POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master's Degree Thesis

Swarm flight for UWB-based reference trajectory tracking

Supervisors

Prof. Marcello CHIABERGE

Dott. Ing. Giovanni FANTIN

Candidate

Matteo CELADA

April 2021

Acknowledgements

First of all, I would like to thank Prof. Chiaberge and the whole PIC4SeR's staff for welcoming me and giving me the opportunity to work on a complex and fascinating project. In particular, my thanks go to Giovanni, who patiently helped me from day one and introduced me to the world of drones and service robotics.
Then, I am grateful to all my friends, both the ones I already knew and the ones I met along the past years spent at Politecnico. Among them, I would like to especially thank Marco and Luca for being always present since kindergarden, and Andrea, with whom I shared endless hours of study during the five years in which we attended the university together.
Moreover, my heartfelt thanks go to my girlfriend Cristina, who proved along the years to be someone I could always rely on, even in the most difficult and stressful times.
Finally, I would like to express my greatest gratitude to my family, which supported me in the educational journey that brought me to this achievement, and encouraged me to always give it my best.

Abstract

Autonomous flying vehicles represent an innovative solution for various service robotics scenarios. In particular, the capability of acting in formation opens to new opportunities in a multiplicity of different fields. One of the many advantages of having multiple Unmanned Aerial Vehicles (UAVs) operating collaboratively is the possibility to sense the relative distances from an object and localize it with respect to their position in space. The case analyzed in this document consists of a reference body, be it terrestrial or aerial, autonomous or human-controlled, that operates in a GNSS-denied environment: computing its position at all times is a fundamental challenge, that is overcome thanks to the information exchanged with a fleet of drones. This work aims to study and implement techniques to perform two combined tasks: maintaining a steady formation of the UAVs and tracking the trajectory of the considered reference body. In particular, a potential-based decentralized algorithm was deployed for the swarming task, while a centralized approach was chosen to fulfill the tracking requirement. The studied procedures make use of the GNSS position of the swarm anchors, and the relative distance between each couple of vehicles, given by the Ultra-WideBand (UWB) sensors mounted on every machine. The achievement of the described objectives, and the data provided by the sensors onboard every vehicle, give the possibility to easily localize the reference body in space, via a multilateration approach. The results of the performed simulations show good tracking capability, while successfully keeping a stable group flight configuration in all tested conditions.

Table of Contents

List of Tables

List of Figures

Chapter 1

Mobile robotics and UAVs

Robotics is one of the most active research subjects in the field of engineering, due to the relatively young age of this branch and to the countless applications that it offers. The main goal of robotics is to study and develop machines that can provide a service by carrying out acts usually made by humans, working either with or in place of them: the term robot itself comes from the Czech word "*robota*", which can be translated as "heavy work".

Since the invention and manufacturing of the first robots, they have been used mainly in the industrial world, where they are known as manipulators, and can be employed to perform tasks that would be difficult for human workers. The intensified utilization of automatic machinery gave an enormous boost to the industrial revolution, as robots allowed to decrease the cost and increase the efficiency of production lines. Indeed, beyond being much cheaper than human manpower, robots were faster and more precise, especially for repetitive tasks.
While in the first years of their deployment robots were thought to substitute individuals in performing a well-defined task or set of actions, thanks to the improvement of the field over the years, robotic systems can be smart enough to complete non-standardized chores. This does not necessarily mean that robots

will substitute human labor: automated systems could help make it even more productive, operating alongside the workers[1].

As time passed by, thanks to research and technological progress, more and more applications were found for robots in a large range of fields other than the industrial one, from military to agriculture, from medicine to the so-called service robotics. Service robotics consists of assisting people in some specific situations of their everyday life, including household chores. According to the standard ISO8373, a service robot is defined as a robot «that performs useful tasks for humans or equipment excluding industrial automation applications»[2]. The official definition is fairly general: service robotics includes cleaning robots, drones, mobile ground robots, and so on. The technological evolution allowed a dramatic increase in the diffusion of these kinds of robots, guaranteeing affordable prices for the items despite their growing complexity.

A fundamental aspect, that allowed many of the usages of robots that we know today, is mobility: mobile robots' main feature is the presence of a mobile base, that allows them to move freely in the environment, be it terrestrial, underwater, or aerial. This analysis will focus mainly on flying autonomous vehicles and their cooperation towards a common goal.

1.1 Unmanned Aerial Vehicles (UAVs)

An Unmanned Aerial Vehicle (UAV), commonly known as a drone, is a flying robot whose defining feature is the absence of a human pilot on board. All of its operations, from takeoff to landing, are entirely carried on remotely either by a pilot or by software specifically designed for the task. Ideated and first studied in the past century for military reasons, UAVs appeared in the civil scenario in the last few years with rapid growth. They can be equipped with various devices and sensors allowing them to take on very different tasks, like monitoring, search and rescue, data collection, aerial photography, and so on. For applications that need

neither high speed nor the transportation of heavy bodies, the so-called quadcopters are the most common solution.

A quadcopter, also known as a quadrotor, is a flying vehicle with four rotors, as represented in figure 1.1. Its structure is very simple, as its motion is commanded directly by controlling the angular velocity of the four rotors[3]. Since it can be extremely small and lightweight, and hence have low inertia, it represents the ideal solution in many situations that make use either of a single drone or multiple quadcopters cooperating amongst each other.

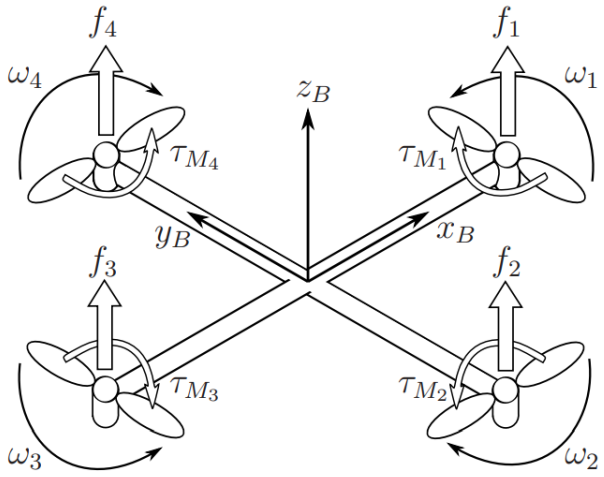


Figure 1.1: The structure of a simple quadcopter with the forces and the torques acting on its body.

The design of the quadcopter provides for two couples of rotors spinning in opposite directions, usually driven by an electric engine that is powered by a battery. The motion of each propeller generates both lift and torque about its center and drag opposite to the drone's direction of motion. This configuration implies that, when all the rotors have the same angular velocity, the net aerodynamic torque about the yaw axis (z_B in figure 1.1) would be null. The rotation about the yaw axis is controlled by tuning the angular velocity of two opposite propellers in the same way, as shown in figure 1.2.

Pitch and roll, instead, are generated by modifying the angular speed of each rotor.

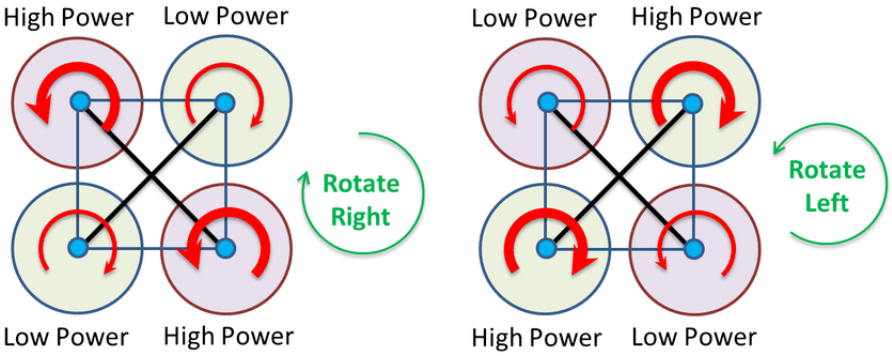


Figure 1.2: The angular speed differential between the two couples of opposite rotors generate a rotation about the yaw axis[4].

Linear motion can be achieved by getting a suitable spatial configuration of the quadcopter, that is combining the rotors' speed to obtain the needed yaw, pitch, and roll angles, as it is exemplified in figure 1.3.

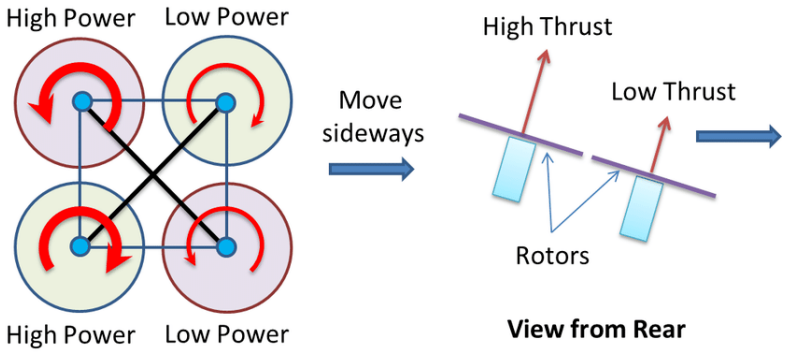


Figure 1.3: The pictured configuration of the rotors' angular speed generates a lateral linear motion[4].

The dynamic model of such a vehicle will not be further analyzed in the current work, as it is handled at a lower level by the drone autopilot, which will instead be treated in the following section.

1.2 The PX4 autopilot

The autopilot used in this project is PX4, a system thought for low-cost autonomous floating robots. Started in 2009, the project was and currently is, developed at Computer Vision and Geometry Lab of ETH Zurich (Swiss Federal Institute of Technology). «PX4 is an open-source flight control software for drones and other unmanned vehicles. The project provides a flexible set of tools for drone developers to share technologies to create tailored solutions for drone applications. PX4 provides a standard to deliver drone hardware support and software stack, allowing an ecosystem to build and maintain hardware and software in a scalable way»[5]. Its main characteristics are:

- modular architecture, both concerning hardware and software, which means that components may be added to the system without influencing the existing modules;
- open source, so that the system can be developed by an extended community, and hence satisfy the needs of a wide range of users, from universities to industry;
- configurability, as it supplies APIs for developers working with integrations, and hence allowing features to be easy to deploy and reconfigure;
- autonomy stack, which makes it ideal to operate with embedded computers for autonomous functionalities[5].

The PX4 autopilot mainly consists of two layers: the flight stack and the middleware.

1.2.1 Flight stack

«The flight stack is a collection of guidance, navigation and control algorithms for autonomous drones»[6]. As it is shown in figure 1.4, the flight stack includes all the blocks that compose the drone and make the proper functioning possible.

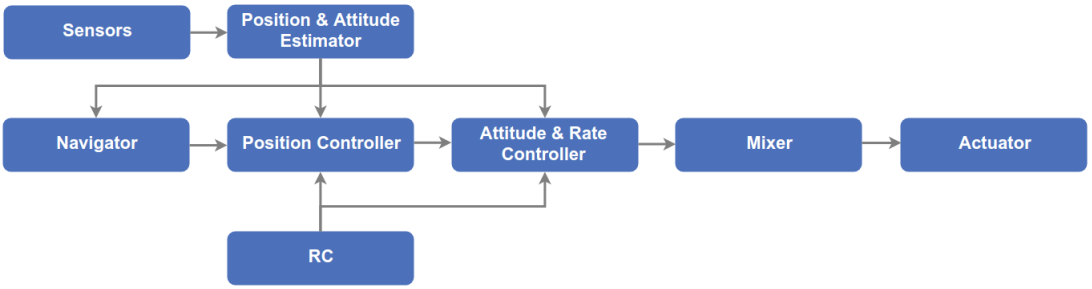


Figure 1.4: An overview of the building blocks of the flight stack[6].

An estimator block takes data from the sensors as input and, starting from them, computes the vehicle state. The controller, in the figure divided into position and attitude blocks, receives a setpoint state either from the navigator or the Radio Control (RC), and the output of the estimator: its algorithm is able to compute the forces needed to suitably correct the state to match the desired setpoint. The mixer block should translate the raw input received from the controller into commands understandable by the hardware: in particular, it has to keep into consideration the specific structure of the drone since the translation is strictly tied to it.

1.2.2 Middleware

The middleware includes mainly the drivers for the sensors, the code dedicated to the communication with the external world, and the uORB publish-subscribe message bus. Moreover, the so-called "simulation layer" included in the middleware is of primary importance: it gives the possibility to run the PX4 code in a simulated environment, simplifying the testing process.

1.2.3 MAVLink messaging protocol

MAVLink is a lightweight messaging protocol used in the communication between drones and internally between their components. It works following a publish-subscribe design: the messages, defined as XML files, are placed in data streams

that are published as topics. This protocol results very efficient, as each packet has just a few bytes of overhead (8 or 14 depending on the used version), and hence it is perfectly suitable and suggested for applications that cope with narrow communication bandwidth. Moreover, it is very reliable, as it has been used since 2009 in many different situations, and it offers valid solutions for possible problems, like packet drop detection, corruption, and authentication[7]. MAVLink is used by the PX4 autopilot to communicate with QGroundControl (QGC) and other ground stations, as well as to exchange messages with components connected to the drone, like companion computers, enabled cameras, and so on[8].

1.3 Global Navigation Satellite System (GNSS)

All vehicles, especially the ones operating automatically, need a continuous flow of information about themselves and the external world. The sensors onboard play an essential role in the correct execution of the machines' tasks.

The acronym GNSS generically refers to the system used to perform global localization of user devices. It indicates a set of satellites transmitting data to receivers, allowing them to univocally compute their own position with a certain accuracy. Some examples of GNSSs are Europe's Galileo, USA's GPS, Russia's GLONASS, and China's BeiDou[9].

1.3.1 GNSS architecture and operation

A GNSS consists of a collection of artificial satellites, called constellation, orbiting around the Earth, at a distance of about 20000 kilometers from the planet's surface. Each satellite follows a precise trajectory and mounts an accurate clock on board, crucial for the success of the operation. Every member of the constellation continuously broadcasts identifiable data, which, after being read and analyzed by the ground stations, are used to send orbit corrections to the satellites and

maintain the system's precision as high as possible.

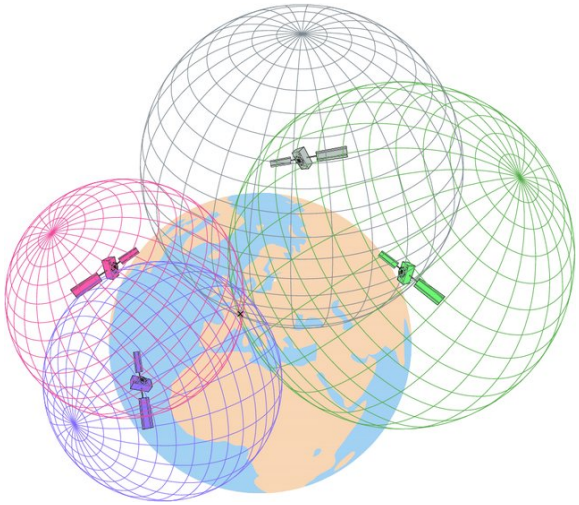


Figure 1.5: GNSS multilateration with four satellites[10].

The position computation is based on a methodology known as multilateration. Such a strategy relies on the fact that it is possible to have an extremely accurate estimate of the location of the satellites. Among the other transmitted data, they communicate the timestamp of the instant the message is broadcast. Knowing that the signal travels at the speed of light, the receivers can easily derive their distance from the transmitter, via the law of linear motion. Then, the receptor certainly knows to be located somewhere on the sphere of such radius centered in the considered satellite. Repeating the same steps for a second transmitter, the possible position is known to be at the intersection of the two obtained spheres. A third and fourth satellite would generate two more spheres crossing the previous ones in a single point, representing the searched location.
Unfortunately, the described situation is ideal and unfeasible in a real scenario, with errors and equipment inaccuracies. In this case, a higher number of satellites is exploited, and the problem is solved approximately: via optimization methods, such as least square, the closest solution to the obtained data is found and used.

1.3.2 Errors and their propagation

As we said, the distances computed by the receivers always carry an error, which can be relatively large and depends on many external factors. In particular, the calculated distance is affected by atmospheric stratification, multipath effect, and clock inaccuracy.

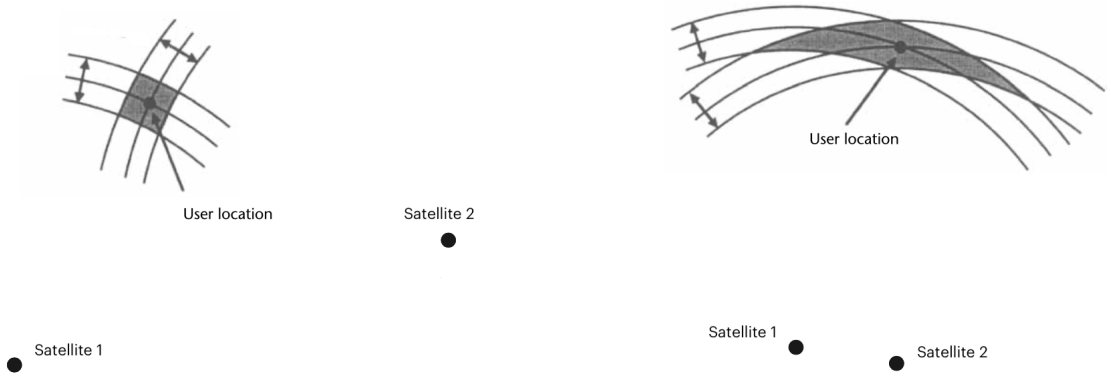
- The composition of the Earth's atmosphere depends on the altitude with respect to the planet's surface and other factors. This implies that assuming the velocity of the signal to be always equal to the speed of light introduces an imprecision. Such a problem is attenuated via specific techniques which consider a precise model of the atmosphere's composition, and how it affects the signals at various frequencies.

- Most of the time, the GNSS receivers are not located in an optimal environment, which would keep the error low. The presence of mountains, tall buildings, or other natural and artificial structures, influences the signal sent by the satellites: indeed, bouncing on these surfaces makes the trajectory of the electromagnetic waves not straight, and consequently, the data may be misinterpreted.

- The clocks mounted on the satellites and the ground control stations are extremely precise, but the GNSS receiver's one is not synchronized with them, so it contributes to the localization error. Because of this difference, the considered time of flight of the signal is not the actual one, and the computed distance from each satellite is affected by a mistake.

Due to the analyzed errors and to others that contribute to the localization loss of accuracy, the imaginary spheres centered in the satellites should be considered to have a radius ranging from $r_i - \Delta_r$ to $r_i + \Delta_r$. Hence, taking into consideration multiple satellites, the intersection is not anymore between the boundaries of the spheres, but between the spherical shells. Figure 1.6 shows an example of what is known as the Dilution Of Precision (DOP) problem in two dimensions: the smaller is the relative distance between the satellites, the wider is the uncertainty area

generated. For this reason, since all the satellites of a constellation orbit at similar altitudes, the GNSS accuracy on the z axis, radial to the Earth surface, is lower than the horizontal one.



(a) Low dilution of precision. **(b)** High dilution of precision.

Figure 1.6: Dilution of precision according to satellites' geometry.

An efficient way to improve the performance of the localization system is to exploit a technique called Real-Time Kinematics (RTK). Such a methodology makes use of a base station, fixed, which through a survey process gets precise and continuously improving data about its position. Then, after suitable processing, it uses the information to send corrections to the enabled receivers, allowing them to improve their accuracy. If the RTK base station is given enough time for the survey process, and it is in an environment with good satellite visibility, then the localization error can fall below the millimeters order of magnitude.

