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 [41]‘. 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 [39] or 3D mapping applications [34] 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 [41]. 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 of Human-Computer Interaction (HCI), passing then to Human-Robot Interaction (HRI) and, finally, Human-Drone Interaction (HDI), what are the purposes, the achievement, and the limitations of this research. Next, we will move to the programming environments for both single drones and swarms of multiple drones, understanding which are the main design paradigms and solutions available.

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 lives, has moved the concept of Human-Computer interaction to a central stage of research. With the massive amount of new designs of technologies and systems, this relation grows exponentially in complexity, and the importance of profoundly understanding it is a key aspect of bringing innovation to the next step and, more importantly, avoiding the risk of incurring harmful and undesired situations.

Human-Computer Interaction (HCI) 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 [40]. 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, which 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 primary 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 [25].

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 [19].

MPH describes the Human as composed of 3 subsystems: the perceptual, motor, and cognitive systems. 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[23]. Therefore, vision, hearing, and touch are the most important senses in HRI, while fingers, voice, eyes, head and body position are the primary effectors.

On the other hand, the computer is a straightforward machine that processes the input data that it receives into outputs. The processor is the leading actor in the data transformation process: it can 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 of the other and vice versa. In the interaction, it is immediately evident that the human outputs are incompatible with the robot's input. 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 need to be translated from body movements 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 make one side easier the lives of 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 [38].

2.1.2. HRI Taxonomy

HRI is a highly heterogeneous field, and 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 [42].

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 this attribute, it can take three values: 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 essential 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's 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 interact at the same time (synchronous) or different times (asynchronous) and while in the same place (collocated) or 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 [42]

2.2. Human-Drone Interaction

Drones, also known as unmanned aerial vehicles (UAVs), 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, achieving a natural human-drone interaction is crucial.

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 [41]. Although some knowledge can be derived from the field of HRI, drones can fly in 3D space, which essentially changes how humans 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 topics in the field of HDI is the role of humans during interaction with drones. 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 an *active controller* of the interaction. In these settings, the user's role 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 [27] or artistic exhibitions [24].

The user acts as a *recipient* when he/she does not control the drone, but he/she benefits from interacting with it. An example of this type of interaction are represented by delivery droneshoppe2019droneOSz [14, 39].

Another type of interaction role is when the drone acts 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 [26], a drone used as a companion for jogging.

The last type of role 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 [22] or aerial photogrammetry [33],

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 tasks in the application domain. Each control interface impacts how the pilot interacts with the drone in various aspects, such as training period, accuracy, latency, and interaction distance.

As drones became available to the public, the major drone 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 [30].

Drone's control interfaces can be classified as follows [41]:

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

Gesture-based interface is a control modality where the user pilots the drone with body movements. 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, and this indicates that the training period of this interface is almost close to zero [20]. 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, this interface is also a natural user interface with a low training period and high usability. They also share the 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 [30]. 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 is decoded using machine learning algorithms to control physical systems using brain waves. Compared to the others, it is the most complex control interface and has the highest accessibility for users with disabilities. The problems in using BCI are the poor control quality 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 complexity and costs a lot.

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 [15]. Given the rapid expansion of such technology, governments worldwide have been caught off guard in recent years. Governments tried to quickly create rules and regulations to control the usage of such technology. This rapid regulation has, in most cases, limited drones too much, slowed their expansion, and reduced their potential. Research in ethics and values about HDI is responsible for producing accurate and reliable results that should guide governments in refining and upgrading drone laws.

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 field started developing and selling their programming environment. Nowadays, most software resources for drone applications are open-source and publicly available. We will dive into the scenario of the drone programming environment, understanding 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 an 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.

