POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Programming Environment for Human Drone Interaction: EasyFly

Tesi di Laurea Magistrale in
Computer Science Engineering

Author: **Matteo Plona**

Student ID: 952967
Advisor: Prof. Luca Mottola
Academic Year: 2023-24

Abstract

TODO

Keywords: here, the keywords, of your thesis

Abstract in lingua italiana

TODO

Parole chiave: qui, vanno, le parole chiave, della tesi

Contents

1 | Introduction

In recent years, the rapid advancement of unmanned aerial vehicles, commonly known as drones, has revolutionized various industries and opened up new possibilities for applications ranging from aerial photography and package delivery to search and rescue operations. As drones become increasingly integrated into our daily lives, it is crucial to explore and understand the dynamics of their interaction with humans.

In modern drone applications, the human figure is marginal with respect to the drone. The former usually plays the supervisor role, while the latter performs almost all requested tasks automatically. This significant discrepancy undoubtedly leads to a decrease in interactions between the two.

This thesis will describe EasyFly, an accessible and high-level programming environment for drone applications. The purpose is to provide a programming environment to allow users with different levels of expertise to experiment with drones. Differently from modern applications, our environment allows humans and drones to work closely together, making EasyFly a perfect tool for conducting research in the field of human-drone interactions.

1.1. The Problem: Programming Human-Drone Interactions

Human-Drone Interaction (HDI) is a branch of the more general field of Human-Robot Interaction (HRI), and it can be defined as 'the study field focused on understanding, designing, and evaluating drone systems for use by or with human user [37]'. While the field of HRI offers valuable insights, the distinctive ability of drones to move freely in three-dimensional space, along with their unique shapes, sets HDI as a distinct and independent area of research.

The rapid technological progress in this field has made drones increasingly efficient and autonomous in performing various tasks. While on the one hand, these advancements enable the integration of drones into everyday life, streamlining processes and reducing the time required for specific activities, on the other hand, modern drone applications do not represent the ideal prototype for conducting research in the field of HDI.

The first limitation of modern drone applications in this discipline's study is that these applications are designed to operate in large environments with minimal human presence. An example can be represented by autonomous delivery drones [35] or 3D mapping applications [30] where the interactions with the human are purposely reduced to the bare minimum.

In these types of applications, interactions between humans and drones are often limited to simple tasks, such as package delivery or crop monitoring. These interactions are often repetitive and lack the diversity and complexity required in research on HDI. Since tasks and interactions are repetitive, users are usually trained to interact with the drone in a specific way. This training can reduce the variability in HDI, making it less suitable for research purposes.

Programming HDI is usually the field's most complex and expensive research phase. It usually requires a specialized team of researchers who can program a custom drone application that addresses the complexity of interactions required. Moreover, the implementation phase is usually the bottleneck of the entire process; every small change to the interaction model can result in days or weeks for implementing the desired behavior.

One of the most significant challenges during the programming of HDI is the testing phase. Modern drones are usually fragile and expensive, while tests are likely to fail. This phase usually introduces a high consumption of resources, both in terms of costs and time needed for repairing the entire setup before another attempt.

1.2. The Solution: EasyFly

To overcome all the issues related to the programming of HDI, we introduce EasyFly, a programming environment for drone applications that address all the research needs in the field of HDI.

To better understand the contribution of EasyFly to this research field, let us take a step back and describe the needs of researchers.

The ideal prototype of a programming environment for researchers studying HDI should address and solve all the problems related to developing drone applications used for research. In particular, this prototype should ultimately reduce the time and costs associated with the development phase and increase the research's effectiveness.

The first characteristic of the ideal prototype is to reduce the level of expertise needed to develop the desired drone application. This feature allows researchers to implement all the required functionality easily without requiring a specialized team of drone programmers. Moreover, this feature would open the doors to a brand-new type of research where the users are questioned to interact with the drone programmed by themselves. EasyFly provides this feature by offering a set of simple operations, which indeed allows the creation of very complex behaviors. In addition to this, EasyFly allows programming in a descriptive fashion; in this way, programs would be self-explaining and easily interpreted by anyone.

As in any other field, research on HDI should be dynamic. In other words, to gather all the possible insights from an interaction, the drone application must rapidly change and adapt to the situation. If the application development cycle is too long, there is the risk of losing many possible opportunities to experiment with possible alternative solutions. The ideal prototype should provide the maximum flexibility in adapting to many possible situations. For this reason, EasyFly has adopted a modular approach for both the hardware and the software components. At any moment, a module (either software or hardware) can be attached or detached to compose the best configuration needed at that specific moment.

Last but not least, facilitating the interaction between the human and the drone should be the primary goal. The ideal prototype should be the first promoter of the interaction. It should offer the best possible condition to allow the two entities to establish an interaction safely and free from any potential bias determined by the programming environment.

To implement EasyFly, we have targeted a specific typology of unmanned aerial vehicles: nano-drones. As the name suggests, nano-drones are simply drones with very small size and weight. Their small size makes them the best choice to facilitate human-drone interaction.

In the first place, nano-drones are less intimidating and intrusive than larger drones, making it easier for researchers to observe how individuals react and interact with them. They also allow minimal disturbance in the observation environment, making them perfect for avoiding any possible noise in the experiment. Given that the drone and the human are supposed to work closely together, any possible malfunction can cause an unexpected drone crash, especially while experimenting with new solutions. It is easy to deduce that the smaller the drone is, the safer the interaction. The last observation is that nano-drones are usually less expensive than bigger ones, allowing researchers to experiment with interactions with multiple drones without affecting their budget.

1.3. The Benchmark: Drone Arena Challenge

In the HDI domain, the research's core part, especially from the computer science perspective, is the experimental phase. During this phase, researchers put their ideas and prototypes to test and assess the practicality of innovative interaction models.

For a programming environment like EasyFly, testing and evaluating in a real research scenario in HDI is essential. The testing in real scenarios can help detect possible weaknesses in the programming environment, allowing for fine-tuning the model.

To best evaluate our EasyFly programming environment, we had the possibility to participate in the Digital Futures Drone Arena project. This project allowed us to perform an in-depth analysis of the impact of using EasyFly while developing human-drone interactions.

Drone Arena is an interdisciplinary research project that aims to create a technological and conceptual platform for interdisciplinary investigations of drones at the intersection of mobile robotics, autonomous systems, machine learning, and Human-Computer Interaction. The project has three inter-related objectives:

1. The constructions of a novel aerial drone testbed that is geared towards application-level functionality rather than low-level control mechanisms.
2. The organization of two challenges where multiple teams are involved and tasked to realize functionality that pushes the state of the art.
3. The conduction of empirical investigation of Human Drone Interactions in the Drone Arena. This includes observation, interviews, and micro-sociological video analysis to inform future competitions in the drone arena and to develop insights from the movement-based explorations of drone piloting.

In these settings, our EasyFly programming environment is focused on the first two objectives of the project. In particular, our programming environment was one of the core parts of the novel aerial drone testbed used for the entire duration of the project. For the second objective of the project, we had the possibility to actively participate in the first of the two challenges organized for the Drone Arena project. During this challenge, we conducted a complete and in-depth evaluation of the impact of using EasyFly; in particular, we compared our programming environment with a simpler and lower-level one.

1.4. State of the Art

The field of HDI is an active and evolving area of research with a focus on improving the ways in which humans and drones interact. It is a multidisciplinary field with two main research areas: the technological and the sociological. Each area focuses on distinct aspects of the interaction between humans and drones.

In the technological area, at the intersection of computer science, mobile robotics, autonomous systems, and machine learning, the key focus is developing and improving the hardware and software components of drones and their interfaces. The main goal of this area is to enhance the capabilities and functionalities of drones to make them more user-friendly and efficient.

In the sociological area, which includes disciplines like social engineering, art, ethics, and political science, the core objective is to understand how the presence and use of drones impact society, individuals, and communities.

The most common drone model used in research in the HDI field is the Parrot ARDrone [37]. Parrot drones provide an easy-to-use software API allowing for quick prototyping, which is likely the reason they are the researcher's first choice. Although the Parrot ARDrone is widely used for research, this model was discontinued by the manufacturer.