1.4 Ultra-WideBand (UWB)

Ultra-WideBand technology is a digital data transmission protocol for short-distance wireless communication. It operates through radio waves at high frequencies, but, unlike Bluetooth and Wi-Fi, it uses a wide portion of the frequency spectrum in the GHz range. In figure 1.7, the difference from other known technologies is

well visible: while occupying the bandwidth going from 3.1 GHz to 10.6 GHz, the needed power spectral density is much lower.

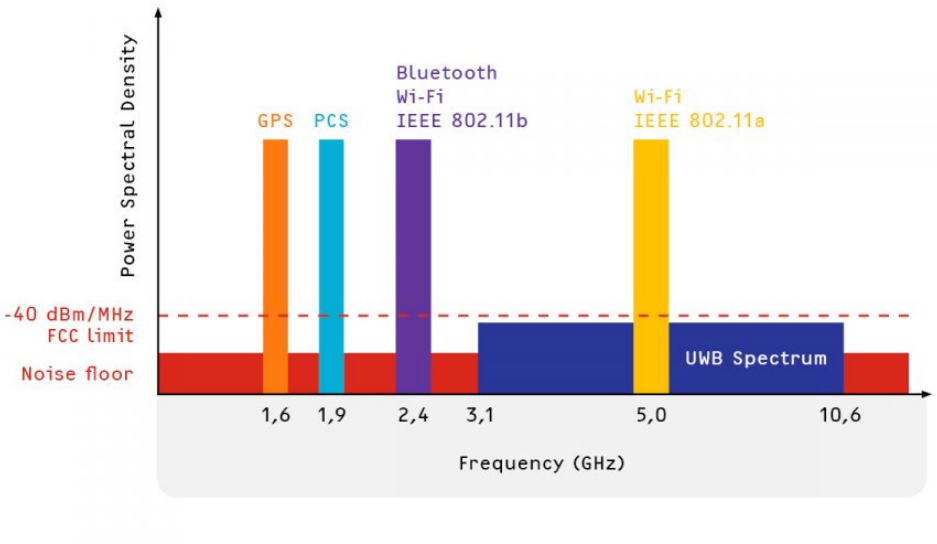


Figure 1.7: UWB frequencies range compared to other wireless communication technologies.

UWB technology is well-suited for capturing very precise spatial and directional information, due to the following specific characteristics.

- It provides very good performance in multipath situations. The problem already described for the GNSS heavily shows up for indoor signals, due to the numerous obstacles which can deviate the electromagnetic wave. Since UWB transmits short pulses over a large bandwidth, it copes well with this issue.

- It has good capability of obstacle penetration. The fact that the low frequencies of the spectrum have large wavelengths, makes UWB communication appropriate for non-line-of-sight scenarios.

- It does not disturb the other wireless communication technologies. The signals' low power spectral density, spread across the large bandwidth, makes them

very difficult to intercept, and able to coexist with other wireless transmission techniques.

- It has good accuracy for ranging operations. Measuring very precisely the generated short pulses, and the time of flight of the messages, it is possible to localize a tag with precision in the order of centimeters.

Because of the listed features, UWB is used in accurate indoor positioning systems, radar-like applications, and it has been expanding widely in the past few years[11][12].

1.5 Organization of the work

After this short introduction, the thesis is organized as follows. Chapters 2 and 3 treat the state of the art of swarming and tracking techniques respectively, chapter 4 analyzes the methodologies and algorithms used in this project, chapter 5 shows the obtained results, and chapter 6 concludes the work, defining some possible following steps.

Chapter 2

Swarming techniques

Among the others, one of the strengths of UAVs is the capability of collaborating towards a common goal. A swarm, or fleet, of drones, is a multi-robot system consisting of a relatively large number of autonomous flying devices. The idea of implementing such an approach was born observing the animal world, taking as examples swarms of bees, flocks of birds, and schools of fish. The basic notion on which autonomous swarms are based is that the involved entities make their own decisions thanks to the information shared with each other, exactly as it happens with animals.

2.1 Swarm flight applications

Just as most of the studies on new promising technologies, its first purpose was tied to military applications, but drone swarming later attracted more and more attention also for civilian usage. The following are examples of situations in which the swarming technique is applied.

Monitoring and surveillance As in warlike context UAVs were often used for surveillance missions, in nonmilitary situations they can be employed for monitoring in various fields, like geophysics and agriculture. Another situation in which flying swarms could be extremely useful is the surveillance of a large facility, which would otherwise require many staff members and hence a much bigger economic effort.

Smart cities and civil protection Like a factory or a company plant, swarming technology could be used also in larger environments, for example in the context of a smart city. The range of feasible applications in this field is extremely wide, as it varies from urban surveillance to weather monitoring, from traffic control to package delivery.

Environmental mapping A swarm of drones could be equipped with the necessary sensors and artificial intelligence to carry on the mapping of a certain area, a task that would be important in the fields of cartography and archaeology.

Precision agriculture This is a field in which it is fundamental to have detailed and precise information about large areas, for example in the context of weather conditions monitoring. The deployment of swarms of UAVs could be useful to get data and be able to analyze them in real-time, reducing the amount of time necessary to put in place the needed corrective actions.

Disaster management In situations that are dangerous for human operators, UAVs can reach the affected areas in a completely safe way and help, for example through cameras or other sensors, the rescuers to analyze the environment and study the needed countermeasures. This is the case of natural disasters, like wildfires, earthquakes, landslides, avalanches, and other situations that would represent a huge risk for people.

Search and Rescue (S&R) Strictly tied to disaster management, in S&R situations coordinated drones can help locate the position of an individual in need

of assistance without putting in danger the life of the rescuers. For example, a drone equipped with a first aid medical kit could be sent to search and bring help to hurt people, significatively increasing the rescue's success chances.

Entertainment On a much lighter note, swarms of flying robots may be exploited for shows and performances, as it already happens in many cities around the world. A very close example is the celebration for the feast of "San Giovanni" in Turin, which has proposed now for two years in a row a spectacle with 200 illuminated drones instead of the traditional fireworks[13][14].

The ones here presented are just a few of the many possible applications of swarms of drones. In addition to these, continuous technological advances will provide new opportunities in a field of engineering that is expected to grow further in the next future.

2.2 Control architecture

There are various approaches to get the desired swarming performance and possibly maintain a stable formation for the whole time of flight. First of all, the architecture of the control could be implemented fundamentally in three different ways: centralized, decentralized, or distributed. The listed logical patterns differ from each other by their specific characteristics, like points of failure, fault tolerance, scalability, and ease of development. In the following sections, we will discuss the various approaches analyzing their proper features.

2.2.1 Centralized control

A centralized architecture provides for a single hub that is responsible for all the needed operations. This master node collects data from every agent involved in

the system, it uses them as input of the control algorithm, and it gives as output the different commands for each component of the network. Despite being easy to maintain, this structure is very fragile due to the presence of a single point of failure: this means that if the central node gets damaged, then the whole system stops working. Networks that use the centralized topology should not be too large, or the traffic of data would be unsustainable for the controller. The development of centralized systems is relatively fast, as they are conceptually rather simple and straightforward.

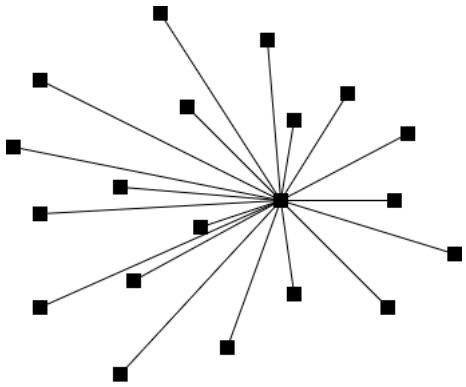


Figure 2.1: The representation of a centralized control architecture.

2.2.2 Decentralized control

In a decentralized architecture, the control intelligence is not concentrated in a single hub, but it is shared between a few nodes, each of which is in turn the master node of a smaller group of agents participating in the system. Essentially, a decentralized control strategy could be defined as a collection of smaller centralized arrangements. Due to their structure, decentralized schemes have a few points of failures (the master nodes of each subsystem), but a malfunctioning would cause just limited damages: indeed, only the actors directly tied to the faulty hub would be affected, while the rest of the network could carry on normally. The scalability of such architecture is surely better than that of the centralized one, because of its

natural subdivision into smaller clusters. Implementing a decentralized network is significatively more complex than realizing its centralized equivalent since the structure is not as plain.

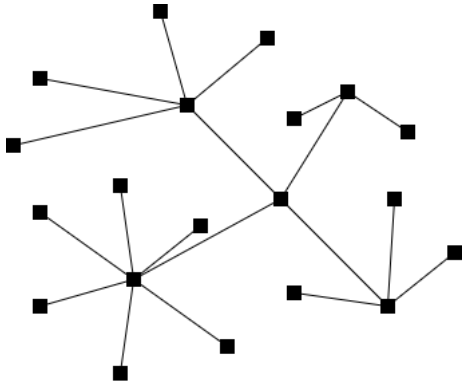


Figure 2.2: The representation of a decentralized control architecture.

2.2.3 Distributed control

The distributed architecture is undoubtedly the most complex of the ones analyzed, but it often guarantees advantages that overcome the project challenges. In such a situation, all the nodes are equivalent, and each one of them can produce the necessary output command exploiting only the information provided by its neighbors. As shown in figure 2.3, the structure of a distributed system reminds the one of a net, in which every node is well connected to the ones close to itself. Due to the numerous connections of each joint, distributed architectures are the most stable ones, because the failures of nodes and links are naturally bypassed through other communication routes. On the other side, maintenance is one of the most critical issues of this type of system: indeed, a simple change in a node could start a waterfall effect exposing a large number of other problems. Despite the obvious design challenges, a distributed system would have unlimited scalability, considering that no nodes would ever be overloaded, even adding new joints[15].

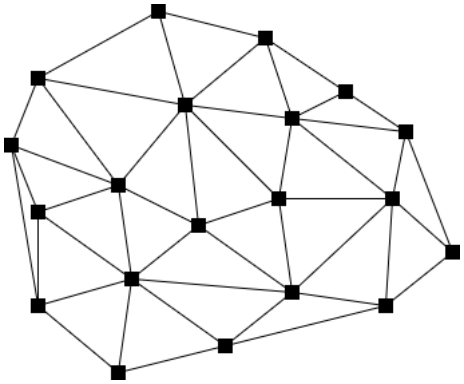


Figure 2.3: The representation of a distributed control architecture.

To conclude this analysis, a system must be carefully examined to be certain to choose the most suitable architecture for the requirements. In the case of a relatively small number of agents, it may be better to use a centralized control design, because the advantages of a decentralized or distributed one would not be enough to justify the higher project effort. Most certainly, it is fundamental to take the needed precautions to avoid or solve the problems that a fault in the system may cause. As we stated before, indeed, a simple defect could be fatal for the whole network. Instead, if we already know that the system will expand in the future, it would be more beneficial to implement a decentralized, or even better a distributed architecture. It is the only way to ensure that the growing network would not overburden the control nodes and the system would have a higher tolerance to joints and communication failures.

2.3 Swarming algorithms

Many different strategies have been implemented in the last few years, with the ultimate goal of guaranteeing the best possible performance, in terms of formation keeping, of a swarm of UAVs. Zhang and Mehrjerdi [16] perform a detailed analysis of various methodologies of coordinated control of flying vehicles. According to such a study, the most common algorithms are based on six main techniques: virtual

structure, behavior-based, leader-follower, artificial potential, graph-based, and intelligent control. In the following section, we will summarize these ideas and present some examples of existing algorithms.

2.3.1 Virtual structure

By definition, «a virtual structure is a collection of elements, e.g. robots, which maintain a (semi-) rigid geometric relationship to each other and to a frame of reference»[17]. The idea behind this methodology is to treat the entire formation as a single body. The expected motion profile is passed as input to the virtual structure, which is, in turn, able to compute the movement of each component of the swarm. The main upside of such a technique is its simplicity, while on the other side, the downside is its strongly centralized architecture, which introduces a single point of failure in the system. The virtual structure scheme is said to have a bidirectional flow of control: it means that the position of the virtual structure commands the position of the robots, which in turn influences the whole system.

Lewis and Tan [17] give a general explanation of such a technique. Assuming a virtual structure to be composed of as many points as the number of robots to be controlled, the first step of this method is the definition of the univocal mapping between each robot and the corresponding point of the virtual structure. It is now possible to define an objective function, whose role is to quantify the positioning error of the robots with respect to the virtual structure. Such a function computes the sum of the distances between each component of the arrangement and its assigned virtual structure point. Then, minimizing the value of the objective function, we get the best possible configuration of the initialized virtual structure. The following phase would be the planning of the movement, both linear and angular, of the virtual structure. It is important to notice that, if the set displacement is not within the reach of the robots, an error will be surely introduced. Hence, the physical limitations of the hardware should be always taken into account to reduce the inaccuracy of the system. Then, knowing the exact position and attitude to be

reached, it is sufficient to compute the desired trajectory and consequently translate it into lower-level commands to be input to the motors of each robot.

Algorithm Virtual structure

```
initialize the virtual structure
while true do
    align the virtual structure with the current robot positions
    move the virtual structure by Δx and Δθ
    for all robots do
        compute the trajectory to move it to the desired virtual structure point
        adjust the velocity to follow the desired trajectory
    end for
end while
```

Table 2.1: Virtual structure formation control algorithm[17].

2.3.2 Behavior-based approach

Behavior-based control is a branch of engineering that makes use of animal systems as an example. Instead of trying to set up a complex model of the environment, which would require quite a lot of time and resources, the intelligent performance is obtained by combining in a constructive way many simple behaviors running simultaneously. A behavior could be defined as the sequence of operations needed to accomplish the desired result. For example, if the goal is to follow a certain object, the behavior would consist of the actions of detecting it, setting the right orientation, and moving towards the objective. Usually, a behavior-based system is also reactive, which means that instead of modeling the world it is operating in, the robot reacts directly to the data coming from its sensors. Combining an initial set of predefined behaviors with the capability of learning from previous experiences, the ability to perform the desired operations is obtained[18].

Xu *et al.* [19] study a behavior-based control to make a swarm of robots keep

a predefined formation. As explained before, the total behavior is obtained as the combination of various sub-behaviors: in this case, such conducts are moving towards the goal, avoiding obstacles, wall-following, avoiding robots and formation-keeping. It is possible to represent the behavior mathematically as a vector product, in this specific case as

$$\boldsymbol{v}_{direction} = \begin{bmatrix} f_1 & f_2 & f_3 & f_4 & f_5 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_{goal} \\ \boldsymbol{v}_{obstacle} \\ \boldsymbol{v}_{wall} \\ \boldsymbol{v}_{avoidance} \\ \boldsymbol{v}_{formation} \end{bmatrix}, \tag{2.1}$$

where $\{f_1,\ f_2,\ f_3,\ f_4,\ f_5\}$ are the weighting factors of the different behaviors. While the first three components of the column vector in equation 2.1 are part of the tracking task of the system, the last two directly influence the formation control.

When two robots get too close to each other, the behavior that manages the avoidance between swarm components comes into play. Assuming the formation to be lying on a plane parallel to the xy plane, we could set $(x_i,\ y_i)$ to be the position of robot R_i and $(x_j,\ y_j)$ the one of robot R_j. Then, the behavior could be computed as

$$\boldsymbol{v}_{avoidance} = \frac{1}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}} \begin{bmatrix} \pm (x_j - x_i) \\ \pm (y_j - y_i) \end{bmatrix}. \tag{2.2}$$

The corresponding weighting parameter is defined as

$$f_4(d_r) = \begin{cases} 0, & b_r \in (b_r, +\infty) \\ a_r d_r, & b_r \in [2r_0, b_r] \end{cases}, \tag{2.3}$$

where d_r is the distance between the two bodies, r_0 is the physical radius of the robots, a_r and b_r are tunable factors, and in particular b_r represents the threshold distance that enables this behavior.

To keep a fixed formation, each robot of the swarm should keep a predefined angle and distance with respect to the others. If the robot R_j uses its neighbor R_i as a

reference, then its ideal position could be defined as

$$\begin{bmatrix} x_{j,id} \\ y_{j,id} \end{bmatrix} = \begin{bmatrix} x_i + d_r \cos(\alpha_{ij} + \theta) \\ y_i + d_r \sin(\alpha_{ij} + \theta) \end{bmatrix}, \tag{2.4}$$

where θ is the direction of the motion of the formation, and α_{ij} is the angle that such a direction makes with the line connecting the two robots. Hence, the behavior is set as

$$\boldsymbol{v}_{formation} = \frac{1}{\sqrt{(x_{j,id} - x_j)^2 + (y_{j,id} - y_j)^2}} \begin{bmatrix} x_{j,id} - x_j \\ y_{j,id} - y_j \end{bmatrix}. \tag{2.5}$$

The weighting parameter related to the formation-keeping behavior is described as

$$f_5(d_f) = \begin{cases} 0, & d_f \in (0, \epsilon) \\ a_f d_f, & d_f \in [\epsilon, +\infty) \end{cases}, \tag{2.6}$$

where d_f is the distance between the position of the robot and its ideal location in the formation, a_f is a tunable factor, and ϵ is the threshold value of d_f that activates the behavior.

In this simplified case with no tracking action, the combination of robot-avoidance and formation-keeping capabilities grants that the swarm is able to maintain its structure, ensuring no collision between its members.

2.3.3 Leader-follower strategy

The leader-follower architecture assumes that one robot is identified as the leader of the swarm, while the others, which are said followers, should react to its moves. In particular, concerning formation control, each follower should maintain a fixed relative position with respect to the leader of the swarm. The strengths of such a methodology are its intuitiveness and simplicity, but it lacks feedback from the followers to the leader, and that is one of the most evident drawbacks of this control architecture. Moreover, in case of failure of the leader (e.g. crash or loss of communication with the rest of the swarm) the formation control would stop working completely[20].

Choi *et al.* [21] analyze and implement a formation control based on the leader-follower architecture: the method used is the so-called separation-bearing, also known as (d, φ)-control. Such an approach relies on the fact that, in order to keep a compact formation, the followers should maintain the desired values of this couple of states with respect to the swarm leader. The first step of the algorithm is the implementation of exact formation control, that is the ability of keeping $d = d^{ref}$ and $\varphi = \varphi^{ref}$. Assuming that R_l is the leader robot and R_f is the follower, it is

$$v_f = v_l \frac{\cos\left(\beta - \varphi^{ref}\right)}{\cos\left(\varphi^{ref}\right)} \tag{2.7}$$

$$\omega_f = v_l \frac{\sin\left(\beta\right)}{d^{ref}\cos\left(\varphi^{ref}\right)}, \tag{2.8}$$

where β is the angular difference between the direction of the leader and the one of the follower, φ^{ref} is the reference value of the angle between the direction of the R_f and the virtual line connecting it to R_l, and d^{ref} is the reference value of the distance between the leader and the follower.

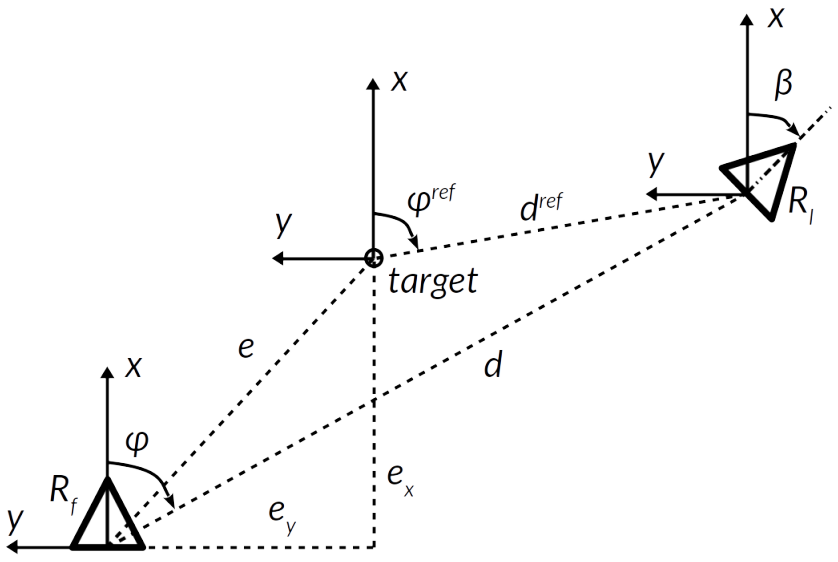


Figure 2.4: The scheme of the described leader-follower architecture.