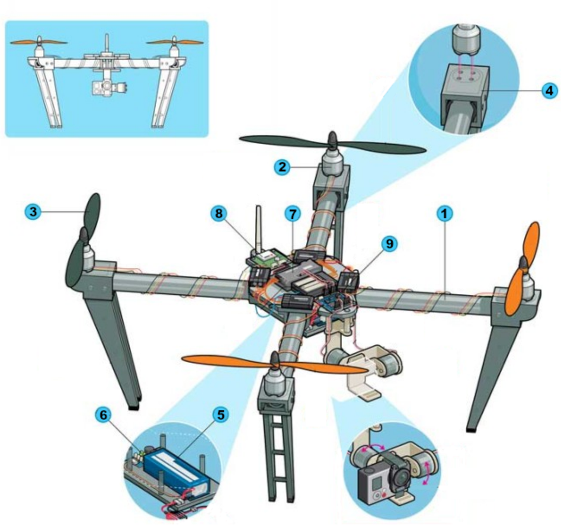


Figure 2.1: The main hardware components of a drone are: 1. Drone's chassis, 2. Motors, 3. Propellers, 4. Motor mount, 5. Battery, 6. Power distribution unit, 7. Computing and memory unit, 8. Communication unit, 9. Sensors unit

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 receives and decodes commands from the off-board system; the signal is then transformed into power set-points from the Core Unit (control loop) with the help of sensor information. The Sensing Unit gathers information from the environment using sensors, 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 [10], DJI [8], and 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 solutions may not be enough; if this is the case, the developer then needs to manually select each hardware component, control the compatibility with each other, and

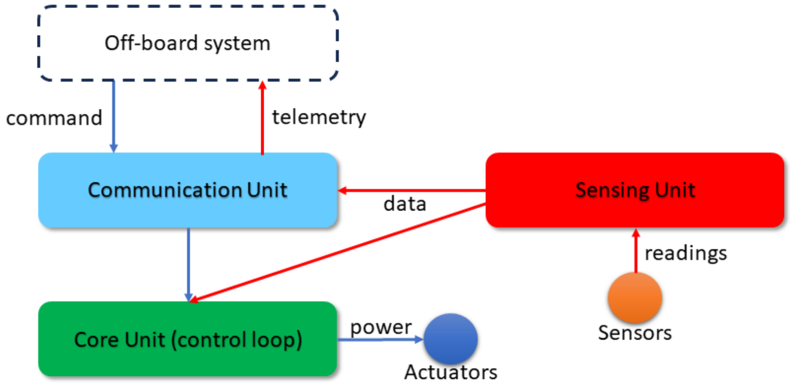


Figure 2.2: The main software components of a drone are: the *Sensing Unit*, the *Communication Unit* and the *Core Unit (control loop)*.

then select (or develop) the software that allows the drone to fly.

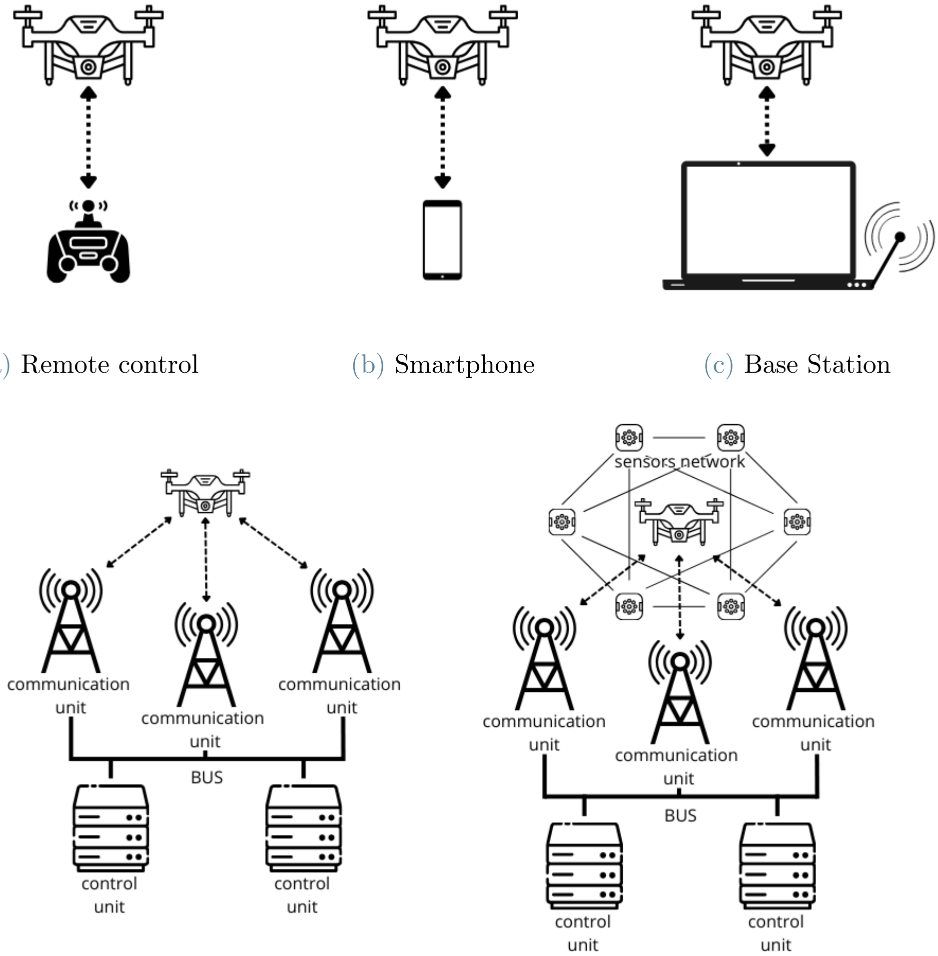
Some producer sell also intermediate solutions [7, 9, 11, 12] between COTS and the completely custom one, these solutions are composed of a microcontroller with usually the basic sensors that compose the Inertial Measurement Unit (IMU) and the autopilot software. These solutions are then extensible with 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, and, in particular, it depends on 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, we can consistently identify two main components in most scenarios: a control unit and a communication unit. In most common situations, 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 acts 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, it 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 sensors that collect atmospheric data, allowing for a better knowledge of the environment in which the drone is deployed.