Especially in research focused on the sociological area, where researchers usually have less familiarity with programming tools, the prototyping phase is the most complex and time-consuming. EasyFly tries to overcome all the issues related to this phase by creating a simple and flexible programming environment for drone applications. Moreover, it tries to offer a new perspective in the investigation of human-drone interactions where the user plays the role of the programmer.

1.5. Overview

TODO at the end

2 | State of The Art

In this chapter we will analyze the literature and background works: we will first examine the more general field Human Computer Interaction (HCI), passing then to the Human Robot Interaction (HRI) and more in details Human Drone Interaction (HDI), what are the purposes, the achievement the limitations of this research. Next we will move to the programming environments for both single drone and swarms of multiple drones, understanding which are the main design paradigm and solutions available. Lastly we will focus on the mostly used Positioning systems that allows to track the position of objects in a 3d space, with particular focus on indoor positioning systems.

2.1. Human Robot Interaction

We live in an era where humans and computers are increasingly in close contact. In the last 40 years, the continuous and exponential growth in computational power and storage capacity, but even more relevant the huge spread of technologies in every aspect of our life, has moved the concept of Human Computer Interaction to a central stage of research. With the huge amount of new designs of technologies and systems, this relation grows exponentially in complexity, and the importance of understanding it profoundly is a key aspect to bring the innovation to the next step, and more important, to avoid the risk of incurring in bad and undesired situations.

Human Computer Interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them [36]. The role of this discipline is to understand in depth the relation between humans and computers, providing the guidelines that allow us to build better user interfaces. On the other side, HCI must also understand the limits and the risks associated with those new interfaces, that should guide governments around the world to regulate and control the expansion of such technologies in a sane manner but avoid limiting their potentiality.

Regarding our work, it is advisable to narrow down the concept of HCI to a subfield of research: Human Robot Interaction (HRI). Actually HRI is not only a subfield of the more general HCI, instead it is a multidisciplinary field with the influence of HCI, Artificial Intelligence, Natural language understanding and social science. The basic goal of HRI is to define a general human model that could lead to principles and algorithms allowing more natural and effective interactions between humans and robots [21].

2.1.1. The Interaction Between Human and Robot

The Human is an extremely sophisticated biological system characterized by an impressive complex and powerful brain that is the main coordinator and central actor in the whole human system. Given this complexity, for our purpose, we can model it using the Model Human Processor (MPH) proposed by Card, Moran and Newell in 1983 [15].

MPH describes the Human as composed of 3 subsystems: the perceptual system, the motor system and the cognitive system. Each of them has a processor and memory. Input in humans occurs mainly through the senses and output through the motor controls of the effectors[19]. Therefore, vision, hearing and touch are the most important senses in HRI, while fingers, voice, eyes, head and body position are the primary effectors.

The computer, on the other hand, is a simple machine that processes the input data that it receives into outputs. The processor is the main actor in the data transformation process: it is able to perform arithmetic and logic operations very fast. Both input and output data for the computer are encoded binary sequences.

The Interaction, with or without a robot, is a process of information transfer between two (or multiple) systems. The outputs of one system are the inputs for the other and vice versa. In the interaction It is immediately evident that the outputs of the human are not compatible with the input of the robot, and also, the outputs of a robot are not easily interpretable by a human. This profound discrepancy between the two is the domain of study for HRI, the user outputs needs to be translated from body parts motion and voice to binary sequences and, on the other way, the robot outputs need to be translated from binary data to images, text and sounds.

Initially, with computers becoming accessible to the public fifty years ago, command-line interaction was the only option for interacting with a computer. Today, interfaces have evolved into sophisticated forms such as advanced GUI, touchscreens, Augmented Reality, Virtual Reality, and Haptic interfaces. All these new interfaces makes on one side easier the life of the humans but, on the other side, they have inevitably brought values and ethics in technology design to the forefront of public debate: questions about the goals and politics of human-designed devices, and whether the social interactions of those devices are good, fair, or just [34].

2.1.2. HRI Taxonomy

Human Robot Interactions is a highly heterogeneous field, understanding the details of interactions between humans and robots is crucial for advancing this dynamic field. To define and better identify the diverse range of interactions between humans and robots, it is important to organize and classify the interactions in a structured taxonomy [38].

Following this taxonomy, HRI can be classified using the following attributes:

Task Type:
It is a high level description of the task to be accomplished. It sets the tone for the system's design and use. Moreover, it implicitly represents the environment for the robot.

Task Criticality:
It measures the importance of getting the task done correctly in terms of its negative effects should problems occur. To mitigate the subjective nature of criticality, it is broken into three categories: high, medium and low.

Robot Morphology:
It can assume three values: anthropomorphic (having a human-like appearance), zoomorphic (having an animal-like appearance), and functional (having an appearance that is neither human-like nor animal-like, but is related to the robot's function). It is an important measure since people react differently based on their appearance.

Interaction Roles:
When humans interact with a robot they can act in 5 different roles: supervisor (monitor the behavior of a robot), operator (control and modify the behavior of the robot), teammate (works in cooperation together to accomplish a task), mechanic/programmer (physically change the robot's hardware or software), and bystander (needs to understand the robot behavior to be in the same space).

Type of Human-Robot Physical Proximity:
When interacting with the robot a human can act at different levels of physical proximity: none, avoiding, passing, following, approaching, and touching.

Decision Support for the Human:
It represents the type of information that is provided to operators for decision support. This taxonomy category has four subcategories: available sensor information, sensor information provided, type of sensor fusion, and pre-processing.

Time and Space:
Divides Human-Robot interaction into four categories based on whether the humans and robots interacts at the same time (synchronous) or different times (asynchronous) and while in the same place (collocated) or in different places (non-collocated).

Autonomy Level and Amount of Intervention:
The autonomy level measures the percentage of time that the robot is carrying out its task on its own; the amount of intervention required measures the percentage of time that a human operator must be controlling the robot.

In Table 2.1 is presented the type of interaction that we are targeting in this thesis work using the taxonomy described above:

Attribute	Values
Task Type	Arbitrary interaction with nano drones in a small indoor environment to investigate Human-Drone relation
Task Criticality	Low
Robot Morphology	Functional
Interaction Roles	Mechanic/Programmer or Operator
Physical Proximity	Any value
Decision Support	Available sensors: [proximity (x, y, z), localization (x, y, z), flow (vx, vy), video]
Time and Space	Synchronous and Collocated
Autonomy Level	Any value
Amount of Intervention	Any value

Table 2.1: Categorization of interaction for target applications in our thesis work following the taxonomy proposed by Yanco and Drury [38]

2.2. Human-Drone Interaction

Drones, also known as unmanned aerial vehicles (UAV), are robots capable of flying autonomously or through different control modalities. Until the early 2000s, drones were complex systems commonly seen in the military world and out-of-reach for civilians. Modern advancements in hardware and software technologies allow the development of smaller, easier to control, and lower cost systems. Drones are now found performing a broad range of civilian activities and their usage is expected to keep increasing in the near future. As drone usage increases, humans will interact with such systems more often, therefore, it is important to achieve a natural human-drone interaction.

Human-drone interaction (HDI) can be defined as the study field focused on understanding, designing, and evaluating drone systems for use by or with human users [37]. Although some knowledge can be derived from the field of HRI, drones can fly in a 3D space, which essentially changes how humans can interact with them, making human-drone interaction a field of its own. This field is relatively new in the research community, but in the last few years the number of publications about HDI has grown exponentially.

2.2.1. The Role of the Human During the Interaction

One of the core topic that compose the field of HDI is the role of the human during the interaction with the drone. Depending on the drone's application and its level of autonomy, humans can play different roles when interacting with drone systems.

When the user pilots the drone to accomplish a given task, by directly controlling the drone through a control interface, the user is considered as an *active controller* of the interaction. In these settings, the role of the user is crucial to complete the given task, the drone instead acts as a mere executor of instructions. Examples of this type of interaction are waypoint navigation [23] or artistic exhibitions [20].

The user act as a *recipient* when he/she does not control the drone, but he/she benefits from interacting with it. Example of this type of interaction are represented by delivery droneshoppe2019droneOSz [10, 35].

Another type of interaction roles is when the drone act as a *social companion* for the user. In this case the user might or might not be able to control the drone movement, but it holds a social interaction with it. An example of this type of interaction is represented by Joggobot [22], a drone used as companion for jogging.

