POLITECNICO DI TORINO

Master degree course in Mechatronic Engineering

Master Degree Thesis

Cloud-Based UASs Traffic Management: Trajectory Tracking and Collision Avoidance

Supervisors
prof. Alessandro Rizzo
prof. Giorgio Guglieri

Candidate
Francesco Polia

Internship Tutor
dott. ing. Stefano Primatesta

Anno accademico 2017-2018

Abstract

Employing unmanned aerial systems (UAS) inside cities to perform different types of operations would definitely bring huge benefits. However allowing such operations requires great attention because of the high density of human beings on those environments. The people safety is in fact among the biggest obstacles to be addressed before such operations could start indeed. So this is the main goal of the project related to this thesis.
Thus the project leads to the creation of a structured low-altitude airspace that allows autonomous operations of low-cost unmanned vehicles, while preserving the safety for people on the ground.
In traditional aeronautics this is done by a category of systems usually referred as Air Traffic Managers (ATM). In this scenario we are facing with vehicles totally different from traditional aircrafts, and a system able to cover the equivalent role of an ATM is required. Such a system is usually called UTM (UAS Traffic Management). Reflecting the characteristics of low-cost unmanned vehicles, an UTM is a system (or ecosystem) able to support safe and efficient UAS operations.
The upcoming generation of mobile networks, 5G, and the improvements in the Cloud Robotics paradigm constitute a suitable framework to address the problem into account here. So, after studying the state of the art of the involved technologies, we propose and developed a framework, called CBUTM (Cloud-Based UTM), to provide a networked control system (NCS) able to accomplish the project's goal.
As showed, an UTM has different aspects to relate with. For this reason the working group was divided into smaller ones, each one with different aspects to treat. What will be covered here is in particular what concerns collision avoidance problem.
Since the sensing capabilities are out the objectives of the project in this phase, the focus is on avoiding collision between vehicles able to communicate with the Cloud Based UTM. These vehicles can be active or passive, depending on whether they are controllable by CBUTM or not.
This problem can be treated ahead of time (planning), when it could occur (reacting), or the combination of both. Taken into consideration the dynamism of our referring context (cities) and the probability of unforeseen circumstances, we decided to start developing a reactive system that results to be yet compatible with a

planning-approach system, when it will be available. This compatibility is achieved unifying trajectory tracking and collision avoidance in a unique problem.
Strategies based on the principles of Receding Horizon Control (RHC) meet well the above defined requirements. Furthermore, such approach is suitable to be implemented in a decentralized manner, reducing the complexity of the optimization problem (essential part of RHC) and making the system easily scalable. Moreover, a priority based mechanism allow to treat with the same system both active and passive agents.
In parallel with problem analysis, we developed an implementation of the outlined framework. The implementation is based on ROS (Robot Operating System), that we encountered here for the first time and that played a key role in this work.
Finally the performed simulations will be shown, substantiating both the choices done and some design limitations while supporting further developments.

Acknowledgements

Even this journey was possible thanks to the support of many. Foremost of my family, that pushed me on this path. The Ente Regionale per il Diritto allo Studio Universitario del Piemonte, which by accomplishing its duty gave me the means to get to the end. Finally, all of the colleagues, and friends, who shared this path with me and who do not need to be itemized here. Your collaboration has been crucial. Thank you.

My last acknowledgment goes to patience. Out of the many who have listened and helped with my dilemmas, Stefano's kind patience deserves a special mention. Finally I would like to extend my appreciation to the reader's patience and my own.

Contents

List of Figures

Acronyms

UAS Unmanned Aerial System

UAV Unmanned Aerial Vehicle

UTM UAS Traffic Management

CBUTM Cloud-Based UTM

GSM Global System for Mobile communication

GPS Global Positioning System

NCS Networked Control System

CCS Cloud Control Station

PTP Predictive Trajectory Planner

MM Map Manager

MPC Model Predictive Control

RHC Receding Horizon Control

Chapter 1

Introduction

The sectors related to unmanned aerial systems (UAS) are currently experiencing a continuous and perhaps no longer surprising growth. The prospects that such systems open in the most disparate fields are such that it is reasonable to believe that this sector will continue to grow in the coming years, without having to report any specific chart with the market growth forecasts. What can be interesting to note is that, in addition to the two major sectors of this market, the military and the consumer ones, a third sector is getting more and more space as drones begin to be used in a range of commercial applications. The commercial segment is in particular the one that is experiencing the fastest growth, above all related to construction, agriculture, insurance and infrastructure inspection. The recent growth of this sector is basically based on two factors.
The first is that the competition generated in the consumer market, from which the commercial segment inherits origins and solutions, has led to a substantial reduction in costs and an improvement in the reliability of unmanned vehicles.
The second factor is the result of ongoing progress, however slow it may seem, that national and international institutions are making about the regulation of these systems and their use. The latter factor, on the other hand, has constituted and certainly constitutes a brake on development, as it is the one that takes charge of a set of issues that the development of such systems raises, first of all the risks for privacy and safety of people. It is precisely on these aspects that the project launched at the Joint Open Lab at the Politecnico di Torino is focused, and consequently this work is based on these.
The idea is that by jointly exploiting the potential offered by cloud computing and next-generation of mobile networks (5G), it is possible to provide an autonomous framework to support commercial fully-autonomous UAS operations in a certain airspace (i.e. a UAS Traffic Management system) and which at the same time offers a unified approach to risk management that these missions involve and is therefore able to ensure a certain level of safety for people on the ground. The concepts present in this last sentence, as we shall see, are in fact more related to

each other than it may seem. A framework that claims to be the only portal to access unmanned missions and the tool with which to control them, on one hand clearly requires a certain computing power, a big amount of available data and the ability to communicate with vehicles beyond visual line of sight (and this brings us back to the Cloud Computing and 5G concepts), on the other hand it translates into a potential advantage for the competent authorities: by equipping the framework with a coherent metric for risk assessment, the authorities will have the tool to monitor and systematically intervene on the execution of these missions. In our opinion, a single access tool for individual users, companies and authorities could lead, even in the short term, to accelerate experimentations and facilitate the adoption of such technologies.

1.1 Thesis Structure and Contributions

The aim of the project is so the design and development of the above mentioned framework (hereafter called Cloud-Based UAS Traffic Management or CBUTM). To do this, the work was divided into sub-groups, each with a specific sub-objective. However, the cooperation between the various groups was not only necessary to ensure consistency and uniformity of the whole work, but was expressly desired because, in our opinion, it is a condition that allows to accelerate the development and to identify better solutions. The overall design of the architecture was in fact conducted by the whole team, aware that the work of each one would be based on this. This also led us to present in the thesis an overall overview of the project, that is, including parts that involved the undersigned only with a certain degree of collaboration and not with direct development. Specifically in this thesis there are several references to the work done with and by Andrea Lorenzini [29] and Enrico Stabile [42].
In the present work we will start, in chapter 2, offering an overview of some of the concepts underlying the project. In particular, after having illustrated the state of the art of the UTM, we will pass to an analysis of the functions that such a system must provide and of the methodologies and technologies that can be adopted to perform these functions. Furthermore, since the project focuses on safety, the concept of risk and the techniques available in the literature to analyze it will be treated.
In chapter 3, the designed framework will be exposed, illustrating its overall architecture and the high-level operating principle of its main components.
In chapter 4 we will discuss the central topic of this thesis: collision avoidance. Starting from the exposure of the reference context, we will first define the problem into account from a technical point of view and then analyze the solutions currently present in the literature to address it. We will then see how we have come to unify this problem with that of the Trajectory Following one, and how the solution of

the combination of the two can actually be applied in the CBUTM's framework.
Parallel to the design phase, the team proceeded to develop a framework implementation. This is based on the ROS development platform, and it is certainly the part that took the most time to complete. In Chapter 5, implementation choices are therefore shown at a high level, mainly focusing on how the trajectory following with collision avoidance mechanism has been achieved.
The entire team finally proceeded to test the functioning of the framework in its different parts, therefore in chapter 6 we will illustrate the results of the conducted simulations as well as the way in which they were carried out.
Finally, in chapter 7 we draw the conclusions of our experience within the project and on the work itself, highlighting the resulting benefits and criticalities and, above all, the ways in which the latter could be mitigated in future work.

Chapter 2

UTM: State of the Art

2.1 Introduction

Being able to perform unmanned missions (vd. 2.2.2) would definitely bring huge benefits, especially in these missions could take place in urban environment. Quoting EU Commissioner for Transport [4]: "*Drones mean innovation, new services for citizens, new business models and a huge potential for economic growth*". However allowing such operations requires great attention because of the high density of human beings in those environments. The people safety is in fact among the biggest obstacles to be addressed before such operations could start indeed. In traditional aeronautics this problem is addressed by a category of systems usually referred as Air Traffic Managers (ATM).
But vehicles and missions we are talking about (unmanned mission performed by low-cost commercial vehicle) are evidently different from those of traditional aviation. That clearly imply a complete redefinition of such a Traffic Management System. For this reason we will refer to a category of systems usually called UTM (UAS Traffic Management). It is important to notice how these two kind of traffic managements system still maintain a substantial convergence of objectives and above all they must be able to cooperate with each other.
There is still not a common definition about what an UTM is and what an UTM has to do exactly. We can state that an UTM is a system aiming to allow multi UAS safe operations into low-altitude airspace, including urban environments. The possibilities that such a system creates are clearly of big impact and it interest both public and private sectors, so it not surprising if different initiatives in the world are investigating this problem and that usually these initiatives involve regulatory agencies and technological industry together, as we will see in the following.
According to the international Civil Aviation (ICAO), an UTM framework will include many components, three of which are fundamental:

- Registration System to allow remote identification and tracking of each UAS

- Communications Systems
- Geofencing-like system

How we will see, these topics are threated in the rest of this work, alongside with other not directly mentioned here.
So, before presenting our framework, we will briefly review the current state of the legislation on the subject and the state of the art of the main UTM solutions under development in the world.

UAVs - rules and limitations

The consolidation of technologies related to unmanned vehicles, with the consequent reduction of costs, makes it imperative to face a series of issues, including security, privacy and regulation. These issues are often more difficult to solve and require the involvement of different entities, compared to those that are properly technological. However, the effort to deal with them still seems inappropriate in most legislations. Quoting the European Aviation Safety Agency: [1].

> Especially smaller drones are increasingly being used in the Europe Union (EU), but under a fragmented regulatory framework. Although national safety rules apply, the rules differ across the EU and a number of key safeguards are not addressed in a coherent way.

The European Aviation Safety Agency (EASA) is an official agency of the European Union (EU) with the regulatory and executive tasks in the field of civilian aviation safety.
On request by the European Commission, member States and other stakeholders, EASA developed a proposals for an operation centric, proportionate, risk- and performance-based regulatory framework for all unmanned aircraft. A general concept, establishing three categories of UAS operations with different safety requirements, proportionate to the risk, was proposed. According to that, UAVs have been classified in:

Open category is a category of UAS operation characterized by low risks and so are operations that can take place without requiring any prior authorization by competent authority nor a declaration by the operator. Safety is so ensured through operational limitations, compliance with industry standards, requirements on certain functionalities, and a minimum set of operational rules.

[1] http://dronerules.eu, https://www.easa.europa.eu/easa-and-you/civil-drones-rpas

Specific category is a category of UAS operation characterized by a medium risk. This kind of operations requires a prior authorization by the competent national authority before it can take place, alongside with a risk assessment and the consequent mitigation measures.

Certified category is a category of UAS operation whose requirements are comparable to those of traditional aeronautics in order to assure an appropriate level id safety: certification of the vehicle, licensed pilot and an operator approved by the competent authority.

Basing on comments received during months of consultation, they recently published a proposal for a new European regulation for UAS operations classified as 'open' or 'specific' and taking into consideration the developments worldwide. The main aspect of this proposed regulation are:

- it provides a framework to safely operate UAS while allowing industry to be agile and to continue to grow. The risks for people on the ground, as well as privacy, security and data protection issues are taken into account.
- it defines technical and operational requirements for the drones like, for example, the remote identification of vehicles, systems of geoawareness, geofencing-like control
- it addresses the pilots' qualification topic. In brief, drone operators will have to register themselves, expect they operate drones lighter than 250g.
- it combines product's legislation with aviation's one
- it allows a high degree of flexibility for EASA member states: they will be able to define restricted and forbidden zones or instead areas where certain requirements are relaxed.

Moreover, the proposed regulation aims to:

- implement an operation-centric, proportionate, risk- and performance-based regulatory framework for all UAS operations conducted in the 'open' and 'specific' categories;
- ensure a high and uniform level of safety for UAS operations;
- foster the development of the UAS market; and
- contribute to addressing citizens' concerns regarding security, privacy, data protection, and environmental protection.

Our work starts from these principles.
However, there are some projects that aim to address all the perspectives introduced by the unmanned missions, involving the interested entities already in the design and development phases. Respectively in USA and Europe, the main ones are:

NASA UTM

The project started thanks to a partnership between the National Aeronautics and Space Administration (NASA) and the Federal Aviation Administration (FAA) of the United States. A specific team, called Research Transition Team, has the task of coordinating the agencies between them and with the industrial partners. The goal of the project are *"exploring concepts of operation, data exchange requirements, and a supporting framework to enable multiple beyond visual line-of-sight UAS operations at low altitudes"*[3]i.e under about 120 meters above ground level, remaining so separate but complementary to the FAA's Air Traffic Management system.

Quoting FAA's referring page [1]:

> UTM is a "traffic management" ecosystem for uncontrolled operations that is separate but complementary to the FAA's Air Traffic Management (ATM) system. UTM development will ultimately identify services, roles/responsibilities, information architecture, data exchange protocols, software functions, infrastructure, and performance requirements for enabling the management of low-altitude uncontrolled UAS operations. [...] Areas of focus include concept and use case development, data exchange and information architecture, communications and navigation, and sense and avoid. Research and testing will identify airspace operations requirements[...].

Finally, we want to underline the importance of the autonomy concept in a UTM, quoting NASA [3]:

> One of the attributes of the UTM system is that it would not require human operators to monitor every vehicle continuously. The system could provide to human managers the data to make strategic decisions related to initiation, continuation, and termination of airspace operations. This approach would ensure that only authenticated UAS could operate in the airspace. In its most mature form, the UTM system could be developed using autonomicity characteristics that include self-configuration, self-optimization and self-protection. The self-configuration aspect could determine whether the operations should continue given the current and/or predicted wind/weather conditions.

In the following we will see how, having designed our UTM to be Cloud-Based, it is definitely projected in an optics of complete autonomy.

U-Space

Another example of a cooperative development of an UTM solution is the U-Space project. It is born inside SESAR (Single European Sky ATM Research), a project

wanted by the European Commission and leaded by a public-private partnership (SESAR Joint Undertaking) whose aim is to overhaul the european airspace and its traffic management. This project seems to be more focused on safety in urban environment respect to the NASA's one, as our project actually is: *The U-Space will help unleash the potential of this new service market while ensuring the safe and secure integration of these drones operations in our urban areas and countryside.*[38] The project and the implementative steps are summarized in the following by SESAR-JU [38]:

> U-space is a set of new services relying on a high level of digitalisation and automation of functions and specific procedures designed to support safe, efficient and secure access to airspace for large numbers of drones. As such, U-space is an enabling framework designed to facilitate any kind of routine mission, in all classes of airspace and all types of environment - even the most congested - while addressing an appropriate interface with manned aviation and air traffic control. The U-space blueprint proposes the implementation of 4 sets of services to support the EU aviation strategy and regulatory framework on drones:
>
> U1: U-space foundation services covering e-registration, e-identification and geofencing.
>
> U2: U-space initial services for drone operations management, including flight planning, flight approval, tracking, and interfacing with conventional air traffic control.
>
> U3: U-space advanced services supporting more complex operations in dense areas such as assistance for conflict detection and automated detect and avoid functionalities.
>
> U4: U-space full services, offering very high levels of automation, connectivity and digitalisation for both the drone and the U-space system

2.2 Basics and Requirements

2.2.1 Cloud Robotics

The expression *cloud robotics* is a relatively new paradigm according to which robots and automatic/autonomous systems in general exchange information and execute computations using a common online network. The main benefits are [19]:

Cloud Computing High computational power exploitable through Internet allows to design and to use complex (and demanding) algorithms on multiple robots at the same time.

Big Data Is an expression commonly referred to massive amount of data (both

structured and non-structured) collected and managed to extrapolate knowledge, and to the systems that allow such operations. The concept is clearly related to the cloud computing.

Collaborative Learning Robots connected to the cloud exchange information between them and with the cloud itself. The knowledge extrapolated by the cloud is indirectly shared by the robots which belong to the network.

Cloud computing paradigm is used in even more applications, as mobile robotics, medical and domestic robotics and industrial systems. Cloud computing also combines two emerging technologies that will play an increasingly important role in the future: machine learning and IoT. Such a paradigm naturally combines with the framework presented here. The idea is in fact a central unit (the Cloud) that collects data by vehicles. Collected data are not only available for other vehicles, but are combined with other available ones (big data) to produce new information and knowledge (big data analysis, machine learning). Through cloud, each vehicle has access to information and knowledge produced by all the others (Collaborative Learning).

2.2.2 Unmanned Missions

We have already seen how the UASs, and more generally all the unmanned systems, are among the most interesting and promising technologies at the moment. The reasons behind the incessant growth in their development is clearly due to the potential that their employment offers in the most disparate fields. This potential is due to that of any unmanned system to perform missions that are dirty, dull or dangerous. We therefore introduce the concept of unmanned mission with the following definition:

Definition 2.1. An unmanned mission consists in a geographical trajectory traveled by an unmanned system while performing a particular task.

The technologies that allow an unmanned mission originally arise from military researches. Nowadays they are clearly common also in civilian applications, like:

- Cargo transport
- Agriculture
- Environmental monitoring
- Recreational pursuit

Moreover, in our framework, the missions are assumed to be fully autonomous, that implies no human in the loop. This condition clearly introduces a lot of issues,

since we have to guarantee a certain safety level without relying on human's ability to manage unexpected circumstances.

Nowadays assuming fully autonomous agents could seems unsuitable. Nevertheless this is clearly the ultimate direction of the researches on unmanned vehicles and will probably be a fact in some years. Internet of Things is another concept that is going to be even more tangible day after day. In a such scenario, sensors moving in the air are fundamental as well as their ability to be autonomous. In short, fully autonomous vehicles and autonomous unmanned missions are going to assume surely an important role in our cities in the next future. However laws, regulation and systems to support that kind of operations while preserving safety, are at most at developing phase. It is in this background that our work set up: a cloud based traffic manager for unmanned vehicles, in fact, has different roles but the main goals are to coherently evaluate and then coherently handle all those critical situations that could happen during an unmanned mission which could result in harming people. It is important to remark the how this job is crucial, especially into an crowded environment like cities are: safety is a fundamental requirement to talk about unmanned missions into national airspace, and it is why risk is a recurring topic in this project.

2.2.3 ROS and Gazebo

Robotics community is experiencing impressive progress day by day, and it is clearly cause and consequence of increase in reliability and cost reduction of the required hardware. In spite of this progress, robotics still present some significant challenges for software developer.

For any single researcher/developer would be unthinkable to develop complex robotics applications, if he need to start from scratch autonomously. At this point we can introduce ROS, quoting its official description:

> ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

So we can have available a distributed architecture that contains the state of the art algorithms implemented and maintained by experts widespread across the world. In this way we can focus only on what we need to develop actually.
Regarding Gazebo, we introduce it by quoting the official website of the project [2]:

[2] http://gazebosim.org/

> A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments.

In this section we were interested in introducing ROS and Gazebo from the point of view of their high-level features, as they were a point of reference for both the design and development phases. More information on their functioning will be provided in the chapter 5.

2.2.4 Autopilots

An Autopilot (or Automatic Pilot) is a system ables to control different aspects related to the flight of a vehicle without human involvement. Referring in particular to commercials UASs, autopilots are software suites commonly used to manage different aspects related to the flight, like stabilization control, geofencing capability, actuators control etc. In other word they can guarantee different levels of autonomy, from direction keeping to fully autonomy. Two of the most widespread example in this field are ArduPilot and PX4.

Ardupilot is an open source software suite (typically referred to as Firmware when it is specifically compiled for the target hardware) running on the vehicle along with ground controlling system, i.e. with the ground based control centre which traditionally provides the facilities for human control of unmanned systems.

For vehicle communications, ArduPilot has adopted a subset of the MAVLink protocol command set, one of the most popular communication protocols between drones and control stations.

Ardupilot capabilities can be extended with ROS (vd. 2.2.3 and 5.1.1) through the use of mavros.