Communication technology and infrastructure are critical topics that can introduce poten-



(a) Remote control (b) Smartphone (c) Base Station

(d) Distributed ground network (e) Distributed with sensor network

Figure 2.3: Off-board ecosystem

tial issues in the off-board environment when it is inadequate for the application scenario. In particular, depending on the flight area's dimension, location, and topography, appropriate technology and communication infrastructure must be deployed to have a properly working drone.

As highlighted in Figure 2.3, the communication infrastructure can be single or distributed. The former is more straightforward to implement and deploy but can be not enough when the flight area is too broad or the topography is irregular; the latter is much more complex but allows for covering all the possible application scenarios.

The most commonly used communication technology in the drone's field are Wi-Fi, radio, Bluetooth, and cellular network [35]. Table 2.2 summarizes the most common communication technologies and their characteristics. Even if the research frontier for drone communication is mainly focused on cellular networks, in particular the 5G network [37], none of the technologies prevails, but the choice depends on the application scenario. Cellular networks can be a great solution around highly populated areas, while another technology must be considered in rural locations.

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 [35]

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 solutions, the vendors usually provide the hardware and software that compose the off-board ecosystem.

Figure 2.4 shows 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 widespread 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 code bugs or hardware problems. Unfortunately, when an error arises, the drone will usually crash; in some unfortunate cases, some parts will break and need to be replaced. Therefore, testing drone applications is a very expensive task both in terms of economic resources and in terms of time.

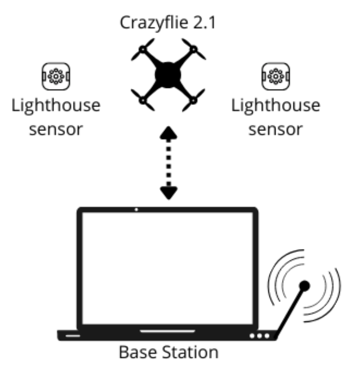


Figure 2.4: EasyFly off-board ecosystem

To overcome this limitation, the main drone producers have developed and distributed a simulation environment that allows testing the software before going into a real scenario. A simulation environment allows running the code written for the planned mission in a graphical simulation that shows the drone performing the tasks. Modern simulation environment [2, 13] usually takes into 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 of using a simulation environment.

2.3.2. Swarm Programming

In modern drone applications, where the tasks to achieve are complex, and the application has to be reliable, a common approach is to deploy multiple drones (a swarm) to complete the requested task collectively. This approach is completely different from single drone programming; in fact, swarm programming introduces new challenges and a different approach to achieving 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 to design 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 [18].