Last type of role that the user can act when interacting with a drone is the role of *supervisor*. Autonomous drones require users to act as supervisors either to pre-program the drone behavior or to supervise the flight itself in case of emergency. In this case examples can be crop monitoring [18] or aerial photogrammetry [29],

2.2.2. The drone's control modality

Usually drones expose a control interface that allows users to control their behavior and eventually complete some task in the application domain. Each control interface impacts how the pilot interacts with the drone in various aspects, such as training period, accuracy, latency, interaction distance.

As drones became available to the wide public, the major drone's producers felt the need to change their control modality from the standard remote controller, to a more natural and easy to use interface. A wide variety of control interfaces are available on the market today, ranging from standard remote controllers to very complex and advanced Brain Controlled Drones [26].

Drone's control interfaces can be classified as follows [37]:

Remote Controller is the standard and most commonly used interface, where the user directly control the drone movements. This control modality provides a low-latency and precise control but on the other side it is less intuitive and usable than natural user interfaces. The usability and easiness of this interface is strongly dependent on the level of autonomy of the drone.

Gesture based interface is a control modality where the user pilots the drone with movements of their body. Usually the drone uses a camera or a Kinect device to extract spatial information and recognize postures. When users are asked to interact with a drone without any instruction, gesture interaction is the primary choice of most users, this indicates that the training period of this interface is almost close to zero [16]. Compared to other control modalities gesture based has a high latency and lower control precision. The flight space for drones that use this control method is sensibly reduced since the pilot needs to be close to the drone during the flight.

Speech based interface is a control modality where the user pilots the drone using vocal commands. As for gesture based, also this interface is a natural user interface with low training period and high usability. As a gesture based interface, speech also has problems of user proximity and high latency of commands.

Touch based interface is a control modality where the user is requested to control the drone using his hands. The drone usually carries proximity sensors that allow it to receive inputs from the user. It is a natural user interface and as the others it has the same pros and cons.

Brain Computer interfaces allows the user to pilot the drone using brain signals [26]. To enable this type of interface the pilot must wear some form of BCI headset, the most common being Electroencephalography (EEG) headsets. These devices measure the brain's electrical activity on a human's scalp, which are decoded using machine learning algorithms to control physical systems using brain-waves. Compared to the others is the most complex control interface and with the highest accessibility also for users with disabilities. The problems in using BCI is the poor quality of the control and higher training period. Further research in this field will probably lead to more usable and better interfaces.

Interactions can also be combined into *multimodal interfaces*. Integrating different interaction methods can combine the advantages of each, however it can increase a lot in complexity and costs.

2.2.3. Values and ethics in HDI

Drones usually carry cameras, and potentially they can fly wherever they want, this introduces a lot of issues of privacy [11]. In recent years, given the really rapid expansion of such technology, governments all around the world have been caught off guard. Governments tried to quickly create rules and regulations to control the usage of such technology. This rapid regulation has, in most cases, limited too much, slowed the expansion, and reduced the potential of drones. The research in ethics and values about HDI has the responsibility to produce accurate and reliable results that should guide governments in refining and upgrading laws on drones.

2.3. Programming Environments

When developing drone applications, the choice of programming environment is crucial to ensure efficient and reliable software. The programming environment is intended as all the resources hardware and software used to accomplish the tasks of the application scenario being developed.

As drone technology expanded, many companies working in the drone's field started developing and selling their programming environment. Nowadays most of the software resources for drone applications are open source and publicly available. We will dive into the scenario of the programming environment for drones, understanding what are the most common and popular solutions for developing a drone application. We will then analyze the more complex scenario of swarm applications.

2.3.1. Single Drone Programming

Exploring the programming landscape for a single drone involves navigating through specialized tools and frameworks specialized for the development and control of individual drones. Whether managing a custom-built drone or utilizing a commercial off-the-shelf

(COTS) model, creating effective combination of hardware and software is crucial. This section will guide you through essential components and considerations for developing software that governs a drone's flight and functionality.

The development of any drone application can be categorized into two main areas: on-board and off-board the drone.

The on-board area comprises all the hardware and the software that composes the drone itself. As we can see in Figure 2.1, usually the hardware of the drone includes: The chassis of the drone, the motors and propellers, the battery and the power distribution unit, the computing unit and the memory unit, the communication unit and the sensors unit.

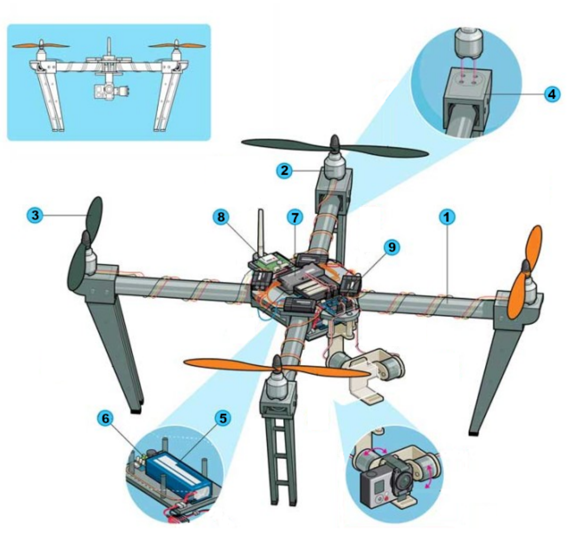


Figure 2.1: The main hardware components of a drone are: 1. Drone's chassis, 2. Motors, 3. Propellers, 4. Motor mount, 5. Battery, 6. Power distribution unit, 7. Computing and memory unit, 8. Communication unit, 9. Sensors unit

The software that runs on the drone, also known as the autopilot software, is usually composed of four main components: the communication unit, the sensing unit, the core control loop unit and the low level control unit. In Figure 2.2 we can see how the software components cooperate together to achieve a controllable and stable flight: the Communication Unit receive and decode commands from off-board system, the signal is then transformed into power set-points from Core Unit (control loop) with the help of sensors information. The Sensing Unit gather information from the environment using sensor, translate sensor readings into readable values, then send this information to the Core Unit and to the Communication Unit to send back telemetry data to off-board systems.

The current landscape of on-board drone solutions ranges from commercial off-the-shelf (COTS) to entirely custom solutions. Commercial producers of drones like Parrot [6], DJI [4], 3DR [1], usually sell COTS solutions where all the hardware resources are supplied with the software needed to run the drone. Depending on the application, these bundled

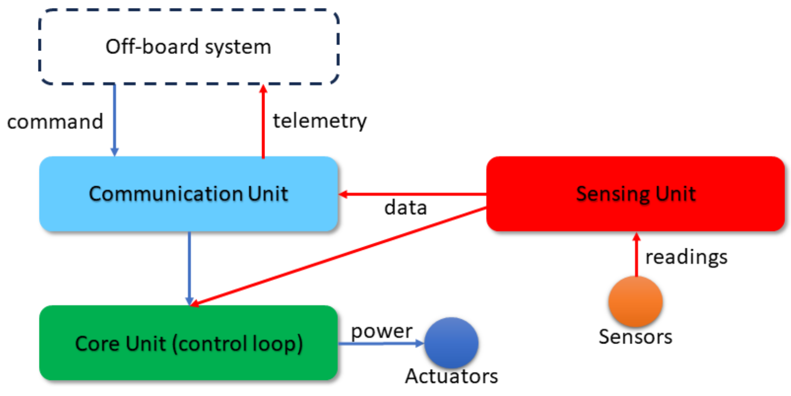


Figure 2.2: The main software components of a drone are: the *Sensing Unit*, the *Communication Unit* and the *Core Unit (control loop)*.

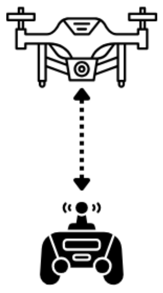
solutions may not be enough, if this is the case, the developer then needs to manually select each hardware component, control the compatibility one with each other, and then select (or develop) also the software that allows the drone to fly.

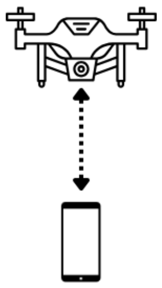
Some producer sell also intermediate solutions [3, 5, 7, 8] between COTS and the completely custom one, these solutions are composed of a microcontroller with usually the basic sensors that compose the Inertial Measurement Unit (IMU) and the autopilot software. These solutions are then extensible with other custom hardware, they provide programming tools to program the behavior of the drone during the flight.