2.3 Risk

We have seen how safety is an essential focus for our project and for any UTM. Although not the main purpose of this thesis, an overview of the risk and the methodologies used to quantify it is necessary.

2.3.1 Unmanned Mission's risk

Here we will analyze the concept itself of risk for an unmanned mission, proposing some definitions and analysis methods. First of all, the term risk is referred to the time frequency in which the drone causes deadly (or very serious) injuries to people on the ground [16]. So in this context we do not consider the risk for the

vehicle themselves. It's clear, and will be remarked in the following, that collisions between UASs can result in damage for people. Risk analysis in avionic field is something the industry has worked on for years. There are actually different ways to modeling the risk, each one with a different level of accuracy [16]. Beyond the differences, there are four main standard criteria that has to be kept into account in every risk model, include ours [15]:

1. Transparency
2. Consistency
3. Clarity
4. Reasonability

The main goal of our analysis is to determine whenever a UAS is able to guarantee a certain level of safety flying over a given geographical zone in a given time window. Moreover, if the result of this analysis is negative, the system should be able to suggest (or directly apply) one o more countermeasures in order to increase the safety level of the mission. The latter aspect is usually called *risk management* [16]. According to [15], the steps to follow here are four:

Mission Definition and Hazard Identification In this phase we have to produce a description of the mission, i.e. a safety bound of it. Since this work is focused on civil operations in an urban environment, the safety level for a mission has to deal with those imposed by the national flight authority. Furthermore, every possible hazard that could happen during each mission have to be identified.

Risk Assessment According to a certain metric, assign to every hazard a corresponding risk value

Risk Reduction and Managment Compare the imposed bounds with the actual safety level. If this latter doesn't meet the requirements, needed countermeasures are computed.

Risk Acceptance Once the risk satisfy the requirements, the mission is approved and can starts.

We can divide hazards in two main categories: due to internal failures and due to external causes. It is important to notice that an hazard analysis of internal possible failure of vehicles is out the aims of this work, also because there are already many tools to perform it. It is however objective of this project aggregate them into a coherent risk analysis.

In our framework, once the risk analysis is complete, a corresponding *risk map* is created. We will discuss further about risk in 2.3 while in 3.3 our qualitative and quantitative approach to risk is exposed.

We therefore begin to define the main hazards that may affect unmanned missions, that can be categorized into [16]:

Drone involuntary mobility this category includes all the accidents that could happen when the drone is not operating, i.e when the vehicle is still on the ground. This scenario will be not treated in this project.

Mid-Air Collisions this category includes the accident due to in flight collision, and may concern two or more UAS or fixed obstacles, such as buildings. Tipically the analysis of this category is based on the "flight's victims" involved in the crash. Since we are dealing with unmanned vehicles, the analysis of mid air collision will take into account the damage for people on the ground due to debris or falling vehicles. This type of risk has been treated in the project and in this thesis.

Early flight termination this category includes all the hazards due to a loss of control of the UAS's flight, and the consequent anticipation of the landing. In some cases, the cloud can control this phase by imposing zones and methodologies for a safe landing.

It is important to underline how, referring to the damages for people, we mean both physical and social damages: a high frequency of accidents, although not lethal, can affect the common perception of such operations and therefore potentially limit their use.
Thus, once again, the need arises to quantify the risk in order to be able to manage it appropriately.
Although there is not yet a commonly accepted definition of risk, we can define it based on a widely held concept in the field of traditional avionics:

Definition 2.2. It is called risk f_F of an Unmanned Aerial Mission the frequency of fatalities, in term of victims per hour of flight, that a given drone, in a certain area will produce.

Evaluating risk, once defined, means carrying out a risk assessment. Let's start by observing that, having defined the risk in terms of frequency, it is implicitly linked to the concept of time.
At this point, ensuring a certain level of safety for a mission means that the risk value is always below the upper-bound of victims per hour imposed by the competent authorities.
Currently, the best way to assess the value of risk is based on statistical considerations [16]. The maximum fatalities rate for a civil operation in an urban context

has been evaluated as:

$$f_{F,Max} = 10^{-5} victims/hour$$

i.e one victim at most each 10^5 hours.

At this this point we have to deduce an analytical expression that given the flight conditions, correctly evaluate the number of victims per flight's hour. Here we start from analyzing the risk due to an uncontrolled landing of the UAS: $f_{F,UFE}$. But notice how this latter event could be due to different causes both externals and due to internal failures. According to [16], a proper analytical expression could be:

$$f_{F,UFE} = N_{exp} \times P(fatality|exposure) \times f_{UFE} \tag{2.1}$$

Where:

$f_{F,UFE}$ fatalities's frequency due to early flight termination $[\frac{peole}{h}]$

N_{exp} number of people exposed to the accident $[peole]$

$P(fatality|exposure)$ probability that a person involved in the UAS's charh will suffer fatal injuries

f_{UFE} frequency of failures that cause an unexpected end of the flight $[\frac{1}{h}]$

Such a simple formula turns out to be the right compromise between a too detailed analysis and an inconsistent one, and contains all the parameters of interest that can affect an unmanned mission. These factors are combined in such a way that the resultant risk analysis is:

- Coherent with the statistical data actually available
- Not expensive from a computational point of view
- Easy to extend in case some new interesting parameters emerges
- Independent from the dimension of the considered area. The analysis can be performed both with very small and big resolution

Some initial considerations on the elements present in the equation 2.1, which will however be investigated further below:
$P(fatality|exposure)$ takes into account the kinetic energy of the drone, the geographical sheltering factor and the vulnerability of the human body to obtain the probability that the collision between an unmanned vehicle and a person is fatal. For what concerns N_{exp}, i.e the number of exposed people, it can be evaluated as $N_{exp} = \rho \times A_{exp}$, where ρ is the population density and A_{exp} is the area involved in the impact. Last note on f_{UFE}, which contains information about the frequency of the UAS's ground impact: in practice, it introduces the time element inside the

equation.
Initially, it was thought to perform a probabilistic analysis to calculate the frequency of crashes, combining external causes with the probabilities of internal failures. However, as we have experimented, an approach of this kind requires the perfect knowledge of all the possible causes of damage and the possibility of quantifying them all. Instead, in agreement with [16], a statistical approach has been preferred.
This apparently small change actually allows to better frame this work: it is not among the purposes of this framework to carry out a hazard analysis on the used components, but we can assume to know the ground impact frequency (from the manufacturer, for example) and focus on providing an instrument able to manage this information in order to guarantee the safety of the missions.

We have seen how the risk of a mission is proportional to population density ρ, so further clarification is needed in this regard. Emerges how this system should know the number of people in danger due to the UAS's flight, and in general should be a dynamic framework capable to collect data and provide a real time risk evaluation. This perfectly suit with the Cloud System we are dealing with. A first idea to evaluate ρ could be to acquire it from Internet. However, in this way it is only possible to obtain information on the number of people living in a certain area, which, however, does not reflect the real location of these people during the course of the day. The level of detail also stops at that of the district. By exploiting the potential of the cloud system at our disposal, we can think of making ρ a function of time. To do this, it is possible to acquire from internet information about the presence of eventual events, and it is also possible to estimate the influx of people looking at the historical series of the participations. Such information can then be combined with those obtained by monitoring the number of users connected to a specific cell of the mobile communication network. The latter is an approach very similiar to that used by some providers of service for cars' navigation. Clearly, being this information time dependent, the system should be able to constantly update this and so to provide information on the population density in function not only of the space, but also of the time.

In the equation 2.1 the term $P(fatality|exposure)$ represents the probability that a person reports serious injuries after the impact with a falling vehicle. To value this probability, several studies have tried to define a model of vulnerability of the human body, taking into consideration factors such as age, physical conformation and posture taken at the time of impact, in order to estimate the damage that the impact itself would have in relation to speed of the vehicle.

However, most of the information necessary to use these models is not usually available. So, starting from those models, we can proceed to a series of simplifications. In the end, we can assume $P(fatality|exposure)$ as a function of the kinetic energy of the falling vehicle and of the sheltering factor of the area in which the crash occurs. Figure 2.1 shows just how $P(fatality|exposure)$ varies with varying

kinetic energy and sheltering factor.

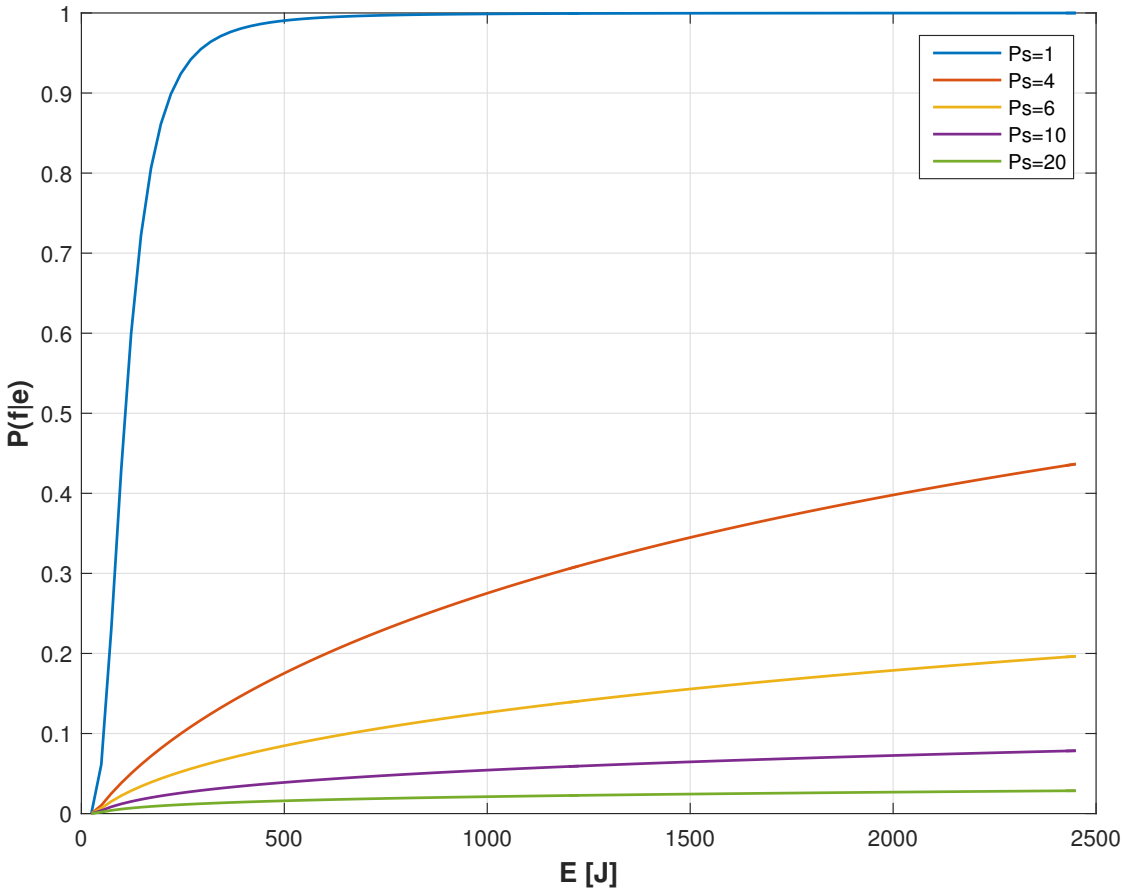


Figure 2.1: $P(fatality|exposure)$ evolution respect to Kinetic Energy and Sheltering

The problem then is the lack of a metric that coherently describes the sheltering factor. First of all, let's start by providing a more precise definition of sheltering factor, citing one of the reference works in this regard [16]:

Definition 2.3. The sheltering factor of a geographical area is its capability of protecting people on ground, through artificial or natural structures, from the fall of an Unmanned Aerial System.

Let's see how this definition, not being quantitative, can induce us to assume $P_S \in [0; \infty]$. However, in this way it is not possible to consider it as a probability and above all it is not known how to assign a value to it.

After carrying out a series of empirical tests, the colleagues found that a credible bound for sheltering factor is $P_S = 10$. In fact, due to the small size of drones in urban areas, the maximum kinetic energy released on impact it will be in the order

of the KJ. By assigning a value greater than 10 to PS, 3.4MJ are required to have a fatal event occurring at 50% probability. For this reason, in the continuation of the work, I will be hired $P_S \in [0; 10]$.

2.3.2 Mid-Air Collisions Risk Modelling

Now let's move on to the analysis of the other, and last, type of hazard previously identified: the mid-air collisions. It is therefore a matter of finding a model for the risk, for people on the ground, of lethal damage due to an in-flight collision of one or more UAS. According to [16], the frequencies of fatalities due to in flight accidents can be modeled as:

$$f_{F,MAC} = N_{exp} \times P(fatality|exposure) \times f_{MAC} \tag{2.2}$$

Where:

$f_{F,MAC}$ frequency of fatalities due to mid-air collisions $[\frac{people}{h}]$

N_{exp} number of people exposed to the accident $[people]$

$P(fatality|exposure)$ probability that a person involved in the UAS's crash will suffer fatal injuries

f_{MAC} frequency of mid-air collisions $[\frac{accudents}{h}]$

So now it's about understanding how to calculate f_{MAC}. Let's remember how, speaking of mid-air collisions, we refer to the collisions that a UAS can have with: one or more other UAS, buildings, other unpredictable obstacles. In formula:

$$f_{MAC} = f_b + f_{UAV} + f_{TBE} \tag{2.3}$$

where:

f_b is the collision rate with unexpected buildings. All the known buildings, in fact, will be handle as no-fly zone (which are explained in the following). So f_b parametrize the number of collision with building due to a lack of precision in the geographical localization

f_{UAV} rate of collisions with other UAS flying in the same area

f_{TBE} parametrize the unpredictable collisions, like those with birds etc

The measure unit of these three parameters is the one of frequency. Note how both f_b and f_{TBE} can be only estimated my means of statistical tools. f_{UAV} therefore remains to be analyzed. We underline that at this stage we are interested in modeling the frequency of collision between vehicles flying in the same area based

on some information known a priori, so we do not want to talk about the methods used to prevent such collisions. This last argument, in fact, is the central part of this work and will be dealt with in chapter 4. Different techniques have been developed to estimate correctly the value of f_{UAV}. According to [48] and [16], we can write:

$$f_{UAV} = E(CT) \times P(collision|CT) \tag{2.4}$$

Where:

$E(CT)$ is the expected value of conflicting trajectories in a given area. The measure unit $\frac{trajectories}{h}$.

$P(collision|CT)$ is the probability of having an impact given two conflicting trajectories

A technique known in the literature to estimate $E(CT)$ is the so called gas model [16], [48]. Here, the functioning of such models will not even be elaborated as, once again, an evaluation of this type is not compatible with the functioning and needs of this context. From the simulations we can instead see how, a typical value for very busy areas turns out to be $E(CT) = 4 \times 10^{-5}[\frac{CT}{h}]$. This value is clearly dependent on the size of both the vehicles and the region of space considered. Therefore, in the final, as seen for the other elements of the risk, a coherent evaluation for EEE can be obtained through a large number of simulations carried out with the Montecarlo method.
To conclude the risk analysis for mid-air collisions, we propose two methods that could be implemented to estimate the value of $P(collision|CT)$:

- quantify the real ability of the collision avoidance algorithms implemented in the conflicting vehicles
- simply assume $P(collision|CT) = 0$ if there is a collision avoidance algorithm implemented in the conflicting vehicles, $P(collision|CT) = 1$ otherwise

In particular, therefore, $P(collision|CT) = 1$ will be followed in consideration of the collision avoidance algorithms developed and shown in the chapter 4.

2.3.3 Ground Impact Risk Modelling: A Complete Framework

We have just seen how, to properly assess the risk, the causes of accidents have been evaluated. They are, we repeat, the loss of control during the flight due to internal failures and the collision in midair. We have just seen how, to properly assess the risk, the causes of accidents have been evaluated. They are, we repeat, the loss of control during the flight due to internal failures and the collision in midair. Both of

these situations can be dangerous for people and for both of them a metric has been proposed to represent the risk. At this point it is therefore necessary to associate each sub-area of the map (cells, as we will see later) with a certain resolution, a single value that combines the causes of risk and represents the overall risk for people on the ground.
To do so, we make the following assumption: any impact on the ground may be caused either by the flight control loss, or by a mid-air collision, not by both at the same time. Furthermore, to carry out a conservative risk analysis, the highest risk factor will be the one that will be taken into account. In formula:

$$f_F = \max(f_{F,UFE}, f_{F,MAC}) \tag{2.5}$$

that can be rewritten:

$$f_F = N_{exp} \times P(fatality|exposure) \times \max(f_{UFE}, f_{MAC}) \tag{2.6}$$

Equation 2.6 can be considered the final formula for calculating the risk of a ground impact, as it contains all the parameters of interest for both crash scenarios.
For our operational context, however, some considerations must be made. In particular, we have seen how f_{MAC} encompasses the risk of mid-air collisions with any type of static and dynamic object. The former are essentially static objects positioned on the path of a drone's mission, but we can assume they are consistently treated by the Risk-Aware Map manager, so collisions with static obstacles may be due to incomplete information. However, by taking advantage of the collective learning capacity typical of a cloud system, it can be assumed that even these possibilities tend to zero. Collisions with other vehicles therefore remain. However, we have seen how it is reasonable to assume that they are independent of a specific geographic area and must be treated by specific collision avoidance algorithms. These algorithms have in fact been developed and will be illustrated below.
In the final then it results that $f_{UFE} \gg f_{MAC}$ and the equation 2.6 can be simplified into:

$$f_F = N_{exp} \times P(fatality|exposure) \times f_{UFE} \tag{2.7}$$

With this equation we are therefore able to offer a qualitative and quantitative definition of the risk in terms of hourly victims, and therefore we can ensure the fulfillment of the security standards usually imposed by the competent authorities. Based on the latter, in the following we will assume that a mission can be considered safe if it turns out:

$$f_F < f_{F,MAX} = 10^{-5}\frac{1}{h}$$

In the next chapter we will see how to move from the definition of risk based on areas to the risk of a mission using the Risk-Aware Map Manager proposed within this project.

2.4 Path Planning

Trajectory planning is a classic problem of robotics, so in literature there are several ways to solve it. According to [13] a common definition of this problem is:

Definition 2.4. The motion planning problem is a term used in robotics to address the process of breaking down a desired movement task into discrete motions that satisfies constraints and possibly optimize some aspect of the movement itself.

In our context, given a description of the environment, the dynamics of the vehicle and the starting and arrival positions of a mission, the block of path planning of CBUTM will have the task of identifying an ordered sequence of points, called waypoints, that the vehicle must follow to arrive at a destination respecting the constraints (the no-fly zone, as shown in the following) and minimizing some parameters (risk and traveled distance).
It should also be noted that, having assumed that the flight quota does not change during the course of a mission, the problem of the path planning can be treated as those of the planar robotics. At this point we can move on to analyze the many solutions offered in the literature. A first, classical, categorization of these algorithms is that between **deterministic algorithms** (which are simple to implement and do not require much computing power) and **probabilistic algorithms** (which involves high dimensional state space and whose execution time can be very high). Despite the analysis and development of these algorithms is out of the scope of this thesis, a high-level overview of these types of algorithms is necessary to understand the choices made in our project and to understand the functioning of the framework we are going to present .
Before proceeding to such analysis of these algorithms, it is important to identify the characteristics of interest for our context. In particular we will deal with:

- high dimension maps. The path planner will have to work on maps of the size of a city and with a rather high resolution. Although the computing power is not a poor resource in a cloud system, the chosen algorithm must be light enough to guarantee a solution in finite time.

- Dynamic environment. Obstacles as well as risk-related parameters change over time, and often even during the course of a mission. The path planner should therefore have the dynamism necessary to change its results in real time.

- Critical situation. In the critical context of flight, response time is crucial. The path planner will therefore have to guarantee acceptable latencies.

- Drone's structure. The calculated route must clearly be feasible for the vehicle to be driven. The path planner must therefore take into account the flight dynamics of the vehicle during the computation.