In particular, the behavior of social animal groups appears 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 examples in the current literature of swarm engineering are Proto [17] and

Meld [16]. 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 languages directly take 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 every single component separately.

When swarm robotics is applied in the field of drones, given the high dynamism in the movements of this type of robot, 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 the first place, the swarm, compared to the single drone, can provide a higher availability: A single drone has a limited flight time, so 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 [22]. 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, the goal of the application is usually defined as a swarm goal. Swarm goals are high-level goals whose 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, the complexity of managing all of them increases a lot as well, introducing a consistent overhead that can affect the application's performance. As in any other engineering problem, we must select and identify the most suitable swarm size for the application to realize.

Depending on the application domain, we can adopt different strategies for 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 finest grain method allows an entirely 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 tricky to use all the swarm resources efficiently to reach the general goal. Moreover, it has been proven [22, 32] that this method is indeed the most complex of the three.

Swarm programming[36], on the other hand, allows writing a single set of rules that are common for all drones, and then every 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 challenging to represent tasks that require explicit drone coordination.

Team level programming [32] 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 exposure to the complexity of concurrent programming, parallel execution, scaling, and failure recovery. More generally, Voltron [32] can be viewed as a set of tasks that must be performed by a set of drones subject to particular timing and spatial constraints.

Our work does not address the complexity of swarm programming, although our programming environment, EasyFly, allows for the deployment of multiple drones.

3 | Tools

The main resource used in this work is Crazyflie, an open platform produced by Bitcraze [3] that offers an ecosystem of products and open-source libraries that allow people to develop new functionality for aerial drones.

The key feature of this platform is that it offers a set of expansion decks that can extend the capabilities of the drone with new sensors. Expansion decks can be mounted on the drones very easily, and they are immediately ready to be 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, high-level environment for programming drones.

The aim of this chapter is to present an overview of the Crazyflie platform to provide basic knowledge of its tools and surrounding environment.

3.1. Ecosystem Overview

The Crazyflie platform comprises a set of devices and tools to allow building drone applications. It has a modular architecture that makes it possible to build very versatile systems that are adaptable to many situations. In Figure 3.1, we can see an overview of the ecosystem of the entire platform.

The ecosystem of this platform gravitates around its leading actor: the Crazyflie 2.1 nano drone. In the basic settings, the drone has minimal sensors and actuators that allow it to fly. To empower the drone capabilities, the platform makes available a set of expansion decks that give 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 a mobile phone with a dedicated App installed.

The communication between the quadcopter and the ground station is handled using a

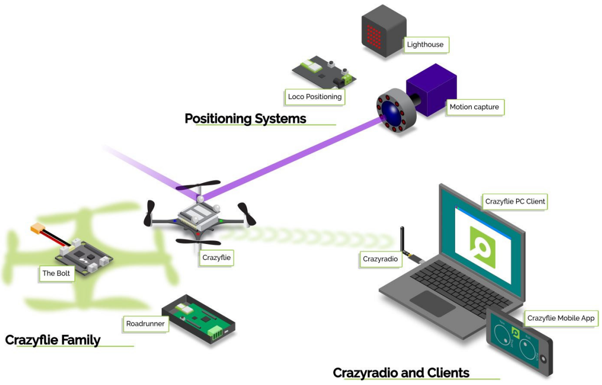


Figure 3.1: This picture represents an overview of the Bitcraze Ecosystem [3]

dongle USB (CrazyRadio) or through Bluetooth when using the mobile App.

The platform also offers multiple absolute positioning systems that allow the drone to have better position estimation in an absolute coordinate system.

3.2. Hardware

In this section, we will provide an overview of the hardware components of the entire Crazyflie platform. We will first 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 of the hardware components that compose the absolute positioning system adopted for the work: the Lighthouse positioning system.

3.2.1. The Quadcopter

Bitcraze produces a family of drones with similar hardware and firmware but different sizes and properties. The target for this work is the principal component of this family, the Crazyflie 2.1, a tiny and versatile quadcopter with a solid and modularized design that falls into the category of nano-drones.

