and downloads dependencies for a new installation making it even simpler to use
- The program is run as a normal application, but the project files can be modified after it's creation (unlike a compiled program)

The logo in oldFigures/ folder can be used as logo if the user wants to set this.

Note: It is important that the program is shut down with the use of the "exit" button, as this triggers the forcing of all open threads. Exact consequence of closing the program otherwise is unknown to author.

B.6.1.3 Running main menu.py directly

It is also possible to run the program by directly running the *main_menu.py* program in a terminal with the virtual environment activated. The reason why running through a shell script is suggested is that this will behave as a normal program, and is a better solution long term, especially since this allows the user to change the program code at runtime, allowing for quick development.

B.6.2 Using the software

No matter the method used to run the framework, should you be met with the main menu program. Here is it possible to run all the desired programs to get from having essentially nothing to datasets, models and using these to running a demonstration.

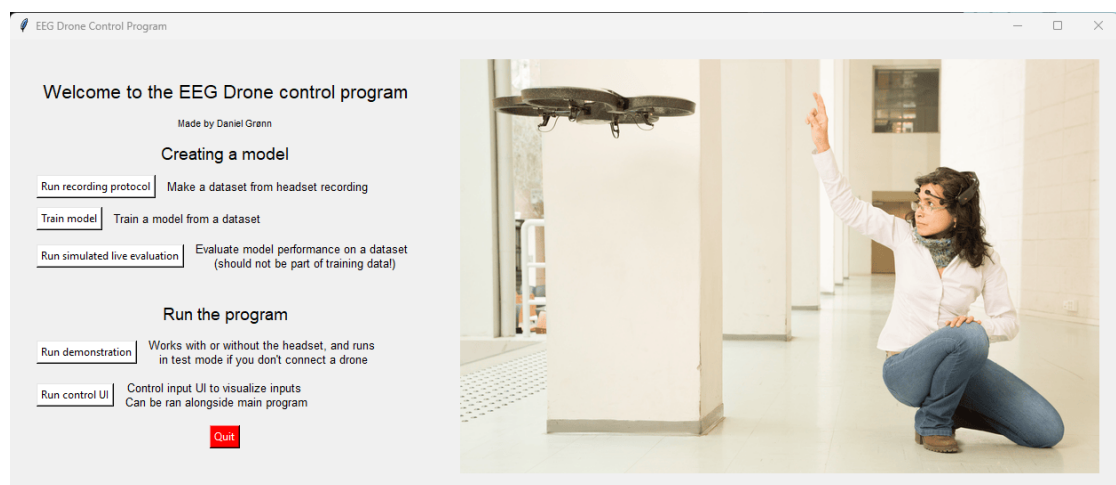


Figure B.16: Main menu where all programs can be ran from

As shown in Figure B.16 is there currently five programs which can be run from the main menu.

- **Run recording protocol** starts a program which will show the user triggers for executing commands to create labeled data. These are red arrows indicating which limb should be activated. It also has a progress bar at the bottom as seen in Figure B.17
- **Train model** which will train the model using the parameters chosen in the GUI shown in Figure B.18
- **Run simulated live evaluation** takes one session of a dataset and runs it through the predictor sequentially as if it was live data. This allows the data to align with the classifier timings. It outputs two confusion matrices like the one in Figure B.19, and an accuracy score in the terminal
- **Run demonstration** runs the live program with a chosen model as classifier. It can be ran with the headset or keyboard as input, or both. It has no GUI, but instead is the next program used
- **Run control UI** used as the UI for live operation. It can run simultaneously as Run demonstration, and echos the keyboard inputs. It can be seen in Figure B.20

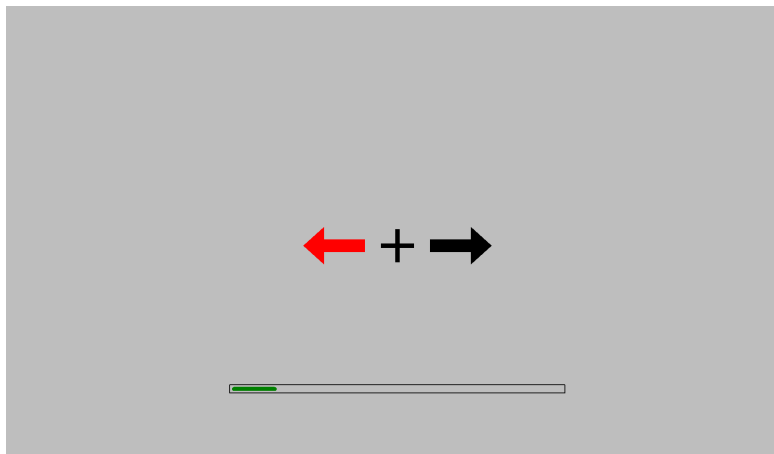


Figure B.17: Recording protocol GUI

Train...