2.4.1 Deterministic Algorithms

The history of this branch of Path Planner starts with E. Dijkstra in 1959, who developed his famous algorithm to find the shortest path between two nodes in a non-negative weighted graph. Since the path planner iteratively follows the edges of the graph till it will find the goal, the Dijkstra one can be classified as a graph search algorithm.
Staring from this, that can be reasonably considered the father of all the motion planning algorithms, many others were developed in the following years. In 1968, Hart, Nilsson and Raphael, developed the A* algorithm, which is still a graph search algorithm but with an heuristic estimate, which classifies each node estimating the best way that passes through it. The result of this process is the shortest path (less costly one) between start and goal [21]. A typical drawback of A* is its computational requirements: on large map (as the one used for drones flight) many states has to be recorded, so a huge memory is needed. During the rest of the century, an impressive amount of deterministic algorithms has been deployed, from scratch or modifying the previous to have a more satisfactory solution. The most important are here resumed:

Dynamic A* or D*, was proposed by Stentz in 1994 [43]. It works exactly has A*, with the main difference that weights between two nodes can changes at run time. Thanks to this feature, it can be used to make real time re-planning of the path, if the robot is supposed to move in a changing environment. The change of cost it's usually detected by sensor mounted on the Unmanned System, and for this reason it is considered a sensor based algorithm.
Stentz itself tried to improve its work, and developed first the Focussed D* and then the D* lite algorithms [44][34]. Both of them are similar to D*, but save computational resources and have beter perfomances.

Theta* developed by Nash et al. [33] is still an extension of A*. The main difference with is predecessor is that for each vertex expansion there must be a line of sight between parent node and its successor. A very similar path planner is phi*. Both of them allows the robot to moves also along diagonal lines: this features, that also provides to them the name of Any Angle Algorithms, really increase the power and the efficiency of the planner.

RA* developed by Guglieri [20], is very most recent one. This algorithm was developed strictly for the risk analysis procedure, so it takes into account parameters as population density.

Many other interesting works in this field has been developed (as Floyd-Warshall, which is a mile stone, or Artificial Potential Field), but each of them has one or more characteristic that doesn't fit with the requirements of our particular scenario.

2.4.2 Probabilistic Algorithms

Probabilistic algorithms are the ones that have a degree of randomness as part of their logic. Usually, the probabilistic (or randomized) algorithms have an auxiliary random input that guide their behaviour in order to have good performance in the average case. The performances oh this kind of systems is a random variable too, as also the running time and the output [14].
Typically, this algorithms works on nonconvex, high dimensional spaces upon which a random space tree is built: in its construction is contained the randomness of the method, that usually takes casual samples from the search space. More the samples, higher the probability of having a correct solution. Furthermore, at every step (when a new sample enter in the tree) an obstacle free trajectory is built, alays checking its feasibility.
Due to their intrinsic difficulty, this algorithms were developed later then the deterministic ones, when computer had already a sufficient computational power to support them. The first path planning randomized algorithm was the probabilistic road map (PRM), developed in 1996 [25]. As Dijkstra was crucial for deterministic algorithms, PRM can be considered the founder of the probabilistic set. It basic idea is the following:

Pre-processing Phase Starting from n random samples, a set (roadmap) of collision free paths is built.

Query Phase The shortest path between start and goal points is found in the roadmap.

The PRM works good in high dimensional space, but has a limit in the construction of the roadmap, that can be challenging and some times infeasible.
In order to provide better performances, a set of new algorithms was developed starting from this. The most important are listed here:

PRM* is a variant of PRM, in which the radius is scaled according to the number of samples. The result is an asymptotic optimality and computational efficiency.

Rapidly Exploring Random Tree or RRT, was developed by LaValle and Kuffner in 1998 [28]. It is probably the most powerful and interesting algorithm presented till now, and one of its variant has been implemented in our Cloud Based Framework. For this reasons, it will be better exposed in a proper section.

Rapidly Exploring Random Graph also called RRG, is an evolution of RRT. Practically, it is capable to provide quickly a first solution to the planing problem, and then monotonically improving it if more time is available.

Ant Colony Optimization developed by Maniezzo in 1992 [31]. It is a classical evolutionary algorithm, since its behaviour was inspired by real biological phenomena: in this case, ants looking for a path between their colony and food. Although it seems incredible, ants always find the shortest path, thanks to a random iterative process that leads to a common knowledge sharing between all the components of the colony.
ACO algorithm is very powerful and can rapidly provide a good solution. However, its converge time to the optimum is uncertain, and this can be a problem in applications like ours.

We have here reviewed the most common path planning algorithms and the principles on which they are based. In section 3.4 we will describe the path planning algorithm chosen for CBUTM: RRT*.

2.5 Collision Avoidance

The problem of collision avoidance, a central part of this work, will be analyzed and dealt with in depth in the chapter 4.

Chapter 3

CBUTM: overall framework

3.1 Introduction and General Architecture

CBUTM is an air traffic management solution based on the Cloud Robotics paradigm and designed for small commercials UAS systems. The aim is the creation of a structured low-altitude airspace which allows UAS operations also in urban environments while preserving a certain level of safety for humans.

In chapter 2 we seen what an UTM is, what are the tasks an UTM has to deal with and finally in which ways these tasks can be accomplished according to literature.

Basing on what seen in the previous chapter, in this one we show the general architecture of CBUTM and the strategies we adopted in order to accomplish our objectives.

The idea is to provide a central gateway to the unmanned missions, both for customers and authorities. One key characteristic of CBUTM is that it work on a standardized risk assessment, and so is able to operate assuring the same safety level for each possible mission in the city. The authorities have at the same time a unified access to monitor and to intervene on all those parameters that condition the missions' requirements and so the characteristics these missions will have.

In our opinion this feature would be of interest particularly in the short term, accelerating the ongoing tests and so the adoption on large scale as well as it accelerates the velocity of the feedback loop for control agencies.

But also changing parameters, the metric could be maintained. This in turn would allow different instances of CBUTM, each one under the control of the respective authorities, and at the same time a common language (the metric itself) among them. In this way, for example, different cities could use different requirements basing on the characteristics of the local environment, while continuing be inside the parameters' range imposed by the national authorities. The same mechanism can be applied even on a larger scale, like among different states of the European Union.

On the other side, a unified portal to access to unmanned missions and to the relative services is offered to the citizenry as well as to private company that would receive huge benefits operating in this field.

In order to understand the overall way of function of CBUTM, it can be useful to have a better idea of what happen when a mission's request is submitted to the system. For this purpose, figure 3.1 is the optimal reference to consider. Please notice that what we are going to do here it's just an introductory overview of the working principles of our Traffic Manager: the detailed explanations of each blocks will be treated in a specific section, in accordance with the aims of this work.
As said before, if a drone wants to fly in the city's airspace it must interact before with the Traffic Manager, in order to join correctly the Cloud-Based UASs Traffic Manager. The first step it has to accomplish is the "Registration". In this phase, the UAV communicates all his structural parameters, its goals and the requirements it must fulfill to successfully complete its mission. The cloud system, from its side, assign to the drone a coherent level of priority, according to its need, and a "virtual identity" that will be the drone image in the CBUTM. Then, it's possible to start the procedure to identify if the requirements of the Unmanned mission are compliant with the safety standard of the Authority that manages the traffic manager. From here till the moment in which it will receive the allowance to fly, the drone has only to wait, since all the procedure will be performed transparently by the cloud itself.
Beside the "Registration" process, there is the Environmental Modelling. This phase is not triggered by the mission's request but instead happens at a constant rate till the CBUTM stays online: it must have, at any time, an update and coherent geographical map of the urban environment where the flights are performed. Although not available for now, it is reasonable to suppose that in future municipal offices will share their data on the city structure. Furthermore, this information are going to be merged by the "Environmental Modelling" block with the ones received in real time by the flying drones: finally, the result is the dynamic map of the city we were looking for. Differently from the rest of the flow chart, this part of the system is independent from the drone that has advanced the request, and remains the same for ever aircraft joining CBUTM.
Once the map is available, and the registration process has began, it is mandatory to start the "Risk Assessment" for the specific UAV. Practically, this means to apply the Risk Modelling techniques (already seen in 2.3) to build a point by point map of the risk that associates at each area of the geographical map its corresponding risk value. How to do this in detail will be described later on in this chapter.
The risk map, function both of the geography and the UAS's constructive parameters, is going to be managed by the "Map Generation" block, which consider it just as a layer of a more complex cost map, built weighting all the other requirements of the unmanned mission. As output, it provides a cost map, this time

function of geography, drone's parameter and mission standard objectives. "Modelling Environment", "Risk Assessment" and "Map Generation" together form the Map Manager structure.
The map coming from "Map Generation" becomes the input of the "Mission Planner", whose final aim it's dual: on one side, it must find the best (lowest cost) path for the drone to follow, on the other it must evaluate if this trajectory is compliant with the standard (for example of safety) imposed by authorities. This two different souls, called "Path Planner" and "Path Validator" co-operate in this block, since the output of the first is the input of the latter.
At this point, three different situation are possible. In the first, the mission's requirements are impossible to be satisfied: the only solution in this case is to abort the mission, that's unfeasible. The second possible way instead happens when the chosen trajectory doesn't satisfy the requirements, but it's still possible to change the weights of the cost map and re-calculate another path. Finally, the last scenario is when the trajectory accomplish the mission's request and the safety bound, so that the drone can be authorized to fly.
In this case, a controller of the in-flight operations is needed: inside CBUTM, the two blocks "Trajectory Following & Collision Avoidance" (TFCA) and "Navigation Management" are implemented for this specific purpose, and will be illustrated and discussed in the next chapter.
The first, as the name says, is responsible for the high-level navigation control and in the following it will be implemented as a node called Predictive Trajectory Planner (PTP). In brief it is a system executed alongside every mission and it's responsible to continuously translate a path (the output of the mission planner) into a proper trajectory (a path plus a time law). The generated trajectory has to be equivalent to the received path as long as this doesn't imply that the distance from other vehicles, that clearly depend on time, goes below a safety threshold. In that case, TFCA has to produce a trajectory different as much as necessary to avoid this circumstance. TFCA works in parallel with the Navigation Management, to which it sends instruction and from which it receives a feedback on the actual state.
Navigation Management system, which is the only one to be on-board, translates high-level instructions into low-level ones, interacting directly with electromechanical actuators and with other devices onboard, like sensors.
That's said, we finally conclude the introduction to the Cloud-Based UASs Traffic Manager. In the rest of this chapter, some of the concepts and blocks just presented will be extended, according to the logical purpose of this thesis.

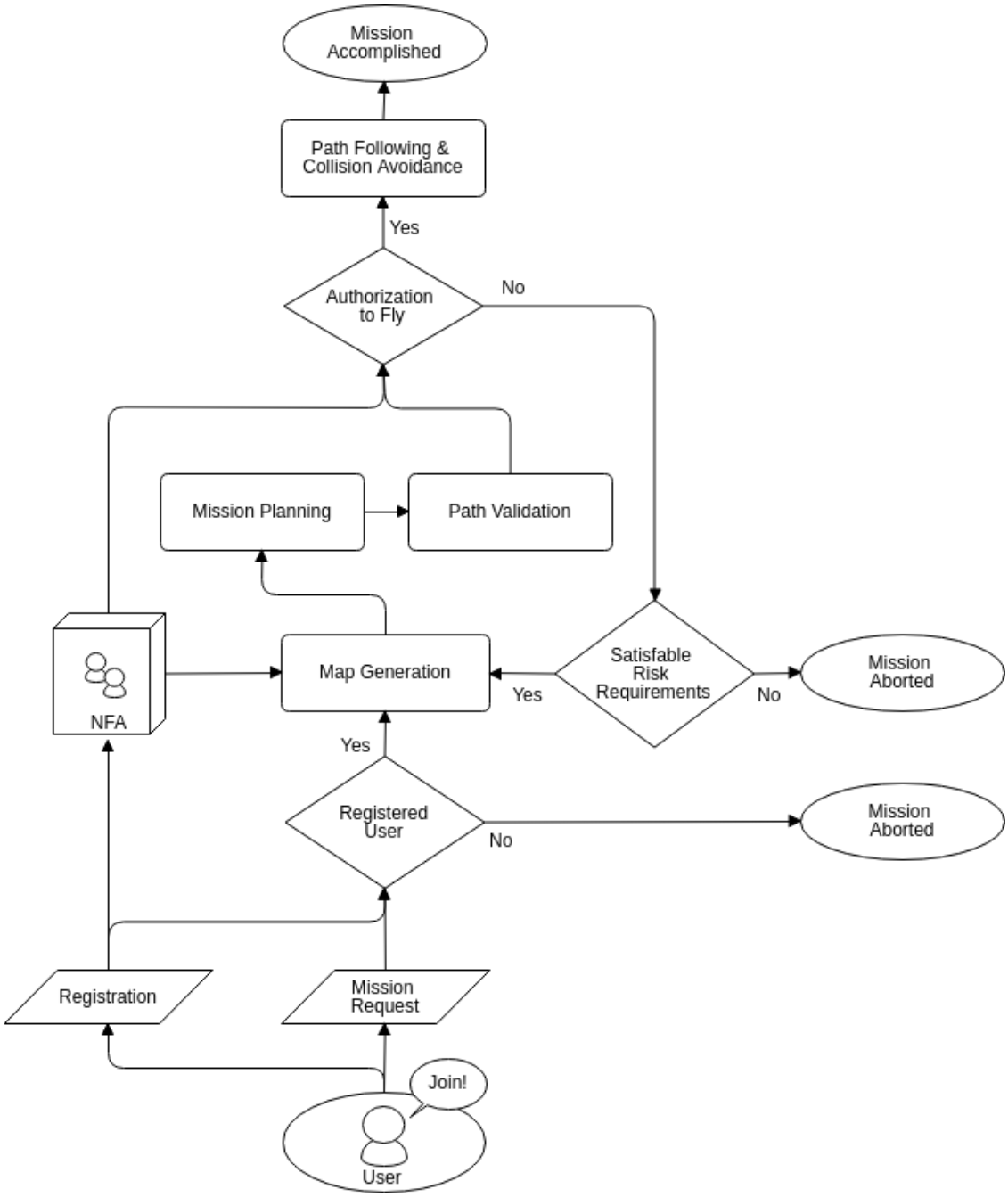


Figure 3.1: CBUTM Flow Chart

3.2 Registration, Identification and Monitoring

As previously stated, CBUTM provides a Networked Control System in charge of coordinate unmanned operation assuring compliance with pre-established rules in order to guarantee a certain level of safety. This clearly require the knowledge of the vehicle belonging to the network and of their characteristics. This information is collected and organized by the cloud system in the so called 'Registration' phase. Registration is the first step of the interaction between the cloud itself and the drone that intends to carry out a mission.
Cloud side, there is a central entity in charge of this role called Cloud Control Station (CSS) node.
When a UAS requests registration to CBUTM, CCS collects from it the information necessary to determine whether it is suitable or not to participate in the network. If so, the information collected is placed in a quick access database maintained by CSS itself. This database meant to collect not only information required for the registration, but also that required by the cloud in order to perform its coordination role and assure the require safety level. There are so also always up-dated information coming from agents of the network. In general information can be both static, like the ID, physical characteristics, its legal owner and so on; or dynamic and change during the mission, like the actual positions, the predicted trajectories and the priority level.
The Cloud Control Station is also in charge of maintaining an always updated list of the UAS active in the airspace. This is done continuing to monitor the registered vehicle through an heartbeat check setted up between the CSS and the UAS's image in the cloud, which in turn is in communication with the vehicle through the MAVROS bridge. In this way any behavior or even an unexpected termination is detected by the Cloud, which will then be able to notify an operator or in general will be able to take the necessary countermeasures directly. This 'Monitoring' phase is also required by other component of CBUTM, and will be detailed below.

Monitoring agents in the airspace is necessary also to perform coordination. The mechanism of trajectory planning will be discussed in section 4.2, but we can anticipate that it will be based on a distributed architecture, in order to exploit all advantages in terms of efficiency and robustness of a distributed networked control system based on a receding horizon control strategy. In particular it is a distributed non-cooperative priority-based networked control system.
This means that the problem of coordination will be decomposed into as many sub-problems as there are agents active in the airspace. Each of these subproblems still requires the results of the other subproblems in order to be able to be solved appropriately, and this requires that the results are easily usable. The result of each sub-problem, as we shall see, is none other than the trajectory that the UAS in question envisages, instant by instant, to carry out. It follows also that every

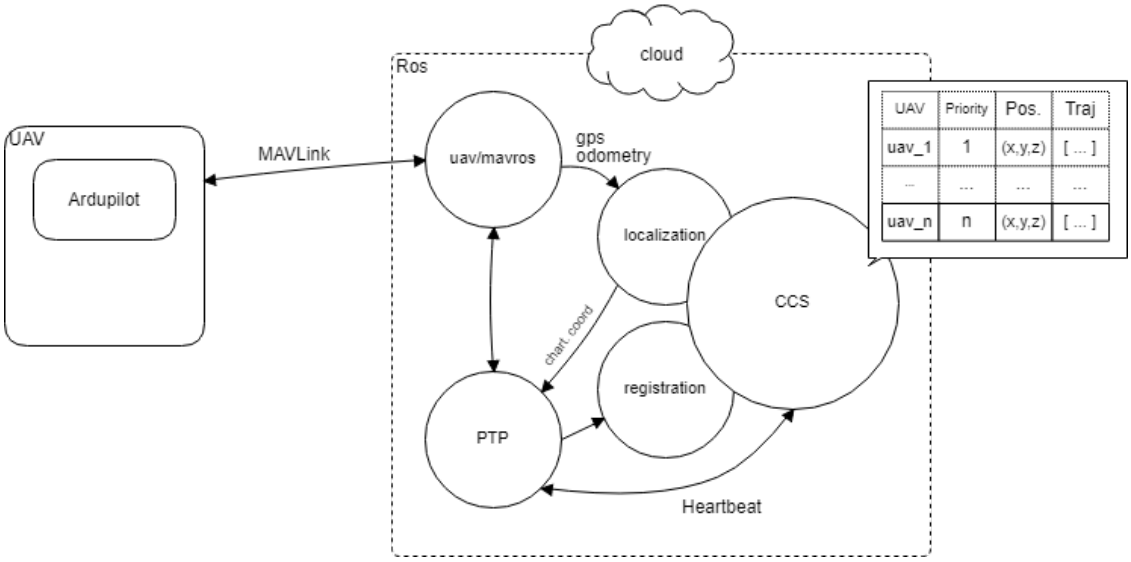


Figure 3.2: CCS-PTP interaction

drone, or rather its image of the cloud, should be able to quickly access this kind of information. This functionality, alongside with those seen in previous sections, is provided by the Cloud Control Station Node, by mean of a specific ordered database constantly updated with the information coming from the localization and with the predicted trajectories.
As depicted in figure 3.2 and 3.3, CSS receives through MAVROS topics, the GPS position data from every agents. These coordinates are so converted into cartesian ones referred to the map reference frame. The converted coordinates are then stored, and updated, into a specific container.
Basing on this container, the distance between different agents is computed and monitored. When this distance goes below a certain threshold, set at 50m, the Cloud Control Station advertise the involved agents with lower priority. This is performed with a message containing the IDs of the vehicles they must take into account and sending to them, or properly to their cloud's images, through a topic they always subscribe. At this point is sufficient to query the database to obtain the needed information and achieve coordination.

3.3 Risk Assessment and Map Generation

A quick overview of what the risk of an unmanned mission is and how it is possible to evaluate it was done in the previous chapter. It is now a question of constructing a risk map for the unmanned mission, ie a map that associates to each geographic point involved in the mission, the corresponding risk value for people on the ground.

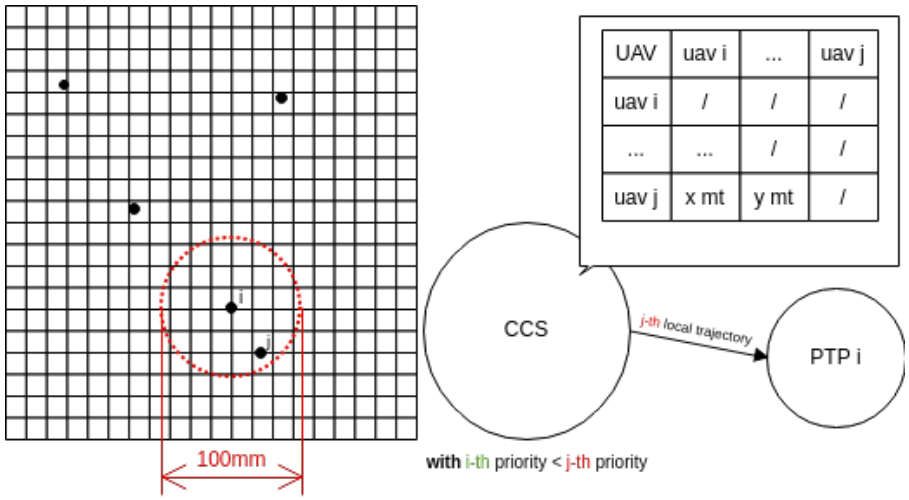


Figure 3.3: Graphical representation of monitoring phase

The resulting map will be used by the path planner in order to find a considerably safe path that allow a UAS to reach its goal positions. Notice how, although we speak of geographical "points", the map will not be punctual, but will be subdivided into sub-areas, called cells, within which the risk will be reasonably assumed to be uniform. In particular we have decided that each of these cells will have an area of $25m^2$. For each of these areas, the cloud system will have to calculate the corresponding risk based on information that it must be able to know and manipulate, and which are:

- population density, varying in time
- the mass of the vehicle
- the flight height
- the sheltering factor of the area
- frequency of ground impact due to unexpected end of the flight
- the number and the positions of the no_fly_zone

Each of these values is contained in a specific layer of the map (we'll talk about it later) and the risk map is nothing more than the weighted overlap of these layers made on the basis of the equations identified in the previous chapter.
The result of these operations will be similar to what shown in the figure 6.10.