As described in Section 1.2, nano-drones are the most suitable typology for conducting investigations around HDI. The modular approach owned by Crazyflie 2.1 constitutes another key advantage in the field of human-drone interactions; the user can easily change the setup to address any possible situation.

This drone has many hardware components hosted on a single, compact, light base. We can identify four main units: Computing Unit, Motor Unit, Sensor Unit, and Power Unit.

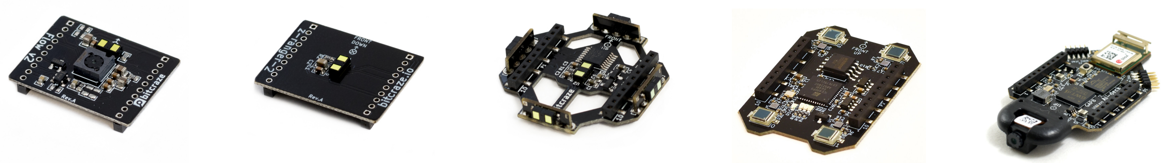
The core of the drone is composed of two Micro Controller Units (MCUs): The first is an STM32F4 MCU that handles the main Crazyflie firmware with all the low-level and high-level controls. The second MCU, NRF51822, handles all the radio communication and power management.

The motor unit of the Crazyflie 2.1 consists of four Coreless DC motors with plastic propellers 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 three coordinates of space plus the angular speed, and a BMP388 sensor, which is a high-precision pressure sensor. The sensor unit of the Crazyflie 2.1, also known as the Inertial Measurement Unit (IMU), is minimal and provides the minimum data that allows the drone to have an almost stable flight.

Its design is robust and simple, easing the assembly and maintenance of its components. 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 it can lift a payload of 15 grams [4].

3.2.2. Expansion Decks

With only two sensors composing the IMU, the drone has a limited capacity to understand the surrounding environment. To overcome this limitation, the drone can be equipped with additional decks that extend its capabilities in sensing, positioning, and visualization. The platform offers a variety of expansion decks, but for the purpose of our work, only a subset of them has been selected. Figure 3.2 shows the expansion decks selected, particularly those that can enable in some way the human-drones interaction.



(a) Flow deck v2 (b) Z-ranger deck v2 (c) Multi-ranger deck (d) Lighthouse positioning deck (e) AI-deck 1.1

Figure 3.2: Expansion decks of Crazyflie 2.1

Flow deck v2:
The Flow deck (Figure 3.2a) allows the Crazyflie to understand when it moves 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 relative to the ground. This expansion deck is a relative positioning system that lets the drone know its position relative to its take-off point.

Z-ranger deck v2:
The Z-ranger deck (Figure 3.2b) is a simplified and cheaper version of the Flow deck v2. It only measures the distance from the floor up to 4 meters using the usual laser sensor VL53L1x ToF.

Multi-ranger deck:
The Multi-ranger deck (Figure 3.2c) gives the Crazyflie the capability to sense the space around it and react when something is close and for instance, avoid obstacles. This is done by measuring the distance to objects in the following five directions: front, back, left, right, and up with mm precision up to 4 meters, using five VL53L1x ToF sensors.

Lighthouse positioning deck:
The Lighthouse deck (Figure 3.2d) is part of the Lighthouse absolute positioning system (See Section 3.2.4). It comprises four TS4231 IR receivers and an ICE40UP5K FPGA to process the signal received. This expansion deck lets the drone know its position in an absolute coordinate system.

AI-deck 1.1:
The AI-deck 1.1 (Figure 3.2e) 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.

3.2.3. 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 an 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 of up to 1km (line of sight). The dongle comes pre-programmed with Crazyflie's 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.

3.2.4. 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 lighten 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 computes the angle of incidence of the IR rays with signal processing. The position and the pose of the Craziflie are then finally computed from the main MCU of the Crazyflie 2.1

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, develop new features, or upgrade existing ones. Each library targets a specific system component and is completely independent of all the others. This section will briefly describe the two main libraries used as a baseline for our work.

3.3.1. crazyflie-lib-python

The crazyflie-lib-python (cflib in short) [6] is a software repository that consists of a Python library for programming scripts that control the behavior of the Crazyflie 2.1. The library provides the base facilities to allow users to define the desired drone behavior in a Python script, abstracting from the low-level control mechanism.