Select Subject:

- ☒ ds: DANIEL_ME
- ☐ fc: DANIEL_MI
- ☐ sv: FILIP_ME

Select Classifier:

- ☒ 2_classes (Classifiers: 2)

Select Sessions:

- ☒ Session 1
- ☒ Session 2
- ☐ Session 3

Model Name:

Name

Select Classifier Type:

- ☐ SVM
- ☐ FBCSPSVM
- ☐ CNN
- ☐ RIEMANN
- ☒ CSPSVM

Submit

Figure B.18: training parameter selection GUI

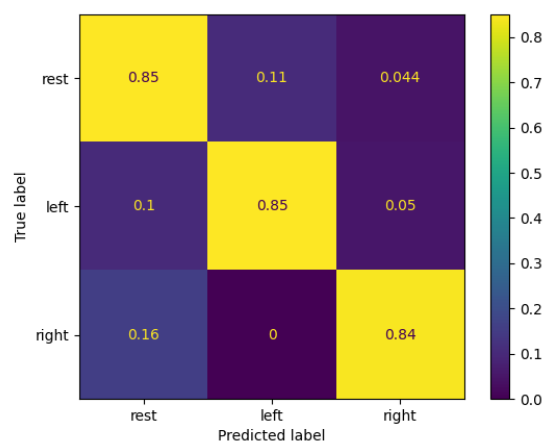


Figure B.19: A confusion matrix

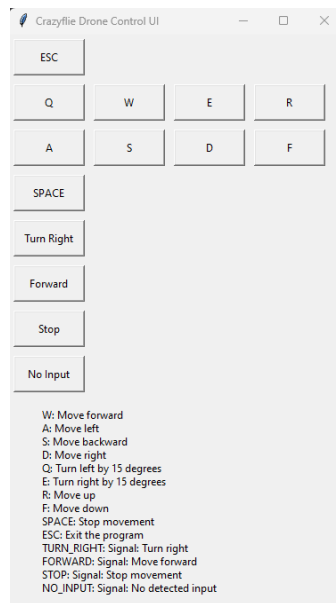


Figure B.20: Control GUI echo

B.6.3 Using the drone

To check the drone connection can the cfclient be used. In any terminal with the library downloaded are you able to write "cfclient" into the terminal and launch said application. It shows live orientation data and some parameters as shown in Figure B.21

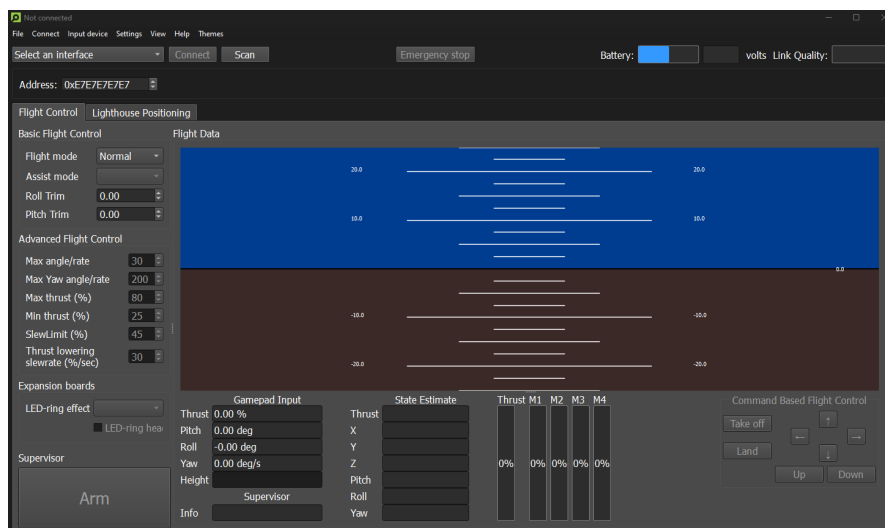


Figure B.21: The cfclient

This software can also be used to check battery level at the top right, as mentioned earlier.

B.6.4 Using the headset

When putting on the headset you have to make sure that all electrodes get a good connection. You do this by using the OpenBCI GUI.

Select Data Source: CYTON (live) -> Pick transfer protocol: Serial (from Dongle) -> Serial connect: Auto-Connect. The software should look like Figure B.22, unless the program has been changed drastically a future version.