It is important to remember one of the assumptions made in this first phase of the project: the navigation of the UAS takes place on a plane parallel to that of the ground. This means that it is possible to consider the flight as a planar

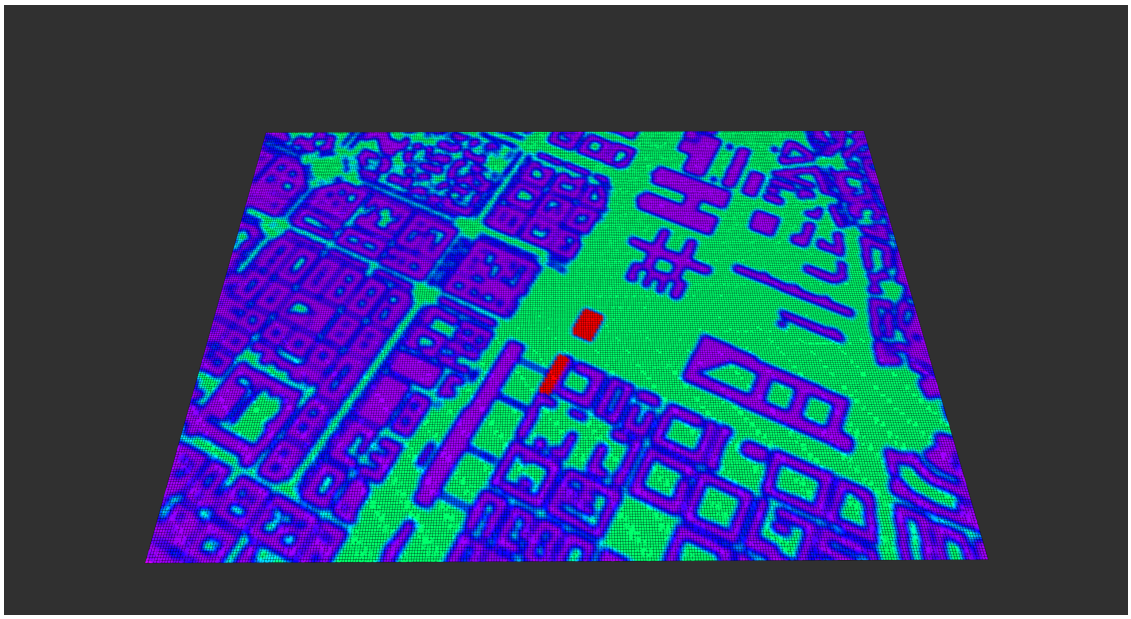


Figure 3.4: Example of Risk Map. Every area of the map has its own colour and height coherent with the risk value calculated. Red squares are no-fly zone

movement on the x and y axes, while the value of the z remains constant during the execution of the mission. Note, however, that despite the height of the flight is assumed constant, the risk of a mission is a function of it. Just think, for example, how the presence of buildings (and therefore no_fly_zone) depends on the height at which it was decided to fly.
In the final, once we have a risk map on which the path planner can decide the best route to assign to a UAS, it is possible to define a metric that allows to determine if the planned path meets or not the safety standards required by the national authorities.
The procedure we adopted to coherently perform a risk assessment procedure, and then building a risk map are described in figure 3.5. In the following part of this section, a quick overview of its main features will be provided.

Any risk map must clearly model consistently the environment it represents, in particular for what concerns the presence of structures and buildings. As already stated during the risk analysis of Section 2.3, and in particular when talking about mid-air collisions, the impact with this type of obstacle must be managed by Risk-Aware Map Manager.
In chapter 5 we will see how to obtain such a model starting from the data currently available publicly.
Besides the structural information from the model, we have to collect as map layer also the distribution of population (density) and the sheltering factor. Finally, the

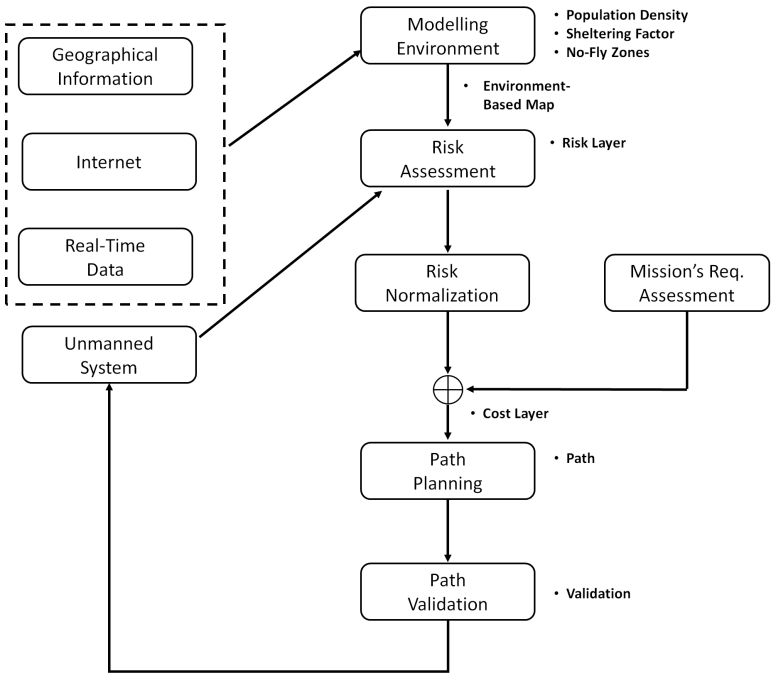


Figure 3.5

risk map will be built applying to every cell:

$$f_{F,i} = N_{i,exp} \times P(fatality|exposure)_i \times f_{UFE,i} \tag{3.1}$$

where each element i is about the i-th cell. At this point it is important to underline how on a risk map there are in some way two possible "risk situations" for each cell. The first is the one in which the risk is calculated using the equation 3.1. The other possibility concerns some particular areas for which, for different reasons, some cells must be marked as no-fly zones and are cells for which the associated risk is assumed infinite. A no-fly zone (NFZ) is trivially defined as an area above which it is not allowed to fly. It possible to distinguish two categories of no-fly zone:

Law-Imposed No-Fly Zone they are areas above which flight is prohibited due to legislative restrictions. The reasons behind these limitations are mostly related to security. Examples are the temporary no-fly zone imposed for major sporting events, or however for events that collect large numbers of people, or above sensitive targets. These areas must be appropriately managed by the cloud system, which can retrieve updated information via the internet, or can be manually entered by an operator. Notice how in this case the no-fly zone is valid at any height.

Structural No-Fly Zone in this categories are included all possible zone in which

is not possible to fly due to the presence of structural items (natural or artificial) that impede the flight. The typical example are the buildings taller than the height of the UAS's flight. It is so obvious that changing the height of flight, the number of this kind of no-fly zone could decrease.

A complete risk map is in fact a large matrix whose cells contain values that most often are within the range $[10^{-7}; 10^{-5}]$. It was therefore decided to normalize these values both for not having to deal with such small numbers, and to be able to relate the risk to other values of interest that are however expressed on a completely different scale.
It was therefore decided to normalize these values both for not having to deal with such small numbers, and to be able to directly relate the risk to other values of interest, which however are expressed on very different scales. To normalize the risk by comparing it to the appropriate scale, a technique derived from image processing technologies is used. It is called gamma correction. As a result, the normalized risk value of each cell is calculated with the following formula:

$$R_i = \left(\frac{f_i}{f_{MAX}}\right)^{\frac{1}{2}} \times 100$$

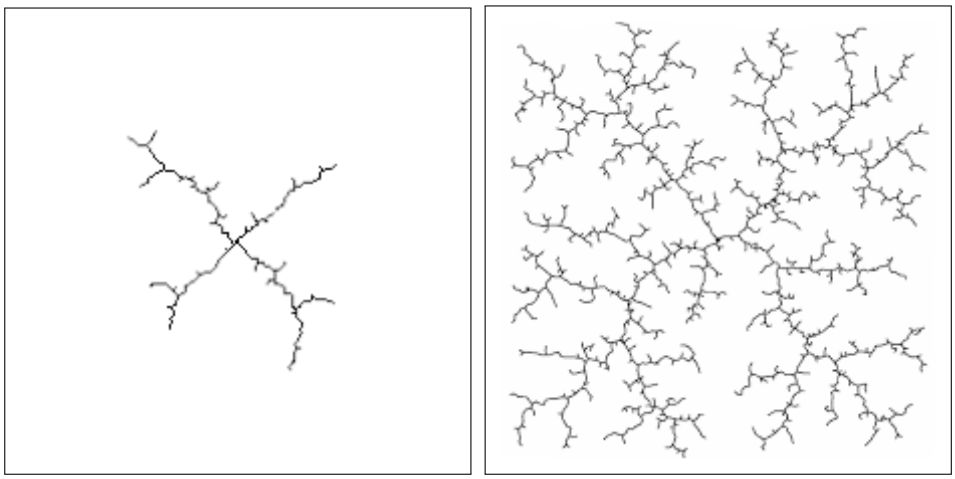
The "Map Manager" structure, which aim is to build the map upon which the Path Planner will works, at this point is capable of providing a coherent risk map, that takes in account both the Drone's building parameter and the environment in which it will fly. Nevertheless, is quite obvious that beside the risk there is also the mission itself, which has is goals and features. When planning the path for the drone to follow, it is important to find also a way that satisfies all the specifics imposed for that mission. According to figure 3.5 , all this aspects are modelized as "Mission's Requirements", and each of them will have its own layer where the "degree of satisfaction" for each zone is contained, normalized in range [0 , 100] as for the risk layer. A typical example, that will be used also in the Simulation part of this work, is the signal quality of service (QoS). The connectivity is a crucial point of this cloud based framework, since without a link with the cloud the UAV is blind. Moreover, we are working on a civil scenario, so let's suppose to have a data streaming from the aircraft to the cloud (maybe a video recording of the flying drone) that in order to be maintained has to provide a signal connection power greater than a lower-bound S_{lb}. In order to accomplish the mission, the path planner have not only to find the less risky path, but also guarantee that this path will have always $S_i \geq S_{lb}$, for every i belonging to the trajectory, where S_i is the signal power of each cell of the map. Practically, extending to a multi-requirements mission, what we want is to to have a mission planner that tries to provide the best trajectory for the UAV in order to satisfy all is goal: then it has to verify if the risk bounds are satisfied and, if not, re-calculate another path. The risk map in this scenario becomes only a layer (the most important one) of a more complex cost map

that will encapsulate all the information useful to accomplish the mission. Since all the layers are of interests are normalized between 0 and 100, it is possible to add them into a single layer called "Cost Layer", which will finally be provided to the Path Planner in order to find the trajectory that best fits with the requirements. However, since the risk is the core of this project, a Path Validation block is needed at the end of this chain: its aim is to verify, at a first stage, that each specifics has been satisfied by the chosen path, and finally to guarantee that risk's upper-bound has not been crossed. Once again the importance of the safety arises: while every particular requirements can have its own priority level, to specify how much it's important is fulfilment, the maximum risk value instead can't never be exceeded, under penalty of not authorizing the flight.

3.4 Path Planning - RRT*

The algorithm we decided to implement to plan the route of our Unmanned Systems is the Optimal RRT (or RRT*), a probabilistic algorithm derived from the classical Rapidly Exploring Random Tree (RRT).
The history of this method is quite recent, since it was firstly developed in 1989, by LaValle and Kuffner Jr, two American computer scientist and professors. It had also many variants and evolutions, among which the most important are RRT* and RRG.
The procedure executed by this algorithm aims to build a tree of open loop trajectories for non linear systems with state constraints: this capability of constructing feasible (for the UAV) paths is one of the key features of RRT. The tree is built extracting random samples from the state space, introducing also a bias to explore in the direction of unsearched areas. Every time a sample is drawn, a connection between it and the nearest state of the tree is attempted: if this link satisfies the constraints (practically, of it is feasible), the sample becomes part of the tree. Some kind of limitations can be introduced for the tree, for example in the length of the connection between the tree and the new state. A classical approach in case where the random sample is too far (i.e, it is farther than the maximum allowed distance) is to substitute it with a new state, at the maximum distance along the line that connects the sample to the tree.
Once this procedure is clear, it's easy to understand the way the algorithm works. The tree borns only with the initial state, than it starts to add random sample to it. This iterative procedure ends as soon as the tree contains a node in the goal state region. In figure 3.6 a typical evolution of RRT's tree is depicted.

The advantages of such approach are many. First of all, it is simple and easy to implement. Moreover, the tree always remains connected and guarantees a feasible path at every step. Finally, it is proved to work better and faster then many other deterministic algorithms. The main drawback instead is due to the need of saving



(a) (b)

Figure 3.6: RRT's tree evolution in time

at each step the overall tree, that implies an increase of the computational time while the algorithm is running.
Starting from this, Karaman et al. developed in 2011 the so called RRT* algorithm, which really increased the optimality of the solution obtained with RRT [24]. The mechanism the rules RRT* are obviously almost the same of RRT, but it introduces two new interesting features: near neighbour search and rewriting tree operation. The first finds the best parent node for the new sample that aims to be inserted in the tree, while the second rebuilds the tree within an area of given radius, in order to maintain always a minimal cost between tree's connections. Thanks to this, RRT* improves asymptotically the quality of its path as the number of samples increases, differently from RRT. In figure 3.7 the differences between the two approaches are shown: it seems evident that the tree built with RRT* is more ordered than the first one, thanks to the operation described before. Obviously, this features have also a computational trade-off, that can however be overcome with an high computational power cloud framework, as the one we have. Finally, after the studies of our path planning team, it seems that RRT* is the best compromise between efficiency and quality of the result. For this reason, it has been implemented in our framework as path planning algorithm for Unmanned Aerial Vehicle.

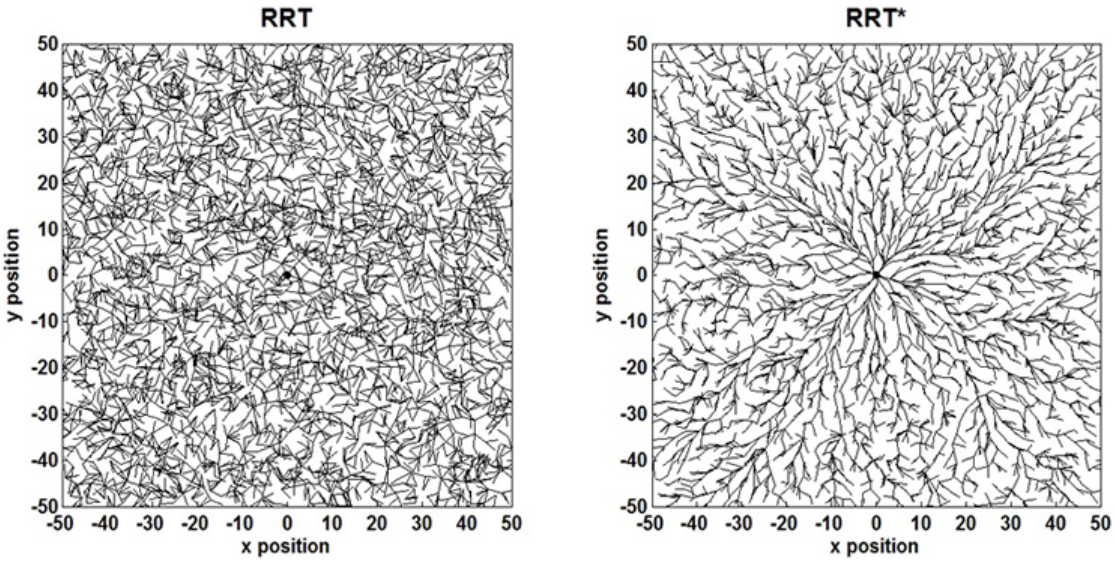


Figure 3.7: Differences between RRT's tree and RRT*'s one

Chapter 4

CBUTM: Trajectory Following

4.1 State of the Art Collision Avoidance

A critical aspect that arises when different UASs share the same airspace to accomplish their own tasks, is of course avoiding collisions. A collision is an accident that happens when a vehicle impact against another object, which can be both stationary (like buildings, trees etc) and moving (like other vehicles). According to [32], collision avoidance problem can be broadly classified into global and local problems.
Typically the solution to the global problem is one path that goes from the actual position of the vehicle to the position it has to reach to perform its mission, avoiding all obstacle that can be assumed to be stationary. A conventional path-planning algorithm just deals with finding this path.
The local problem, that is the one usually simply referred as collision avoidance problem, deals with avoiding unforeseen obstacles. Clearly this requires sensing capabilities on-board. In [18] and [17] avoiding collisions, that are autonomous or not, is a task that has to be assigned to each UAS and its on-board sensing system, sustaining that *"a UAS must handle the collision avoidance autonomously using sensors to detect obstacles and change the route as well as the altitude by itself. This works in the same way as autonomous cars in the research projects and the actual civil aviation"*[17].

On the other hand, since we have a Cloud to referring with, someone could instead thinks to assign to it all the aspects related to flight management and so also to collisions. In this way no particular sensing capabilities would be required, increasing autonomy and decreasing equipment costs' for each single flight. There are unfortunately different scenarios in which this mechanism cannot be useful:

- no connection (even temporarily) between the vehicle and the Cloud system
- the collision happens before the items can react according to the solution the Cloud computes and applies to them

- the colliding object doesn't belong to the Cloud control, but suddenly appear in its airspace (i.e. an emergency helicopter)
- the colliding object is not controllable (i.e. birds)

In other words, despite collecting data from vehicles and use those data to extract knowledge that vehicles can inherit is one of the aim of the cloud system, it's often not possible assume that it is sufficient to prevent any possible collision, especially in real dynamical environment like cities. For these reasons some sensing capabilities are obviously required on board, but they are out the purpose of this work.

Between all possible moving obstacles we surely find other autonomous vehicles, in the following called also *active agents*, about which we can make different assumptions respect any other kind of obstacles. The main assumption in this context, is that there are not antagonistic vehicles, i.e. each vehicle in the network communicates required data and follow received instructions. Another category of interest is the that of the so called *passive agents*. Those are vehicle able to communicate their actual and goal positions, but they don't receive any command. It's the case of vehicles piloted remotely or directly on board, how those used for emergency or security purposes.

At this point we have to solve the problem of collisions between an active agent with on or more active/passive ones, without relying on sensing capabilities. In particular, if an active agent is involved in the occurring collision, we will be able to control it to prevent such a circumstance. If only passive agents are involved, we will be only able to generate an alert for respective pilots.

So, the aim of the Cloud-Based UTM is to provide, when it is possible, a redundant layer against collisions. In particular it has to prevent that the real trajectories (i.e paths with relative time laws the UASs follow in practice, not only the originally planned ones) of two or more UASs under its control intersect in a dangerous manner.

In literature a lot of different approaches are proposed with this aim. In the following we investigate the majors of them to define the one we will use in our project. The simplicity of the solution, at this stage, will be accounted as an important design requirement.

According to [41] [23], two different approaches exist to achieve (local) collision avoidance in robotics:

Planning provides feasible collision-free trajectories to each agent in the airspace ahead of time.

Reacting consists in an online system able to responde adequately to dangerous situations when they arise.