The second step is an adjustment of equations 2.7 and 2.8 to make the position

error disappear at steady state. First of all, it is possible to decompose the error into its components parallel to the x and y axes of the follower reference system, as

$$e_x = d\cos(\varphi) - d^{ref}\cos\left(\varphi^{ref}\right) \tag{2.9}$$

$$e_y = d\sin(\varphi) - d^{ref}\sin\left(\varphi^{ref}\right), \tag{2.10}$$

where d is the distance between the leader and the follower and φ is the angular difference between the direction of R_f and the virtual line connecting it to R_l. Notice that the total error is computed as $e = \sqrt{e_x^2 + e_y^2}$.
Knowing the error, it is feasible to perform a correction to the target linear and angular velocities computed in equations 2.7 and 2.8 and set

$$v_f = v_l \frac{\cos\left(\beta - \varphi^{ref}\right)}{\cos\left(\varphi^{ref}\right)} + k_v e_x \tag{2.11}$$

$$\omega_f = v_l \frac{\sin(\beta)}{d^{ref}\cos\left(\varphi^{ref}\right)} + k_\omega e_y, \tag{2.12}$$

where k_v and k_ω are tunable parameters.

What was explained for just a single follower could be extended to larger swarms. As stated before, the problem of such a structure is the lack of feedback communication from the followers to the leader. Moreover, each follower drone only has knowledge about the leader of the formation, but it has no information about the position and the state of its peers in the swarm.

2.3.4 Artificial potential

A very interesting method to control the shaping and keeping of a formation of UAVs is based on the concept of artificial potential.

Definition[22]**.** *A scalar function $V(x) : \Omega \to \mathbb{R}$ is an artificial potential field defined at each point $x \in \Omega$, where Ω is a subset of $\mathbb{R}^m$.*

The gradient $\boldsymbol{\nabla}_x \boldsymbol{V}(x)$ of a potential field $V(x)$ is defined as

$$\boldsymbol{\nabla}_x \boldsymbol{V}(x) = \left[\frac{\partial V}{\partial x_1} \quad \frac{\partial V}{\partial x_2} \quad \cdots \quad \frac{\partial V}{\partial x_m}\right]^T, \tag{2.13}$$

and it determines a vector field in Ω, which is the starting point of the algorithm.

Howard *et al.* [23] study a system using a potential-based architecture to handle the autonomous distribution of a set of sensors with motion capability into a building. The presence of an artificial potential field U means that each body in the scheme experiences the effect of a force $\boldsymbol{F}$, derived as the gradient of the scalar field U, that is

$$\boldsymbol{F} = -\nabla U. \tag{2.14}$$

It is convenient to consider an analogy with an environment with N nodes carrying an electric charge each: the resulting electric potential field at the position of node i would be

$$U_i = k \sum_{\substack{j=1 \\ j \neq i}}^{N} \left(\frac{1}{\|\boldsymbol{r}_{ij}\|} \right), \tag{2.15}$$

where k is a constant value indicating the intensity of the potential field, and $\boldsymbol{r}_{ij}$ is the vector going to node i from node j.
The gradient of the artificial potential field can be rewritten and solved using the chain rule, as

$$\begin{aligned} \boldsymbol{F}_i &= -\frac{\mathrm{d}U_i}{\mathrm{d}\boldsymbol{x}} = \\ &= -\sum_{\substack{j=1 \\ j \neq i}}^{N} \left(\frac{\mathrm{d}U_i}{\mathrm{d}\|\boldsymbol{r}_{ij}\|} \cdot \frac{\mathrm{d}\|\boldsymbol{r}_{ij}\|}{\mathrm{d}\boldsymbol{x}} \right) = \\ &= -k \sum_{\substack{j=1 \\ j \neq i}}^{N} \left(\frac{1}{\|\boldsymbol{r}_{ij}\|^2} \cdot \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|} \right). \end{aligned} \tag{2.16}$$

Once the forces acting on each node have been defined, a suitable model to describe the motion is used. In particular, the chosen equation is

$$\ddot{\boldsymbol{x}}_i = \frac{\boldsymbol{F}_i - \nu \dot{\boldsymbol{x}}_i}{m_i}, \tag{2.17}$$

where $\ddot{\boldsymbol{x}}_i$ represents the acceleration of vehicle i, $\dot{\boldsymbol{x}}_i$ is its velocity, m_i denotes its mass and ν is the viscous friction coefficient.

Starting from a situation in which a certain velocity $\boldsymbol{v}$ is commanded to the vehicle at time t, then the goal of the control is to get the corrective value $\Delta\boldsymbol{v}$ to modify the velocity input at time $t + \Delta t$. Applying the stated idea to equation 2.17, and assuming the considered time interval to be small enough, it is

$$\Delta\boldsymbol{v}_i = \frac{\boldsymbol{F}_i - \nu\dot{\boldsymbol{x}}_i}{m_i} \cdot \Delta t. \tag{2.18}$$

According to the physical limitations imposed by the structure of the vehicle, it is possible, if necessary, to saturate the computed acceleration to the desired value. In particular, it is

$$\Delta\boldsymbol{v}_i = \begin{cases} \Delta\boldsymbol{v}_i, & \|\Delta\boldsymbol{v}_i\| \leq a_{max} \\ \dfrac{\Delta\boldsymbol{v}_i}{\|\Delta\boldsymbol{v}_i\|} a_{max}, & \|\Delta\boldsymbol{v}_i\| > a_{max} \end{cases}. \tag{2.19}$$

Following the definition of a suitable acceleration value, the correction is applied to the velocity command as

$$\boldsymbol{v}_i^{t+\Delta t} = \boldsymbol{v}_i^t + \Delta\boldsymbol{v}_i. \tag{2.20}$$

As it was done for the acceleration, the velocity needs to be saturated to a maximum value, specific for each vehicle:

$$\boldsymbol{v}_i = \begin{cases} \boldsymbol{v}_i, & \|\boldsymbol{v}_i\| \leq v_{max} \\ \dfrac{\boldsymbol{v}_i}{\|\boldsymbol{v}_i\|} v_{max}, & \|\boldsymbol{v}_i\| > v_{max} \end{cases}. \tag{2.21}$$

As usual, the computed velocity command is then sent to the lower-level controller, that in turn translates it into a message understandable by the vehicle motors.

An architecture based on the notion of artificial potentials is conceptually a strong control scheme. First of all, it is naturally oriented towards a distributed control strategy, and hence it is suitable also for large systems. Moreover, the idea implemented for the formation keeping task could be easily extended to other situations, for example obstacle avoidance, exploiting the data perceived by the sensors onboard.

Algorithm Artificial potential

while *true* **do**
    read the distances from all other robots
    compute the resulting virtual force
    calculate the needed velocity variation Δv
    if $\|\Delta \boldsymbol{v}\| > a_{max}$ **then**
        saturate the velocity variation
    end if
    compute the new theoretical velocity command
    if $\|\boldsymbol{v}\| > v_{max}$ **then**
        saturate the velocity
    end if
    send the command to the lower-level control
end while

Table 2.2: Artificial potential formation control algorithm[23].

2.3.5 Graph-based algorithm

Graph theory is the branch of mathematics that studies graphs.

Definition. *A graph G is a pair of finite sets (V, E). The elements of* $V = \{v_1, v_2, \ldots, v_n\}$ *are called the vertices (or nodes) of G, while the elements of* $E = \{e_1, e_2, \ldots, e_m\}$ *are called the edges (or arcs) of G. Moreover, each edge* $e \in E$ *is a set of two distinct nodes, usually denoted by* $e = (v, w)$ *with* $v, w \in V$*.*

Control theory is one of the various fields of application of the study of graphs. In particular, graphs are ideal to model and represent the complex structures that can be found in the analysis of multi-vehicular systems.

Marjovi *et al.* [24] deploy a control system whose goal is to make a group of vehicles keep a predefined formation on a highway. Although studied for a very specific scenario, this algorithm could be extended to a wider set of situations, including UAV's swarming.

Keeping into consideration N vehicles randomly placed in space, their position

could be indicated with the undirected graph $\mathcal{G} = (V, E)$, where the vertices V correspond to the vehicles, and the edges E represent the communication links. A first solution to the initial problem is given by

$$\dot{\boldsymbol{x}} = -\left(\boldsymbol{\mathcal{L}} \otimes \boldsymbol{I}_2\right)\left(\boldsymbol{x} - \boldsymbol{b}\right), \tag{2.22}$$

where $\boldsymbol{\mathcal{L}} = \boldsymbol{\mathcal{I}} \cdot \boldsymbol{\mathcal{W}} \cdot \boldsymbol{\mathcal{I}}^T$ is the Laplacian matrix, $\boldsymbol{\mathcal{I}}$ is the incidence matrix defining the edges of $\mathcal{G}$, $\boldsymbol{\mathcal{W}}$ is a weight matrix, $\boldsymbol{I}_2$ is the identity matrix, $\boldsymbol{x}$ includes the position of each vehicle and $\boldsymbol{b}$ indicates the desired offset of each vehicle from the center of the formation[25].

Although its proven stability, such a control algorithm is limited by the fact that it is static: this means that it is not able to properly react to a vehicle joining or leaving the formation after a certain time. The listed capabilities are added by implementing a second dynamic control algorithm.

The local neighborhood $\mathcal{N}_D$ of a node defines the vehicles close to itself according to the concept of topological distance instead of the classic euclidean one. For example, $\mathcal{N}_1$ denotes all the vehicles in the first layer around the reference, while $\mathcal{N}_2$ includes also the ones in the second tier, as depicted in figure 2.5. Each drone

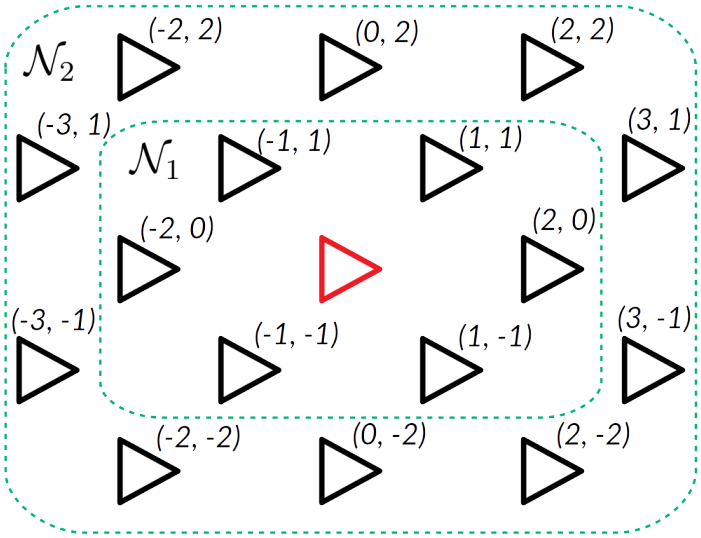


Figure 2.5: A vehicles' formation and the neighborhood definition.

denotes the others by a couple of values indicating the topological position with respect to itself, plus a global identifier, unique to each vehicle. Assuming that all the agents of the formation are capable of measuring the distance $e_{m,n}$ and the angle $\alpha_{m,n}$ to the other elements of a certain local neighborhood, the following dynamic control algorithm can be designed:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \sum\limits_{(m,n)\in\mathcal{N}_D} w_{m,n} \cdot \left(e_{m,n} \cos\left(\alpha_{m,n}\right) - m \cdot \left(\frac{b_x}{2} \right) \right) \\ \sum\limits_{(m,n)\in\mathcal{N}_D} w_{m,n} \cdot \left(e_{m,n} \sin\left(\alpha_{m,n}\right) - n \cdot b_y \right) \end{bmatrix}, \tag{2.23}$$

where (m, n) indicates the topological position of the considered UAV, $w_{m,n}$ is a tunable parameter, and b_x and b_y are the desired offset distances on the x and y axes respectively.

If compared to the previously proposed static algorithm, this one presents some fundamental advantages. First of all, it is based only on the relative position of the vehicles with respect to each other, hence it is not GNSS-dependent. Moreover, each vehicle establishes a communication only with the machines in its local neighborhood, without being influenced by possible changes happening in the rest of the structure. This feature makes the control algorithm fully decentralized, meaning that each vehicle computes its velocity command based only on the information coming from its own sensors. For this reason, the control is scalable to large formations of vehicles, possibly with dynamic changes.

2.3.6 Intelligent control

Intelligent control relies on the idea of emulating the human way of reasoning to bring the system to the desired state of stable equilibrium, and to be able to keep such a situation in time. Imitating the human brain means being able to adapt to situations even in the case of large uncertainties, learn from experience, and manage big sets of data to handle complex systems[26].

Rezaee *et al.* [27] implement a leader-follower fuzzy algorithm to control the formation flight of multiple UAVs. Fuzzy logic is opposed to traditional boolean

logic: the latter considers a variable to be either true or false (1 or 0), while the former accepts also intermediate values, which means a variable can be partially true or false. One of the biggest advantages of fuzzy controllers with respect to traditional ones is the usage of linguistic variables, which make the project more human-readable and relatively easy to implement even in case of partial knowledge of the system model.

Each UAV in the system is modeled as

$$\begin{aligned}
\dot{x} &= V\cos(\gamma)\cos(\psi) \\
\dot{y} &= V\cos(\gamma)\sin(\psi) \\
\dot{z} &= V\sin(\gamma) \\
\dot{V} &= \frac{T-D}{m} - g\sin(\gamma) \\
\dot{\psi} &= \frac{a_y}{V\cos(\gamma)} \\
\dot{\gamma} &= \frac{-g\cos(\gamma)}{V} + \frac{a_p}{V},
\end{aligned} \tag{2.24}$$

where $(x,\ y,\ z)$ indicate the position of the vehicle, V is the longitudinal speed, γ and ψ represent the pitch and yaw angles, T is the thrust, m denotes the mass of the UAV, g is the gravitational acceleration, D indicates the drag effect, and a_p and a_y are the pitch and yaw accelerations. Moreover, l and f subscripts are used to denote leader and followers.

The desired position of the follower vehicle with respect to the leader of the formation is defined as

$$\begin{aligned}
x_{fl} &= -r\cos(\zeta+\gamma_l)\cos(\chi+\psi_l) \\
y_{fl} &= -r\cos(\zeta+\gamma_l)\sin(\chi+\psi_l) \\
z_{fl} &= -r\sin(\zeta+\gamma_l),
\end{aligned} \tag{2.25}$$

where r is the reference distance between the leader and the follower, ζ represents the angle between the xy plane of the follower and the virtual line connecting it to the leader, and χ is the angle between the xz plane of the follower and that same virtual line.

Consequently, to keep the wanted position, the velocity command of the follower is determined to be

$$\begin{aligned} v_x &= \lambda\left(x_f - x_l - x_{fl}\right) + \dot{x}_l + \dot{x}_{fl} \\ v_y &= \lambda\left(y_f - y_l - y_{fl}\right) + \dot{y}_l + \dot{y}_{fl} \\ v_z &= \lambda\left(z_f - z_l - z_{fl}\right) + \dot{z}_l + \dot{z}_{fl}, \end{aligned} \tag{2.26}$$

where λ is a tunable parameter.

From equations 2.26, it is possible to define the desired values of the control outputs as

$$\begin{aligned} V_{fd} &= \sqrt{v_x^2 + v_y^2 + v_z^2} \\ \psi_{fd} &= \arctan2\left(v_y, v_x\right) \\ \gamma_{fd} &= \arctan2\left(\frac{v_z}{\sqrt{v_x^2 + v_y^2}}\right), \end{aligned} \tag{2.27}$$

and hence the errors of the state variables as

$$\begin{aligned} e_V &= V_f - V_{fd} \\ e_\psi &= \psi_f - \psi_{fd} \\ e_\gamma &= \gamma_f - \gamma_{fd}. \end{aligned} \tag{2.28}$$

The studied fuzzy algorithm is designed to have four inputs (e_V, e_ψ, e_γ, and V_f) coming from the previously described model, and three outputs (T_f, a_{yf}, and a_{pf}) that are then sent to the actuators of the vehicle. The following step consists in the definition of the membership functions of the specified variables, that are illustrated in figures 2.6, 2.7, and 2.8.

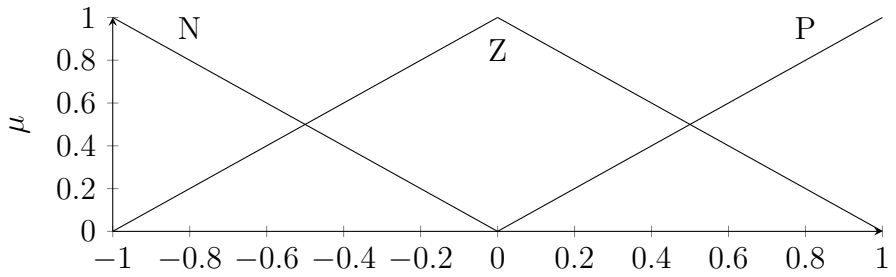


Figure 2.6: Membership function of e_V, e_ψ, e_γ and T_f[27].

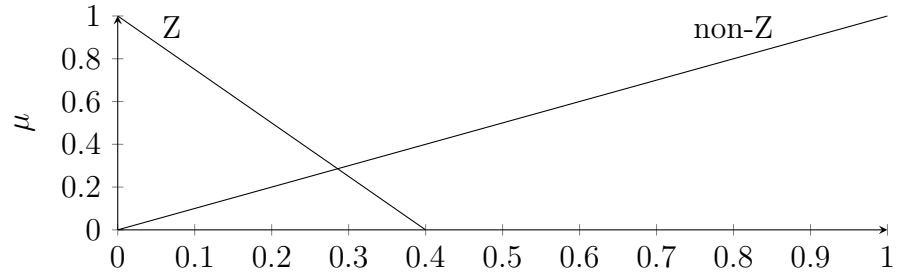


Figure 2.7: Membership function of V_f[27].

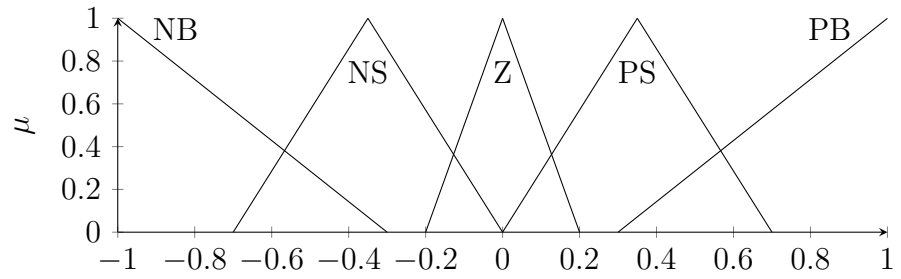


Figure 2.8: Membership function of a_{yf} and a_{pf}[27].

Then, the algorithm is designed to give a suitable linguistic value of the output depending on the analyzed inputs, following the logic rules defined in tables 2.3, 2.4, and 2.5. Notice that NS stands for negative small, N for negative, NB for negative big, Z for zero, non-Z for non-zero, PS for positive small, P for positive, and PB for positive big.

e_V \ V_f	Z	non-Z
P	N	N
N	P	P
Z	Z	Z

Table 2.3: Fuzzy rules for T_f[27].

e_ψ \ V_f	Z	non-Z
P	NB	N
N	PB	P
Z	Z	Z

Table 2.4: Fuzzy rules for a_{yf}[27].

e_γ \ V_f	Z	non-Z
P	NB	N
N	PB	P
Z	Z	Z

Table 2.5: Fuzzy rules for a_{pf}[27].