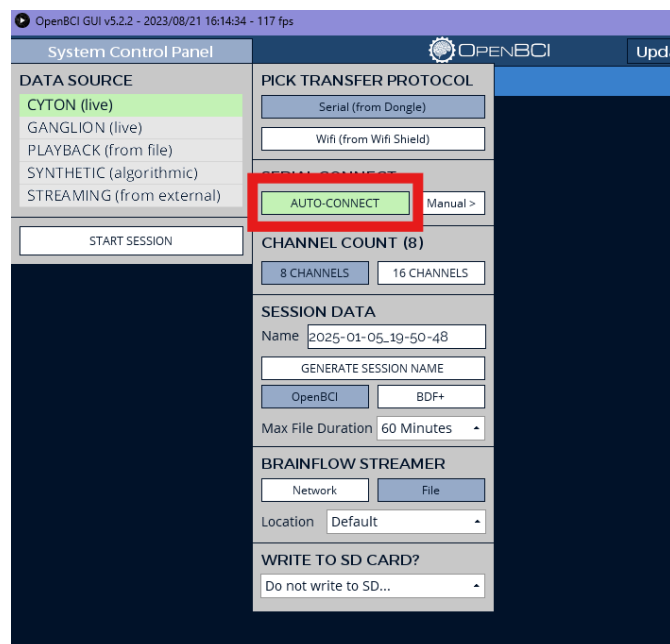


Figure B.22: OpenBCI GUI Autoconnecting to headset

To then adjust for impedance on all electrodes, should the Cyton signal element be selected, see bottom right quadrant of the window in Figure B.23.

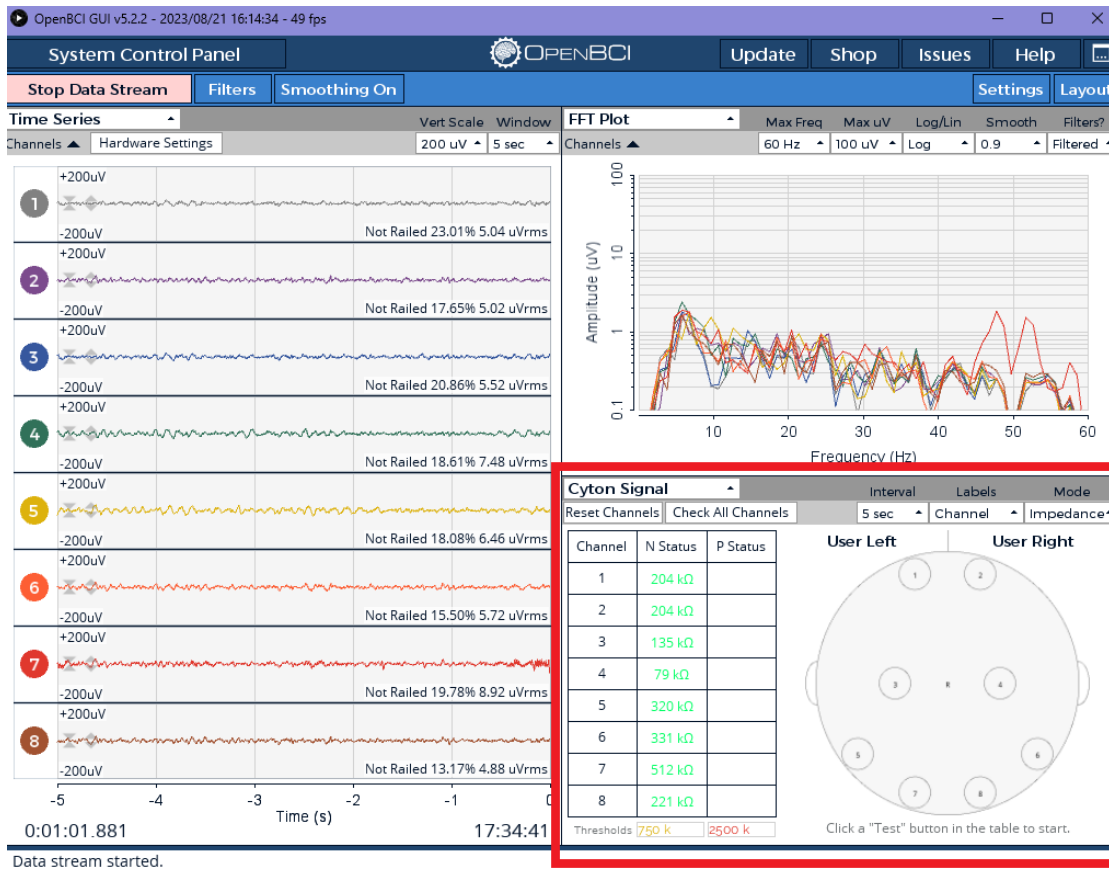


Figure B.23: After adjusting the electrodes to get good impedance values

Note: Always disconnect the electrodes before adjusting them (rotating to move in or out). If this is not done will the wires break near the electrodes in the cap. This is time consuming to fix and expensive to replace.

It might be necessary to find out which port the headset is connected to manually. This is also done through OpenBCI GUI. Select Data Source: CYTON (live) -> Pick transfer protocol: Serial (from Dongle) -> Serial connect: Manual, and the port will show up on the top right of this GUI element.