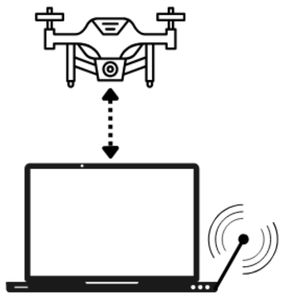
Regarding our setup, the on-board system that we used is a nano drone named Crazyflie 2.1 produced by Bitcraze, it is a COTS solution but with a lot of space for customization for both hardware and software components.

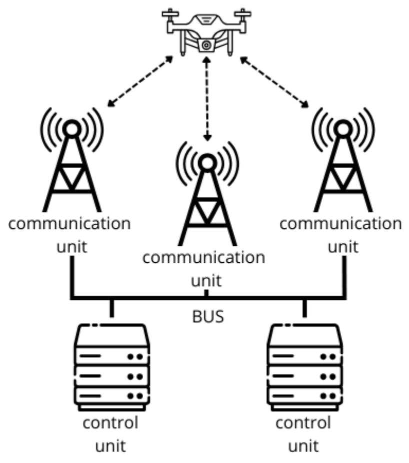
Off-board the drone, the environment is strongly related to the application scenario, in particular is dependent from the level of autonomy request for the drone, the flight area dimension and the complexity of the operation.

Despite the heterogeneity of off-board systems, in most of the scenarios we can consistently identify two main components: a control unit and a communication unit. In most common situation, control and communication units are hosted on a single device. Example of these devices are remote controls (Figure 2.3a), smartphones (Figure 2.3a) or a computer that act as base (ground) station 2.3c. When the scenario is more complex and the fight area is very broad, we can have a distributed ground network of control and communication units (Figure 2.3d). Additionally, in combination with a distributed ground network can also be deployed a sensor network to gather more information of the drone in its environment(Figure 2.3e). For example the sensor network can be composed of senors that collects atmospheric data allowing for a better knowledge of the environment in which the drone is deployed.

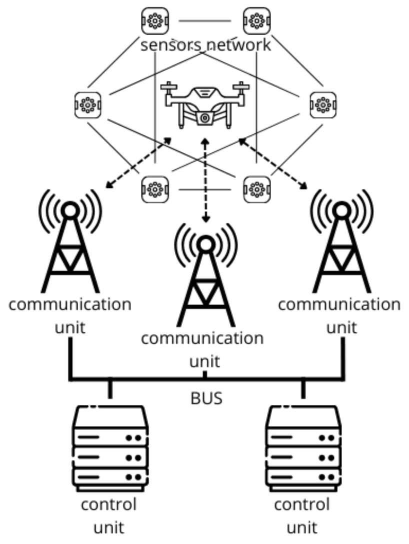

(a) Remote control


(b) Smartphone


(c) Base Station



(d) Distributed ground network



(e) Distributed with sensor network

Figure 2.3: Off-board ecosystem

The communication technology and infrastructure is a critical topic that can introduce potential issues in the off-board environment, when is inadequate for the application scenario. In particular, depending on the dimension, location and topography of the flight area, an appropriate technology and infrastructure of communication must be deployed in order to have a properly working drone.

The infrastructure of the communication, as highlighted in Figure 2.3, can be single or distribute. The former is simpler to implement and deploy but can be not enough when the flight area is too wide or the topography is irregular; the latter is much more complex but allow to cover all the possible application scenarios.

The most commonly used communication technology in the drone's filed are Wi-Fi, radio, Bluetooth and cellular network [31]. In Table 2.2, the most common communication technologies are summarized along with their characteristics. Even if the research frontier for drone communication is mostly focused on cellular network, in particular the 5G network [33], none of the technologies prevails, but the choice depends on the application scenario. Cellular network can be a very great solution in around highly populated areas, while in rural location another technology must be considered.

Technology	Range	Weight	Complexity	Cost
Wi-Fi	MED [100m]	MED	HIGH	MED
Radio	SHORT-LONG [10-1000m]	LOW	LOW	LOW
Bluetooth	SHORT-MED [<25m]	LOW	MED	LOW
Cellular	LONG [8000m]	LOW	HIGH	MED

Table 2.2: Communication technologies used in drone applications [31]

The software that runs off-board is usually apt to coordinate all the resources of the environment to finally achieve and complete the task needed for the application. When using COTS or intermediate solution, usually the vendors provide also the hardware and software that compose the off-board ecosystem.

In Figure 2.4 is shown the off-board environment used in this work. It consists of a single base station with a USB dongle radio and an external positioning system (Lighthouse positioning system) composed of two sensors.

A very common problem encountered while dealing with drones is the testing phase of the application in a real environment. As in any other programming environment, drone programming is not immune to bugs in the code or problems with the hardware. Unfortunately, when an error arises, usually, the drone will crash and in some unfortunate case some parts will break and need to be replaced. Therefore, testing drones application is a very expensive task both in terms of economic resources and in terms of time.

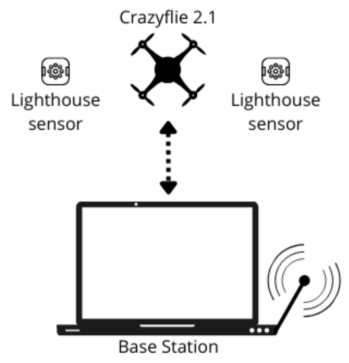


Figure 2.4: EasyFly off-board ecosystem

To overcome this limitation the main drones producers have developed and distributed a simulation environment that allows testing the software before going on a real scenario. A simulation environment allows to run the code written for the planned mission in a graphical simulation that shows the drone performing the tasks. Modern simulation environment [2, 9] usually takes also in consideration atmospheric phenomena like pressure and wind to make the simulation closer to the real deployment environment.

In our work we built our custom simulation tool, allowing us for a better evaluation of the work itself and, moreover, allowing users of our programming environment to have all the benefits in the use of a simulation environment.

2.3.2. Swarm Programming

In modern drone applications, where the task to achieve are complex, and the application have to be reliable, a common approach is to deploy multiple drone (a swarm) to collectively complete the requested task. This approach is completely different from the single drone programming, in fact, the swarm programming introduces new challenges and a different approach to achieve the tasks of the application.

Swarm programming is a branch of the more general Swarm Engineering which tries to take advantage of using multiple resources to achieve the application goal with better performance.

Swarm Robotics and more in general Swarm Engineering is an emerging discipline that aims at defining systematic and well-founded procedures for modeling, designing, realizing, verifying, validating, operating, and maintaining a swarm robotics system. Taking inspiration from the self-organized behaviors of social animals, it makes use of simple rules and local interactions, for designing robust, scalable, and flexible collective behaviors for the coordination of large numbers of robots. The inspiration that swarm robotics takes from the observation of social animals (ants, bees, birds, fish, . . .) is that starting from simple individuals, they can become successful when they gather in groups, exhibiting a sort of swarm intelligence [14].

In particular, the behavior of groups of social animals appears to be robust, scalable, and flexible. Robustness is the ability to cope with the loss of individuals. In social animals, robustness is promoted by redundancy and the absence of a leader. Scalability is the ability to perform well with different group sizes. The introduction or removal of individuals does not result in a drastic change in the performance of a swarm. In social animals, scalability is promoted by local sensing and communication. Flexibility is the ability to cope with a broad spectrum of different environments and tasks. In social animals, flexibility is promoted by redundancy, simplicity of the behaviors and mechanisms such as task allocation.

Two great example in the current literature of swarm engineering are Proto [13] and Meld [12]. The former is a spatial computing language that allows programming swarms of robots starting from a mathematical model called amorphous medium. The latter is a declarative programming language that uses logic programming to enable swarm programming. Both of the language takes directly the swarm programming from the point of view of aggregate behaviors, i.e., their approach is to program the entire swarm behavior instead of programming each single component separately.

When swarm robotics is applied in the field of drones, given the high dynamism in the movements of this type of robots, the swarm management and control is much more complex with respect to the single drone, but the capability of the swarm may increase the application's performance.