The last step of the studied control algorithm is the so-called defuzzification, that is the process of translating the linguistic values previously obtained into quantifiable data, according to some conversion scale specific for each application.

The used methodology is relatively easy, as it could be noticed by analyzing the straightforward logic behind the used fuzzy rules. Moreover, it has the major advantage of being computationally not expensive, and such a peculiarity would guarantee good performances even for bigger swarms. On the other side, the leader-follower architecture specific of this study introduces the typical limits that have been described in section 2.3.3.

Chapter 3

Tracking techniques

Trajectory tracking is a central topic in the discussion about potential applications of autonomous drones, and more in general mobile vehicles. This capability is present also in some commercial drones, which use various techniques to detect the objective and follow its motion. The different approaches strongly depend on the available information coming from the onboard sensors, in addition to the number of UAVs working collaboratively to chase the same reference. The possible applications of such a feature are countless, ranging from recreational scopes to military reasons.

3.1 Tracking algorithms

In this section, some of the procedures to get tracking ability already present in the state of the art will be analyzed. In particular, some details will be discussed about multilateration techniques, visual-based algorithms, and iterative approaches.

3.1.1 Multilateration

True-range multilateration indicates a procedure used to compute the position of a reference body by exploiting the geometry of the system. This approach is built on the knowledge of the absolute distance to the object to be localized from N anchors, whose position is well-known[28]. Once the localization issue is solved, then the tracking command can be easily obtained via various control techniques, the simplest of which is a proportional control on the position.

Sang *et al.* [29] explore one possible way to solve multilateration problems for UWB-based positioning systems. In an ideal situation, localizing an object univocally would require intersecting three circles in 2D, and four spheres in 3D. Though, since each ranging measurement comes with an error, an exact unique solution to the proposed system is very unlikely to be found. Then, to overcome the issue, the goal becomes finding the solution that better approximates the real one. For this reason, the more anchors are included in the architecture, the more the localization algorithm gains precision and accuracy.
Considering a local reference system, let us define the unknown position of the objective, that is the UWB tag, as $(x_t,\ y_t,\ z_t)$. Then, the spheres centered in every anchor, whose radii are the distances from the tag, can be denoted analytically as

$$r_i^2 = (x_i - x_t)^2 + (y_i - y_t)^2 + (z_i - z_t)^2 , \tag{3.1}$$

where i is an indicator of the considered anchor, and r_i is the range between station i and the tag.
Taking into account two of the system anchors, it is possible to linearize the equation by subtracting equation 3.1 referred to i to the same considering j:

$$\begin{aligned} (x_j - x_i)\, x_t + (y_j - y_i)\, y_t + (z_j - z_i)\, z_t = \\ = \frac{1}{2}\left[r_i^2 - r_j^2 + \left(x_j^2 + y_j^2 + z_j^2\right) - \left(x_i^2 + y_i^2 + z_i^2\right)\right] \end{aligned} \tag{3.2}$$

Repeating this subtraction for each couple of anchors, the resulting equations could be grouped to form a matrix equation:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{3.3}$$

where

$$\boldsymbol{A} = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \\ \vdots & \vdots & \vdots \\ x_{N-1} - x_0 & y_{N-1} - y_0 & z_{N-1} - z_0 \end{bmatrix}, \qquad \boldsymbol{x} = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix},$$

$$\boldsymbol{b} = \frac{1}{2} \begin{bmatrix} r_0^2 - r_1^2 + (x_1^2 + y_1^2 + z_1^2) - (x_0^2 + y_0^2 + z_0^2) \\ r_0^2 - r_2^2 + (x_2^2 + y_2^2 + z_2^2) - (x_0^2 + y_0^2 + z_0^2) \\ r_0^2 - r_3^2 + (x_3^2 + y_3^2 + z_3^2) - (x_0^2 + y_0^2 + z_0^2) \\ \vdots \\ r_0^2 - r_{N-1}^2 + \left(x_{N-1}^2 + y_{N-1}^2 + z_{N-1}^2\right) - (x_0^2 + y_0^2 + z_0^2) \end{bmatrix}.$$

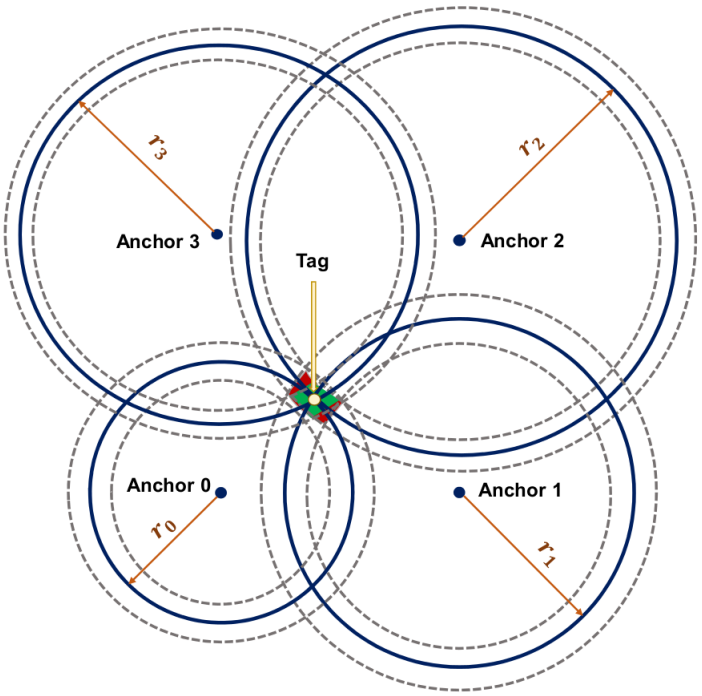


Figure 3.1: Localization of a tag using 4-anchor multilateration[29].

Once the problem is set, it is possible to find the best possible solution, keeping into consideration the Gaussian errors that affect the ranging measurements. The only condition to be verified is the invertibility of the matrix $\boldsymbol{A}^T\boldsymbol{A}$, and it is done by evaluating the column rank of matrix $\boldsymbol{A}$. In particular, the problem has a solution

if the anchors do not lie on the same line in 2D, and if they are not on the same plane in 3D. Then, the best possible solution is found applying the least-squares method, that is

$$\boldsymbol{x} = \left(\boldsymbol{A}^T \boldsymbol{A}\right)^{-1} \boldsymbol{A}^T \boldsymbol{b}. \tag{3.4}$$

Usually, to improve the quality of the performed localization, some weighting and filtering techniques are used, to remove the outliers and get better and coherent results.

3.1.2 Visual-based method

Another tracking methodology relies on the data coming from a camera to find and follow the objective. Cameras are often already present on many drones, because they are useful to carry on lots of different tasks. Images, if analyzed properly, provide more than enough information to solve the tracking issue. To reduce the burden on the onboard computational unit, a detail that stands out and is easily recognizable can be used to characterize the reference body: some examples are peculiar shapes, QR codes, or other particular stickers.

Sato [30] implements a system to control the motion of a UAV, according to what is filmed by an onboard camera. The pictures taken by the drone are sent via ROS to a ground station, where they are analyzed by a computer running OpenCV. After correcting the image distortion, the first goal is to detect the recognizable features of the chased body. Then, supposing that the ideal situation is to have the found feature in the center of the picture, it is possible to define the errors on the x and y axes of the image, measured in pixels, as

$$e_x = x_c - x_f \tag{3.5}$$

$$e_y = y_c - y_f, \tag{3.6}$$

where $(x_c,\ y_c)$ is the center of the image, and $(x_f,\ y_f)$ are the coordinates of the characteristic feature, as pictured in figure 3.2.

Concerning the z axis, the objective is to keep a fixed reference altitude r_z, hence the vertical error is defined as

$$e_r = r_z - z_f, \tag{3.7}$$

where z_f is the altitude of the drone with respect to the ground, measured by an ultrasound sensor.

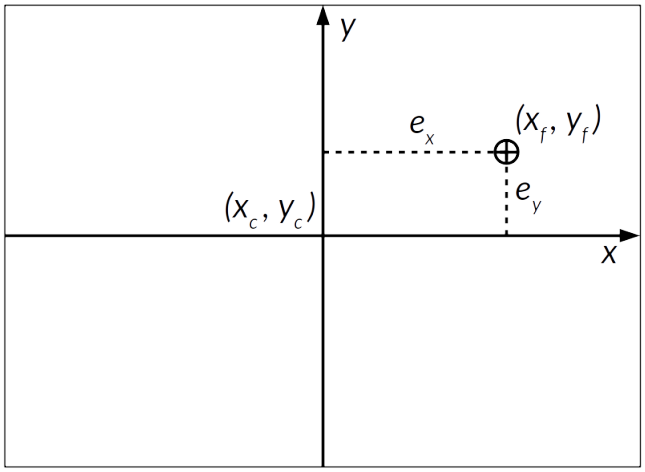


Figure 3.2: The captured image with the coordinates of the recognizable feature on the reference body.

The following step is the conversion of the distances from pixels to metric units, that is done experimentally, by moving the reference horizontally of regular intervals in the x and y directions. Then, from the obtained couples of values in pixels and metric units, a mathematical relationship is extrapolated via least-squares method. At this point, the control law can be defined as a function of the introduced errors:

$$\begin{aligned} u_x(t) &= k_{P_x} e_x(t) - k_{D_x} \frac{\mathrm{d}x_f(t)}{\mathrm{d}t} \\ u_y(t) &= k_{P_y} e_y(t) - k_{D_y} \frac{\mathrm{d}y_f(t)}{\mathrm{d}t} \\ u_z(t) &= k_{P_z} e_z(t), \end{aligned} \tag{3.8}$$

where k_{P_x}, k_{D_x}, k_{P_y}, k_{D_y}, and k_{P_z} are tunable parameters.
The computed commands are then sent to the autopilot of the drone, that will automatically transform them into suitable motor inputs.

The analyzed methodology, though, presents some disadvantages if compared to the others. Since it is based on images, it works only if a clear line of sight is available between the UAV and its objective. Moreover, image processing requires a discrete amount of computational power, which in the analyzed case is overcome by sending the data to a ground station. Considering multiple anchors, sending all the data to the same computing station would introduce a dangerous single point of failure in the system. If possible, this situation should always be avoided.
The algorithm is summarized in table 3.1.

Algorithm Visual-based tracking

1: estimate the relationship between a pixel in the image and the equivalent in the real world
2: **while** *true* **do**
3: take a picture from the drone
4: correct the image distortion
5: detect the recognizable feature of the target
6: compute the x and y errors from the image (in pixels)
7: compute the z error from the ultrasound sensor (in meters)
8: convert the x and y errors to meters
9: apply the control law
10: **end while**

Table 3.1: Visual-based tracking control algorithm[30].

3.1.3 Iterative approach

In situations with multiple chasing vehicles acting collaboratively towards the same goal, it is possible to exploit distance data coming from different locations in space. Starting from a scenario very similar to the one presented when introducing the multilateration approach, it is possible to get a solution via successive iterations.

Jiang *et al.* [31] propose a localization algorithm that relies on iterative estimation of the centroid of the anchors. Let us consider N known nodes, each of which

is in position $(x_i,\ y_i,\ z_i)$, and one unknown target to be located, denoted with $(x,\ y,\ z)$. With the described information, it is possible to analytically compute the center of the swarm, that is the first step of the algorithm, as

$$\begin{aligned} x_{O_1} &= \frac{1}{N}\sum_{i=1}^{N} x_i \\ y_{O_1} &= \frac{1}{N}\sum_{i=1}^{N} y_i \\ z_{O_1} &= \frac{1}{N}\sum_{i=1}^{N} z_i. \end{aligned} \tag{3.9}$$

The distance between the found centroid and the target could be indicated by

$$d_{O_1} = \sqrt{(x_{O_1} - x)^2 + (y_{O_1} - y)^2 + (z_{O_1} - z)^2}, \tag{3.10}$$

which through some mathematical passages simplifies to

$$d_{O_1} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (d_i)^2 - \frac{1}{N^2}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} (d_{ij})^2}, \tag{3.11}$$

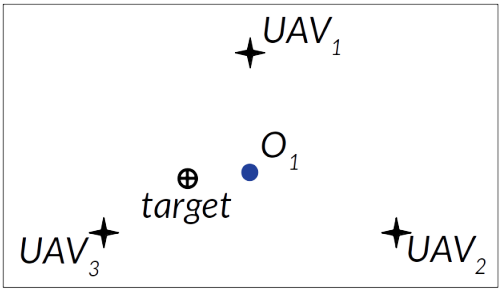
where d_i denotes the distance between anchor i and the target, and d_{ij} is the distance from node i to node j.

It is immediate to prove that there exists at least one anchor that is further away from the target than the computed centroid. Then, the same passage is repeated after substituting the node with the biggest d_i with a virtual node in position O_n. The successive goal is the definition of a stopping criterion, to univocally set a condition for the algorithm to stop iterating. In particular, the analyzed study derives it by comparing two successive centroids obtained via iteration, and setting a suitable threshold. When such a condition is fulfilled, the point O_n is a sufficiently good approximation of the position of the tracked body. Hence, the stopping criterion is defined as

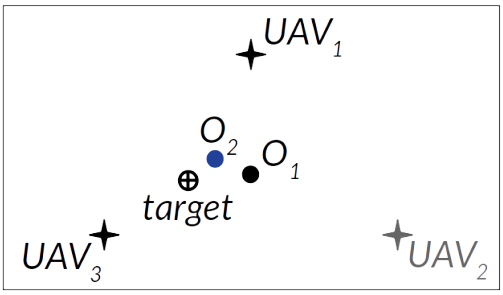
$$d^2_{O_{n-1}} - d^2_{O_n} < \epsilon, \tag{3.12}$$

where ϵ is the tunable threshold.

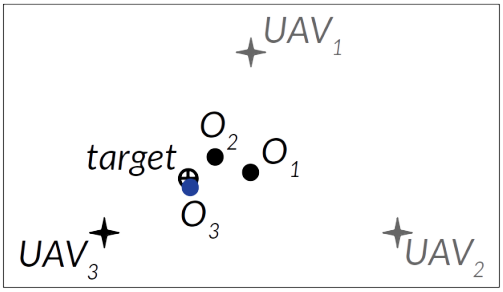
Once the position of the target is found with enough accuracy, it is not difficult to



(a) 1st iteration.



(b) 2nd iteration.



(c) 3rd iteration.

Figure 3.3: The first three iterations of the iterative localization algorithm.

define a simple control architecture to compute the commands in the x, y, and z directions.

This study was realized thinking about the localization of a target moving inside the space delimited by the sensors. Although very accurate and efficient in the standard situation, it does not work correctly if the objective body leaves that

space. The reason is not difficult to verify intuitively: indeed, the centroid that approximates the searched position can never be outside of the space delimited by the anchors.

The algorithm is summarized in table 3.2.

Algorithm Iterative tracking

```
1: define a stopping criterion
2: while true do
3:     read the anchors' position
4:     read the distances between the anchors
5:     read the distance between the anchors and the target
6:     repeat
7:         compute the centroid of the swarm
8:         compute the distance of the centroid from the target
9:         substitute the further node with the centroid
10:    until the stopping criterion is not fulfilled
11:    apply a simple control law on the position
12: end while
```

Table 3.2: Iterative tracking control algorithm[31].

Chapter 4

Project development

This project was developed at *PoliTO Interdepartmental Centre for Service Robotics (PIC4SeR)*. PIC4SeR is a research center whose goal is the integration of researchers with different fields of expertise to carry on studies on service robotics and innovative applications.

The work of the current thesis aims at studying, implementing, and simulating a system to successfully perform formation flight of a small number of UAVs. The first goal of the project is to test the capability of the drones to maintain a compact formation while moving towards a desired GNSS location. The subsequent challenge of the swarm is to chase the trajectory of a reference body, which could be either terrestrial or aerial, whose global position is unknown. The only available information is the GNSS location of the drones, also denoted as anchors, and the output of the UWB sensors mounted on each vehicle involved in the system, which give the distance between each couple of bodies.

Possible applications of such a system include deploying a set of autonomous drones to follow a reference entity that operates in GNSS-denied environments. Moreover, having a sufficient number of anchors, they could operate conceptually like satellites, allowing to compute the position of the followed body via multilateration or other

localization techniques.

4.1 Control architecture

After evaluating various possibilities, analyzing the current state of the art, we chose to implement a partially decentralized velocity control to succeed in the fulfillment of the described objectives.
The algorithm is conceptually subdivided into two parts: the first one has the goal of formation-keeping, the second one instead aims at guaranteeing stable tracking of the reference, be it a GNSS position or a physical body. The output of each of the two pieces of the algorithm is a velocity vector: by summing the two obtained velocities, a fully functioning control is obtained as

$$\boldsymbol{v} = \boldsymbol{v}^{swarm} + \boldsymbol{v}^{track}, \tag{4.1}$$

where $\boldsymbol{v}^{swarm}$ and $\boldsymbol{v}^{track}$ will be better defined in sections 4.2 and 4.3 respectively. The swarming part of the algorithm is implemented in a decentralized fashion: after receiving all the needed data from the other anchors, each UAV computes on-board its own velocity command. On the other side, the tracking algorithm is naturally oriented towards a centralized structure. Since the tracking velocity is the same for each component of the formation, it is computed only once by the ground station, and then provided to each anchor, which will apply equation 4.1 and transmit the command to the lower-level controllers.

4.2 Swarming control

Due to the structure of the tracking algorithm, based on the partial information that the robots have about the system, the formation-keeping task is fundamental, not only to avoid collisions but also to guarantee the correct pursuit of the objective.

First of all, each anchor should collect the needed information, either from its own sensors, or from the networks connecting it to the other agents. In particular, it should receive the GNSS position of all the other anchors, that will be used to compute the direction unit vectors between itself and its peers. Knowing the position of every anchor, the following step consists in the computation of the compass angle between itself and every other UAV. It is determined as

$$\theta_{ij} = \operatorname{arctan2}\left\{\lambda_i - \lambda_j,\ \log\left(\frac{\tan\left(\frac{\varphi_i}{2} + \frac{\pi}{4}\right)}{\tan\left(\frac{\varphi_j}{2} + \frac{\pi}{4}\right)}\right)\right\}, \tag{4.2}$$

where φ and λ denote latitude and longitude respectively, and the subscript i denotes the anchor on which the swarming algorithm is running.
Once computed the angle θ_{ij}, it is possible to compute the unit vector in the direction that goes from vehicle i to vehicle j, that is

$$\boldsymbol{u}_{ij} = \begin{bmatrix} \sin(\theta_{ij}) \\ \cos(\theta_{ij}) \\ 0 \end{bmatrix}, \tag{4.3}$$

where the first term of the vector represents the component in the east direction, and the second represents the north direction.
Moreover, to improve the performance of the algorithm even in the case of low GNSS precision, we use the data from the UWB sensors to compute the distance norm. Since the goal of this project is to maintain a planar formation, what we actually need is the horizontal distance between each couple of UAVs. To correct the absolute distance coming from the UWB sensor, the best solution is to avoid using the GNSS altitude data, which has usually lower precision than the latitude and longitude ones. For this reason, a laser sensor pointing downwards is used to get the relative altitude of the drone, and the horizontal distance between any two bodies is computed as

$$d_{hor,ij} = \sqrt{(d_{UWB,ij})^2 - (d_{laser,i} - d_{laser,j})^2}, \tag{4.4}$$

assuming a planar ground surface.