B.7 Setting up for further development

Much of the motivation to organise the interface and make GUIs was to ease use, but also ease future development. If you plan to do this, should read the previous master's theses in the project thoroughly. Some first steps toward further development include:

As mentioned at the end of Section B.6, can also `main_menu.py` be used directly. All files can be ran in this fashion ofcourse. However, if the shortcut with the powershell solution is implemented correctly can this potentially save valuable time. It is still therefore suggested to use that solution even when further developing the framework.

However, if it is necessary to use a different terminal might the `ModuleNotFoundError` appear. See Section B.8.1.

B.8 Issues and Fixes

This section will describe some encountered issues and their proposed solutions. Their order is based on when in the setup/developing process they were encountered, and their likelihood of happening to a new user.

B.8.1 ModuleNotFoundError

```
ModuleNotFoundError: No module named 'recording_protocol'
```

Listing 9: ModuleNotFoundError

This error appears because the folders which contains imported code is not a part of the available paths for the python instance. To temporarily fix this issue write this into the terminal (each time you open a new one):

- **cmd:** set PYTHONPATH=%CD%
- **PowerShell:** \$env:PYTHONPATH = Get-Location

This fix is built into the *Run_Demo.ps1* file and is therefore ran every time to avoid the issue with normal use.

A possible permanent fix: All folders containing scrips which are imported into other scrips need to be added as environment variables. This has not been done yet due to time limitations.

B.8.2 BCI headset connection problems

B.8.2.1 Unresponsive headset

If the headset is unresponsive but on, try disconnecting the headset from it's battery as this can sometimes fix the problem.

B.8.2.2 High noise and low impedance in OpenBCI GUI

If one channel appears to have very high (or noisy) measurements in the OpenBCI GUI, but has low impedance, try closing and opening OpenBCI GUI again.

B.8.2.3 Importing wrong serial library breaks OpenBCI class

Importing the package "serial" brakes the serial code, since this project uses pyserial. If you install serial package, you need to reinstall pyserial. One option to guarantee nothing is broken is to reinstall all packages in the .venv environment.

B.8.3 Drone connection problems

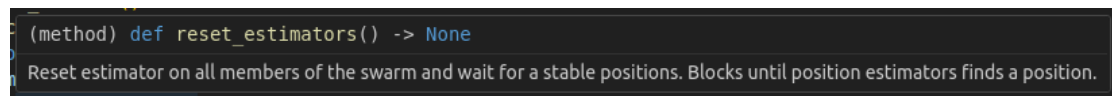
B.8.3.1 Connected, but not flying

The drone could connect properly, but refuse to fly. This might be caused by the software on the drone needing flashing, or rather, the memory needs flashing. A guide to how to do that is provided here. Note: this requires linux or wsl: <https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/building-and-flashing/build/>

A possible other solution could be to change the cfclient libraries to a different version if the issue is followed by an update of either software.

B.8.3.2 Program connecting to drones, but not getting to the command loop stage

As a part of testing connection, logging and information flow has it been useful to connect drones and the headset outside of the flying lab. However, with the swarming programs will the program then get stuck in reset estimator stage of the program, because this command blocks until the position system has found a stable position. Getting a position is of course not possible when the position system is missing.



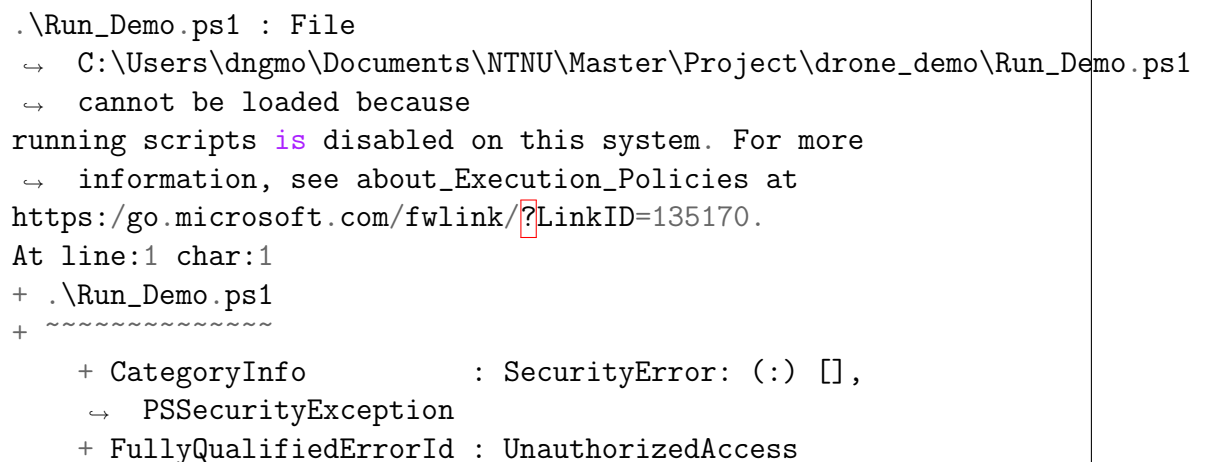
```
(method) def reset_estimators() -> None
Reset estimator on all members of the swarm and wait for a stable positions. Blocks until position estimators finds a position.
```