In first place the swarm compared to the single drone can provide a higher availability: a single drone has a limited time of flight, then its batteries need to be recharged or replaced. A swarm instead can dynamically deploy and retire drones to be always active on the field. In addition, a swarm can also scale when the request increases e.g., a phenomenon to sense has a peak of occurrence [18]. In the same situation, a single drone application can miss the peak of the phenomena because for example it can be stuck at the charging station.

With swarms, usually the goal of the application is defined as a swarm goal. Swarm goals are high-level goals, which achievement is independent of the success or the failure of the single task of a swarm component. The separation between application (swarm) goals and drone tasks allows the application to be scalable and fault-tolerant. Of course the advantages of the swarm with respect to the single drone hide inside a huge complexity. As the number of components of the swarm increases, complexity in managing all of them increases a lot as well, introducing a consistent overhead that can affect the performance of the application. As in any other engineering problem, we need to select and identify the most suitable swarm size for the specific application to realize.

Depending on the application domain, we can adopt different strategies in coordinating the resources available. We can identify three main programming models that can be applied to swarm programming in the field of drones: Drone level programming, Swarm programming and Team level programming. Drone level programming is the most straightforward approach; it expects to develop a single application for each component of the swarm, taking into account all the possible interactions between them. This is the finest grain method that allows a completely independent, customizable, and deterministic single-drone behavior. Since the application has a swarm goal that is not directly related to the single drone's task, with this method, it is usually difficult to use all the swarm resources efficiently to reach the general goal. Moreover, it has been proven [18, 28] that this method is indeed the most complex between the three.

Swarm programming[32], on the other hand, allows writing a single set of rules that are common for all drones and then each single drone executes that instruction in its local state. The swarm programming model explicitly forbids a shared or global state. This programming model is easier to use and to set up and scale up with multiple drones, but it is difficult to represent tasks that require explicit drone coordination.

Team level programming [28] is a programming model in between swarm and drone level programming, in which users express sophisticated collaborative sensing tasks without resorting to individual addressing and without being exposed to the complexity of concurrent programming, parallel execution, scaling, and failure recovery. More generally Voltron [28] can be viewed as a set of tasks that have to be performed by a set of drones subject to particular timing and spatial constraints.

In our work we do not address the complexity of swarm programming although our programming environment EasyFly allows the deployment of multiple drones.

3 | Tools

The main resource used in this work is Crazyflie, an open platform produced by Bitcraze [Bitcraze] that offers an ecosystem of products and open source libraries that allows people to develop new functionalities for aerial drones. The key feature of this platform is that it offers a set of expansion decks that can extend the drones capabilities with new sensors. Expansion decks can be mounted on the drones in a very easy way and they are immediately ready for being used to compose the desired configuration in the application of interest. Given its modularity and high versatility, this platform perfectly fits as a baseline for developing an easy and high level environment for programming drones. In this chapter we present an overview of the Crazyflie platform to give a basic knowledge of its tools and its surrounding environment. We will first present all the hardware used in this work, then we will give an overview of all the software libraries that compose the platform and finally we will describe in detail the main software components.

3.1 Ecosystem Overview The Crazyflie platform is composed of a set of devices and tools FIG 3.1, it has a modular architecture that makes it possible to build very versatile systems, adaptable to many possible situations. The ecosystem of this platform gravitates around its main actor: the Crazyflie nano drone. The drone brings with itself the minimal set of sensors and actuators that allows it to fly. To empower the drone capabilities, the platform makes available a set of expansion decks which gives the drone additional sensors, making it possible to adapt it to many possible situations. To coordinate the drone's operations, the platform needs a ground station that can be hosted on any computer with a python script interpreter or on a mobile phone with installed a dedicated App. The communication between the quadcopter and the ground station is handled using a dongle usb (CrazyRadio) or through Bluetooth when using the mobile App. The platform also offers some absolute positioning system that allows the drone to have a better position estimation in an absolute coordinate system. 3.2 Hardware In this section we will provide an overview on the hardware components of the entire Crazyflie platform. We will firstly analyze the characteristics of the Crazyflie quadcopter used in this work and then we will briefly introduce the relevant expansion decks. We will then give an overview on the hardware components that compose the absolute positioning system adopted for the work:

the Lighthouse positioning system. 3.1.1 The Quadcopter Bitcraze produces a family of drones with similar hardware and firmware, but with differences in size and properties. The target for this work is the principal component of this family, the Crazyflie 2.1, an incredibly small and versatile quadcopter with a solid and modularized design that falls into the category of nano drones. Since our programming environment is meant to be used also by non-expert users, we need to target the most robust drone and with a very easy set up. For this reason we have targeted nano drones for our work and in particular the Crazyflie 2.1. Moreover given its size it is also possible to set up a swarm of drones with ease. This drone consists of many hardware components that are hosted on a single, compact and light base. We can identify four main unit: Computing unit Motor unit Sensor unit Power unit The core of the drone is composed of two Micro Controller Units (MCUs): the first is an STM32F4 MCU that handles the main Crazyflie firmware with all the low-level and high-level controls. The second MCU is a NRF51822 that is in charge of handling all the radio communication and power management. The motor unit of the Crazyflie 2.1 consists of four Coreless DC motors with plastic propellers, that are fixed at the corners of the base with the help of plastic supports. To control the flight the drone is equipped with two sensors: a BMI088 sensor which measures the acceleration along the 3 coordinate of the space plus the angular speed, and a BMP388 sensor that is an high precision pressure sensor. The sensor unit of the Crazyflie 2.1, known also as Inertial Measurement Unit (IMU), is minimal and provides the minimum data that allows the drone to have an almost stable flight. Finally the power unit is constituted by a 240mAh LiPo battery that allows a flight duration of about 7 minutes. The total weight of the drone is 27 grams and is able to lift a payload of 15 grams. Its design is robust and simple, easing the assembly and the maintenance of its components. 3.1.2 The Expansion Decks The drone itself, with only 2 sensors composing the IMU, has a limited capacity of understanding the surrounding environment. To overcome this limitation, it can be equipped with additional decks that extend its capabilities in sensing, positioning and visualization FIG 3.2. The platform offers a variety of expansion decks, but for the purpose of our work only a subset of them has been selected, in particular, those that can enable in some way in the human-drones interaction. Here we will list all the expansion decks used in this work with a brief description for each of them. Flow deck v2 The Flow deck gives the Crazyflie the ability to understand when it's moving in any direction. It mounts two sensors: the VL53L1x ToF measures the distance to the ground with high precision up to 4 meters and the PMW3901 optical flow sensor measures the relative velocity in the x-y plane in relation to the ground. This expansion deck is a relative positioning system that allows the drone to know its position relative to its take off point. Lighthouse positioning deck The Lighthouse deck is part of the Lighthouse absolute positioning system (See section 3.1.4). It is composed of four TS4231 IR receivers and a ICE40UP5K FPGA to process the signal received. This expansion deck allows the drone to know its position in an absolute coordinate system. Multi-ranger deck The Multiranger deck gives the Crazyflie the capability to sense the space around it and could react when something is close and for instance avoid obstacles.This is done by measuring the distance to objects in the following 5 directions: front, back, left, right and up with mm precision up to 4 meters, using five VL53L1x ToF sensors. Z-ranger deck v2 The Z-ranger deck is a simplified and cheaper version of the Flow deck v2, it gives only the measurement of the distance from the floor up to 4 meters using the usual laser sensor VL53L1x ToF. AI-deck 1.1 The AI-deck 1.1 extends the computational capabilities and will enable complex artificial intelligence-based workloads to run onboard, with the possibility to achieve fully autonomous navigation capabilities. It mounts an Himax HM01B0 (ultra low power 320×320 monochrome camera), GAP8 (ultra low power 8+1 core RISC-V MCU), NINA-W102 (ESP32 module for WiFi communication) and it has 512 Mbit HyperFlash and 64 Mbit HyperRAM memories. LED-ring deck The LED-ring deck has 12 RGB LEDs disposed of in a ring (W2812B module) each of which can be controlled independently to build colorful animation. In addition it has 2 more LEDs facing front that emit more than 1800 mcd helping in light up the environment.