Concerning this part of the algorithm, we opted for a control based on the concept of artificial potential, in particular applying a slightly modified version of what was studied by de Vries and Subbarao in [32]. The potential function is chosen to have a quadratic attractive part and an exponential repulsive part, as

$$J_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \left(\frac{1}{2} a \left(d_{hor,ij} \right)^2 + \frac{bc}{2} e^{-\frac{\left(d_{hor,ij} \right)^2}{c}} \right), \tag{4.5}$$

where N is the number of members of the formation, and a, b, and c are positive tunable parameters[33].

Consequently, the velocity command is obtained by differentiating the artificial potential function 4.5, obtaining

$$\boldsymbol{v}_{hor,i}^{swarm} = -\nabla J_i = -\sum_{\substack{j=1 \\ j \neq i}}^{N} \left[d_{hor,ij} \left(a - b e^{-\frac{\left(d_{hor,ij} \right)^2}{c}} \right) \boldsymbol{u}_{ij} \right]. \tag{4.6}$$

In the described velocity command, it is possible to distinguish between a linear attractive component and an exponential repulsive part. Such a design choice was made to guarantee higher reactivity in case of two drones getting too close to each other, with respect to the situation of a couple of UAVs incrementing too much their relative distance. Indeed, the primary goal is to avoid collisions that would cause structural damages to the vehicles.

Since the previously described control algorithm commands only the horizontal velocity of the anchors, it is necessary to implement a mechanism to maintain the desired swarming altitude. To reach such a goal, we used a simple proportional controller: the vertical command is defined as

$$v_{z,i}^{swarm} = k_h \left(h_{des} - d_{laser,i} \right), \tag{4.7}$$

where k_h is a tunable parameter, and h_{des} is the desired swarming altitude.

Hence, the swarming velocity vector is finally defined as

$$\boldsymbol{v}_i^{swarm} = \boldsymbol{v}_{hor,i}^{swarm} + \begin{bmatrix} 0 \\ 0 \\ v_{z,i}^{swarm} \end{bmatrix}. \tag{4.8}$$

The analyzed steps are summarized in table 4.1.

Algorithm Swarming

1: **while** *true* **do** ▷ The following algorithm runs on each anchor
2: read the current GNSS position and relative altitude
3: read the GNSS position and relative altitude of every anchor
4: derive the direction unit vectors between itself and every other anchor
5: read the distance between itself and every other anchor
6: compute the horizontal distance between itself and every other anchor
7: apply the horizontal swarming velocity algorithm
8: compute the vertical velocity
9: send the command to the lower-level control
10: **end while**

Table 4.1: Algorithm for collision-free formation hovering.

4.2.1 Equilibrium distance

In the simple case of $N = 2$, it is possible to compute the equilibrium distance between the vehicles by finding the value for which equation 4.6 goes to zero. Solving the equation, we obtain

$$d_{eq,2} = \sqrt{c \log\left(\frac{b}{a}\right)}. \tag{4.9}$$

Instead, assuming to have $N = 4$ drones being part of the formation, which is the situation we studied and simulated, we can compute the side of the regular polygon that the UAVs will maintain over time. This value will be useful later on to evaluate the performance of the swarming algorithm in the simulation and testing phases.

Proof. The side of the square will be computed by solving the velocity algorithm for the vehicle laying on the positive side of the y axis, but a similar proof could be done for the other drones.
Due to the symmetry of the formation, it is clear that $v_x = 0$ for all possible values

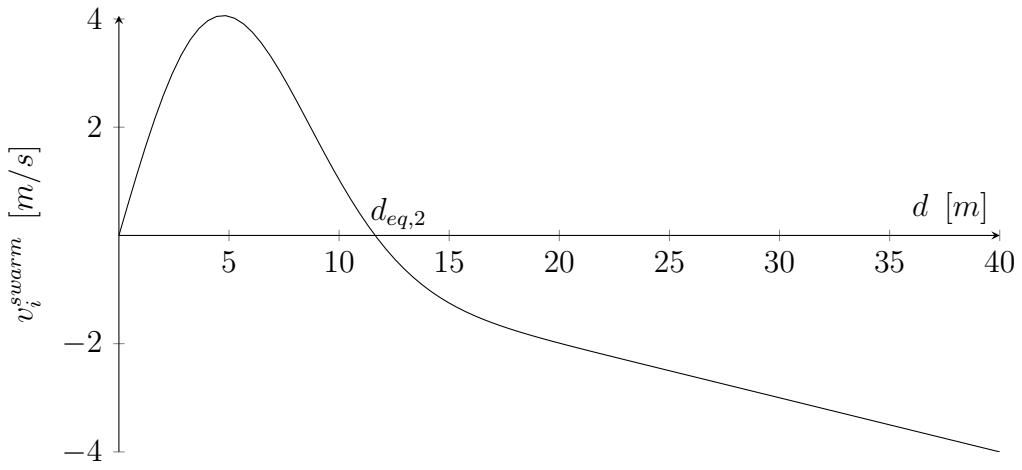


Figure 4.1: Swarm velocity function with $N = 2$, $a = 0.1$, $b = 1.5$, and $c = 50$.

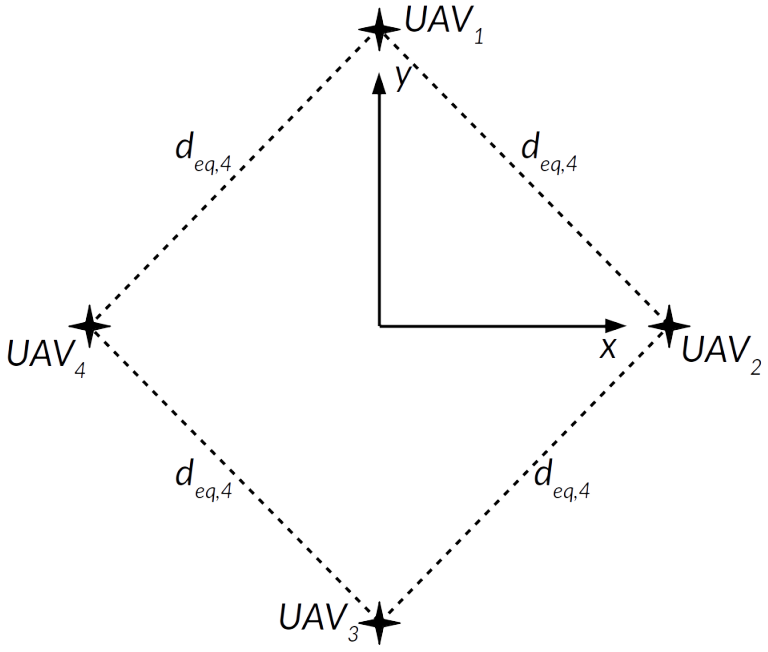


Figure 4.2: The stable structure of the formation of UAVs.

of the side of the square. Instead, concerning the velocity in the y direction, it is

$$v_y = \underbrace{\frac{d_4}{\sqrt{2}}\left(a - be^{-\frac{d_4^2}{c}}\right)}_{UAV_2} + \underbrace{2\frac{d_4}{\sqrt{2}}\left(a - be^{-\frac{\left(2\frac{d_4}{\sqrt{2}}\right)^2}{c}}\right)}_{UAV_3} + \underbrace{\frac{d_4}{\sqrt{2}}\left(a - be^{-\frac{d_4^2}{c}}\right)}_{UAV_4} =$$

$$= \sqrt{2}d_4\left(a - be^{-\frac{d_4^2}{c}}\right) + \sqrt{2}d_4\left(a - be^{-\frac{2d_4^2}{c}}\right) =$$
$$= \sqrt{2}d_4\left[2a - b\left(e^{-\frac{d_4^2}{c}} + e^{-\frac{2d_4^2}{c}}\right)\right].$$

Then, setting $v_y = 0$, we get

$$\sqrt{2}d_{eq,4}\left[2a - b\left(e^{-\frac{d_{eq,4}^2}{c}} + e^{-\frac{2d_{eq,4}^2}{c}}\right)\right] = 0 \qquad \Longrightarrow \qquad d_{eq,4} = 0, \text{ unfeasible}$$

$$e^{-\frac{d_{eq,4}^2}{c}} + e^{-\frac{2d_{eq,4}^2}{c}} = \frac{2a}{b}.$$

Substituting $\gamma = e^{-\frac{d_{eq,4}^2}{c}}$ into the previous equation, it is

$$b\gamma^2 + b\gamma - 2a = 0$$
$$\gamma = \frac{-b \pm \sqrt{b^2 + 8ab}}{2b}.$$

Discarding the negative solution and substituting back, we obtain

$$-\frac{d_{eq,4}^2}{c} = \log\left(\frac{-b + \sqrt{b^2 + 8ab}}{2b}\right)$$
$$d_{eq,4} = \pm\sqrt{-c\log\left(\frac{-b + \sqrt{b^2 + 8ab}}{2b}\right)}.$$

Discarding also in this case the negative solution, it is

$$d_{eq,4} = \sqrt{c\log\left(\frac{2b}{\sqrt{b^2 + 8ab} - b}\right)}. \tag{4.10}$$

□

4.3 Tracking control

The second part of the algorithm consists of a subtask running on a central node, possibly on a ground station or on a companion computer mounted on the physical followed reference. For what regards the tracking, it is necessary to distinguish between two situations: traveling in formation towards a known GNSS goal, or chasing the trajectory of a moving body, without information about its global position.

4.3.1 GNSS position tracking

The situation that we are going to describe in this section is conceptually the easier of the two. Since the position of all the members of the formation is known, it is possible to compute the geographic angular coordinates of the center of the swarm, as

$$\varphi_c = \frac{1}{N}\sum_{i=1}^{N}\varphi_i \tag{4.11}$$

$$\lambda_c = \frac{1}{N}\sum_{i=1}^{N}\lambda_i, \tag{4.12}$$

where φ and λ denote latitude and longitude respectively.
Knowing both the start and destination positions, it is possible to compute the direction unit vector between the two, using equations 4.2 and 4.3. It is worth noticing that the third component of the vector, that represents the z axis radial to the Earth surface, is always null because the altitude control is handled onboard by the swarming algorithm.
Then, the distance between the current position of the center of the swarm and the destination point is determined using the `Inverse` function, from the `geographiclib.geodesic` library[34].
Having all the information that is needed, a simple proportional control on the distance to be covered is implemented, determining the east and north components of the required velocity. Mathematically, the control is denoted as

$$\boldsymbol{v}_{track} = k_p d_{cd} \boldsymbol{u}_{cd}, \tag{4.13}$$

where k_p is a tunable parameter, and d_{cd} is the distance to the destination as computed in the previous step, and $\boldsymbol{u}_{cd}$ indicates the direction to be followed.
Notice that, for big values of d_{cd} in equation 4.13, the presented algorithm outputs a proportionally big value of the command velocity. Although, the input to the rotors is automatically saturated to a maximum value by lower-level controllers, always guaranteeing a suitable input to the engines, as shown in figure 4.3.
The analyzed steps are summarized in table 4.2.

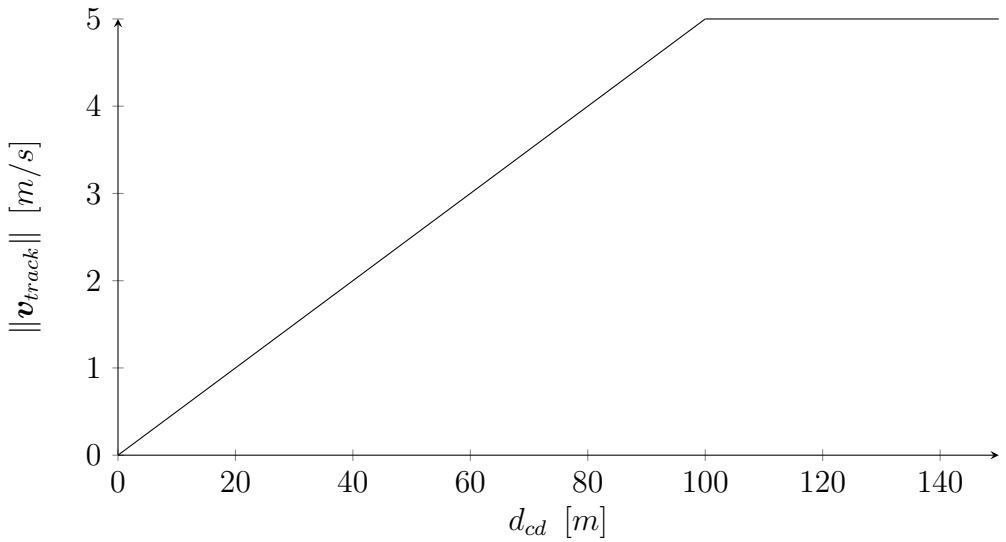


Figure 4.3: Tracking velocity function with $k_p = 0.05$, and $\|\boldsymbol{v}\|_{max} = 5\ m/s$.

Algorithm GNSS position tracking

```
read the GNSS position to be reached
if no destination has been received then
    set the current position as destination
end if
while the position is not reached do
    read the GNSS position of every anchor
    compute the coordinates of the swarm center
    calculate the route compass angle
    derive the direction unit vector
    compute the distance to be traveled
    calculate the tracking velocity vector
    send the command to each anchor
end while
```

Table 4.2: Algorithm for GNSS position tracking.

4.3.2 UWB target tracking

Tracking a moving reference, instead, requires a more complex algorithm, in particular if the condition imposes to perform the tracking operation without

knowing the GNSS position of the body. In addition to the information about the global position of the UAVs, their relative distance, and their altitude with respect to the ground, we should consider to have the absolute distance between each anchor and the vehicle to be followed, given by the UWB sensors. The first step, hence, is to collect all the data from the drones' sensors and compute the direction unit vectors using equations 4.2 and 4.3.

The tracking algorithm that we implemented is based on the assumption that a specific formation is maintained. In our case, such an assumption is valid, because the previously described swarming algorithm ensures the stability of the formation at steady state. Considering these conditions, the examined situation is similar to the one described in figure 4.4, analyzing the scenario with four anchors in the swarm. The desired action is the motion of the center of the swarm towards

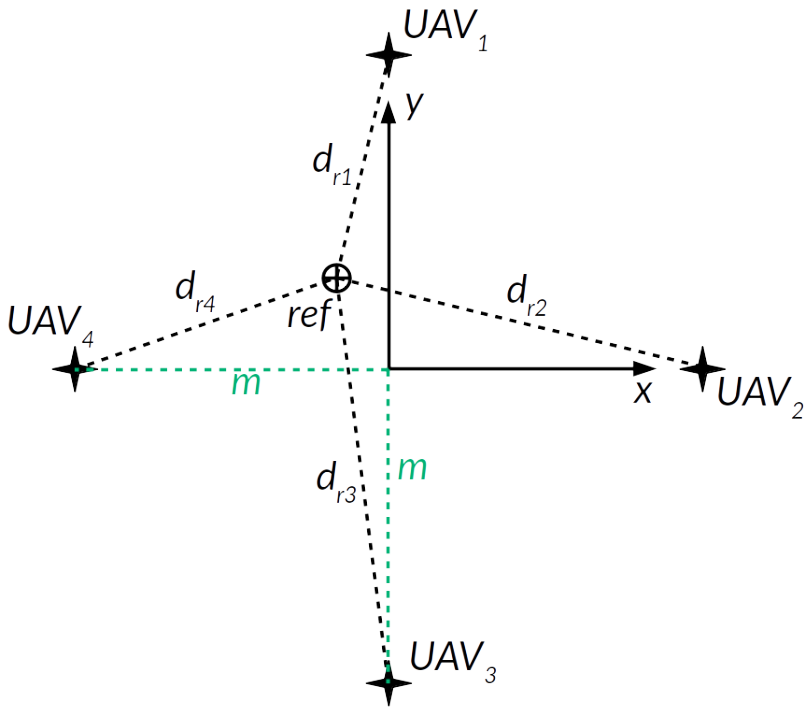


Figure 4.4: The structure of the swarm that deploys the tracking algorithm on the xy plane.

the projection of the reference position on the formation plane. Before moving

to an analytical description, we will try in the following paragraph to explain the intuitive reasoning behind the implemented tracking algorithm.
Considering the simplest case pictured in figure 4.5, which is a formation of two drones, we could track the position of the reference body only if it moves on the line connecting the anchors. In such a case, it is sufficient to compare the distance of each UAV from the followed body to understand the direction of motion to be commanded. If the distance from vehicle i is smaller than the distance from vehicle j, then the center should move towards the position of vehicle i, otherwise, the opposite should happen. The extension to four anchors allows to chase the reference in the three-dimensional space, following the presented basic idea: each couple of UAVs is considered, and the vectorial sum of the obtained components gives a command that is proportional to the velocity to be followed, as will be proved later.

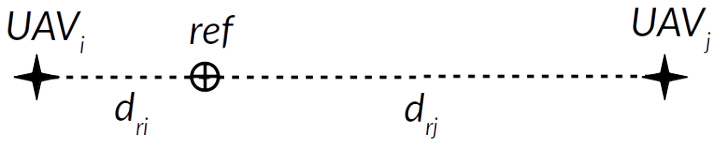


Figure 4.5: The structure of the swarm that deploys the tracking algorithm.

The formal definition of what was intuitively described in the previous paragraph is

$$\boldsymbol{v}_p^{track} = \frac{1}{\beta} \sum_{i=1}^{N} \left(\sum_{\substack{j=1 \\ j \neq i}}^{N} (d_{ri} - d_{rj})\, \boldsymbol{u}_{ji} \right), \tag{4.14}$$

where β is a tuning parameter, N is the number of vehicles in the swarm, d_{ri} is the euclidean distance between the reference and drone i, d_{rj} is the euclidean distance between the reference and drone j, and $\boldsymbol{u}_{ji}$ is the unit vector going from drone i to drone j.