As shown in Figure 3.3, the Python scripts are executed on the ground station. The library contains the code to create communication packets sent through the Crazyradio reaching the Crazyflie, which will eventually execute the commands requested.

3.3.2. crazyflie-firmware

The crazyflie-firmware [5] is a software repository that contains all the firmware of Crazyflie 2.1. The firmware is written in C++ and 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 of the cflib.

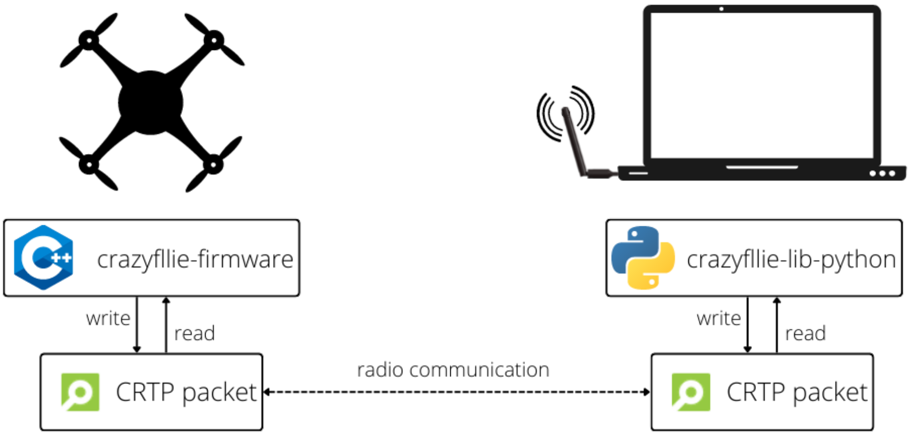


Figure 3.3: The two main software libraries are *crazyflie-firmware* and *crazyflie-lib-python*. The former, written in C++, runs on the drone; the latter, written in Python, runs on the base station. The communication protocol that enables the cooperation of the two is the Crazy Radio Transfer Protocol (CRTP)

3.4. Positioning Systems

Positioning systems represent the core sensing task of every drone application. Knowing the drone's position in the flight space is essential for achieving any possible goal. We can identify two main categories of positioning systems: relative and absolute.

Relative positioning systems are the simplest: starting from the drone's position at take-off, they estimate the position during the flight on the base of the movements made by the drone in the flight space. The measurement of the movements is usually made with the IMU (Inertial Measurement Unit) or with some additional sensors that track the difference in position over time.

The main issue related to this type of positioning system is that the error continually accumulates, and after a certain period of flight, they can lead to consistent mispositioning.

Absolute positioning systems provide coordinates that define the exact location of an object within a specific coordinate system. These systems typically rely on external references or signals from fixed environmental points. The sampling of the position in this type of positioning system is independent of any initial measurements, so the error in the positioning usually does not accumulate over time.

The Crazyflie platform offers multiple positioning systems, both absolute and relative. To select the best system for building our programming environment, 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 three relative positioning systems. The first system is represented by the Inertial Measurement Unit, which is provided on the base drone without expansion decks. From our experience, this system has very poor accuracy and cannot be used as the only positioning system of the entire application. However, it contributes its information to obtain the position estimate.

The Z-ranger deck (See 3.2.2) is another positioning system that the platform offers; it provides an estimate only for the z-coordinate with pretty good accuracy, but, as typical for every relative positioning system, it has a high cumulative error during the time.

The Flow deck v2 (See 3.2.2) represents an empowered version of the Z-ranger, 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 good sampling accuracy and an acceptable cumulative error rate over time. Flow deck v2 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

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 act as a GPS satellite, and a Tag, namely, the Loco positioning deck, which acts as a GPS receiver. As indicated in the specification of this system, 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 system knows the layout of the markers on the drone, 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 that the native environment provides only the markers and relies on third-party systems for the entire MCS. Moreover, the position is computed in an external node and needs to be sent to the Crazyflie, increasing the communication load.