3.1.3 Crazyradio PA As previously described, the Crazyflie platform expects two nodes of computation: the ground station and the Crazyflie 2.1 itself. To communicate with the Crazyflie 2.1 which has its integrated radio, the ground station needs an external radio dongle. The platform provides a low-latency and long range USB radio dongle, the Crazyradio PA. The CrazyRadio PA is based on the nRF24LU1+ from Nordic Semiconductor and it features a 20dBm power amplifier giving a range up to 1km (line of sight). The dongle comes pre-programmed with Crazyflie compatible firmware. The communication protocol used to communicate is the Crazy Radio Transfer Protocol (CRTP) which is a custom communication protocol of the Crazyflie platform (See section 3.7). 3.1.4 Lighthouse Positioning System Hardware The Lighthouse positioning system is one of the possible solutions that Bitcraze offers to have an absolute positioning system for understanding the drone's coordinates inside the flight space. The hardware of this system is composed of 2 or more HTC-Vive/SteamVR Base Station 2.0. The role of these Base Stations is to light up the flight space with periodic InfraRed (IR) beams. Onboard the Crazyflie 2.1, the Lighthouse expansion deck allows it to capture these IR beams thanks to four TS4231 IR receivers. The signal captured is then passed to an ICE40UP5K FPGA that with signal processing it computes the angle of incidence of the IR rays. The position and the pose of the crazyflie is then finally computed from the main MCU of the Crazyflie

2.1 (See section 3.4.x). 3.3 Software libraries As previously anticipated, the Crazyflie environment is completed by a set of open source libraries, publicly available on GitHub, which allow people to program all its components and devices, to develop new features or upgrade existing ones. Each library targets a specific component of the system and is completely independent from all the others. In this section we will briefly describe the two main libraries that have been used as baseline for our work. 3.3.1 crazyflie-lib-python The crazyflie-python-lib (cflib in short) [crazyflie-lib-python] is a software repository which consists of a python library for programming scripts which control the behavior of the Crazyflie 2.1. The library provides the base facilities to allow users to define in a python script the desired behavior of the drone, abstracting from the low level control mechanism As shown in figure FIG 3.3, the python scripts are executed on the ground station, the library contains the code to create communication packets that are sent through the Crazyradio reaching the Crazyflie which will eventually execute the commands requested.

3.3.2 crazyflie-firmware The crazyflie-firmware [crazyflie-firmware] is a software repository that contains all the firmware of the Crazyflie 2.1. The firmware is written in C++ and it handles the main autopilot on the STM32F4, it contains the driver of each possible expansion deck and controls all the communication on the opposite side with respect to the cflib.

Fig 3.3

3.4 Positioning systems Positioning systems represent the core sensing task of every drone application. Knowing the position of the drone in the flight space is essential for achieving any possible goal. The Crazyflie platform offers multiple positioning systems, both absolute and relative (See section 2.5). To select the best system for our application we analyzed the following metrics for each possibility that the platform offered: Relative or Absolute Positioning system Accuracy in sampling Cumulative of the error during time 3.4.1 Relative Positioning Systems The Crazyflie platform offers 3 relative positioning systems. The first system is represented by the Inertial Measurement Unit which is provided on the base drone without expansion decks. This positioning system has a very poor accuracy and a cumulative error even worse. From our experience this system cannot be used as the only positioning system of the entire application, however it contributes with its information to obtain the position estimate. The Z-ranger deck is another positioning system that the platform offers, it provides an estimate only for the z-coordinate with a quite good accuracy but, as typical for every relative positioning system, it has a high cumulative of the error during time. An empowered version of the Z-ranger is represented by the Flow deck which instead is a positioning system capable of measuring the distance from the takeoff point for the three coordinates x, y and z. This last system has a really good accuracy in sampling and an acceptable cumulative of the error during time. Flow deck is the only relative positioning system that allows the crazyflie to fly with acceptable precision in the flight space.

3.4.2 Absolute Positioning Systems As we have seen before relative positioning systems suffer from a high cumulative of the error during time. When the application needs that the estimate of the drone drone position does not drift over time, we need to consider the more robust absolute positioning systems. The environment provides three different absolute positioning systems solutions with different characteristics, performance and costs. The first solution proposed, the Loco Positioning System (LPS) is based on Ultra Wide Band radio that is used to find the absolute 3D position of objects in space. Similarly to a miniature GPS system, it uses a set of Anchors, namely Loco positioning nodes (from 4 up to 8), that acts as a GPS satellite and Tag, namely Loco positioning deck, that acts as a GPS receiver. The accuracy of this system is probably the main limitation and is estimated to be in the range of 10 cm. The second solution proposed is the Motion Capture System (MCS), which uses cameras to detect markers attached to the Crazyflies. Since the layout of the markers on the drone is known by the system, it is possible to calculate the position and orientation of the tracked object in a global reference frame. It is a very accurate positioning system but the two main limitations are: the native environment provides only the markers and it relies on third party systems for the entire MCS. Moreover the position is computed in an external node and it needs to be sent to the crazyflie increasing the communication load. The last solution is the Lighthouse positioning system (Lighthouse), is the newest introduced in the environment and is the one selected for the work because it overcomes all the limitations of the previous. Lighthouse is an optically-based positioning system that allows an object to locate itself with high precision indoors. The system uses the SteamVR Base Station (BS) as an optical beacon. They are composed of spinning drums that shine the flight space with infrared beams in a range of 6 meters. The crazyflie on the other hand, with a Lighthouse positioning deck that has 4 optical IR receivers (photo diode) is capable of measuring the angle of incidence of the IR beams. Knowing the position and orientation of the BS enables the Crazyflie to compute its position onboard in global coordinates. The knowledge of the position and the orientation of the BS is called system geometry and is composed by a vector in the three dimensions and by a rotation matrix. Bitcraze officially supports only up to two BSs, which results in having a flight area of approximately 5x5 meters without losing in accuracy. This can be seen as a little bit constraining but with appropriate modifications the system supports up to 16 BSs covering in practice almost all the indoor situations. As previously anticipated, Lighthouse positioning system overcomes the limitations that LPS and MCS has by taking the best characteristics form each of them, in fact, it computes the position onboard like LPS reducing the communication overload, and it has a very high accuracy comparable to the MCS, moreover is also the cheapest solution between the three choices. 3.5 State estimate and control All the information given by the positioning systems and by the additional decks are simply data, to become useful, they need to be used in a clever way to control the stability of the drone and make it possible to fly. In this section we will describe all the software components that act in the control loop enabling the drone to fly. In the crazyflie platform, the control loop is managed inside the crazyflie firmware from the sensor read to the motor thrust FIG 3.4.

FIG 3.4 3.5.1 Sensors The read of data from sensors is the first step of the process and consists in the collection of the data measured by the sensors available and that are significant to the next step. Of course, the more data is gathered during this step, the more accurate would be the whole process. Given the choices described in the previous section regarding positioning systems and expansion decks, the sensors that are used in this step are: IMU Accelerometer: acceleration in body fixed coordinates x-y-z [m/s2] Gyroscope: angle rate in roll pitch and yaw [rad/s] Pressure Sensor: Air pressure [mBar] Flow Deck v2 ToF sensor: Distance to a surface [mm] Optical flow sensor: The detection movement of pixels in [px/s] Z ranger deck ToF sensor: Distance to a surface [mm] Lighthouse deck IR receivers: Sweep angle of Steam VR base stations 2.0 [rad] 3.5.2 State Estimator The State Estimator, as the name suggests, is the component in charge of computing the state estimate starting from the data read by the sensors. The state to be estimated consists of 4 main variables: Position (x, y, z) [m] Velocity (vx, vy, vz) [m/s] Attitude absolute (roll, pitch, yaw) [rad] Attitude rate (rollrate, pitchrate, yawrate) [rad/s]

This component inside the crazyflie firmware has 2 concrete implementation with different performance and accuracy: Complementary Filter Extended Kalman Filter (EKF) The Complementary Filter is a very lightweight and efficient state estimator. It uses only data coming from the IMU and from the ToF sensor (Flow deck or Z-ranger). The estimated output is only a portion of the crazyflie state: the Attitude and Position for the z coordinate. The EKF is a recursive filter that estimates the current state of the Crazyflie based on incoming measurements (in combination with a predicted standard deviation of the noise), the measurement model and the model of the system itself. It is a step up in complexity with respect to the other estimator, it accepts as input all the possible sensor's data.If enough sensor data are available, it can compute as output a complete state estimation: Attitude, Position and Velocity in all the directions. The choice of which state estimator to use can be either forced by the user or automatically set given the sensors available at boot time. By default the firmware uses the lighter Complementary Filter and switches to the EKF if more sensors are available. 3.5.3 State Controller After the state has been estimated, the next step is to try to understand which actions to take in order to bring the state near to the desired state. This is the task of the State Controller component, which starting from the estimated state and the desired setpoint of the state it outputs the commands for the power distribution. As it was for the estimation stage, also here there are multiple options available: Proportional Integral Derivative controller (PID) The Incremental Nonlinear Dynamic Inversion controller (INDI) Mellinger controller The simplest and lightest controller is the PID, on the opposite the Mellinger is the most complex one. By default the PID controller is used and the user can select to change the controller on the base of his/her particular needs. 3.5.4 Commander The Commander component acts as an interface with the outside world. It is responsible for gathering from all possible sources the setpoints requested and also managing the priority among those sources. Before sending the setpoints to the controller, it checks its validity and, if necessary, discards or modifies it. The setpoint structure is defined by 2 levels of control: Position and Attitude that can be combined with 3 modality of control: mode absolute, mode velocity and mode disabled. In the following table we can see how to combine levels and modality of control to define a setpoint for the desired state variable.