Proof. Assuming to have $N = 4$ drones in the swarm, which is the scenario shown in figure 4.4, they will position themselves to form a square formation. It is then

possible to decompose the tracking velocity into its x and y components.

$$
\begin{aligned}
v_x = {} & (d_{r1} - d_{r2}) \cdot \cos(-45°) + (d_{r1} - d_{r3}) \cdot \cos(-90°) \\
& + (d_{r1} - d_{r4}) \cdot \cos(-135°) + (d_{r2} - d_{r1}) \cdot \cos(135°) \\
& + (d_{r2} - d_{r3}) \cdot \cos(-135°) + (d_{r2} - d_{r4}) \cdot \cos(180°) \\
& + (d_{r3} - d_{r1}) \cdot \cos(90°) + (d_{r3} - d_{r2}) \cdot \cos(45°) \\
& + (d_{r3} - d_{r4}) \cdot \cos(135°) + (d_{r4} - d_{r1}) \cdot \cos(45°) \\
& + (d_{r4} - d_{r2}) \cdot \cos(0°) + (d_{r4} - d_{r3}) \cdot \cos(-45°) = \\
= {} & \sqrt{2}(d_{r1} - d_{r2}) - \sqrt{2}(d_{r1} - d_{r4}) - \sqrt{2}(d_{r2} - d_{r3}) - 2(d_{r2} - d_{r4}) \\
& - \sqrt{2}(d_{r3} - d_{r4}) = \\
= {} & -2\left(\sqrt{2} + 1\right) d_{r2} + 2\left(\sqrt{2} + 1\right) d_{r4} = \\
= {} & 2\left(\sqrt{2} + 1\right)(d_{r4} - d_{r2})
\end{aligned}
$$

$$
\begin{aligned}
v_y = {} & (d_{r1} - d_{r2}) \cdot \sin(-45°) + (d_{r1} - d_{r3}) \cdot \sin(-90°) \\
& + (d_{r1} - d_{r4}) \cdot \sin(-135°) + (d_{r2} - d_{r1}) \cdot \sin(135°) \\
& + (d_{r2} - d_{r3}) \cdot \sin(-135°) + (d_{r2} - d_{r4}) \cdot \sin(180°) \\
& + (d_{r3} - d_{r1}) \cdot \sin(90°) + (d_{r3} - d_{r2}) \cdot \sin(45°) \\
& + (d_{r3} - d_{r4}) \cdot \sin(135°) + (d_{r4} - d_{r1}) \cdot \sin(45°) \\
& + (d_{r4} - d_{r2}) \cdot \sin(0°) + (d_{r4} - d_{r3}) \cdot \sin(-45°) = \\
= {} & \sqrt{2}(d_{r1} - d_{r2}) - 2(d_{r1} - d_{r3}) - \sqrt{2}(d_{r1} - d_{r4}) - \sqrt{2}(d_{r2} - d_{r3}) \\
& + \sqrt{2}(d_{r3} - d_{r4}) = \\
= {} & -2\left(\sqrt{2} + 1\right) d_{r1} + 2\left(\sqrt{2} + 1\right) d_{r3} = \\
= {} & 2\left(\sqrt{2} + 1\right)(d_{r3} - d_{r1})
\end{aligned}
$$

Let us denote with m the horizontal distance from each anchor to the center of the swarm, with h the vertical distance between the reference and the center of the

formation, and with the couple $(x_r,\ y_r)$ the position of the reference with respect to the origin.

$$\begin{aligned}
\|\boldsymbol{v}\|^2 &= v_x^2 + v_y^2 = \\
&= \left[2\left(\sqrt{2}+1\right)(d_{r4} - d_{r2})\right]^2 + \left[2\left(\sqrt{2}+1\right)(d_{r3} - d_{r1})\right]^2 = \\
&= 4\left(\sqrt{2}+1\right)^2 \left[(d_{r4} - d_{r2})^2 + (d_{r1} - d_{r3})^2\right] = \\
&= 4\left(\sqrt{2}+1\right)^2 \left\{ \left[\sqrt{(-m - x_r)^2 + y_r^2 + h^2} - \sqrt{(m - x_r)^2 + y_r^2 + h^2}\right]^2 \right. \\
&\qquad \left. + \left[\sqrt{x_r^2 + (m - y_r)^2 + h^2} - \sqrt{x_r^2 + (-m - y_r)^2 + h^2}\right]^2 \right\} \approx
\end{aligned}$$

Performing the 4^{th} order Taylor expansion of the obtained equation around the origin, that is the center of the swarm, we get

$$\begin{aligned}
&\approx 4\left(\sqrt{2}+1\right)^2 \frac{4m^2}{m^2+h^2}\left(x_r^2 + y_r^2\right) = \\
&= \frac{16m^2\left(\sqrt{2}+1\right)^2}{m^2+h^2}\left(x_r^2 + y_r^2\right)
\end{aligned}$$

To get the norm of the velocity, it is sufficient to compute the square root of the previous equation, obtaining

$$\|\boldsymbol{v}\| \approx \frac{4m\left(\sqrt{2}+1\right)}{\sqrt{m^2+h^2}}\sqrt{x_r^2 + y_r^2}$$

This proves that, if the goal is well inside the square formed by the moving anchors, the norm of the computed velocity is directly proportional to its distance from the center of the formation.

$$\begin{aligned}
\angle\boldsymbol{v} &= \operatorname{arctan2}\{v_y,\ v_x\} = \\
&= \operatorname{arctan2}\{d_{r3} - d_{r1},\ d_{r4} - d_{r2}\} = \\
&= \operatorname{arctan2}\left\{\sqrt{x_r^2 + (-m - y_r)^2 + h^2} - \sqrt{x_r^2 + (m - y_r)^2 + h^2},\right. \\
&\qquad \left.\sqrt{(-m - x_r)^2 + y_r^2 + h^2} - \sqrt{(m - x_r)^2 + y_r^2 + h^2}\right\} \approx \qquad \star
\end{aligned}$$

Performing the 3rd order Taylor expansion of v_x and v_y around the origin, it is

$$v_x \approx \frac{2m}{\sqrt{m^2+h^2}} x_r \qquad\qquad v_y \approx \frac{2m}{\sqrt{m^2+h^2}} y_r$$

$$\star \quad \approx \arctan 2 \left\{ \frac{2m}{\sqrt{m^2+h^2}} y_r,\ \frac{2m}{\sqrt{m^2+h^2}} x_r \right\} =$$
$$= \arctan 2 \left\{ y_r,\ x_r \right\}$$

It means that the direction of the vector going from the center of the formation to the position of the reference is well approximated by the algorithm if the goal is sufficiently close to the center of the swarm. □

From a practical point of view, the error introduced by the tracking algorithm is small enough that, when the reference body is close to the center of the swarm compared to the size of the formation polygon, the system acts as a proportional control on the position. As pictured in figure 4.6, considering $m = 10$, $h = 15$, and a circle of radius equal to 1 meter around the origin, then the magnitude error is always lower than 7 millimeters. With the same data, figure 4.7 shows that the angle error is at most 0.007°: moreover, it is experimentally proved that this error never exceeds 0.7°, even if the reference is far away from the swarm, meaning that the formation is always able to reach the objective.
Besides, the more the inter-anchors distance and the vertical swarm altitude with respect to the reference increase, the more the tracking algorithm is precise. Figures 4.8 and 4.9 show the maximum value of the magnitude and angular errors respectively, when the reference lays inside a circle with a unitary radius. Analyzing the graphs, it is evident how increasing m and h drastically helps to improve the algorithm performances.

In order to guarantee the convergence to zero of the tracking error at steady state, though, algorithm 4.14 is not sufficient. For such a reason, an integral component (mathematically represented as a sum, since we consider to be in discrete time) is

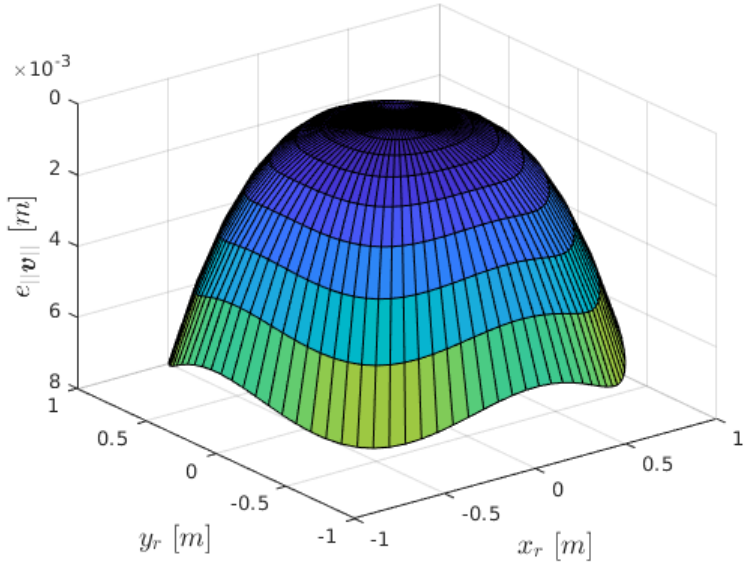


Figure 4.6: The magnitude error introduced by the tracking algorithm for $m = 10$ and $h = 15$.

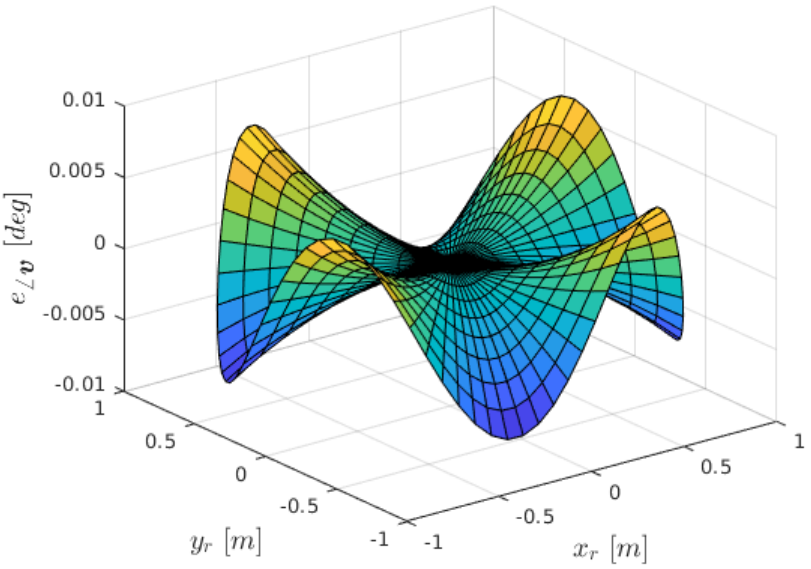


Figure 4.7: The angle error introduced by the tracking algorithm for $m = 10$ and $h = 15$.

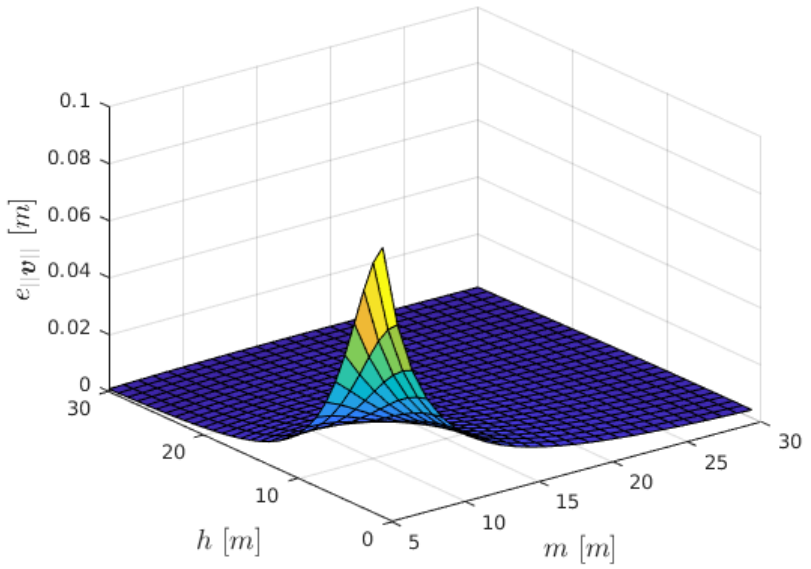


Figure 4.8: The maximum value of the magnitude error introduced by the tracking algorithm for $(x_r,\ y_r)$ inside a circle of radius equal to 1 meter.

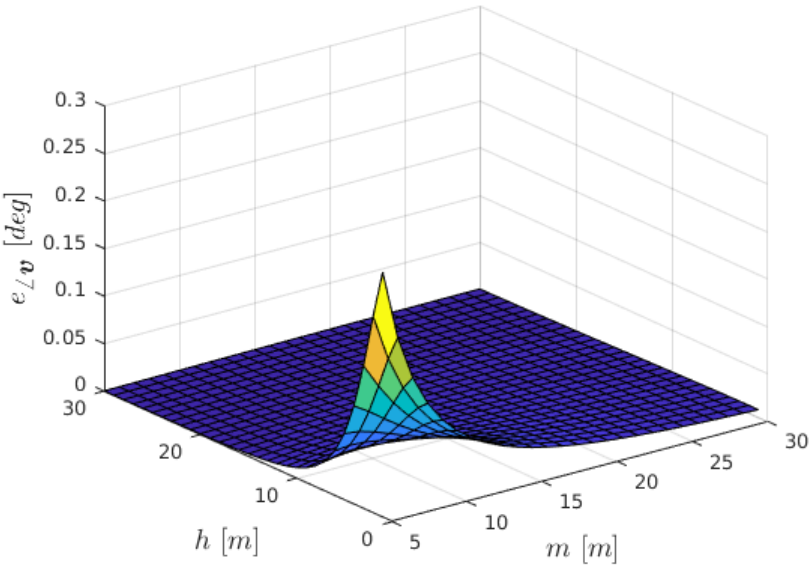


Figure 4.9: The maximum value of the angular error introduced by the tracking algorithm for $(x_r,\ y_r)$ inside a circle of radius equal to 1 meter.

introduced: the complete algorithm, then, becomes

$$\boldsymbol{v}^{track}(t) = \frac{1}{\beta}\sum_{i=1}^{N}\left(\sum_{\substack{j=1\\ j\neq i}}^{N}(d_{ri}-d_{rj})\,\boldsymbol{u}_{ji}\right) + \frac{1}{\gamma}\sum_{\tau=0}^{t-1}\boldsymbol{v}^{track}(\tau)\,, \tag{4.15}$$

where γ is a tuning parameter.

The analyzed steps are summarized in table 4.3.

Algorithm Reference tracking
while *true* **do**
read the GNSS position of every anchor
compute the direction unit vectors
read the UWB data from every sensor
apply the tracking algorithm
send the command to each anchor
end while

Table 4.3: Algorithm for reference tracking.

4.4 Attitude control

Since the vehicles used in this work are quadcopters, they do not have a preferred direction of motion on the xy plane. Due to their structure, indeed, they can move with the same effort in all the possible ways on a plane. Although, also considering future applications and improvements of the system, we implemented an attitude controller. Looking at the prospect of mounting a camera on the UAVs, it would be convenient to have the possibility to automatically adjust the orientation of the drones as desired. Due to the lack of precise indications about the requirements of the control, in this project, we implemented it in such a way that each drone is oriented in the direction of the tracking velocity.

The information needed to perform the described attitude control is the same that is output by the tracking algorithm. Once the tracking velocity vector $\boldsymbol{v}^{track}$

is known, it is sufficient to compute the desired compass angle as

$$\theta_{des} = \arctan2\left\{v_y^{track},\ v_x^{track}\right\}. \tag{4.16}$$

Then, the yaw error is computed by comparing the desired value to the orientation θ given by the compass mounted onboard, as

$$e_\theta = \theta_{des} - \theta. \tag{4.17}$$

If it is necessary, the yaw error should be corrected by adding or subtracting 360°, in such a way that it is always included in the interval $-180° < e_\theta \leq 180°$. Moreover, an ulterior check is needed on the tracking velocity. Indeed, to avoid undesired rotations of the UAVs, a threshold value is introduced to enable or disable the attitude control:

$$\omega_z = \begin{cases} 0, & \left\|\boldsymbol{v}^{track}\right\| < v_{th} \\ -\rho \cdot e_\theta, & \left\|\boldsymbol{v}^{track}\right\| \geq v_{th} \end{cases}, \tag{4.18}$$

where ρ and v_{th} are tunable parameters.

Algorithm Attitude control

```
1: while true do
2:     read the tracking velocity vector
3:     compute the tracking velocity angle
4:     read the onboard compass data
5:     compute the compass error
6:     if the error is out of (−180°, 180°] then
7:         add or subtract 360° to correct it
8:     end if
9:     if the tracking velocity is above the threshold then
10:        compute the rotational velocity
11:        send the command to the lower-level control
12:    end if
13: end while
```

Table 4.4: Algorithm for attitude control.

The threshold should be adjusted in such a way that, when the swarm reaches a stable position above an unmoving reference, a small corrective value of the tracking velocity would not trigger an attitude change.
The analyzed steps are summarized in table 4.4.

Chapter 5

Simulations

The following step is to set up a simulation environment to verify the correct functioning of the system, tune the parameters, and evaluate the overall performance of the used algorithm. Everything is implemented using the Robot Operating System (ROS), a set of frameworks suitable for the development and programming of robots. Each UAV runs a ROS node, that is a function executing at a given rate, able to read the needed data, perform the desired computations, and output the correct control command. The ROS nodes communicate with each other via publishing and subscribing to topics, which are data channels dedicated to predefined subjects. On every UAV, it is mounted a PX4 autopilot that communicates with the drone (including motors, sensors, and so on), using a very lightweight messaging protocol, called MAVLink[35]. The way these messages can be caught also by the ROS nodes, is through a package named MAVROS, that «is the "official" supported bridge between ROS and the MAVLink protocol. It enables MAVLink extendable communication between computers running ROS, MAVLink enabled autopilots, and MAVLink enabled ground control stations»[36]. The last fundamental element to set up the simulation is Gazebo, a common simulation interface, that allows to visually evaluate the time evolution of the tested events[37].

5.1 Sensors and parameters

For simplicity reasons, the simulations were performed by substituting the UWB and laser sensors with a ROS node, which geometrically computes the distance between two points starting from their GNSS positions. Anyway, this modification does not improve the performance of the system, because the precision of the simulated GNSS sensor is lower than the one of the UWB.

Referring to the notation introduced in chapter 4, the performed parameters tuning process yielded

Parameter	Value
N	4
a	0.1
b	1.5
c	50
k_h	1
h_{des}	5
k_p	0.05
β	4
γ	50
ρ	1
v_{th}	0.5

Table 5.1: Simulation parameters.

where N is the number of anchors in the swarm, a, b, c are the parameters of the swarming algorithm, k_h is the proportional factor of the altitude control, h_{des} is the desired swarming altitude, k_p is the proportional multiplier in the GNSS tracking algorithm, β and γ are the reciprocal of the proportional and integral factors in the reference tracking algorithm, ρ is the proportional parameter in the attitude controller, and v_{th} is the threshold velocity to activate the attitude control.
Applying equation 4.10 with the data of table 5.1, we get that the side of the formation square, in equilibrium conditions, is

$$d_{eq,4} \approx 10.31 \text{ meters},$$

and hence the distance of two anchors lying on opposite corners, that is the diagonal of the square, is

$$\sqrt{2}d_{eq,4} \approx 14.59 \text{ meters.}$$

5.2 Results

In the next pages, some significant results of the performed simulations will be depicted and analyzed. We combined different motion profiles of the reference body, with different starting configurations of the anchors. In particular, about the motion of the reference, we considered hovering, motion along a straight line, a broken line, according to pseudo-random velocities, and following a circular path. About the initial position of the anchors, instead, we placed them to form a square around the reference, a line close to the objective, and relatively distant from it.

5.2.1 Hovering

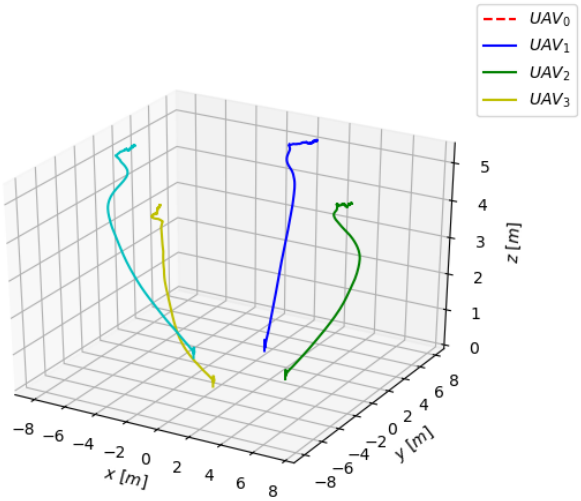


Figure 5.1: Local trajectories of the hovering drones.

This simulation is useful to evaluate the performance of the swarming algorithm.

Indeed, there is no reference body to be followed, and no objective GNSS position is given. Hence, the only goal of the anchors is to form the swarm and maintain a stable formation in time. In this case, the anchors, start from a square configuration. In figure 5.1, it is possible to appreciate the motion of the UAVs, and we can notice that they slightly drift. Though, figure 5.2 shows that, after a short adjustment phase in the first seconds after the takeoff, the distance between each couple of vehicles reaches and keeps the expected equilibrium value, with good precision and excellent stability.