Figure B.24: The function description for the reset_estimators() command

The issue looks very similar to that of the main loop being busy with something else, but in this case has the main loop gotten stuck on this, right before entering the command loop.

B.8.4 Running downloaded scripts disabled error

If the program quits quickly or the following error is displayed:



```
.\Run_Demo.ps1 : File
↳ C:\Users\dngmo\Documents\NTNU\Master\Project\drone_demo\Run_Demo.ps1
↳ cannot be loaded because
running scripts is disabled on this system. For more
↳ information, see about_Execution_Policies at
https://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\Run_Demo.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [],
  ↳ PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

Listing 10: Scripts Disabled Error

The issue could be caused by the Execution policy being set to restricted. This is a safety measure to restrict execution of malicious software from being run on the machine.

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope
↳ LocalMachine

-- OR --

Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope
↳ LocalMachine
```

Listing 11: Set-executionPolicy for local machine

These commands change the execution policy system wide; for all users. Remote-Signed option ensures that remotely written executables need signing, and Unrestricted allows all programs to be ran. The latter is less safe, but more flexible. This solution comes from Rudy Mens' article: [60].

B.8.5 Automatically detecting the port of the Cyton board

Previously, like many other parameters in this project, was it necessary to hard-code the port of the BCI headset dongle. This has been fixed as a new function:

```
% file: src/command_system/headset/RECORDER.py
```

```
def find_cyton_port():
    """
    Automatically finds the serial port for the Cyton board.
    Returns the port name if found, else None.
    """
    ports = serial.tools.list_ports.comports()
    for port in ports:
        # Check for a specific keyword associated with the Cyton board
        # Common Cyton identifiers include "USB Serial Device" or "FTDI"
        if "FTDI" in port.manufacturer:
            return port.device # e.g., 'COM3' or '/dev/ttyUSB0'
    return None
```

Listing 12: Automatically finds the Cyton board

B.8.6 Export requirements to text file

Here is a useful command to export the requirements to a text file so that it is easy to set up on the next computer. Although this might be an outdated

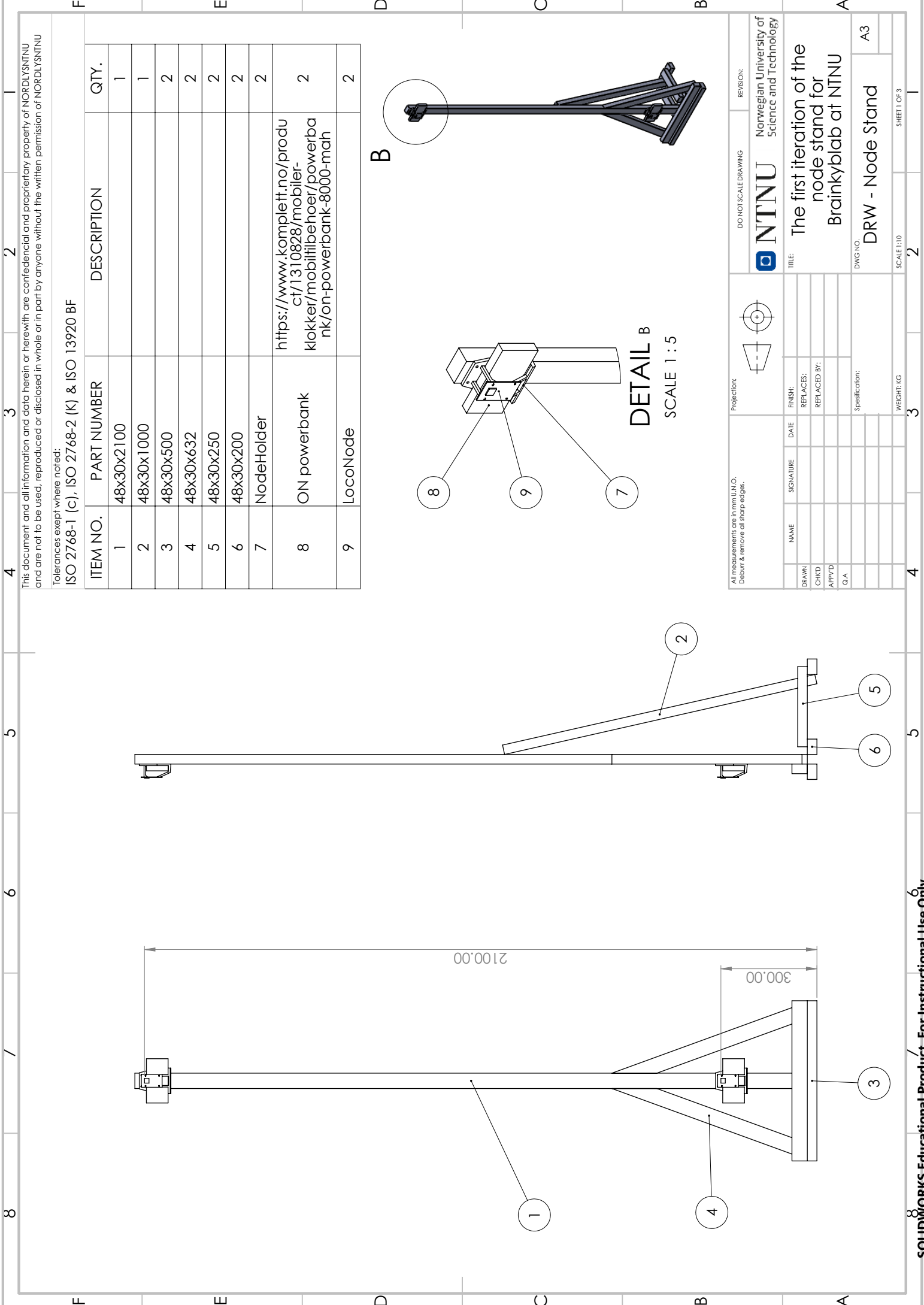
way to handle packages, does it still work pretty well. It could be changed to a newer/better option in the future.

```
pip3 freeze > requirements.txt
```