State Variable Position control Attitude control Position mode absolute mode disabled Velocity mode velocity mode disabled Attitude absolute mode disabled mode absolute Attitude rate mode disabled mode velocity

The sources from which the commander can listen for commands can be either a ground station or an internal component named High Level Commander. The High Level commander is a component that generates setpoints starting from a predefined trajectory saved in memory. 3.5.5 Power Distribution and Motors The power distribution component is the component responsible for translating the commands received from the controller into thrust to give to the motors. The Motors will then finally produce a movement that changes the real state of the Crazyflie. 3.6 Flight control On the other side of the communication channel, the ground station needs a way to interact with the drone and give information on which actions to take to fly in the desired way. In this section we will briefly describe the main alternatives that the cflib offers to write scripts that will eventually make some crazyflie fly. Inside the cflib the responsible for controlling the flight is a module named Commander Framework. The Commander Framework can be viewed as composed of 2 layer: The first layer provides low level operations that allow writing setpoints (according to TABLE1) and sending them with the custom CRTP protocol. The second layer, which is built upon the first, is more abstract and adds some general functionalities e.g., take off, land, move to. Inside the Commander framework there is also a parallel version for the 2 layers that is meant for interacting with another type of commander: the High Level commander (See section 3.5.4). This version provides functionality for starting/stopping a pre-loaded trajectory and some operations to modify this trajectory at runtime. To complete the overview on the flight control, the cflib offers also another module, the Swarm module which provides the functionalities to use the Commander Framework on multiple Crazyflie 2.1 simultaneously. 3.7 Communication The Crazyflie platform has its own customized packet protocol: Crazy RealTime Protocol (CRTP). It was designed to allow packet prioritization to help real-time control of the Crazyflie. Each CRTP packet carries 32 bytes: one port number between 0 and 15 (4 bits), a channel number between 0 and 3 (2 bits) as well as a payload data buffer of up to 31 bytes. The link layer of the protocol guarantees strict packet ordering within a port whereas for different ports, packets can be re-organized and sent out of order. All communication links are identified using an URI build up of the following: InterfaceType://InterfaceId/InterfaceChannel/InterfaceSpeed/Address e.g., radio://0:80:2M:E7E7E7E7E7. Built up on this communication protocol, the Crazyflie implements a logging and a parameters framework that allows the user to read and modify the information on the crazyflie. Logging allows the user to define inside a log configuration a set of state variables to be logged periodically with a certain period. Each log configuration has a maximum size of 26 Bytes and the minimum period allowed is 10 milliseconds. Parameters are instead configuration variables that can be read and/or written to the crazyflie. Although it may seem that the two concepts overlap because they can both read variables, in reality they are profoundly different since logging tackles variables used to know periodically the current state of the crazyflie that usually changes during the flight instead parameters address configuration variables that are usually set before take off and then rarely changes. To avoid any possible ambiguity, the environment provides two Table of Contents (ToC), one for each framework, that contains all the possible variables that each framework can work on, grouped by functionalities. Both the 2 framework works asynchronously and so they allow to set up a callback function that will be called: when the values of the associated variable changes for the parameter framework; periodically according to the associated configuration log's period instead for the logging framework.

4 | Chapter one

In this chapter additional useful information are reported.

4.1. Sections and subsections

Chapters are typically subdivided into sections and subsections, and, optionally, subsubsections, paragraphs and subparagraphs. All can have a title, but only sections and subsections are numbered. A new section is created by the command

```
\section{Title of the section}
```

The numbering can be turned off by using `\section*{}`.
A new subsection is created by the command

```
\subsection{Title of the subsection}
```

and, similarly, the numbering can be turned off by adding an asterisk as follows

```
\subsection*{}
```

4.2. Equations

This section gives some examples of writing mathematical equations in your thesis.

Maxwell's equations read:

$$\begin{cases} \nabla \cdot \boldsymbol{D} = \rho, & \text{(4.1a)} \\ \nabla \times \boldsymbol{E} + \dfrac{\partial \boldsymbol{B}}{\partial t} = \boldsymbol{0}, & \text{(4.1b)} \\ \nabla \cdot \boldsymbol{B} = 0, & \text{(4.1c)} \\ \nabla \times \boldsymbol{H} - \dfrac{\partial \boldsymbol{D}}{\partial t} = \boldsymbol{J}. & \text{(4.1d)} \end{cases}$$

Equation (4.1) is automatically labeled by `cleveref`, as well as Equation (4.1a) and Equation (4.1c). Thanks to the `cleveref` package, there is no need to use `\eqref`.

Remember that Equations have to be numbered only if they are referenced in the text.

Equations (4.2), (4.3), (4.4), and (4.5) show again Maxwell's equations without brace:

$$\nabla \cdot \boldsymbol{D} = \rho, \tag{4.2}$$
$$\nabla \times \boldsymbol{E} + \frac{\partial \boldsymbol{B}}{\partial t} = \mathbf{0}, \tag{4.3}$$
$$\nabla \cdot \boldsymbol{B} = 0, \tag{4.4}$$
$$\nabla \times \boldsymbol{H} - \frac{\partial \boldsymbol{D}}{\partial t} = \boldsymbol{J}. \tag{4.5}$$

Equation (4.6) is the same as before, but with just one label:

$$\begin{cases} \nabla \cdot \boldsymbol{D} = \rho, \\ \nabla \times \boldsymbol{E} + \dfrac{\partial \boldsymbol{B}}{\partial t} = \mathbf{0}, \\ \nabla \cdot \boldsymbol{B} = 0, \\ \nabla \times \boldsymbol{H} - \dfrac{\partial \boldsymbol{D}}{\partial t} = \boldsymbol{J}. \end{cases} \tag{4.6}$$

4.3. Figures, Tables and Algorithms

Figures, Tables and Algorithms have to contain a Caption that describe their content, and have to be properly reffered in the text.

4.3.1. Figures

For including pictures in your text you can use `TikZ` for high-quality hand-made figures, or just include them as usual with the command

```
\includegraphics[options]{filename.xxx}
```

Here xxx is the correct format, e.g. `.png`, `.jpg`, `.eps`, ...

Figure 4.1: Caption of the Figure to appear in the List of Figures.

Thanks to the `\subfloat` command, a single figure, such as Figure 4.1, can contain multiple sub-figures with their own caption and label, e.g. Figure 4.2a and Figure 4.2b.

(a) One PoliMi logo. (b) Another one PoliMi logo.

Figure 4.2: This is a very long caption you don't want to appear in the List of Figures.

4.3.2. Tables

Within the environments `table` and `tabular` you can create very fancy tables as the one shown in Table 4.1.

Title of Table (optional)

	column 1	**column 2**	**column 3**
row 1	1	2	3
row 2	α	β	γ
row 3	alpha	beta	gamma

Table 4.1: Caption of the Table to appear in the List of Tables.