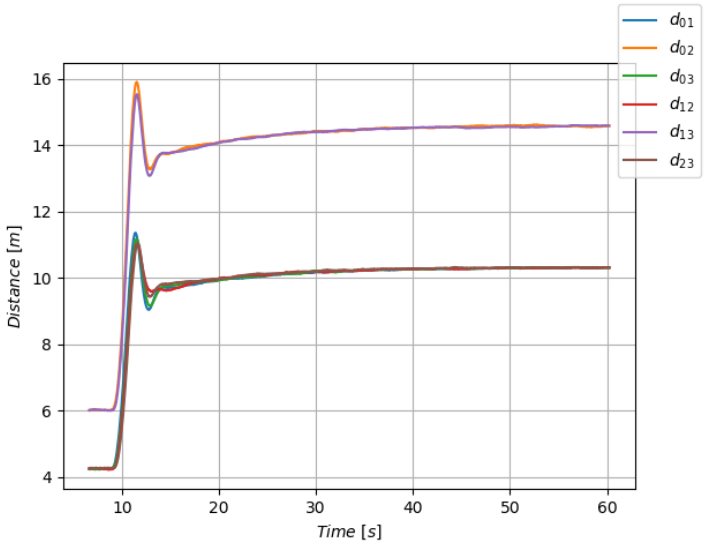


Figure 5.2: Relative distances between the UAVs.

5.2.2 Straight line

This simulation includes the reference body, hence it gives the possibility to evaluate also the tracking algorithm. It takes into consideration the simplest reference motion, that is at constant velocity along a straight trajectory.

Figure 5.4 shows that, after an initial time interval for adjustments, the equilibrium values for the inter-UAVs distances are reached. In figure 5.5, we could notice that the tracking error goes to 0 at steady state, verifying what was expected by

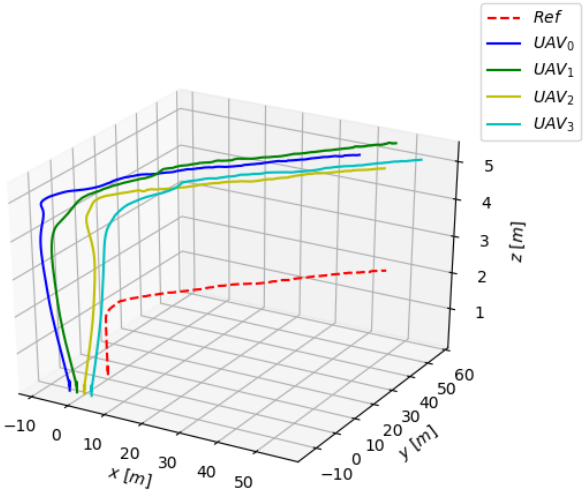


Figure 5.3: Trajectories of the anchors in the local reference system.

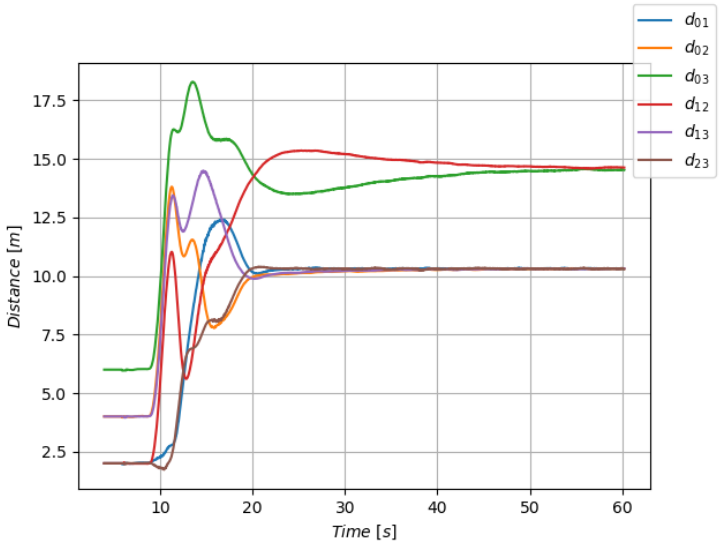


Figure 5.4: Relative distances between each couple of UAVs.

the trajectory tracking algorithm. Moreover, the correct operation of the attitude controller is confirmed by the curves depicted in figure 5.6, which represent the heading errors tending to 0 after the takeoff phase. Complexively, in this very simple scenario, the overall system behaves as it was theoretically analyzed in

chapter 4.

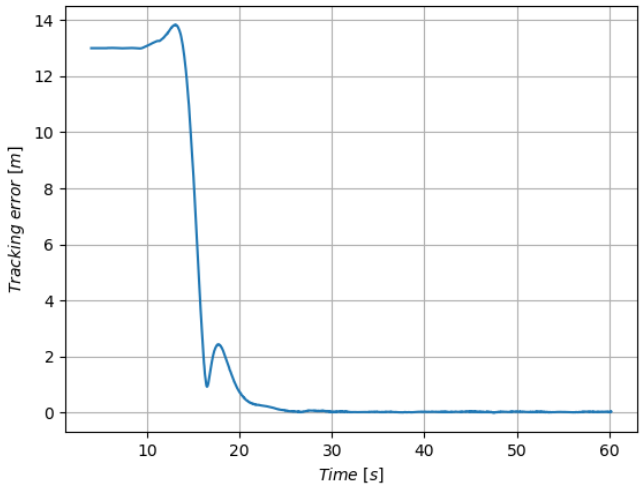


Figure 5.5: Reference tracking error.

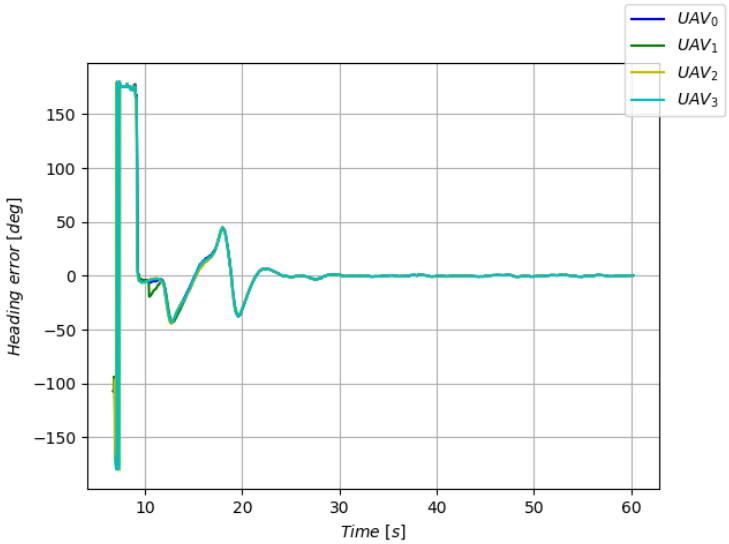


Figure 5.6: Heading error of each anchor.

5.2.3 Broken line

The following simulation introduces a new term of complexity, which is a sudden change of trajectory in the motion of the reference body.

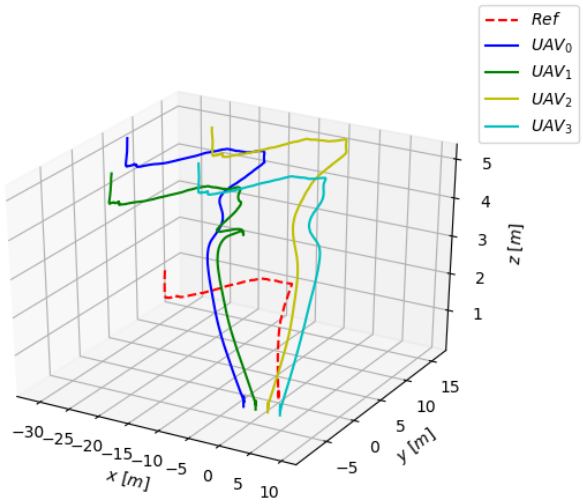


Figure 5.7: Anchors' path in the local reference system.

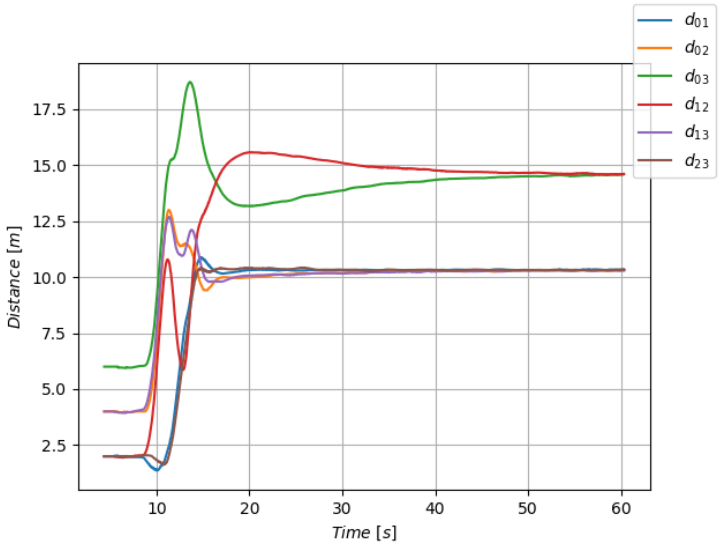


Figure 5.8: Formation inter-drones distances.

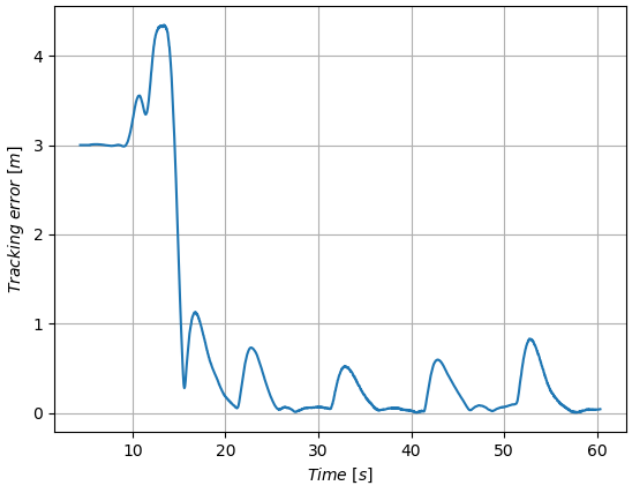


Figure 5.9: Objective tracking error.

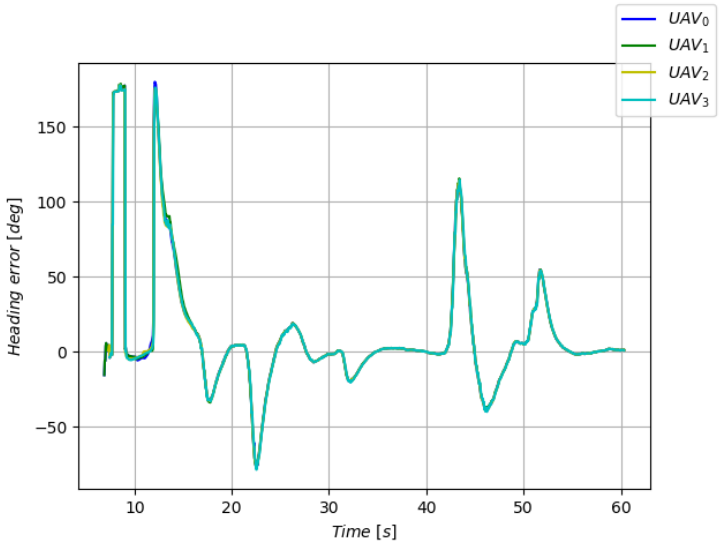


Figure 5.10: Attitude error of every formation member.

As it can be seen in figure 5.7, every 10 seconds, the reference body changes both the direction and the norm of its velocity, generating a sequence of straight lines. The curve depicted in figure 5.9 well represents this motion: indeed, when the reference changes its trajectory, the tracking error rapidly increases until the swarm

reacts to what happened and gets to the correct location. It is worth noticing that the tracking error always stays below 1 meter: considering the square diagonal value of 14.59 meters computed before, the reference is well inside the formation, as desired. The same behavior occurs to the attitude control, as pictured in figure 5.10.

5.2.4 Random movement

In this configuration, both the horizontal direction and the magnitude of the velocity are updated every time the autopilot receives a new command. In the current case, the frequency at which the ROS nodes run is equal to 20 Hz, hence the motion of the reference body randomly changes every 50 milliseconds.

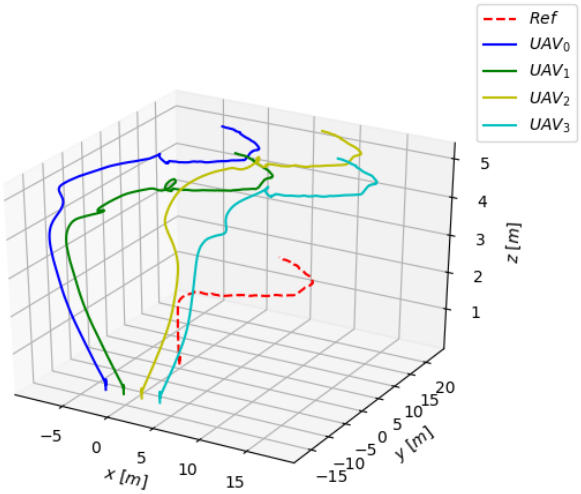


Figure 5.11: Local random trajectory of the system components.

Figure 5.13 shows that also in this case, the tracking algorithm guarantees the expected behavior of the swarm with respect to the reference to be chased. The random movement makes it impossible to bring to 0 the tracking error, but the simulation proves that the horizontal distance between the center of the swarm and the reference remains small compared to the size of the anchors' formation.

From figure 5.14, it is noticeable that also in the case of the attitude control, the heading can not get permanently to the desired value, anyway it swings around it as expected.

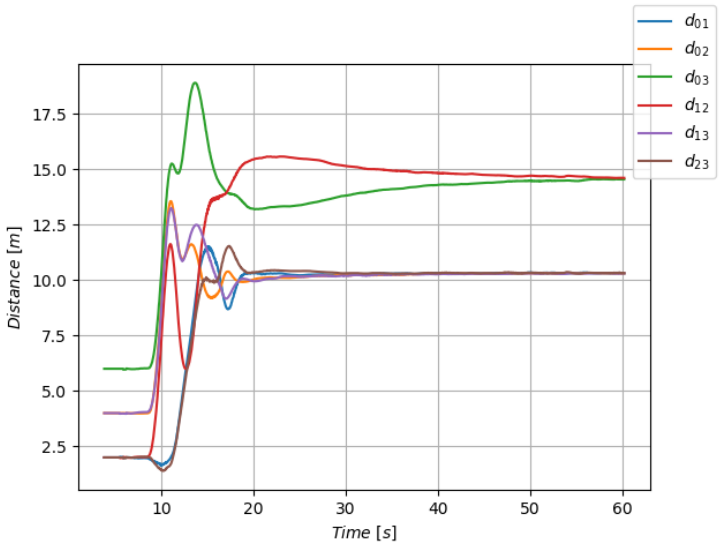


Figure 5.12: Length of sides and diagonals of the polygon generated by the drones.

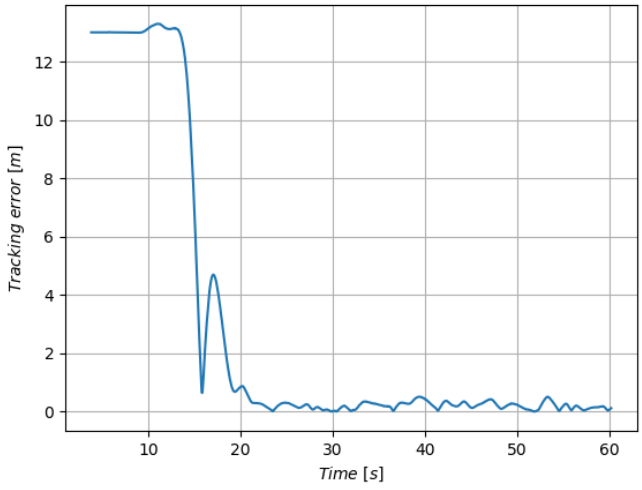


Figure 5.13: Tracking error of a randomly moving object.

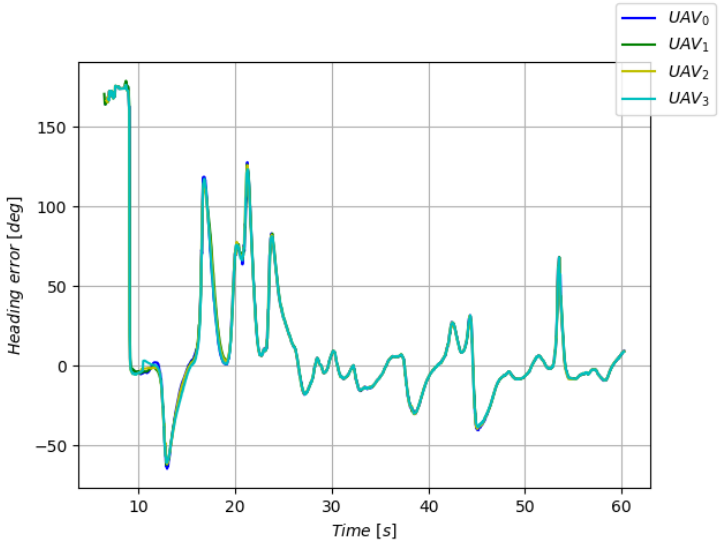


Figure 5.14: Orientation error of the UAVs.

5.2.5 Circular path

The last analyzed simulation takes into account a circular trajectory of the objective, as is evident by figure 5.15.

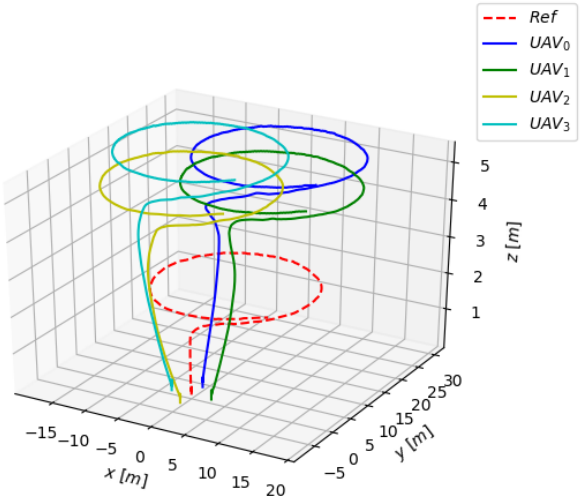


Figure 5.15: System's bodies trajectories with respect to a local origin.

Taking off from an advantageous configuration, the anchors need very little adjustments to create the correct formation, and there are no complications in getting to the right distances from one to another, as illustrated in figure 5.16.

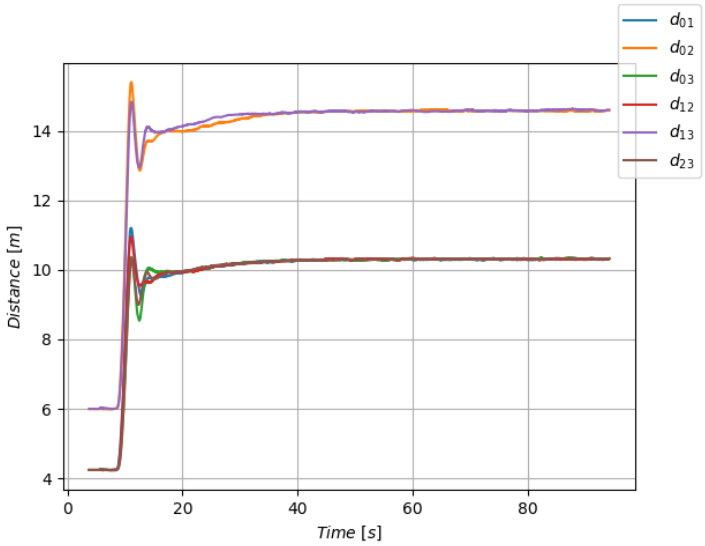


Figure 5.16: Relative spacing between each pair of anchors.