Listing 13: Loading package requirements into separate file

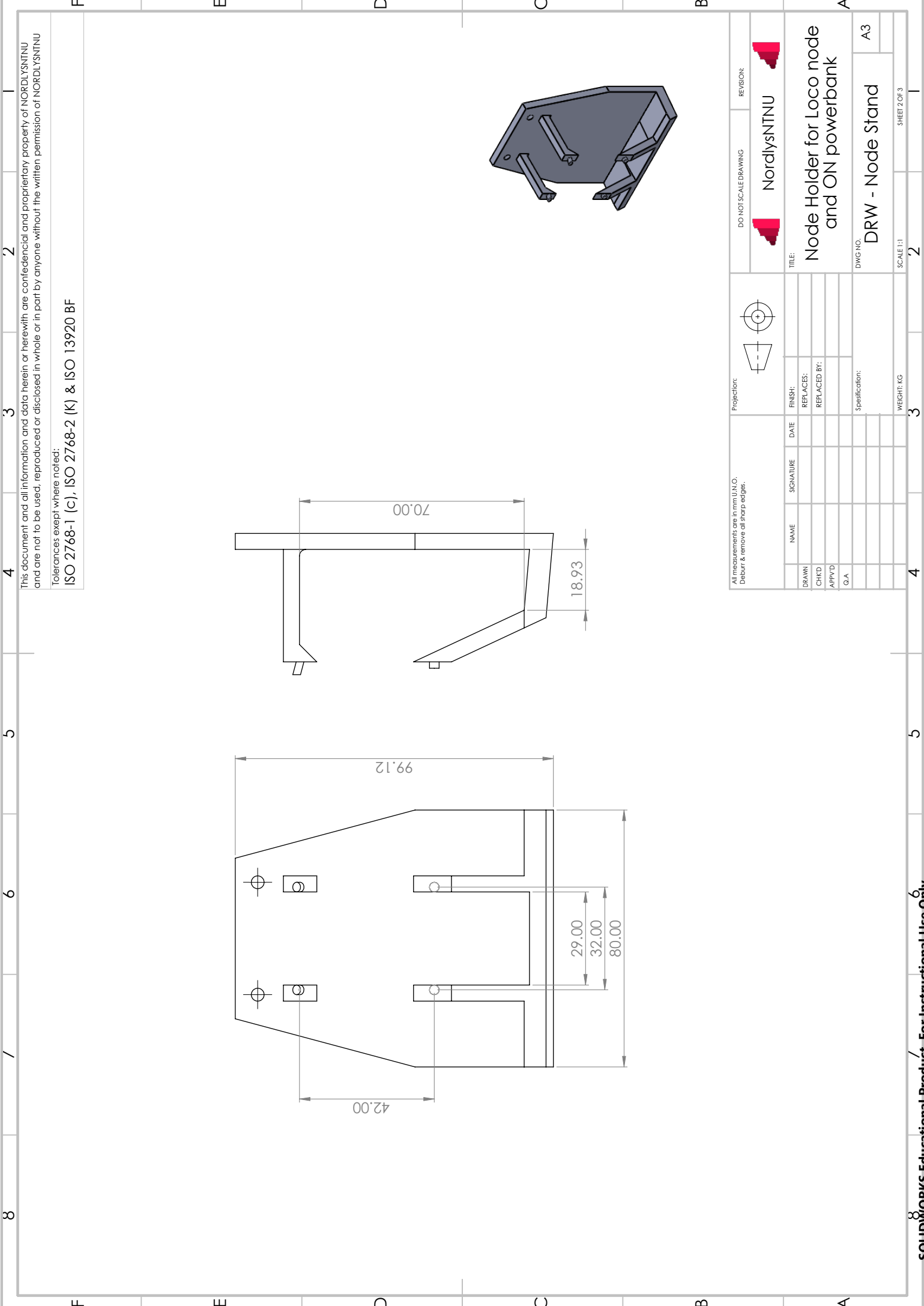
NODE STAND - TECHNICAL DRAWINGS

NB: Drawings are scaled from A3 to A4, only for visualizations.



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	48x30x2100		1
2	48x30x1000		1
3	48x30x500		2
4	48x30x632		2
5	48x30x250		2
6	48x30x200		2
7	NodeHolder		2
8	ON powerbank	https://www.komplett.no/produskt/1310828/mobiler-klokker/mobilitilbehoer/powerbank/on-powerbank-8000-mah	2
9	LocoNode		2

All measurements are in mm U.N.C. Debur & remove all sharp edges.		Projection:		DO NOT SCALE DRAWING		REVISION:	
NAME		SIGNATURE		DATE		NTNU Norwegian University of Science and Technology	
DRAWN		FINISH:		TITLE:		The first iteration of the node stand for Brainkyblab at NTNU	
CHKD		REPLACES:		DWG NO.		A3	
APPVD		REPLACED BY:		DRW - Node Stand			
Q.A.		Specification:		SCALE 1:10		SHEET 1 OF 3	
WEIGHT: KG		3		2		1	