You can also consider to highlight selected columns or rows in order to make tables more readable. Moreover, with the use of `table*` and the option `bp` it is possible to align them at the bottom of the page. One example is presented in Table 4.2.

	column1	**column2**	**column3**	**column4**	**column5**	**column6**
row1	1	2	3	4	5	6
row2	a	b	c	d	e	f
row3	α	β	γ	δ	ϕ	ω
row4	alpha	beta	gamma	delta	phi	omega

Table 4.2: Highlighting the columns

	column1	**column2**	**column3**	**column4**	**column5**	**column6**
row1	1	2	3	4	5	6
row2	a	b	c	d	e	f
row3	α	β	γ	δ	ϕ	ω
row4	alpha	beta	gamma	delta	phi	omega

Table 4.3: Highlighting the rows

4.3.3. Algorithms

Pseudo-algorithms can be written in LaTeX with the `algorithm` and `algorithmic` packages. An example is shown in Algorithm 4.1.

Algorithm 4.1 Name of the Algorithm

```
 1: Initial instructions
 2: for $for - condition$ do
 3:     Some instructions
 4:     if $if - condition$ then
 5:         Some other instructions
 6:     end if
 7: end for
 8: while $while - condition$ do
 9:     Some further instructions
10: end while
11: Final instructions
```

4.4. Theorems, propositions and lists

4.4.1. Theorems

Theorems have to be formatted as:

Theorem 4.1. *Write here your theorem.*

Proof. If useful you can report here the proof.

4.4.2. Propositions

Propositions have to be formatted as:

Proposition 4.1. *Write here your proposition.*

4.4.3. Lists

How to insert itemized lists:

- first item;
- second item.

How to insert numbered lists:

1. first item;
2. second item.

4.5. Use of copyrighted material

Each student is responsible for obtaining copyright permissions, if necessary, to include published material in the thesis. This applies typically to third-party material published by someone else.

4.6. Plagiarism

You have to be sure to respect the rules on Copyright and avoid an involuntary plagiarism. It is allowed to take other persons' ideas only if the author and his original work are clearly mentioned. As stated in the Code of Ethics and Conduct, Politecnico di Milano *promotes the integrity of research, condemns manipulation and the infringement of* *intellectual property*, and gives opportunity to all those who carry out research activities to have an adequate training on ethical conduct and integrity while doing research. To be sure to respect the copyright rules, read the guides on Copyright legislation and citation styles available at:

`https://www.biblio.polimi.it/en/tools/courses-and-tutorials`

You can also attend the courses which are periodically organized on "Bibliographic citations and bibliography management".

4.7. Bibliography and citations

Your thesis must contain a suitable Bibliography which lists all the sources consulted on developing the work. The list of references is placed at the end of the manuscript after the chapter containing the conclusions. We suggest to use the BibTeX package and save the bibliographic references in the file `Thesis_bibliography.bib`. This is indeed a database containing all the information about the references. To cite in your manuscript, use the `\cite{}` command as follows:
Here is how you cite bibliography entries: [24], or multiple ones at once: [25, 27].
The bibliography and list of references are generated automatically by running BibTeX [17].

5 | Conclusions and future developments

A final chapter containing the main conclusions of your research/study and possible future developments of your work have to be inserted in this chapter.

Bibliography

[1] 3dr. URL `http://3dr.com/`.

[2] Dij flight simulator. URL `https://www.dji.com/it/simulator`.

[3] Cubepilot. URL `https://www.cubepilot.com/`.

[4] Dij. URL `https://www.dji.com`.

[5] Navio2 emlid. URL `https://docs.emlid.com/navio2/`.

[6] Parrot. URL `https://www.parrot.com`.

[7] Pixhawk. URL `https://pixhawk.org/`.

[8] Px4 autopilot. URL `https://px4.io/`.

[9] Sphinx. URL `https://developer.parrot.com/docs/sphinx/index.html`.

[10] Wing drones. URL `https://wing.com/`.

[11] C. Anderson. How i accidentally kickstarted the domestic drone boom. *Wired Magazine*, 22:2012, 2012.

[12] M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, and P. Pillai. Meld: A declarative approach to programming ensembles. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2794–2800. IEEE, 2007.

[13] J. Bachrach, J. Beal, and J. McLurkin. Composable continuous-space programs for robotic swarms. *Neural Computing and Applications*, 19:825–847, 2010.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[15] S. Card, T. MORAN, and A. Newell. The model human processor- an engineering model of human performance. *Handbook of perception and human performance.*, 2 (45–1), 1986.

[16] J. R. Cauchard, J. L. E, K. Y. Zhai, and J. A. Landay. Drone & me: an exploration into natural human-drone interaction. In *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*, pages 361–365, 2015.

[17] CTAN. BiBTeX documentation, 2017. URL `https://ctan.org/topic/bibtex-doc`.

[18] K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh. Programming micro-aerial vehicle swarms with karma. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 121–134, 2011.

[19] A. Dix. Human–computer interaction: A stable discipline, a nascent science, and the growth of the long tail. *Interacting with computers*, 22(1):13–27, 2010.

[20] S. Eriksson, K. Höök, R. Shusterman, D. Svanes, C. Unander-Scharin, and Å. Unander-Scharin. Ethics in movement: Shaping and being shaped in human-drone interaction. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2020.

[21] D. Feil-Seifer and M. J. Mataric. Human robot interaction. *Encyclopedia of complexity and systems science*, 80:4643–4659, 2009.

[22] E. Graether and F. Mueller. Joggobot: a flying robot as jogging companion. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pages 1063–1066. 2012.

[23] M. Hoppe, M. Burger, A. Schmidt, and T. Kosch. Droneos: A flexible open-source prototyping framework for interactive drone routines. In *Proceedings of the 18th International Conference on Mobile and Ubiquitous Multimedia*, pages 1–7, 2019.

[24] D. E. Knuth. Computer programming as an art. *Commun. ACM*, pages 667–673, 1974.

[25] D. E. Knuth. Two notes on notation. *Amer. Math. Monthly*, 99:403–422, 1992.

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

[27] L. Lamport. *LaTeX: A Document Preparation System.* Pearson Education India, 1994.

[28] L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi. Team-level programming

of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 177–190, 2014.

[29] F. Nex and F. Remondino. Uav for 3d mapping applications: a review. *Applied geomatics*, 6:1–15, 2014.

[30] F. Nex and F. Remondino. Uav for 3d mapping applications: a review. *Applied geomatics*, 6:1–15, 2014.

[31] G. Pantelimon, K. Tepe, R. Carriveau, and S. Ahmed. Survey of multi-agent communication strategies for information exchange and mission control of drone deployments. *Journal of Intelligent & Robotic Systems*, 95:779–788, 2019.

[32] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[33] A. Sharma, P. Vanjani, N. Paliwal, C. M. W. Basnayaka, D. N. K. Jayakody, H.-C. Wang, and P. Muthuchidambaranathan. Communication and networking technologies for uavs: A survey. *Journal of Network and Computer Applications*, 168:102739, 2020.

[34] K. Shilton et al. Values and ethics in human-computer interaction. *Foundations and Trends® in Human–Computer Interaction*, 12(2):107–171, 2018.

[35] S. R. R. Singireddy and T. U. Daim. Technology roadmap: Drone delivery–amazon prime air. *Infrastructure and Technology Management: Contributions from the Energy, Healthcare and Transportation Sectors*, pages 387–412, 2018.

[36] G. Sinha, R. Shahi, and M. Shankar. Human computer interaction. In *2010 3rd International Conference on Emerging Trends in Engineering and Technology*, pages 1–4. IEEE, 2010.

[37] D. Tezza and M. Andujar. The state-of-the-art of human–drone interaction: A survey. *IEEE Access*, 7:167438–167454, 2019.

[38] H. A. Yanco and J. Drury. Classifying human-robot interaction: an updated taxonomy. In *2004 IEEE international conference on systems, man and cybernetics (IEEE Cat. No. 04CH37583)*, volume 3, pages 2841–2846. IEEE, 2004.

A | Appendix A

If you need to include an appendix to support the research in your thesis, you can place it at the end of the manuscript. An appendix contains supplementary material (figures, tables, data, codes, mathematical proofs, surveys, . . .) which supplement the main results contained in the previous chapters.

B | Appendix B

It may be necessary to include another appendix to better organize the presentation of supplementary material.

List of Figures

List of Tables

List of Symbols

Variable	**Description**	**SI unit**
$\boldsymbol{u}$	solid displacement	m
$\boldsymbol{u}_f$	fluid displacement	m

Acknowledgements

Here you might want to acknowledge someone.