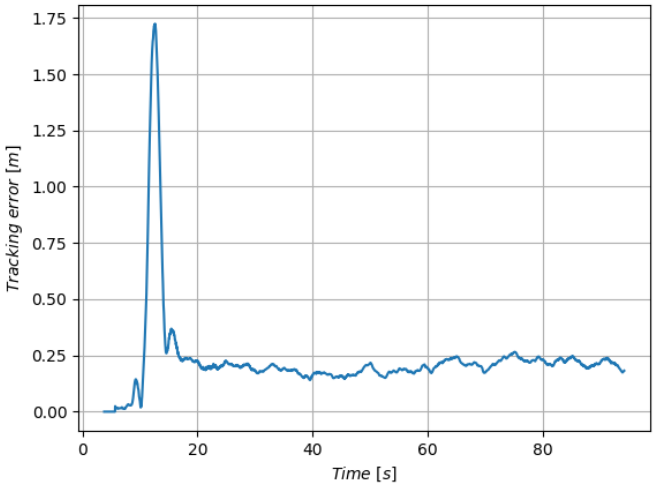


Figure 5.17: Circular path's tracking error.

Looking at the behavior of the tracking error in figure 5.17, it is immediate to

notice that we are only able to lower it to a certain extent. This occurs because of the lack of a derivative component in the control algorithm. Although, we made such a choice because the oscillations of the GNSS and UWB data would make it extremely difficult, if not impossible, to obtain a helpful contribution from a derivative control. Notice that, considering the swarm square to have the computed 14.59 meters diagonal, the value of the error does not represent a problematic situation for the system capabilities. The same issue is present also in the attitude control, as depicted in figure 5.18.

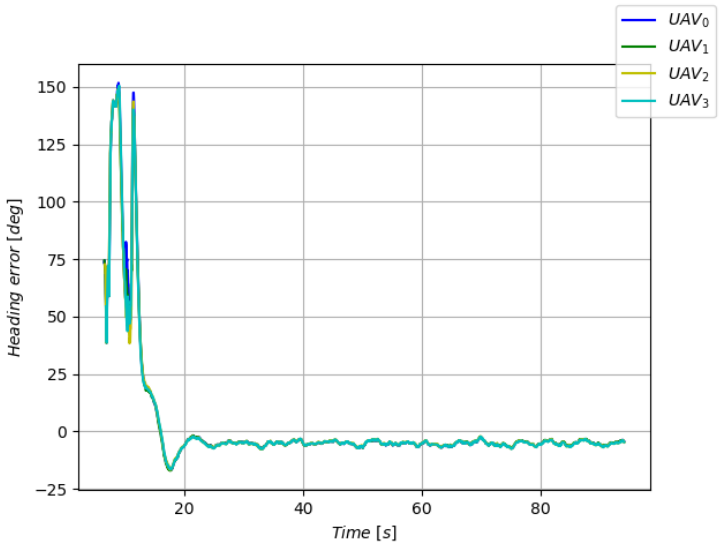


Figure 5.18: Direction error of every drone.

5.3 Hardware configuration

The very last phase of the work included the first steps towards the testing process: mounting and configuring the drones and the onboard sensors needed to correctly run the swarming and tracking algorithms.

We decided to use four quadcopters, choosing in particular the *DJI F450* airframe[38]. Connected through a *Holybro PM02* power module[39], each arm hosts a *RCS TRX 400* brushless motor, which rotates an *APC 10×4.7* propeller[40], able to guarantee

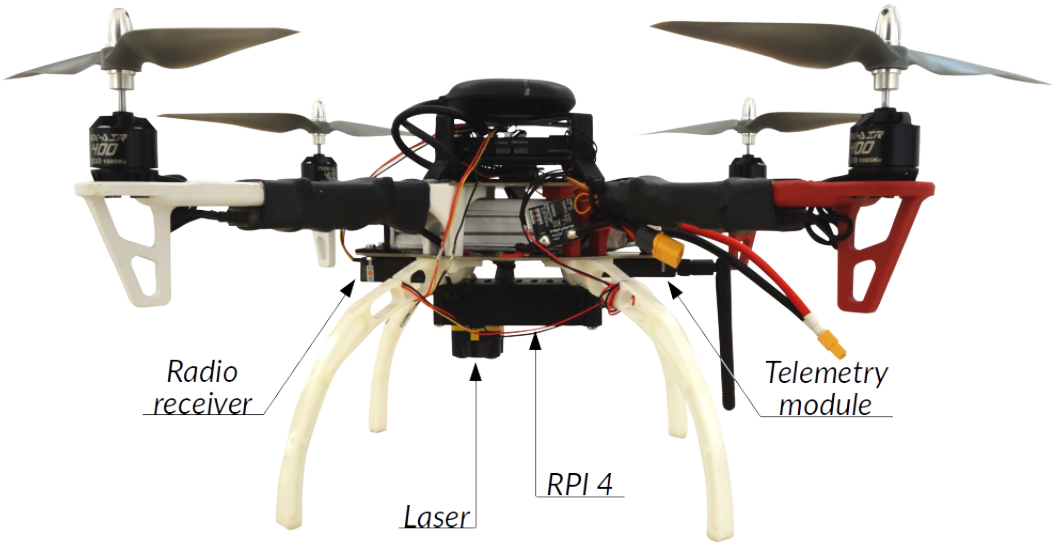


Figure 5.19: Bottom view of the mounted drone.

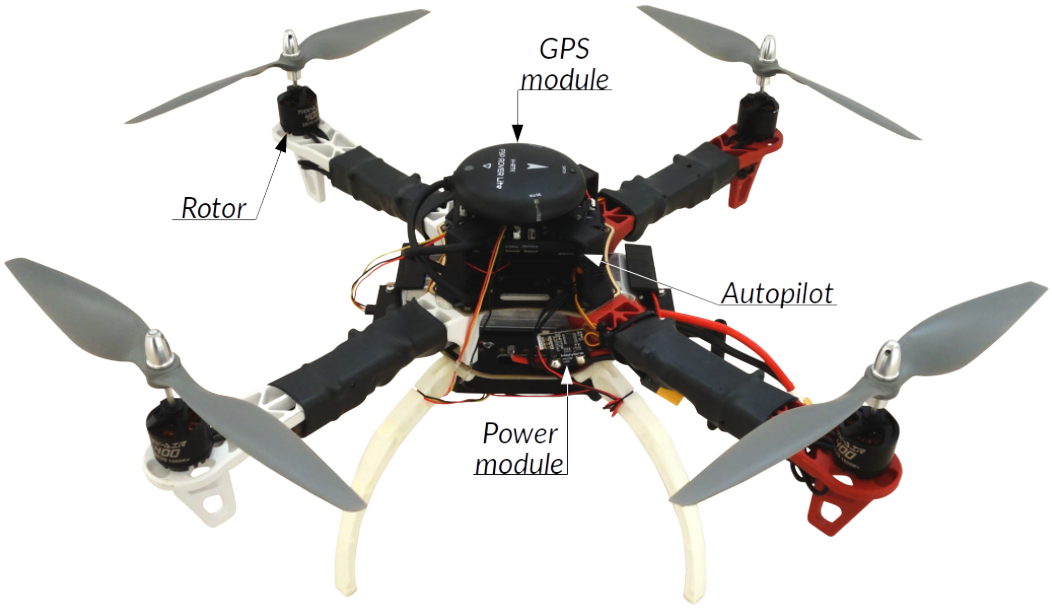


Figure 5.20: Side view of the UAV.

enough thrust to lift the drone. The quadrotors are equipped with a *Pixhawk 4* autopilot[41], designed and produced by *Holybro* in collaboration with the PX4 team. The autopilot receives the information it needs from various sensors added

Figure 5.21: Top view of the assembled vehicle.

to the configuration: in our case, the most important ones are the *H-RTK F9P Rover Lite* GNSS receiver[42], and the *TeraRanger Evo 60m* distance sensor[43]. The *Holybro Transceiver Telemetry Radio V3 433MHz*[44] mountend onboard has a crucial role within the system: it communicates with a paired transceiver connected to the ground station, allowing the transmission of RTK corrections, and the capability to monitor the state of the vehicle from a proper software (for example QGroundControl). A *FrSKY S8R* radio receiver[45] is added in order to have the possibility to control the UAV manually, via a *FrSKY Taranis X9D Plus* radio transmitter[46], for safety reasons, in case of necessity. The final main component of

the vehicle we put together is a *Raspberry Pi 4*[47], an onboard computational unit, able to communicate both to the ground station via wi-fi, and to the autopilot via the configured pins. Such a computer, used with the *Ubuntu Server 18.04.5* operating system[48], *ROS Melodic Morenia*[49], and *MAVROS*[50], is able to perform the needed computations, and send the high-level control commands directly to the autopilot. The vehicle and all its components are powered by a *FullPower Li-Po 3S 11.1 V / 5200 mAh* high discharge battery.

The described configuration should allow to run the swarming and tracking algorithms, and perform some tests necessary to verify that what works in the simulation is also correct for the hardware.

Chapter 6

Conclusions

The presented work introduced the state of the art of formation keeping and target tracking techniques, currently used for autonomous mobile vehicles. In the project, we then developed a methodology to maintain a stable swarm of drones using a potential-based algorithm, exploiting their GNSS information, and improving it with the UWB distance data. The algorithm was subsequently completed by adding a second component, whose goal is to derive the velocity necessary to successfully follow the trajectory of an ulterior reference body. Such a tracking algorithm, based only on the GNSS location of the anchors and on their absolute UWB distance from the target, was analytically proved to converge towards the correct solution. Moreover, a simple attitude control was inserted to keep the front of the vehicle always in the direction of its motion, but different configurations could be easily implemented.

The simulations, conducted in the Gazebo environment, show good performance of the swarm of quadcopters in carrying out its tasks. First, we verified the ability to successfully reach and maintain a stable spatial configuration, taking off from non-specific positions. Then, various motion profiles of the target were simulated, to tune the control parameters, monitor the expected behavior of the system, and improve as much as possible its performance. In general, the results were

satisfactory, exhibiting excellent tracking capability, even in the case of irregular random motion of the reference body.
The last part of the work consisted of putting together four quadrotors, equipped with the components needed for the correct functioning of the system.

6.1 Future work

The following step would be to test what was implemented on the physical bodies. After checking experimentally the correctness of the algorithms via simulation tools, it is fundamental to check if the obtained parameters are valid also for the chosen hardware. Most likely, an ulterior fine-tuning phase would be needed, in order to get the best possible performance of the overall system.

Glossary and acronyms

API Application Programming Interface

DOP Dilution Of Precision

GLONASS Global'naya Navigatsionnaya Sputnikovaya Sistema

GNSS Global Navigation Satellite System

GPS Global Positioning System

MAVLink Micro Air Vehicle communication protocol

MAVROS MAVLink to ROS bridge

OpenCV Open Source Computer Vision Library

PIC4SeR PoliTO Interdepartmental Centre for Service Robotics

PX4 Open source autopilot

QGC QGroundControl

QR Quick Response

RC Radio Control

ROS Robot Operating System

RTK Real-Time Kinematics

S&R Search and Rescue

UAV Unmanned Aerial Vehicle

UWB Ultra-WideBand

XML eXtensible Markup Language

Bibliography

[1] Michael Decker, Martin Fischer, and Ingrid Ott. «Service Robotics and Human Labor: A first technology assessment of substitution and cooperation». In: *Robotics and Autonomous Systems* 87 (2017), pp. 348–354. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2016.09.017. URL: http://www.sciencedirect.com/science/article/pii/S0921889016306285.

[2] International Federation of Robotics (IFR). URL: https://ifr.org/service-robots/.

[3] Teppo Luukkonen. «Modelling and control of quadcopter». In: *Independent research project in applied mathematics, Espoo* 22 (2011).

[4] Namrata Gupta, Mangal Kothari, and Abhishek Abhishek. «Modeling and Control of Inverted Flight of a Variable-Pitch Quadrotor». In: (Sept. 2017).

[5] *Software Overview - PX4 Autopilot*. https://px4.io/software/software-overview/.

[6] *PX4 Architectural Overview - PX4 Developer Guide*. https://dev.px4.io/master/en/concept/architecture.html.

[7] *Introduction - MAVLink Developer Guide*. https://mavlink.io/en/.

[8] *MAVLink Messaging - PX4 Developer Guide*. https://dev.px4.io/master/en/middleware/mavlink.html.

[9] *What is GNSS? | European Global Navigation Satellite Systems Agency.* https://www.gsa.europa.eu/european-gnss/what-gnss.

[10] François Peyret, Pierre-Yves Gilliéron, Laura Ruotsalainen, and Jesper ENGDAHL. *SaPPART White paper : Better use of Global Navigation Satellite Systems for safer and greener transport.* Jan. 2015.

[11] *The UWB Technology Story and How it Makes Sense for Indoor Positioning.* https://www.eliko.ee/uwb-technology-indoor-positioning/.

[12] *What is ultra-wideband technology and how does it work?* https://insights.samsung.com/2021/01/25/what-is-ultra-wideband-and-how-does-it-work-2/.

[13] Anam Tahir, Jari Böling, Mohammad-Hashem Haghbayan, Hannu T. Toivonen, and Juha Plosila. «Swarms of Unmanned Aerial Vehicles — A Survey». In: *Journal of Industrial Information Integration* 16 (2019), p. 100106. ISSN: 2452-414X. DOI: https://doi.org/10.1016/j.jii.2019.100106. URL: http://www.sciencedirect.com/science/article/pii/S2452414X18300086.

[14] Hanno Hildmann, Ernö Kovacs, Fabrice Saffre, and A. F. Isakovic. «Nature-Inspired Drone Swarming for Real-Time Aerial Data-Collection Under Dynamic Operational Constraints». In: *Drones* 3.3 (2019). ISSN: 2504-446X. DOI: 10.3390/drones3030071. URL: https://www.mdpi.com/2504-446X/3/3/71.

[15] Saurabh Goyal. *Centralized vs Decentralized vs Distributed | Delta Exchange | Medium.* https://medium.com/delta-exchange/centralized-vs-decentralized-vs-distributed-41d92d463868.

[16] Y. Zhang and H. Mehrjerdi. «A survey on multiple unmanned vehicles formation control and coordination: Normal and fault situations». In: *2013*

International Conference on Unmanned Aircraft Systems (ICUAS). 2013, pp. 1087–1096. DOI: 10.1109/ICUAS.2013.6564798.

[17] M Anthony Lewis and Kar-Han Tan. «High precision formation control of mobile robots using virtual structures». In: *Autonomous robots* 4.4 (1997), pp. 387–403.

[18] Mattias Wahde. «Behavior-based robotics». In: *Introduction to autonomous robots*. Department of Applied Mechanics, Chalmers University of Technology, Goteborg, Sweden, 2012. Chap. 3.

[19] Dongdong Xu, Xingnan Zhang, Zhangqing Zhu, Chunlin Chen, and Pei Yang. «Behavior-Based Formation Control of Swarm Robots». In: *Mathematical Problems in Engineering* (June 2014). DOI: 10.1155/2014/205759.

[20] Z. Zhao and J. Wang. «Leader-Follower Formation Control of Multiple Quadrotors». In: *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*. 2018, pp. 1–6. DOI: 10.1109/GNCC42960.2018.9019167.

[21] I. Choi, J. Choi, and W. Chung. «Leader-follower formation control without information of heading angle». In: *2012 IEEE/SICE International Symposium on System Integration (SII)*. 2012, pp. 842–846. DOI: 10.1109/SII.2012.6427361.

[22] P. Dunias. «Autonomous robots using artificial potential fields». English. PhD thesis. Department of Electrical Engineering, 1996. Chap. 5. ISBN: 90-386-0200-6. DOI: 10.6100/IR470384.

[23] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. «Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem». In: *Distributed Autonomous Robotic Systems 5*. Springer, 2002, pp. 299–308.

[24] Ali Marjovi, Milos Vasic, Joseph Lemaitre, and Alcherio Martinoli. «Distributed graph-based convoy control for networked intelligent vehicles». In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 138–143.

[25] Mehran Mesbahi and Magnus Egerstedt. *Graph theoretic methods in multiagent networks*. Vol. 33. Princeton University Press, 2010.

[26] Panos J Antsaklis. «Intelligent Control». In: *Wiley Encyclopedia of Electrical and Electronics Engineering* (2001).

[27] H. Rezaee, F. Abdollahi, and M. B. Menhaj. «Model-free fuzzy leader-follower formation control of fixed wing UAVs». In: *2013 13th Iranian Conference on Fuzzy Systems (IFSC)*. 2013, pp. 1–5. DOI: 10.1109/IFSC.2013.6675677.

[28] C. L. Sang, M. Adams, M. Hesse, T. Hörmann, T. Korthals, and U. Rückert. «A Comparative Study of UWB-based True-Range Positioning Algorithms using Experimental Data». In: *2019 16th Workshop on Positioning, Navigation and Communications (WPNC)*. 2019, pp. 1–6. DOI: 10.1109/WPNC47567.2019.8970249.

[29] C. L. Sang, M. Adams, T. Korthals, T. Hörmann, M. Hesse, and U. Rückert. «A Bidirectional Object Tracking and Navigation System using a True-Range Multilateration Method». In: *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. 2019, pp. 1–8. DOI: 10.1109/IPIN.2019.8911811.

[30] K. Sato. «Simple Autonomous Flight Control of a UAV Flying Above a UGV Using Onboard Camera Vision*». In: *2019 12th Asian Control Conference (ASCC)*. 2019, pp. 1643–1648.

[31] Rui Jiang, Xin Wang, and Li Zhang. «Localization algorithm based on iterative centroid estimation for wireless sensor networks». In: *Mathematical Problems in Engineering* 2018 (2018).

[32] Erik de Vries and Kamesh Subbarao. «Cooperative control of swarms of unmanned aerial vehicles». In: *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. 2011, p. 78.

[33] Veysel Gazi and Kevin M Passino. «Stability analysis of swarms». In: *IEEE transactions on automatic control* 48.4 (2003), pp. 692–697.

[34] *GeographicLib: GeographicLib::Geodesic Class Reference*. https://geographiclib.sourceforge.io/html/classGeographicLib_1_1Geodesic.html#afdca5eb7c37fa2fecf124aecd6c436fd.

[35] *Introduction · MAVLink Developer Guide*. https://mavlink.io/en/.

[36] *MAVROS | PX4 User Guide*. https://docs.px4.io/master/en/ros/mavros_installation.html.

[37] *Gazebo*. http://gazebosim.org/.

[38] *Flame Wheel ARF KIT - DJI*. https://www.dji.com/it/flame-wheel-arf.

[39] *Power Module(PM02 V3) – Holybro*. http://www.holybro.com/product/power-modulepm02-v3/.

[40] *APC Propellers*. https://www.apcprop.com/product/10x4-7sf/.

[41] *Pixhawk 4 – Holybro*. http://www.holybro.com/product/pixhawk-4/.

[42] *H-RTK F9P – Holybro*. http://www.holybro.com/product/h-rtk-f9p/.

[43] *Long Range TOF Sensor | Distance Sensor | Level Sensing*. https://www.terabee.com/shop/lidar-tof-range-finders/teraranger-evo-60m/.

[44] *Transceiver Telemetry Radio V3 433MHz – Holybro*. http://www.holybro.com/product/transceiver-telemetry-radio-v3/.

[45] *S8R - FrSky - Lets you set the limits*. https://www.frsky-rc.com/product/s8r-2/.

[46] *Taranis X9D Plus - FrSky - Lets you set the limits.* https://www.frsky-rc.com/product/taranis-x9d-plus-2/.

[47] *Raspberry Pi 4 Model B – Raspberry Pi.* https://www.raspberrypi.org/products/raspberry-pi-4-model-b/.

[48] *Ubuntu 18.04.5 LTS (Bionic Beaver).* https://releases.ubuntu.com/18.04/.

[49] *Melodic - ROS Wiki.* http://wiki.ros.org/melodic.

[50] *Mavros - ROS Wiki.* http://wiki.ros.org/mavros.