The last solution is the Lighthouse positioning system (Lighthouse), the newest introduced in the environment and 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 with four optical IR receivers (photodiode), can measure 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 of a vector in the three dimensions and a rotation matrix.

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 cleverly to control the drone's stability and make it possible to fly.

This section will describe the principal 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 (See 3.3.2) from the sensor read to the motor thrust. In Figure 3.4, we can see a representation of the control loop of the Craziflie 2.1.

As in any feedback control system, also for the Crazyflie control loop, we can identify two principal components in Crazyflie 2.1: the state estimator and the state control. The synergy between the state estimate and the state control in a control loop is fundamental for achieving accurate and effective system control.

The state estimator is responsible for computing the best possible estimate of the system's current status. This component inside the Crazyflie 2.1 firmware has two concrete implementations with different performances and accuracy:

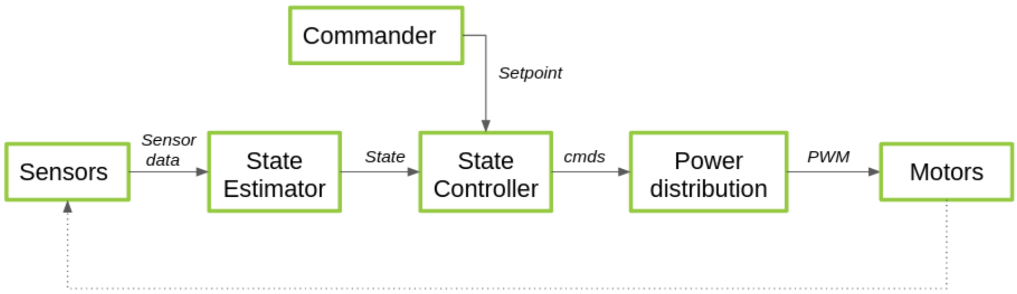


Figure 3.4: The control loop of the Craziflie 2.1 [3]

- Complementary Filter
- Extended Kalman Filter (EKF)

The Complementary Filter is a very lightweight and efficient state estimator. It can only partially use the available sensors; in particular, it uses only data from the IMU and the ToF sensor (Flow deck or Z-ranger). The estimated output is only a portion of the Crazyflie state: the Attitude (roll, pitch, and yaw) and the z coordinate relative to the starting point.

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 all the possible sensors' data as input. If enough sensor data are available, it can compute a complete state estimation as output: Attitude, Position, and Velocity in all directions. The choice of which state estimator to use can be forced by the user or automatically set. By default, the firmware uses the lighter Complementary Filter and switches to the EKF if more sensors are available.

The state control uses this estimate to elaborate the actions needed to change the current state to a target state. On the Craziflie's 2.1 firmware, we have three possible alternatives, ordered by complexity:

- Proportional Integral Derivative Controller (PID)
- Incremental Nonlinear Dynamic Inversion Controller (INDI)
- Mellinger controller

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.

Inside the cflib the one responsible for controlling the flight is a module named Commander Framework. The Commander Framework can be viewed as composed of two layers:

The first layer provides low-level operations that allow writing setpoints and sending them with the custom CRTP protocol.

The second layer, built upon the first, is more abstract and adds some general functionalities, e.g., take-off, land, and move-to.

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:

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

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

Initial instructions
for $for - condition$ **do**
  Some instructions
  if $if - condition$ **then**
    Some other instructions
  end if
end for
while $while - condition$ **do**
  Some further instructions
end while
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: [28], or multiple ones at once: [29, 31].
The bibliography and list of references are generated automatically by running BibTeX [21].

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] Bitcraze. URL `https://www.bitcraze.io/`.

[4] Crazyflie 2.1, . URL `https://www.bitcraze.io/products/crazyflie-2-1/`.

[5] crazyflie-firmware, . URL `https://github.com/bitcraze/crazyflie-firmware`.

[6] crazyflie-lib-python, . URL `https://github.com/bitcraze/crazyflie-lib-python`.

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

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

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

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

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

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

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

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

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

[16] 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.

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

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

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

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

[30] 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.

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

[32] 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.

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

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

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

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

[42] H. A. Yanco and J. Drury. Classifying human-robot interaction: an updated taxon-

omy. 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.