As we will see, the two approach can be combined into hybrid solutions.
Starting from planning approach, we can cite [26], where collision-free trajectories for a team of robots is achieved solving a mixed integer quadratic programming problem in a centralized fashion. A similar approach was presented in [8], where instead sequential quadratic programming technique are employed. However, different version of this complex problem are proved to be PSPACE Hard even in static environment [40].
Furthermore, this kind of approach puts different limits to the adaptability of the missions since it could require re-planning of all trajectories in the case one of these needs to be adjusted [23]. But adaptability is crucial especially considering the high dynamism of our referring context. Since at JOL the research on path planning strategies and solutions continues, and in future steps could include some technique relative to the planning approach to collision avoidance, we decided to start developing a reactive online system to prevent collisions. It is important to remark again that in future the two different approaches can be combined to provide the pros of each one. As we will be clear in the following, our reactive system is compatible yet with a path planner of this type.
Also to maintain this compatibility, we decided to unify the trajectory tracking and collision avoidance in a distributed manner, as just seen in [23]. In this way, the trajectory generated by the path planner, however it works, can be applied directly to our system without any modifications. This system will be able to carry out trajectory tracking while, reactively, it will be also able to detach itself from it, as much as necessary to avoid collisions with other vehicles. A path planner able to provide collision-free trajectories, if available, would not require any modification in the reactive system but clearly would decrease the risk of collisions and would increase the safety of operations in the airspace.
Having opted for a reactive approach, we found in [7] something particularly related to our context: a centralized control which, knowing the objects to be controlled and their positions, uses a model predictive control to find the smaller steering angle to be applied to the involved vehicles such that the occurring collision is avoided.
In fact, a cloud system is a central controller with whom each vehicle has to refer with, communicating its parameters (position, velocity, mass of payload etc) and receiving instruction to accomplish its task. So the cloud system has not only an updated state of involved agents, but also the capability to modify their trajectory.
In this work the computational time is appointed as the major drawback. As suggested by the same author, distributing the problem makes it easier to solve and the system more robust at the same time, as explained in more details in [6] and below.
The problem of collision avoidance, as we will see later, is very often approached in literature with control techniques based on the model predictive control [23], [40] and for the generation of collision-free local trajectories [39].
In the following we are going into detail for what concern model predictive control,

showing its working principle and why it results to be particularly adapt to accomplish control, in particular distributed control, into a Networked Control System (NCS).

4.1.1 Networked Control System

Now we introduce the concept of Networked Control System (NCS), which consists of dynamic subsystems interacting with each other using communication networks over which they can exchange data. It immediately becomes clear how this concept is particularly suited not to our project, but in general it lends itself to being a reference paradigm with the advent and diffusion of the Internet of Things. A more formal definition is present in [6]:

Definition 4.1. A NCS is an integration of network and actions, i.e.,the fields of communications, computation, and control. It consists of multiple agents, which communicate and interact over a time-variant network. Each agent has its own controller and aims to achieve its goal while coordinating with other agents in the NCS.

According to [10], advances in communications have ended up with affecting the platforms usually used for control systems. There are, in particular, applications that more than others seem to benefit from what results to be the advantages of a "networked" approach. These applications are those that benefit from a control system that is particularly prepared to be:

large-scale using a communication network, has both the benefit of making control loop actions independent of the physical distance that separates the different entities involved in the loop, and offers also the possibility of forming additional control loops. This latter implies that it is possible to define further control objectives at the same time. These possibilities make a NCS suitable for use on a large-scale systems.

openess unlike most traditional control systems, in a NCS system it is possible to reconfigure runtime for the operation of the system itself. This means that it is possible for some subsystems to join the network and for others to leave it without compromising the functioning of the control system. In many situations, in fact, the control flows can be readjusted dynamically to the current configuration. Note how this leads to benefits also for the modularity of the overall system, including the possibility to easily update the control algorithms.

These are fundamental characteristics for any UTM and therefore also for that object of this work.
Interacting subsystems in an NCS are also called agents, and can be broadly classified into two categories:

Passive agents are those dynamic subsystem that can communicate their data (like current and future states) but that are without networked control.

Active agents use data exchanged trough the network as one of the input for their controller, so that the controller can find the appropriate input to achieve their goal while considering other agents, both active and passive.

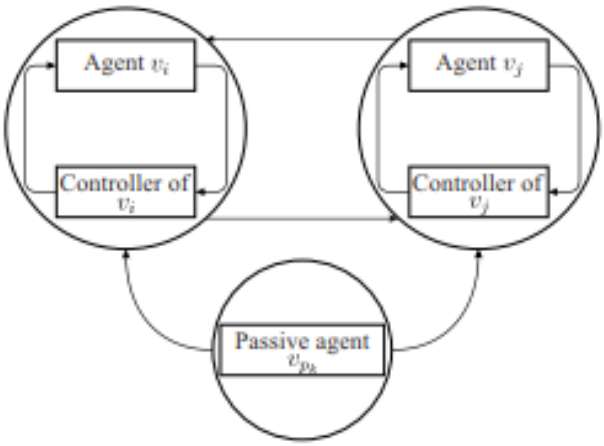


Figure 4.1: Passive and Active agents in NCS

A traffic control system is one of the classic examples of networked control systems with active and passive agents.

A networked control system can also be classified, according to the agents participating, in homogeneous or heterogeneous. To be homogeneous, all agents of an NCS must have the same dynamic.
Another important feature, which will then be examined in more detail below, concerns the interaction mode between the agents of an NCS: they can be dynamically coupled or decoupled.

In an interconnected system, there are several ways in which control action can be generated. These different modalities can be summarized in 3 different approaches to the control problem itself, each of which leads to different architectures of the overall system. The advantages and disadvantages of each will be dealt with in the following but, for the moment, we only intend to offer a quick overview:

centralized The control action is calculated from a single central entity, to which all the agents are connected. In particular, the agents send the data produced to this central unit and receive from it the control command to be applied to pursue their objectives.

decentralized each agent has his own controller and his own control action that is independent from that of the other agents.

distributed it is a hybrid approach of the first two. each agent still has a dedicated controller, but control action is achieved by interacting with the other agents.

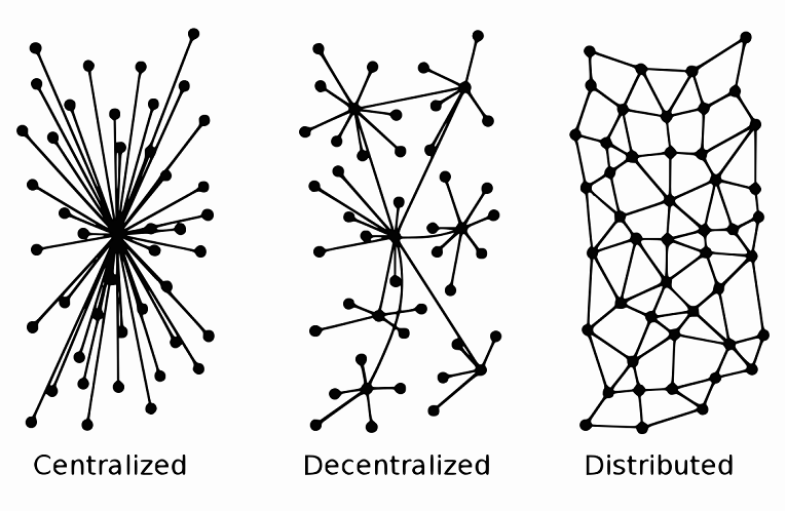


Figure 4.2: Classical representation of differences between centralized, decentralized and distributed approach

A recap of the classification criteria for a NCS is presented in figure 4.3

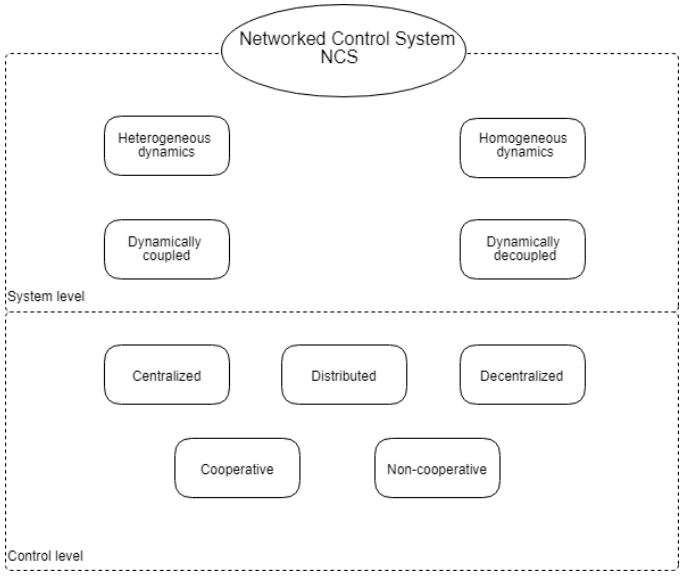


Figure 4.3: Graphical classification of NCSs

Once the overview on Networked Control Systems is over, we would like to quote [6]: *In recent years, there has been much interest in Networked Control Systems (NCS) with time delays and constraints. MPC can be considered as a natural control framework to handle NCS with coordinated and distributed agents as it can deal with the action of other agents with respect to their future intention and even their objective functions while making decisions of its own control inputs.*

The MPC to which the author refers is an acronym for a category of control algorithms called Model Predictive Control or Receding Horizon Control. It is therefore appropriate to give an overview of the principles of operation of this control strategy.

4.1.2 Model Predictive Control

In short, the principle behind this control methodology consists in choosing the control action to be applied, by evaluating its effects on a model of the system to be controlled on a finite time-horizon.
In other terms, starting from the state of the system sampled at the time t, iteratively applies a series of possible control actions on a dynamic model of the system itself, thus predicting the future states in the time interval $[t; t + H_P]$, where H_P is said *horizon of prediction.*
The simulations performed are then evaluated on the basis of a function specifically defined, called the cost function, which assigns to each predicted state, and to the actions required to reach them, a numerical value, said cost.
The best control strategy is therefore that which turned out to be at minimal cost in the time interval $[t; t + H_P]$.
In particular, however, at the instant of time t only the first step of this strategy will actually be applied to the system to be controlled. The others are discarded. Then the system is sampled again and the procedure starts from the beginning. This mechanism is represented in figure 4.4

Choosing the best control sequence means solving an optimization problem.

The optimization problem is usually constituted, as we will see in more detail in section 4.2 , by an objective function to be minimized (or maximized) and by constraints.
The objective function is often formulated as a cost to be minimized, the constraints represent the limit on the system states and input.
Traditionally a cost function J relative to the optimization part of a receding horizon control algorithm, is defined:

$$J(\mathbf{u}(k), \mathbf{x}(k|k)) = \sum_{i=0}^{H_p-1} c(\mathbf{x}(k+i|k), \mathbf{u}(k+i|k) + \Phi(\mathbf{x}(k+H_p|k))$$

where:

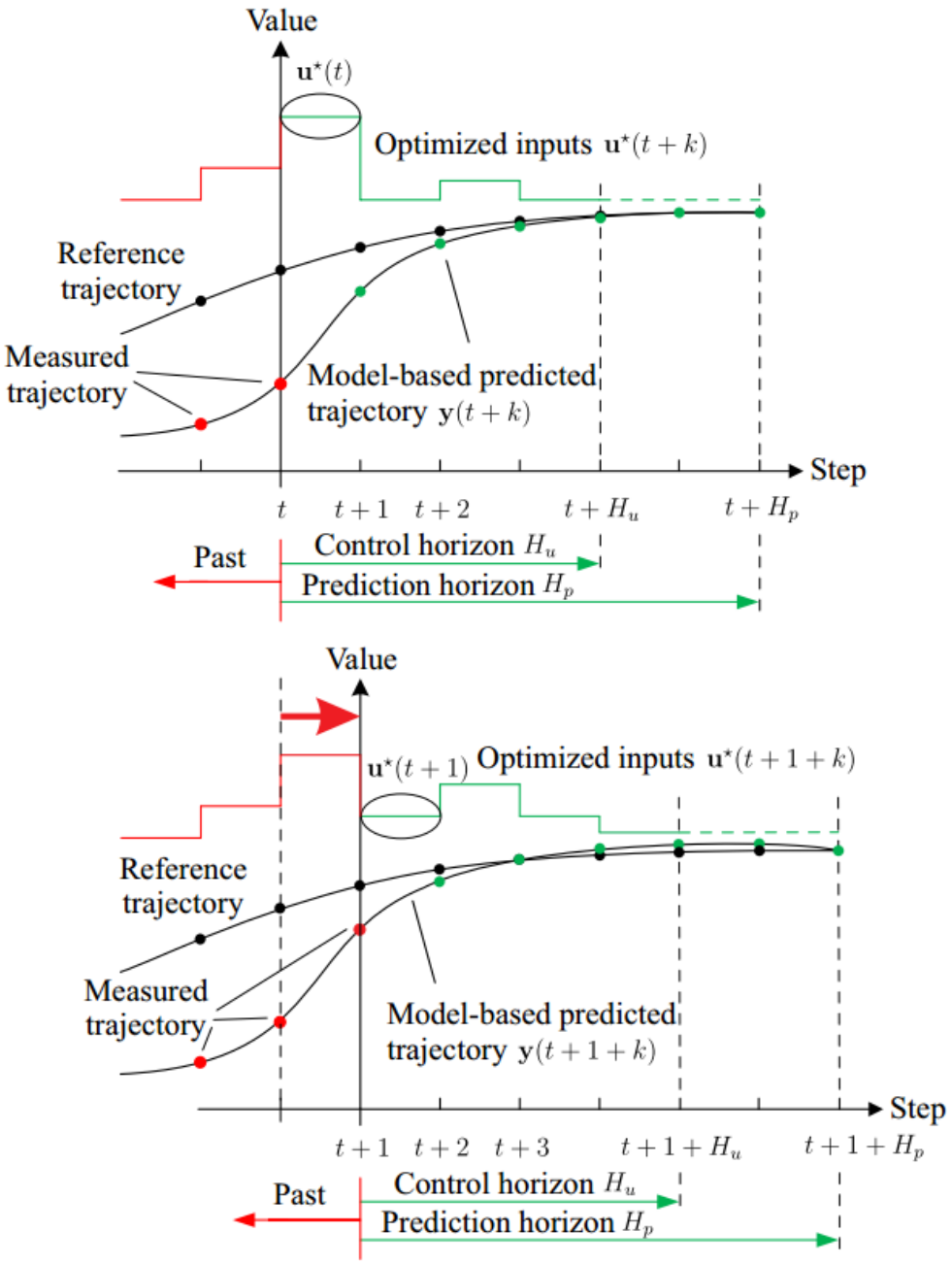


Figure 4.4: MPC and the receding horizon principle taken from [6]

$\mathbf{x}(k)$ is the state of the system at time k

$\mathbf{u}(k)$ is the control input at time k

H_P is the prediction horizon

c is the per-stage weighting function

Φ is the weight for the last state of prediction

$\mathbf{x}(k+i|k)$ indicates the state at time $k+i$ computed with the data known at k

The control vector $\mathbf{U}(k)$ can be computed solving the optimization problem:

$$\begin{aligned} \mathbf{U}^*(k) = \min_{\mathbf{U}(k)} \quad & J(\mathbf{U}(k), \mathbf{x}(k|k)) \\ \text{subject to} \quad & \mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)) \\ & \mathbf{U}(k) \in \mathbb{R} \\ & \mathbf{x}(k+1|k) \in \mathbb{X} \end{aligned}$$

where the first constraint is the system dynamics, the second represent the constraints on control inputs and the third on the states.

The ability to limit future states and control actions simply by adding appropriate constraints in the optimization problem is one of the advantages of MPC-based strategies.
So what we obtain solving the optimization problem is a sequence of feasible control input, one for each step over the prediction horizon. As previously said, only the first is actually applied to the system. The problem is solved at every sample time of the controller because of the obvious differences from the predicted states to the real ones.
From a control point of view, the mechanism of prediction incorporates a feedforward logic, while the receding horizon computation introduces the feedback in MPC.

In the course of these overviews we have seen why the concepts of Networked Control System and Receding Horizon Control are suitable to be combined with each other and are suitable to address the problem that is the subject of this thesis. We can define:

Definition 4.2. A Net-MPC is a Networked System controlled by means of a Model Predictive Control logic.

4.1.3 Networked-MPC

Now we are going to analyze different strategies of Networked-MPC (from now Net-MPC).

Centralized MPC

All agents communicate with the central entity, which resolve the unique optimization problem. In this problem objective functions and constraints of all agents are

taken into account at the same time. In particular, the resulting objective function is the summation of single objective functions (one for each agent) as well as the resulting set of constraints is the union between the sets of constraints (one set for each agent).

Solved the problem, it communicates to each agent its relative control input just computed.

The disadvantages of this approach, according with [6], are:

- it has a high computational time that results in a long sample time which affects the NCS performance
- in case of a large scale system it requires a complex communications network
- single point of failure

Due to these disadvantages, a centralized approach is often not applicable [6]. To divide the problem into smaller parts, decentralized and distributed MPC are taken into consideration.

Decentralized MPC

In a decentralized MPC, agents do not communicate with each other and no coupling is accounted for in the optimization phase. Each one solves its own minimization problem, considering only its objective function and its own constraints. This means that: «Often, it is not possible to implement a functional decentralized MPC as the coupling between agents must be taken into consideration» [6].

Distributed MPC

Distributed MPC is a hybrid between centralized and decentralized MPC. The agents communicate and each agent has its own controller, which takes other agents into consideration.
We distinguish two kind of Distributed MPC: Cooperative and Non-Cooperative.

Cooperative Distributed MPC

It can be seen as a decomposition of a Centralized MPC problem into smaller problems. What cooperations means in this scenario is that resolving its own optimization problem, each UAS considers its own benefits alongside with benefits of other agents. This means also that the communication graph and the coupling graph coincide. In other words each agent considers only its neighbors and the coupling between them. If those graphs are not fully connected, the predictions computed by an agent for its neighbors differs from the predictions computed by those agents for themselves [6]. This is because those agents consider their own

neighbors that are not necessary the same of the agent taken into account. If instead the graphs are fully connected, the Coop. DMPC is equivalent to a CMPC except for the communication structure. In this case results that:

- high computational time
- the communication network is simpler than in CMPC's case
- reduces the safety hazard (a failure in one controller does not led to a failure of all system)

The main drawback is increasing the number of agents leads to an increasing of decision variables and constraints of the optimization problem that every agent has to solve, and so to an higher computational complexity. Furthermore the advantages are not so consistent due to errors in predictions and disturbances. One solutions to this problem is adopting a Priority-Based Non-Cooperative Distributed MPC, which is debated below.

Non-Cooperative Distributed MPC

Like in Coop. DMPC, also in this case each agents has an own controller which solves a part of the overall optimization problem. This controller solves the optimization problem relative to the agent itself, which takes into account only its own objective function and constraints and coupling objective and constraints with its neighbors. One of the main differences with the above case is that in Non-Coop DMPC, agents don't compute predictions for their neighbors, but instead use the communication network to acquire predictions from their neighbors and to respect coupling with them. This kind of approach fit quite well our requirements and objectives. What still miss is a coordination method for adjust trajectory.

Example 4.1. Consider two vehicles traveling along the same line one towards the other.

Step 1 As long as they are distant enough, each one will follow its initial trajectory.

Step 2 At a certain point the predicted trajectory of one will encounter the predicted trajectory of the other. At this point, based on the trajectory received from the other vehicle, each will elaborate a new trajectory in such a way that it is collision-free.

Step 3 At the next controller step, each vehicle, seeing that the trajectory of the other has changed, could decide, for example, to relocate its predicted trajectory on the initial straight line.

Step 4 Acting both in the same way, the predicted trajectories would meet again generating a risk of collision, as happened two steps before. This mechanism could be repeated indefinitely.

This problem is known in literature and different solutions were presented. One way, for example, is to solve the optimization problems of different agents in sequence and to iterate until they converge to a solution [27]. Another solution consider to apply a constraint to the predicted control input that limits its deviation from the control inputs sent to other agents [45]. These approaches succeeded but have some drawback: the first approach requires a higher computational time when agents became more; the second approach may limit the control actions also in cases in which they are necessary according to the real actual state. In [6] a Priority Based approach is proposed to solve this problem.

Priority-Based Non-Cooperative Distributed MPC

Priority is one of the possible way to manage the problem arises from Example 4.1. To each agent v_i of the network is assigned a priority value $p(v_i)$ by mean of an injective function, which associate a different value of priority for every different agent of the network. At this point every agent considers, alongside with its own objective function and constraints, only the coupled objectives and constraints of its neighbors with higher priority. In this way each agent communicates its optimized trajectory to its lower priority neighbors and only one iteration is required to converge to a solution. Assigning to passive agents (i.e. those outside the network's control but of which the network knows position and velocity) the highest values of priority in the network, they will be properly considered by all active agents.

Example 4.2. Consider the same scenario of Example 4.1. Adding a priority mechanism, we have that at the **Step 2** only the agent with lower priority detects the collision and adjusts its predicted trajectory according to the higher priority's one. **At Step 3** both predicted trajectories are collision-free.

Therefore, PB-Non-Coop DMPC reduces the size of the optimization, and so requires less computational time in comparison with other Non-Coop DMPC.

The Net-MPC strategies illustrated here are suitable for contexts in which the subsystems (agents) are dynamically coupled and independently actuated [6]. In fact, coupling takes place at the level of objective function and constraints. These in particular are a function of both the states of the agent in question and a subset of the states of its neighboring agents. This strategy is versatile, as well as for collision avoidance (which we will investigate) for other problems such as flight formation, platoon control and others.