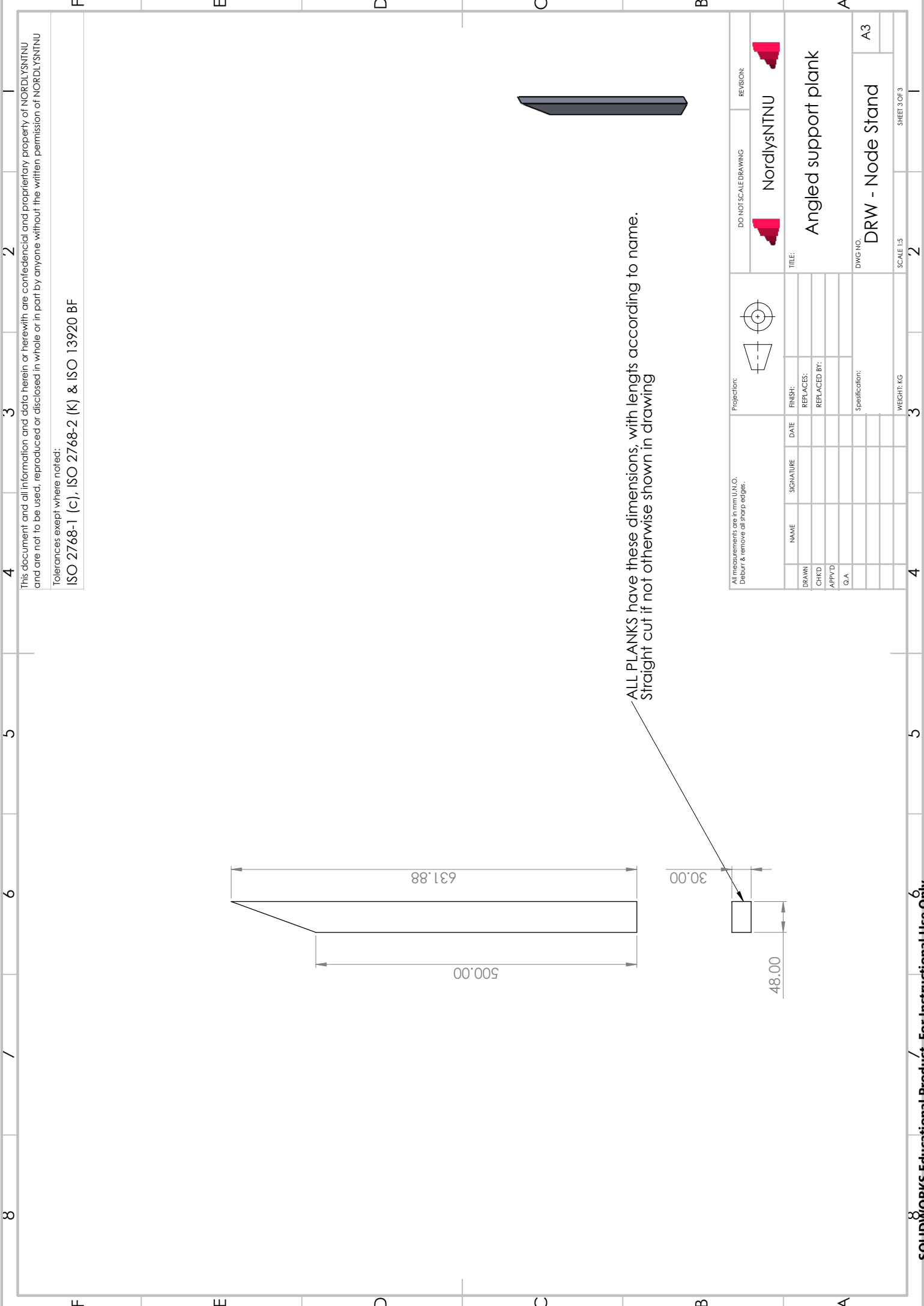


This document and all information and data herein or herewith are confidential and proprietary property of NORDLYSNTNU and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of NORDLYSNTNU

Tolerances except where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

All measurements are in mm U.N.I.C. Deburr & remove all sharp edges.				Projection:		DO NOT SCALE DRAWING		REVISION:	
								NordlysNTNU	
								TITLE:	
								Node Holder for Loco node and ON powerbank	
								DWG NO.	
								DRW - Node Stand	
								A3	
								SCALE 1:1	
								SHEET 2 OF 3	



This document and all information and data herein or herewith are confidential and proprietary property of NORDLYSNTNU and are not to be used, reproduced or disclosed in whole or in part by anyone without the written permission of NORDLYSNTNU

Tolerances exept where noted:

ISO 2768-1 (c), ISO 2768-2 (K) & ISO 13920 BF

All measurements are in mm U.N.O. Debur & remove all sharp edges.				Projection: 		DO NOT SCALE DRAWING		REVISION: NordlysNTNU	
NAME		SIGNATURE		DATE		FINISH:		TITLE:	
DRAWN						REPLACES:		Angled support plank	
CHKD						REPLACED BY:			
APPVD									
Q/A						Specification:		DWG NO.	
								DRW - Node Stand	
								A3	
								SCALE 1:5	
								WEIGHT: KG	
								SHEET 3 OF 3	

APPENDIX

D

AI DECLARATION

Declaration of AI aids and -tools

Have any AI-based aids or tools been used in the creation of this report?

- ☐ No
☒ Yes

If yes: please specify the aid/tool and area of use below.

Text

- ☒ **Spell checking.** Are parts of the text checked by:
Grammarly, Ginger, Grammarbot, LanguageTool, ProWritingAid, Sapling, Trinkai.ai or similar tools?
- ☐ **Text generation.** Are parts of the text generated by:
ChatGPT, GrammarlyGO, Copy.AI, WordAi, WriteSonic, Jasper, Simplified, Rytr or similar tools?
- ☐ **Writing assistance.** Are one or more of the reports ideas or approach suggested by:
ChatGPT, Google Bard, Bing chat, YouChat or similar tools?

If yes, use of text aids/tools apply to this report - please specify usage here:

ChatGPT has been used to fix grammar and formalize formulations

Codes and algorithms

- ☒ **Programming assistance.** Are parts of the codes/algorithms that i) appear directly in the report or ii) have been used to produce results such as figures, tables or numerical values been generated by: *GitHub Copilot, CodeGPT, Google Codey/Studio Bot, Replit Ghostwriter, Amazon CodeWhisperer, GPT Engineer, ChatGPT, Google Bard* or similar tools?

If yes, use of programming assistance aid/tools apply to this report - please specify usage here:

ChatGPT has been used to help generate simple parts of code for efficiency, and GitHub Copilot has been used (although rarely) to autocomplete function calls etc

Images and figures

- ☐ **Image generation.** Are one or more of the reports images/figures generated by:
Midjourney, Jasper, WriteSonic, Stability AI, Dall-E or similar tools?

If yes, use of image generator aids/tools apply to this report - please specify usage here:

- ☐ **Other AI aids or tools.** Have you used other types of AI aids or -tools in the creation of this report?
If yes, please specify usage here:

- ☒ I am familiar with NTNU's regulations: *Submitting a report generated with the assistance of AI tools and claiming the work to be partially or fully my own, is not permitted. I therefore declare that any use of AI aids or tools are explicitly stated i) directly in the report or ii) in this declaration form.*

Daniel N. Gran / 05.07.25 / Trondheim, Norway
Signature/Date/Place