4.1.4 Application of Net-MPC strategies to the Collision Avoidance Problem

Due to their predictive nature, receding horizon techniques results to be commonly used to address a problem like path following with collision avoidance. In literature such a control architecture is used to generate the input of motion actuators of the vehicle into account, or to find the minimum deviation to be applied, respect to the original a-priori trajectory, so that the collision is avoided. In the case of wheeled robots, as in [6], the control parameter is typically the steering angle.

In our scenario we can refer to the autopilot of each vehicle, which already offers the possibility to manage the vehicle and the flight at an higher control level, as seen in 2.2.4 and depicted in fig. 4.5. In section 4.2 we will see how to apply this techniques to our framework.

4.2 Trajectory Following with Collision Avoidance in CBUTM

In section 4.1 we defined the problem into account here and found a suitable approach to solve it in section 4.1.3.
In that section we identified the collision of agents that interacts with the platform as the problem to be addressed here, justifying why we opted for a reactive and distributed approach that doesn't rely on the sensing capabilities on-board to be useful. This led us to choose a receding horizon strategy for the networked control system we are facing with. Receding horizon strategies, in fact, result to address quite well problems related to navigation [30].
According to [47], algorithms based on receding horizon strategies are the most common control method used for trajectory tracking, mainly because:

- are easy to model
- have rolling optimization strategy with good dynamic control effect
- can correct the output by feedback, this improves the robustness of the control system
- being a computer optimization control algorithm, it is easy to realize on a computer

Some advantages of a distributed controller arise yet in section 4.1 analyzing different strategies to address collision avoidance in networked control systems. In particular:

Less computational power required: a unique and complex optimization problem is discomposed into smaller ones

Scalability: also increasing the number of involved vehicles the complexity of the controller doesn't increase

Robustness to failures: with a centralized approach there is a single point of failure for the system

In this section we will try to apply all of this to our referring framework, that is the one depicted in figure 4.5.

This imply some substantial differences. The output of the optimization problem, that is the next control input of the system itself, will not to be applied directly to electromechanical actuators, as usually seen in literature, but is the input for Ardupilot (or for other autopilot systems, vd. 2.2.4). That allowed us to stay focused on the problem from high-level point of view.

So it is with a such system that our receding horizon controller has to relate with. The input accepted by Ardupilot are defined according to the MAVlink protocol, and so it is according to MAVROS that we have to define our output. See [11] for the complete list of command available in MAVROS.

We tried different approaches. They are showed, alongside with the analysis of their differences, in the following subsections.

4.2.1 Context and Assumptions

The different approaches we tried maintain however a common topology, that is depicted in figure 4.5.

We have seen in section 4.1.2 how an optimal problem is usually defined, showing the classic formulation of the objective function and the optimization problem and describing the terms that compose them. Now we report only the general form of the objective function and the minimization problem because on these we have to make some additional considerations with respect to those already made above related to the reference context.

$$J(\mathbf{u}(k), \mathbf{x}(k|k)) = \sum_{i=0}^{H_p-1} c(\mathbf{x}(k+i|k), \mathbf{u}(k+i|k) + \Phi(\mathbf{x}(k+H_p|k))$$

$$\begin{aligned} \mathbf{U}^*(k) = \min_{\mathbf{U}(k)} \quad & J(\mathbf{U}(k), \mathbf{x}(k|k)) \\ \text{subject to} \quad & \mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)) \\ & \mathbf{U}(k) \in \mathbb{R} \\ & \mathbf{x}(k+1|k) \in \mathbb{X} \end{aligned}$$

Recalling that: $\mathbf{u}$ is the control vector to be found by minimizing J; the constraints are relative, respectively to the system dynamics, the control inputs and the states.

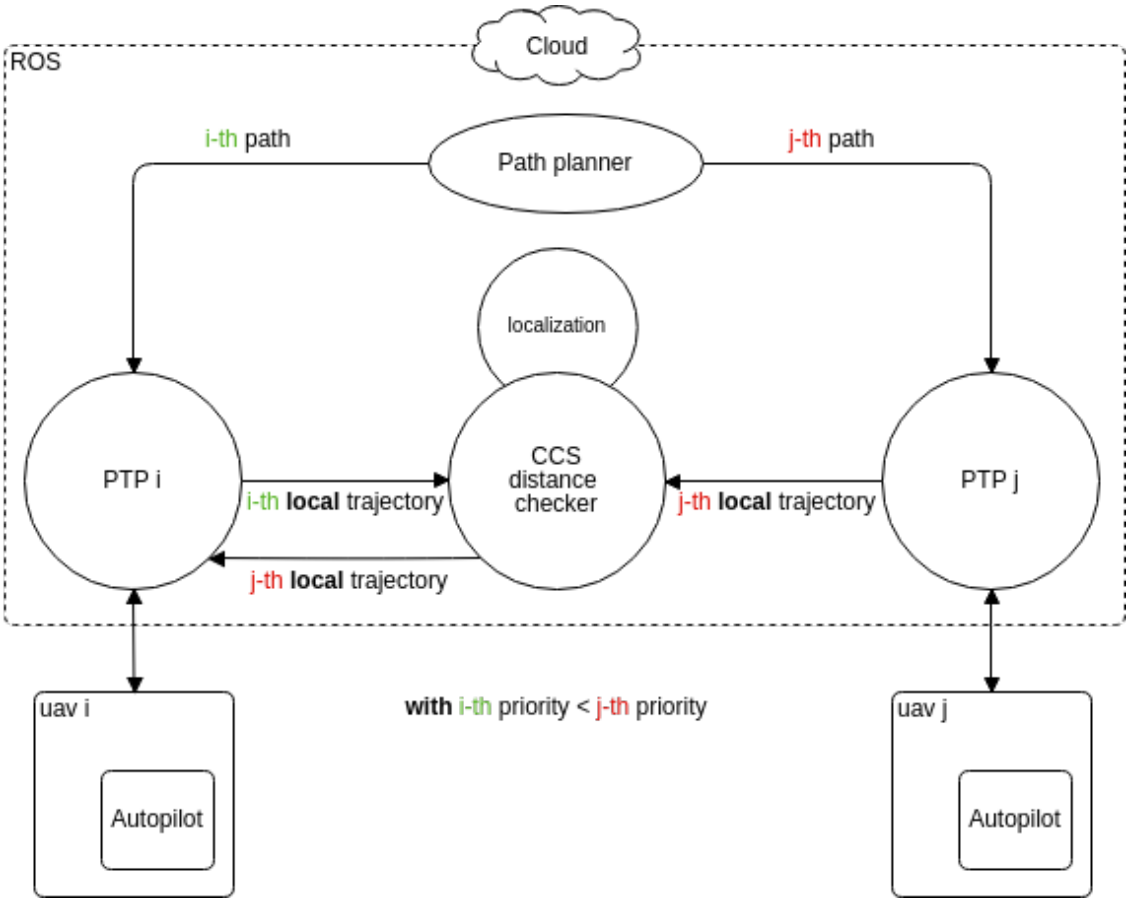


Figure 4.5: PTP Topology

The state **x** is always assumed to be the position of the agent, expressed in a certain reference frame, so will be also called **p**. The control input **u**, instead, will vary depending on the approach we followed, so it will be provided in the following sections. Note also that we assumed a 2D environment for agents and their maneuvers, i.e. the plane resulting from a cut at a certain height (the height of flight) of a real 3D environment.
Despite the current strategy of trajectory following and collision avoidance is conceptually independent from the type of vehicle used, what follows has been specifically calibrated, according to what emerged from the risk analysis, for copter-like vehicles more than for fixed wing ones.
The following assumptions are also made, to which reference will be made for each of the tried and tested approaches:

Goal Position: remembering the topology of this block (vd. fig 4.5)), we assume a path coming from the Mission Planner, i.e. an array of poses said waypoints, as input. So we call "goal position" the position of the next waypoint the

agent must reach. In the following, without loss of generality, we will assume that the goal position is outside the horizon of prediction, i.e, the destination is the same for each step of the prediction $\mathbf{x}(k+i)$

Forbidden set $\mathbb{S}$: is the union set of all the forbidden areas for a certain agent. It is comprehensive of areas occupied by buildings and all the safety area of a certain radius relative to other agents, around both actually occupied positions and around those they plan to occupy.

Agents: as stated in the previous chapter, agents are distinguished between passive and active. Both of them communicates their positions and plans to CBUTM.

Obstacles: Every passive agent or every active agent with higher priority is considered an obstacle.

.

4.2.2 Equilibrium Point

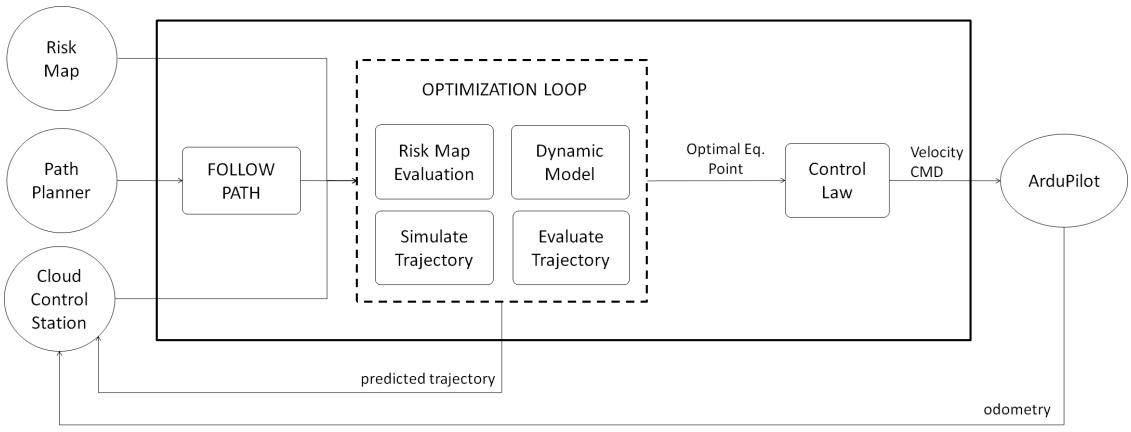


Figure 4.6: Equilibrium Point Functional Block Diagram

The first approach we tried is partially based on [35] and is depicted in figure 4.6.
The optimization problem is used here to find the coordinates of a point (said equilibrium point), in a radius of 40 meters from the current position of the vehicle, such that: the trajectory that leads from the current position of the drone to that point is the trajectory at minimal cost.
The evaluation of the trajectory takes place here by evaluating the succession of the poses that compose it. In particular, the cost of each of these poses is calculated in accordance with the following cost function:

$$J(\mathbf{p}(k), \mathbf{p_G}(k)) = \sum_{i=1}^{H_p-1} \|\mathbf{p}(k+i|k) - \mathbf{p_G}(k)\|^2$$

where:

$\mathbf{p} = \begin{bmatrix} x & y \end{bmatrix}^T$ is the predicted state at time $(k+i)$, representing the position of the agent in a cartesian reference frame

$\mathbf{p_G}(k)$ is the goal position

Note that what we still obtain from this problem are the succession of poses that minimize the cost function, not directly a control input. But this will be obtained from the dynamic model of the system.

So the optimization problem is in the form:

$$\begin{aligned} \mathbf{P}^*(k) = \min_{\mathbf{P}(k)} \quad & J(\mathbf{P}(k), \mathbf{p_G}(k|k)) \\ \text{subject to} \quad & \mathbf{p}(k+1) = f(\mathbf{p}(k), \mathbf{p_G}(k)) \\ & \mathbf{P}(k) \in \mathbb{P} \\ & \mathbf{p}(k+i|k) \notin \mathbb{S}, \forall i = 0, \ldots, H_p \end{aligned}$$

where:

$f(\cdot)$ is the dynamic model of the system

$\mathbb{S}$ is the forbidden set

$\mathbb{P}$ is the set of feasible states in the radius of 40 meters from the agent's position

Here the control input is defined: $\mathbf{u}(k) = \begin{bmatrix} v_x & v_y \end{bmatrix}^T$, where v_x is the linear velocity along x and v_y is the linear velocity along y.
Found $\mathbf{P}^*(k)$, we can obtain $\mathbf{u}(k)$ by means of the dynamic model of the system.

The computed control input is send to the autopilot, according to figure 4.5 and figure 4.6.

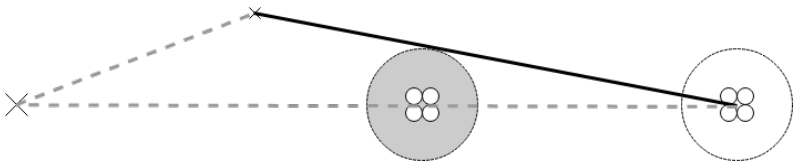


Figure 4.7: sim of equilibrium point

As a result of the above, we have that the predicted trajectories, in the case in which there is an obstacle between the agent and its goal, result as in figure 4.7.

The trajectories produced here, in fact, not only are not accurate, but do not transmit the correct information content to those who use it.
This approach result so to be suitable for navigation and exploration purposes, but since the quality of our distributed collision controller is based on the quality of the predicted trajectories, since basing on them other agents will compute their own trajectory, we started looking for something else.

4.2.3 Step by Step

In this case the optimization problem aims to find singularly the next waypoints for the agent. Iteratively, each found waypoint constitutes the starting point for the next optimization problem, till we reach the horizon of prediction or the goal position.

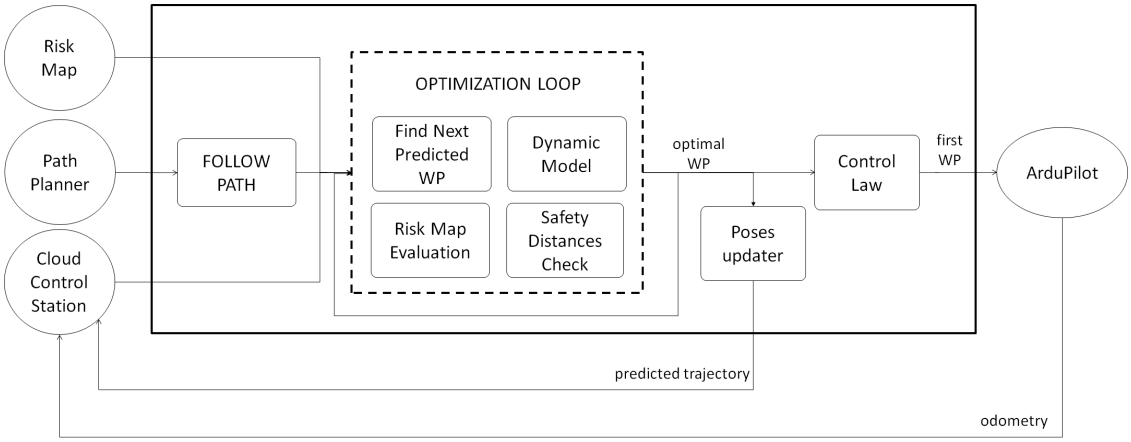


Figure 4.8: Step by Step functional block diagram

For each step of the prediction, starting from the real agent's position, the optimization problem look for a position, in the neighborhood of 4 meters as close as possible to the goal and out of the forbidden areas. Found such a pose, this is assumed as a starting point for the next optimization problem and so on.

The cost function is here so defined:

$$J_i(\mathbf{p}(k), \mathbf{p_G}(k)) = \|\mathbf{p}(k+i|k) - \mathbf{p_G}(k)\|^2, \qquad \forall i = 1, \ldots, H_p - 1$$

The absence of the classical summation defining J, is due to the fact that an independent optimization is performed every time we need to determine the successive position of the vehicle, so for every step of the predicted trajectory.

The optimization problem results to be:

$$\begin{aligned} \mathbf{p}^*(k) = \min_{\mathbf{p}(k)} \quad & J(\mathbf{p}(k), \mathbf{p_G}(k|k)) \\ \text{subject to} \quad & \mathbf{p}(k+1) = f(\mathbf{p}(k), \mathbf{p_G}(k)) \\ & \mathbf{P}(k) \in \mathbb{P} \\ & \mathbf{p}(k+i|k) \notin \mathbb{S}, \forall i = 0, \dots, H_p \end{aligned}$$

As before, the command $\mathbf{u}(k)$ is derived from $\mathbf{P}(k)$ by means of the dynamic model $f(\cdot)$.
The drawback of this approach is easily showed in figure 4.9. The problem is due to the fact that each pose is evaluated alone, so it will be as close as possible to the goal. In the case of an obstacle long its original path, however, it results in a late deviation from the original trajectory, that not only means a not efficient (then, smooth) trajectory, but also a not effective approach, since the vehicle is not able to change its velocity vector as soon as possible. This translates into a trajectory close to the obstacles' safety area.

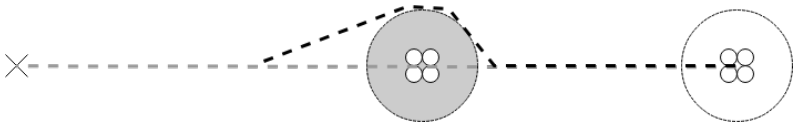


Figure 4.9: sim Step by Step Functional Block Diagram

4.2.4 Step by Velocity

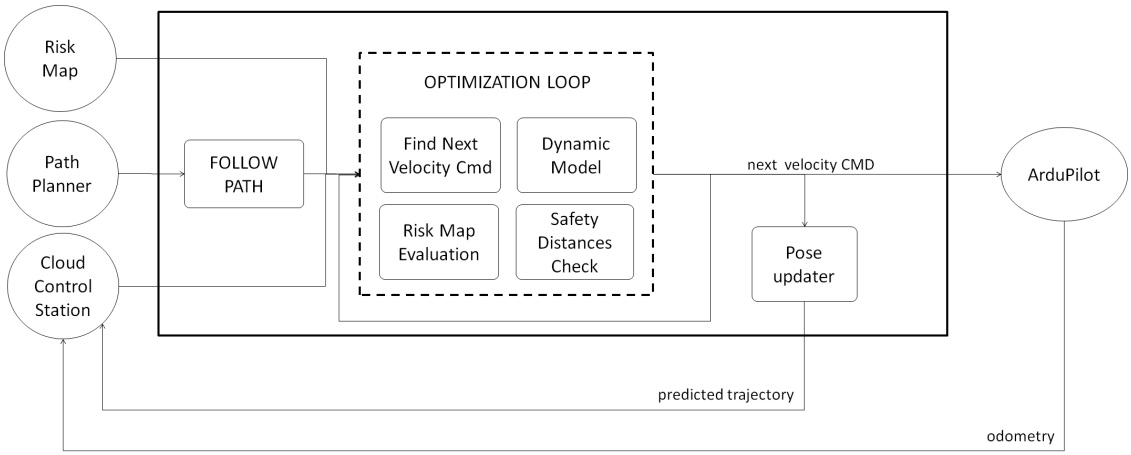


Figure 4.10: Step by Velocity Functional Block Diagram

This approach is similar to the previous, but introduce one main innovation: the control vector $\mathbf{u}(k)$ is now directly subject to optimization.

So each position of the predicted trajectory is still found independently from the others into an iterative process, as before, but now these pose are computed as result of the action of a control input applied to their starting point (i.e. the previous predicted pose) by means of the dynamical model of the agent itself.
As control input we still use a velocity command, to be sent to the autopilot to properly apply it to the on-board actuators, so: $\mathbf{u}(k) = \begin{bmatrix} v_x & v_y \end{bmatrix}^T$, where v_x is the linear velocity along x and v_y is the linear velocity along y.

The problem is set to find $\mathbf{u}(k)$ such that it leads to a position that is as close as possible to the goal.

So we have as cost function:

$$J(\mathbf{p}(\mathbf{u}(k)), \mathbf{p_G}(k)) = \|\mathbf{p}(k+i|k) - \mathbf{p_G}(k)\|^2$$

where is made explicit the relation between $\mathbf{p}$ and $\mathbf{u}$. The optimization problem as the form:

$$\begin{aligned} \mathbf{U}^*(k) = \min_{\mathbf{p}(k)} \quad & J(\mathbf{p}(\mathbf{u}(k)), \mathbf{p_G}(k|k)) \\ \text{subject to} \quad & \mathbf{p}(k+1) = f(\mathbf{p}(k), \mathbf{p_G}(k)) \\ & \mathbf{U}(k) \in \mathbb{U} \\ & \mathbf{p}(k+i|k) \notin \mathbb{S}, \forall i = 0, \ldots, H_p \end{aligned}$$

This approach has brought some benefits respect to the previous one, but it has also highlighted the problems deriving from a velocity control action in such a framework. Controlling by velocity a system require an accurate system dynamic model and a control frequency much higher than those at our disposal. Furthermore, the quality of predicted trajectories was still not satisfying.
This has led us to change our approach and try to make better use of a component we are already interacting with, but which can offer us vehicle control features at a higher level: the autopilot on board. We will then pass, in the next versions, from controlling the agent by velocity, to controlling it by position.

4.2.5 Optimizing-Traiettoria

In addition to the already mentioned novelty of position control, the following approach illustrated here introduced another major change.
As shown in figure **??**, the output of the optimization problem is now a trajectory. This means that are not single poses or velocities to be optimized, but the succession of way-points (WP) itself.
In this way the maneuverer needed to avoid a collision is properly addressed in prediction phase and this results in a smoother predicted trajectory.

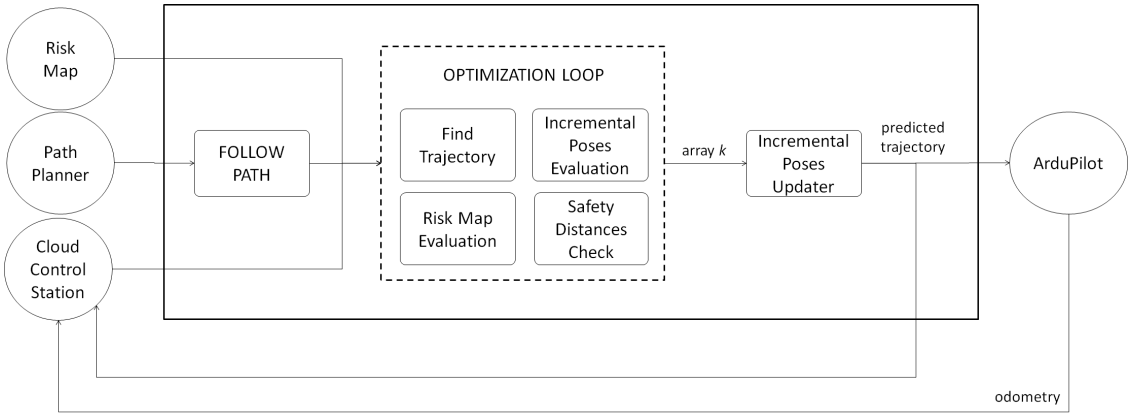


Figure 4.11: Optimizing-Traiettoria Functional Block Diagram

The method actually implemented to optimize a trajectory will be explained in detail in the section 5.4.2. For the moment we want to know that the result of the optimization is an array that contains as many pairs of incremental polar coordinates as there are steps of prediction. This array is the one indicated with $arrayk$ in the figure 4.11.
Note that the predicted trajectory is directly send as input for the autopilot: being able to send to Ardupilot the entire trajectory predicted in the form of a "mission" was a merely implementation step, but allowed us to obtain significantly better results than the other attempts, in which the control action imparted to the autopilot was a velocity command or a goal position.
As usual, this control input is constantly updated according to the evolution of the state variables. So we have that:

$$\mathbf{U}(k) = \mathbf{P}(k+1|k)$$

Defined Ψ and Φ as the weights of the overall trajectory and of the last predicted state respectively, we can write the cost function:

$$J(\mathbf{P}(k), \mathbf{p_G}(k)) = \sum_{i=1}^{H_p-1} \Psi \left\|\mathbf{P}(k+i|k) - \mathbf{p_G}(k)\right\|^2 + \Phi \left\|\mathbf{P}(k+Hp-1|k) - \mathbf{p_G}(k)\right\|^2$$

The optimization problem has still the form:

$$\begin{aligned} \mathbf{P}^*(k) = \min_{\mathbf{P}(k)} \quad & J(\mathbf{P}(k), \mathbf{p_G}(k|k)) \\ \text{subject to} \quad & \mathbf{p}(k+1) = f(\mathbf{p}(k), \mathbf{p_G}(k)) \\ & \mathbf{P}(k) \in \mathbb{P} \\ & \mathbf{p}(k+i|k) \notin \mathbb{S}, \forall i = 0, \ldots, H_p \end{aligned}$$

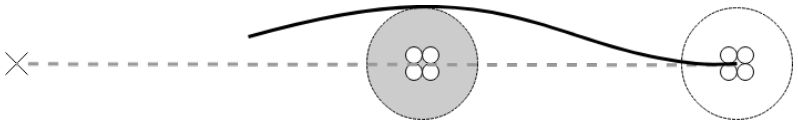


Figure 4.12: Sim Optimizing Traiettoria

4.2.6 Maximizing

In the tuning phase of the cost function, ie when we have quantitatively evaluated the weights to be given to the different terms that compose the function itself, it has emerged as $\|P - P_G\| \in [0; \inf)$. The absence of an upper limit for this quantity implies that the result of the optimization process, and therefore the behavior assumed by the vehicle, is a function of the distance from the waypoint. However, the waypoint position is decided by the path planner and is beyond our control. We have therefore looked for ways to obtain costs in closed intervals.

The first solution was to relate this distance not directly to the goal, but to a point that it was the projection of the goal itself on a circumference of a certain radius. We called the projected point compass and explained how it works in the section 5.4.2.
Another solution was identified, and although it yielded promising results, it required further analysis and experimentation to be included in the final implementation of this work. It will therefore only be presented in this section as a theoretical approach. Defined:

$$dist_1 = \|\mathbf{p_G} - \mathbf{p_i}\|$$

$$dist_2 = \|\mathbf{p_G} - \mathbf{p_{i+1}}\|$$

$$reduction = dist_1 - dist_2;$$

It turns out that $reduction \in [-\rho; \rho]$, where ρ is the maximum distance between two consecutive poses. In this way it is possible to determine a more precise metric for the costs to be assigned and make the behavior of the optimizer independent of the distance from the next waypoint.
The problem is then overturned by the one of minimizing a distance to the one of maximizing the approach to the goal. which should prevent some unexpected behaviors encountered, such as the tendency to deviate slightly behind the reference trajectory. We expect that this will prevent some unexpected behaviors encountered, such as the tendency to deviate slightly late from the reference trajectory.
The cost function is thus formulated:

$$J(\mathbf{P}(k), \mathbf{p_G}(k)) = \sum_{i=1}^{H_p-1} \left[\|\mathbf{P}(k+i|k) - \mathbf{p_G}(k)\|^2 - \|\mathbf{P}(k+i+1|k) - \mathbf{p_G}(k)\|^2\right]$$

The optimization problem has the form:

$$\begin{aligned} \mathbf{P}^*(k) = \max_{\mathbf{P}(k)} \quad & J(\mathbf{P}(k), \mathbf{p_G}(k|k)) \\ \text{subject to} \quad & \mathbf{p}(k+1) = f(\mathbf{p}(k), \mathbf{p_G}(k)) \\ & \mathbf{P}(k) \in \mathbb{P} \\ & \mathbf{p}(k+i|k) \notin \mathbb{S}, \forall i = 0, \ldots, H_p \end{aligned}$$

Chapter 5

Implementation

Our work can be broadly classified into two parts: one theoretical, the other practical.
The analysis of the context, of literature and of the state of the art were clearly the first steps of our work. After that we started developing a theoretical and qualitative approach to the problem into account.
But in the course of this development we certainly could not exempt ourselves from considering an essential as crucial part of our work: the implementation. So we decided working in parallel on the two parts, clearly maintaining when possible the theoretical part some steps ahead.
In this way we managed to study and then to propose something that was both original and concretely achievable with the time and tools at our disposal. Also the implementation, as the rest of the work, was conducted cooperatively by all the team's members and it was an opportunity for personal and professional growth.
Based on what we saw in chapter 2, after a short time from the start of the project it immediately seemed clear to us how development should have started from an instrument, completely unknown to us, which turned out to be as powerful as it was indispensable: ROS.
Starting from scratch, understanding the functioning of ROS and becoming familiar with this tool in order to develop what is necessary, was the first important step in this part of the work, which clearly taken some time.
In the next section we will see ROS more in-depth, a necessary step to understand the functioning of CBUTM, but here we can say that choosing such a framework as a basis for our work has certainly made it more robust, integrable and expandable.

In chapter 3 and 4 we saw the main components of CBUTM and in figure 3.1 how these components link together logically.
Here we will show how the blocks were implemented singularly and how they are interconnected in practice, following the real flow of data that are required, manipulated and exchanges between these blocks.

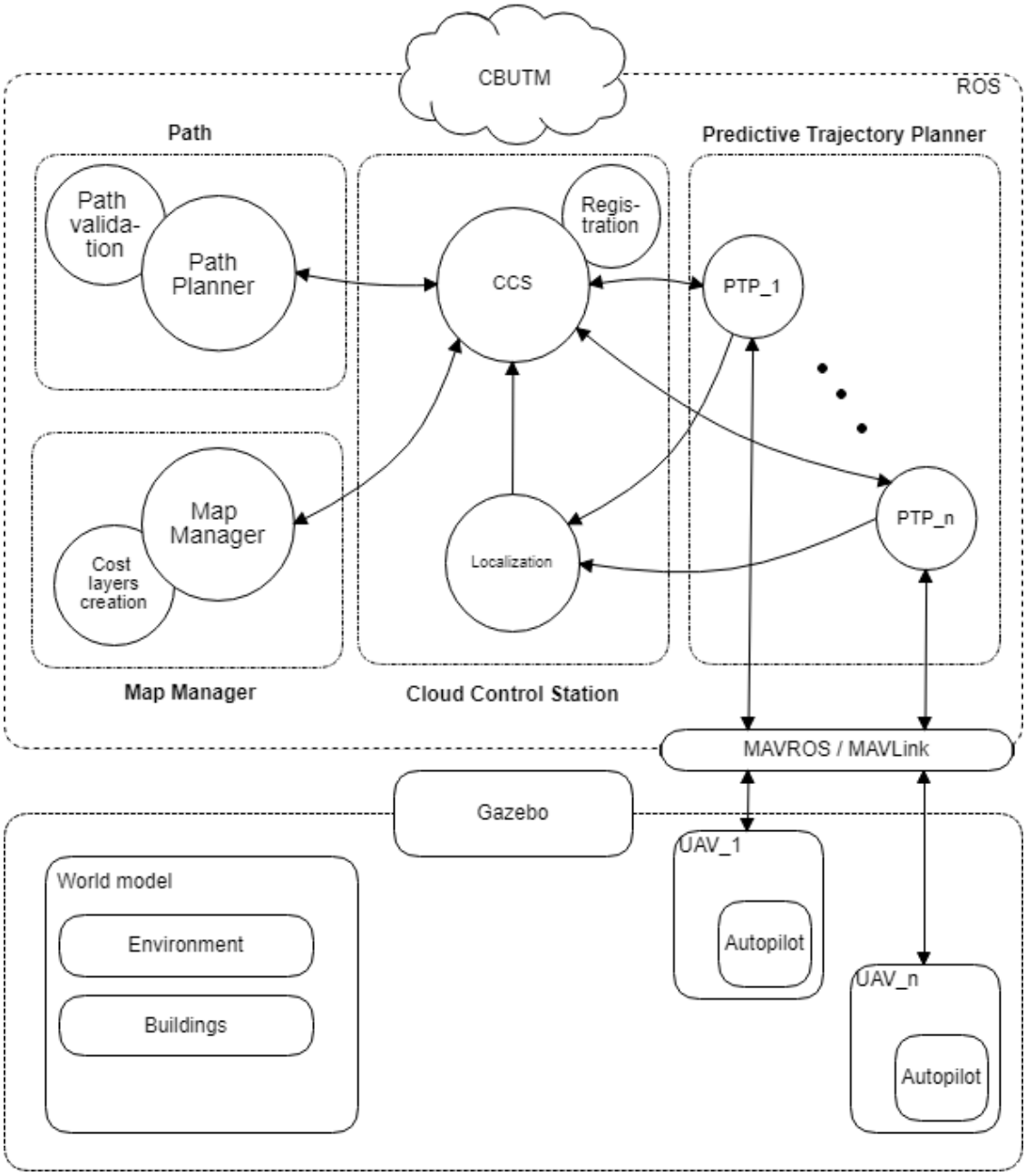


Figure 5.1: CBUTM topology

5.1 Software Environment

5.1.1 Robot Operative System (ROS)

We briefly introduced ROS in 2.2.3 with its official description. Here we are going more in depth. As previously stated, understanding ROS's main elements is a key

step since on them and with them CBUTM works.
The reasons why ROS changed the robotics panorama, and in just few years is widely adopted by academia and industry, are that it provides a distributed architecture and contains state of the art algorithms, implemented and maintained by a large community.
The main concept of ROS we are going to use are [12],[36]:

NODES are independent executables and the respective processes able to communicate with other processes using topics, services, or the parameter server. Using nodes is a way to separate the code and the functionalities providing a robuster and simpler system. A node can be written using libraries of Python and of C++.

TOPICS Topics are buses used by nodes to transmit data in unidirectional way. Using topics the production and consumption of data are decoupled.Each topic, in fact, has a publisher and can have an undefined number of subscriber. A topic is strictly characterized by the ROS type of the message they transmit. Once a node publish a topic with a certain type, other node can only subscribe it as it is and as long as they have the same message type. Actually the communication protocols are based on TCP/IP and UDP.

SERVICES When a communication can not be unidiretional, i.e. a reply is needed, you have to use services. It is a request/reply messaging model. With services, that are defined by the user (no default services exists) nodes can expose functionalities to other nodes or exploits that provided by others.

PARAMETERS are another mechanism in ROS to get information to nodes. Their use requires the presence of a central entity, called Parameter Server, which keeps track of a collection of values. Nodes can can query the Parameter Server if they are interested in the value of any parameter. This communication method is more suitable for information that will not vary to much over time. We make use of parameters, for example, to make more instance of a same node running in parallel from the others.

The working principle of the Robotic Operating System itself, with different instances, namely nodes, running independently and communicating by means of topics and services, seems to be perfect for describing how the CBUTM system works. It can be seen, in fact, as the cooperation of 4 main nodes, each one main subject of the work of some colleague within the Joint Open Lab, but in the whole result of the cooperation of each. We also want to underline the importance of the open source nature of ROS: this encourages collaborative work and software development, making it simpler or even feasible within a global horizon.

5.1.2 Gazebo

Gazebo is another open source project aimed at providing the tools needed to simulate a population of drones in any environment, outdoors or indoors. Using an XML-based syntax, it is possible to create accurate, scalable and self-descriptive descriptions of the environment (world files) and the robot (model files). A Gazebo component, called Gazebo Server, parses these files and simulates the desired environment using a powerful physical engine. Thanks to another component, called Gazebo Graphical Client, it is possible to interface to the server to visualize the simulation itself through 3D rendering. The features of Gazebo can also be extended through the use of plugins. The community itself develops and releases plugins, like the one that simulates a GPS sensor used in our project. Thanks to its characteristics, Gazebo can be used alongside ROS for the development of robotics applications. More details on the simulation environment will be provided in the appropriate chapter.

5.2 Cloud Control Station

How we seen in previous chapter, the Cloud Control Station node is meant to be the central entity of CBUTM, in charge of collecting relevant data and linking together different components of CBUTM itself by managing the communication steams within the system.
In particular it is responsible for what we defined before as the Registration and Monitoring problems.
If a user want to accede CBUTM, it has to call the 'registration service'. To do this it has to use a proper client, using the registration.srv message shown in 5.1.

Listing 5.1: registration.srv message type definition definition

```
id_message uav_id
---
bool registration_status
float priority_level
bool response
```

Using this custom defined ROS message, the user can transmits its identity and receives a priority level and the confirm it is among the active agents of the Cloud system.
Identity is a type of information defined in another custom message, which is used to collect different types of information about the vehicle itself. Without going into the details of all the parameters defined in the message called **id_message** we can classify them in macro areas to understand their role:

UAS parameters as the ID, the kind of vehicle, the weight, the mean time between failures, the nominal speed, etc

Mission parameters as start position, the goal position, the global path, the local predicted trajectory, the mission type, the odometry of the flight, the maximum acceptable risk, etc

service parameters as the registration status, the computed priority level, the authorization to fly etc

It is important to underline how the point here is the functioning principle and not the exact information exchanged, which can be easily expanded according to the network requirement and the available information.
The priority level, for example, is be implemented in a static manner and it assigned to the agent in the registration phase. Thanks to the underlying architecture it would be easy to update this information dynamically, according to some logic or particular events.
Besides the registration, CSS is responsible for collecting and distributing relevant data among components of the network. To do this each agent is inserted in a quick access database alongside with the collected data. In practice, objects of class map and set are used. These classes allow to define associative containers that store elements elements formed, as usual, by:

key used to uniquely identify the element

value that store the information value relative to the element

The container contains elements sorted following a specific ordering criteria, defined by an internal comparison object. Furthermore, the container uses an allocator object to dynamically handle its storage needs, assuring an high rate access also with a growing number of elements contained. In this way elements are inserted according to a fixed order and, although this method is slower to access individual elements by their keys than that with unordered_set containers, it results suitable for direct iteration on subset based on that order. This latter characteristics result to be useful when a UAS need to obtain all the position of higher priority agents.
The insertion of an agent in the list of active agents, involves a direct link between the CSS and the agent itself, aimed to make the CSS monitor the connection status of the agent and to check whenever its relative process is terminated, both intentionally or due to a crash.
This behavior is obtained by mean of a specific topic, on which the agent has to transmit to CSS a periodic signal containing its id reference, and it is what is called an heartbeat.

Registration Service
CHECK_ID( )
COMPUTE_PRIORITY_LEVEL( )
CREATE_LAYER( )
START_HEARTBEAT_CHECK( )

Every time an heartbeat is received by CSS, it restarts a timer. The timer, set at 5 seconds, determine the amount of time the CSS will be waiting for a signal from the agents, past which a timeout callback is ran, advertising for an expected malfunctioning and updating the list of agents active in the airspace.
This dynamic mechanism of insertion-removal UAS from the list of operative agents, allows CSS to perform an efficient **tracking system**.
In this way CSS is not only aware of the active drones and their respective positions, but one of its functions is precisely that of verifying, from time to time, the proximity (in a certain radius) of two or more agents. In this case the collision avoidance procedure described below is started, simply by entering the ID of the highest priority drones in the reference topics for each lower priority drone. Once the IDs of interest are known, each agent will be able to reconstruct the predicted positions and trajectories of the vehicles corresponding to these IDs, simply by querying the database maintained by CSS.

5.3 Map Management

5.3.1 Grid Map

Grid Map is officially defined as *"a C++ library with ROS interface to manage two-dimensional grid maps with multiple data layers"* [2]. It is particularly useful in mobile robotics and in all those applications that require a supporting map framework. Usually a map is a region of space divided into cells, for each of which you can assign a value. The meaning of this value depends on the use that must be made of the map. Frequently, in the case of navigation for the mobile robotics, these cells are assigned a value that indicates the presence or absence of an obstacle in the position that that cell represents. In this case we talk about occupancy grid. Using Grid Map it is instead possible to define an arbitrary number of information for each cell of the map. The peculiarity of Grid Map lies, in fact, in the possibility of defining an arbitrary number of layers, all referred to the same region of space, as depicted in figure 5.2. The layer mechanism is a way to add different types of information about a given location on the map, and in particular one type of information for each layer of the Grid Map.
In addition to the structure based on the layers, Grid Map provides a series of libraries that allow you to efficiently perform operations of storage, manipulation and acquisition of data from the maps. Cell values can be modified at run-time if necessary, making the map dynamic.
For what concerns this project, for example, the values contained in the cells of the different layers are the inputs for the risk analysis seen in section 3.3. In particular, the layer structure allows us to assign each of the parameters that characterize the risk procedure assessment to a different layer. The results of this procedure

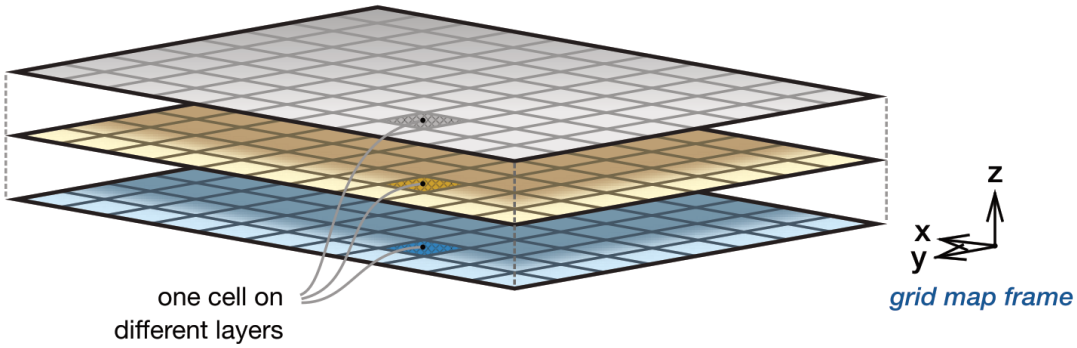


Figure 5.2: grid-map

constitute an additional layer, in which each cell contains the cost in terms of risk for a UAS that should fly over the area the cell itself represents. In turn, this cost layer will be used as input from the path planner to find the best trajectory.

5.3.2 Risk-Aware Map Manager

We have said how each layer of the Grid Map, in this project, contains the information related to the parameters that characterize the risk analysis. Now we can distinguish between two groups of layers.

Environment-related layers

A first node, devoted to the creation of this type of layer, is called "Create Map Service" (CMS). This node is provided with a 3D description of the environment in which it is desired to operate unmanned missions. When it is triggered, CMS makes a cut of this 3D model at a given height h, returning a planar map on which two different functions are applied:

CreateNoFlyZone returns the occupancy map relative to the buildings higher than h, which then will constitute the layer of the no-fly zones illustrated above and necessary in this project to prevent collisions between vehicles and buildings.

CreateCoveredArea returns an object gridmap containing all and only the structures with a height inferior to h. The usefulness of this object is related to the evaluation of the sheltering factor. Consider, for example, the fact that it is certainly safer to fly over a building than over a street.

CMS is invoked by another node, called "Risk_Map_Node" (RMN), which is the

core of the map management block. Its purpose is the management and manipulation of the various layers, as well as the creation of further layers, such as the one related to risk. As already mentioned, each layer relates to a particular information that is of interest for the risk assessment procedure. Clearly the number and type of layers present may vary depending on the information used in the procedure itself. At the moment the following layers are created and managed by the Map Manager:

- no_fly_zone
- covered_area
- pop_density
- sheltering_factor
- signal_layer
- risk_layer
- cost_layer

While the first 4 layers are created each time the RMN node is launched, the other 3 are related to the individual vehicle, and are created during the registration process of the vehicle itself.

UAS-related layers

Three different services are defined within RMN in order to provide the respective layers related to each UAS. In particular, when an unmanned system requires registration, the Cloud Control Station node calls these functions sequentially in order to build the signal, risk and cost layers. The first layer is the one related to the signal, and is the simplest of the three because no particular calculation has to be made. It basically translates a map of the 5G signal coverage, obtained from the service providers, into a layer for the grid map according to the on-board antenna capabilities.
The second layer is the one related to the risk, whose value is calculated in accordance with what is seen in chapter 3. The cost layer must necessarily be built last, as it aggregates those related to the signal and the risk that therefore must have been already created.

5.4 Trajectory Following and Collision Avoidance

5.4.1 The Solver: NLopt

As expected using a receding horizon control strategy, we deal with optimization problems. This led to the need of a proper solver which can be used to obtain the optimal control input according to a proper cost function. Without implementing ourself a solver, we decided to use NLopt.
NLopt is an open-source library for nonlinear optimization, callable from a variety of programming language among which C++, which provides a common interface for a number of different optimization routines [22]. The common interface consists in a standard way to define all aspects relative to an optimization problem, in particular:

An objective function by mean of a function/method written in the used programming language and which return a valuable output (i.e a double);

Bound constraints by mean of two consistent arrays, one containing the upper bounds of the values the variables can assume, the other the lower bounds;

Non linear contraints by mean of a proper function/method which allows to determine if the constraint, written ad $f(\cdot) \leq 0$ is satisfied or not, regardless of the degree of $f(\cdot)$;

Stopping criteria which can be a maximum number of iteration performed, a certain value of the cost function, when an optimization step changes the objective function value by less that fixed or relative threshold or when the computational time exceeds a predefined value;

Global and local optimization NLopt contains implementation of a variety of well know optimization algorithms. They can be also broadly classified into global and local algorithm. Global refers to finding optimal value of a certain cost function among all possible solution, local optimization instead finds the optimal value within the neighboring set of candidate solution [46].

5.4.2 Predictive Trajectory Planner

In section 4.2 we depicted the architecture and the functioning principles of the trajectory planner we designed and whose current implementation is illustrated below. The ROS node relative to this functionality is called Predictive Trajectory Planner (PTP), and it's topology is depicted in figure 5.3.

As discussed in sections 4.1 and 4.2, we decided to address the trajectory planning problem in a distributed manner. Remember how the CBUTM architecture

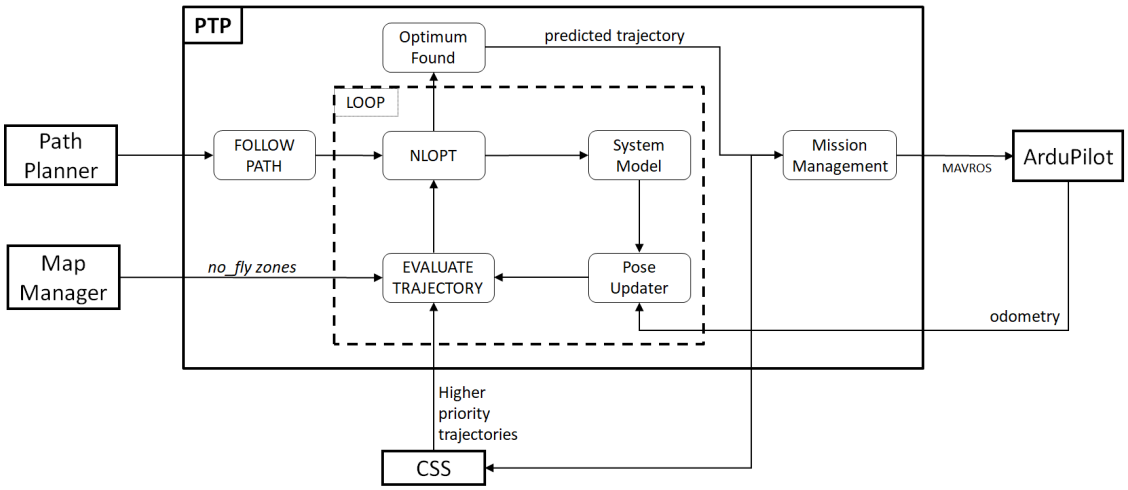


Figure 5.3: PTP functioning

actually require only an autopilot (i.e. Ardupilot) onboard, so every kind of computation is carried out by the Cloud itself.

This imply that we must have a different instances of the same node running in parallel in the Cloud, and in particular one instance for every active agent in the airspace. In other words, every active agent has its virtual image in the cloud, aimed to performed all the needed computations for its navigation. In ROS this set up can be achieved using parameters and roslaunch.

The role of the optimization problem was properly described in previous chapters, now we will see trough pseudo-code how any instance of PTP actually works. We will begin by first showing a high-level description of the PTP node, then a description of the main functions that compose it.

Being PTP the CBUTM interface for Ardupilot, it is responsible for requesting registration of the vehicle, i.e inserting the vehicle in the list of active agents. This is done exploiting the *registration service* (offered by the node CCS, as seen in section 5.2) and so providing it with a custom message called $is\_message$ containing its identity and other required characteristics of the vehicle. As response the vehicle can be logged in the system or not.
A mission can be planned once the vehicle has been successfully registered, but it cannot start until the PATH VALIDATOR node (PV) communicate the authorization to fly. The authorization can be grant or not by PV, depending on the risk analysis of the vehicle and of the mission it has to perform (vd. section 5.3).
Once the vehicle has received the authorization, then its mission can begin. Control of the mission is now entrusted to the FOLLOW_PATH() function for as long as the mission requires.

Node PTP

Require: *path* from Path Planner
Ensure: Trajectory Following

```
INIT( )
registered ← VEHICLE_REGISTRATION( )
if (registered) then
    path ← PATH PLANNER
    authorized ← PATH VALIDATOR(path)
    if authorized then
        MISSION START( )
        while node is running do
            trajectory ← FOLLOW_PATH()
            SEND_MISSION(trajectory)          ▷ Send trajectory to ArduPilot
            publish(trajectory)               ▷ Communicate the predicted trajectory
        end while
    end if
end if
```

The trajectory predicted are communicated to the CCS to be usable by other PTP instances.

Function 1 FOLLOW_PATH

Require: *path* from Path Planner ▷ Array of waypoints
Require: index i ▷ index of the current waypoint
Ensure: the trajectory the vehicle has to follow

```
position ←vehicle position in meters
goal ← path[i]
distance ← GET_DISTANCE(position, goal)
if distance ≥ goal_tollerance then
    trajectory ← FIND_TRAJECTORY(goal)
else if i < path.size() then
    i ← i + 1
    goal ← path[i]
    trajectory ← FIND_TRAJECTORY(goal)
else
    mission completed
end if
return trajectory
```

Function 1 receives the path of the mission computed by the Path Planner, scanning the path in accordance with the progress of the mission and providing function 2 with the position of the next goal to be reached.

In Function 2 we find the optimization part. The definition of the optimization problem was stated in section 4.2.5, here we implement that problem according to NLopt (vd section 5.4.1).

What we want to achieve is the succession of poses that the drone plans to assume in order to reach the next goal while avoiding obstacles. In order to obtain this poses, we define an array k containing as many couples (ρ, α) as there are steps of prediction. Where:

ρ is the module and determine the distance of the new pose from the previous one.

α is the angle respect to a fixed reference frame

Using such variables, we can easily impose that the evaluation of the predicted trajectory take place at the most every $\rho = 4$ meters, which is approximately the same as evaluating it every second in the flight simulation since we are assuming a constant cruise velocity $v = 4m/s$. Note as with this assumption the time is implicitly considered in the prediction.
The function OPTIMIZE is provided by NLopt, and require (vd. section 5.4.1) an

Function 2 FIND_TRAJECTORY

Ensure: optimal trajectory

$\rho\_ub \leftarrow 4$ ▷ bounds on optimization parameters
$\rho\_lb \leftarrow 0$
$\alpha\_ub \leftarrow 2\pi$
$\alpha\_lb \leftarrow 0$
NLOPT.SET_UPPER_BOUNDS(ρ_ub, α_ub)
NLOPT.SET_LOWER_BOUNDS(ρ_lb, α_lb)
NLOPT.SET_MAX_ITER(6000) ▷ Stopping Criteria
$numStep \leftarrow 10$
define vector $k[numStep]$ ▷ $k[i]$ contains ρ and α
$k \leftarrow$NLOPT.OPTIMIZE(global alg, evaluate_trajectory, k)
$k \leftarrow$NLOPT.OPTIMIZE(local alg, evaluate_trajectory, k)
$ithpose \leftarrow actual\_pose$
for $i \leftarrow 1, numStep$ **do**
    $ithpose \leftarrow ithpose +$ POLAR_TO_CARTESIAN($k[i]$)
    $trajectory[i] \leftarrow ithpose$
end for
return $trajectory$ ▷ optimal trajectory

initialized vector variables (k in this case), which will be fulfilled with the optimized value, an algorithm to perform the computation and a cost function, in this case function 3.
The optimization itself is performed in lines 10 and 11, before with a global algorithm and later with a local algorithm to make it more refined.
Note how, in order to be correctly evaluated by a function that assign cost to geographical points, as function 3 does, this array k (ρ, α) has to be converted into a series of poses expressed in the map reference frame. This is done starting from the actual position of the vehicle, expressed in the map frame yet, and incrementally adding to it the two variables converted into cartesian quantities, how showed in Function 3 (at lines 14 and 15) and graphically in figure 5.4.

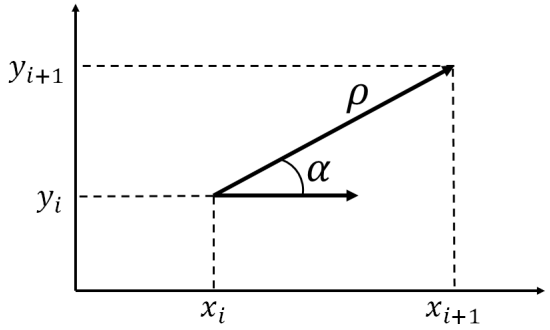


Figure 5.4: Incremental position computation using polar coordinates

At line 16 of function 3, you can find the core of the optimization problem, i.e. the quantity identified in 4.2.5 as the one to be minimized to reach the goal. Note as the goal position is replaced by another position, called *compass*.
Compass is the goal itself if it is distant less than 40 meters from the pose to be evaluated; otherwise is the projection of the goal on a circle of radius 40 meters centered in the pose $(i-1)$-th, as expressed in function 4.
In this way we solved a problem found tuning the weight of the cost function: $goal\_distance$ had a lower bound, zero, but the upper bound depended from the path planner and so was unknown. This problem had already been dealt with in section 4.2, where there is also an alternative solution. However it is not acceptable that the behavior of the system in the presence of an obstacle is conditioned by the distance between the waypoints provided by the path planner, so we introduced $compass$ and results $goal\_distance \in [0; 44]$, allowing us to make different considerations about the weight to assign to it.

Note also that the constraint identified in section 4.2.5 require the respect of a safety distance between two agents. This translates into a circular area of radius $safety\_distance = 10m$ built around the position of the obstacle, which are localized by the system on the map and so in cartesian coordinates.

Function 3 EVALUATE_TRAJECTORY

Require: k vector of variables ▷ output of solver
Require: $actual\_position$ ▷ received from mavros
Ensure: the trajectory cost of the trajectory
$ithpose \leftarrow actual\_position$
$numStep \leftarrow 10$
$Q \leftarrow 20$ ▷ weights
$F \leftarrow 120$
$S \leftarrow 100$
$M \leftarrow 100$
for $i \leftarrow 0, numStep$ **do**
if $distanceToGoal > 40$ **then**
$compass \leftarrow$ COMPUTE_COMPASS(ithpose,goal)
else
$compass \leftarrow goal$
end if
$(\rho; \alpha) \leftarrow k[i]$
$ithpose.x \leftarrow ithpose.x + \rho\cos(\alpha)$
$ithpose.y \leftarrow ithpose.y + \rho\sin(\alpha)$
$goal\_distance \leftarrow$ GET_DISTANCE$(ithpose, compass)$
for each drone with higher priority d **do**
$d\_traj \leftarrow$ its trajectory from CCS ▷ vector of waypoint
$safety\_cost \leftarrow$ ASSIGN_SAFETY_COST(ithpose,d_traj)
end for
$map\_cost \leftarrow map.no\_fly\_zone[ithpose]$
if i = numStep **then**
$last\_dist \leftarrow distance$
end if
$cost \leftarrow cost + Q\left\|goal\_distance\right\|^2 + F\left\|last\_dist\right\|^2 + S\left\|safety\_cost\right\|^2 + M\left\|map\_cost\right\|^2$
end for
return $cost$

Function 4 COMPUTE_COMPASS

Require: *ithpose* and *goal* positions
    $distance \leftarrow$ GET_DISTANCE(ithpose,goal)
    if $distance > 40$ **then**
        $dx \leftarrow goal.x - ithpose.x$
        $dy \leftarrow goal.y - ithpose.y$
        $angle \leftarrow$ ATAN2(dy,dx)
        $compass.x \leftarrow ithpose.x + 40\cos(angle)$
        $compass.y \leftarrow ithpose.y + 40\sin(angle)$
    else
        $compass \leftarrow goal$
    end if
    return *compass*

So we decided to traduce these constraints into the cost function in form of huge costs, as it was the simpler way to implement a circular constraint into a cartesian reference frame.
The presence of fixed obstacles in a certain cell off the map is read from the map the layer no_fly_zone (as seen in section 3.3), which eventually return a $maxCost$. So each possible pose is evaluate according to:

- the distance with the goal (compass)
- the fact that is or not the last pose of prediction.
- the proximity to vehicles with higher priority
- the presence of no_fly_zones

The proximity to vehicles with higher priority is taken into account by function 5. Currently, for purely implementation reasons, the check over the reciprocal distances of vehicles is statically performed inside this node for a limited number of vehicles. However, according to what saw in the previous sections, the implemented monitoring methodologies make it easy to overcome this implementation limitation. As you can see, in order to model the uncertainty on the execution time of the mission, the proximity of an i-th pose is compared not only with the i-th poses of the higher priority vehicles, but also with their $i-1$-th and $i+1$-th pose.

Function 5 ASSIGN_SAFETY_COST

Require: the pose to be evaluated $ithpose$
Require: the trajectory of a neighbor agent (d) with higher priority: d_traj
Ensure: the safety cost of $ithpose$ relative to the presence of d
    $dist\_i \leftarrow \text{DISTANCE}(ithpose, d\_traj[i])$
    $dist\_ip1 \leftarrow \text{DISTANCE}(ithpose, d\_traj[i+1])$
    $dist\_im1 \leftarrow \text{DISTANCE}(ithpose, d\_traj[i-1])$
    $dist \leftarrow \text{MIN}(dist\_i, dist\_im1, dist\_ip1)$
    if $dist \leq strictly\_forbidden\_dist$ **then**
        $safety\_cost \leftarrow maxSafetyCost$
    else if $dist \leq safety\_area$ **then**
        $safety\_cost \leftarrow maxSafetyCost/sqrt(dist)$
    end if
    return $safety\_cost$

From simulations emerged how, in particularly congested situations, is better if the cost of proximity is not a constant with a huge value but rather a value that, after a certain radius (named $strictly\_forbidden\_dist$ and set equal to 5 meters), decrease with the distance from the obstacle itself. Otherwise for the optimizer it

would be equally "expensive" go through few centimeters away from the obstacle rather than several meters, and that ,in particular circumstances, could lead to a dangerous scenario.
The function we finally use to link the distance (from the obstacle) to the corresponding cost is plotted in figure 5.5.

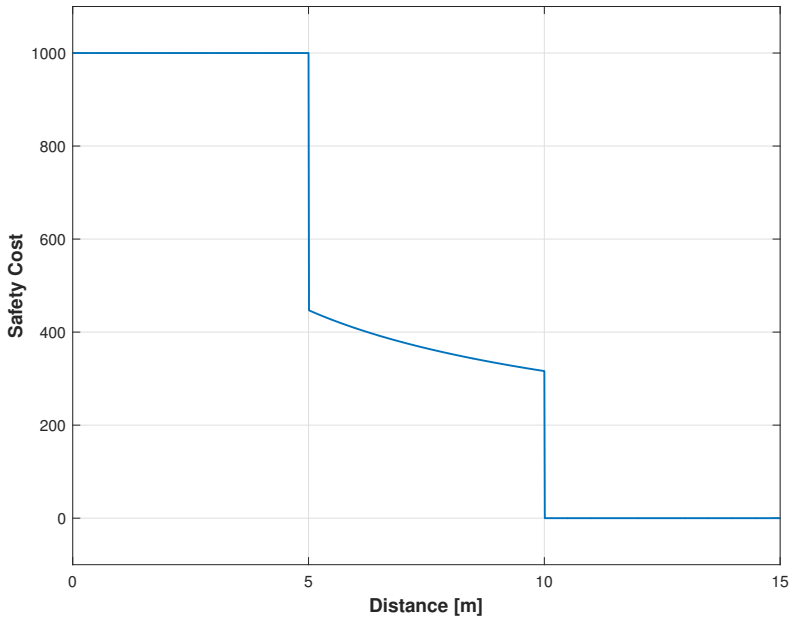


Figure 5.5: SafetyCost values

Chapter 6

Results and Simulations

6.1 Simulation Environment

In this chapter, a complete simulation of a typical Traffic Management scenario will be provided: we will show how the processes described in previous chapters really works, considering all components of the Cloud-Based Traffic Manager, from the registration to the in-flight control.
Before starting the simulation, let's have a quick presentation of the main tools we used in it.
Simulation, testing and debugging have been performed in Gazebo 7, presented in section 5.1.2, running on a ROS distribution 5.1.1.
Our needing was to have a simulation environment, in which vehicles as much as some on-board sensors (like the gps transmitter) were simulated, and could send and receive messages and commands through the MAVLink protocol: this one is a communication protocol, very common for Micro Aerial Vehicles and already presents in most of the ones actually commercialized.
For this purpose, UCTF [1] turned out to be very useful. It is actually nothing more than a game developed in the ROS environment, in which swarms of drones are flown in an Unmanned Capture The Flag (UCTF, precisely) match, that can be played both in the real world and in the simulator. It consists of a complex system in which the simulator, the autopilot software and a mission management interface are linked together, built up as provided by 3D Robotics in a software-in-the-loop (SITL) environment.

Starting from the provided configurations, some modification have been applied in the JOL, in order to adapt it to our case. From figure 6.1, it can be seen that a

[1] https://github.com/osrf/uctf

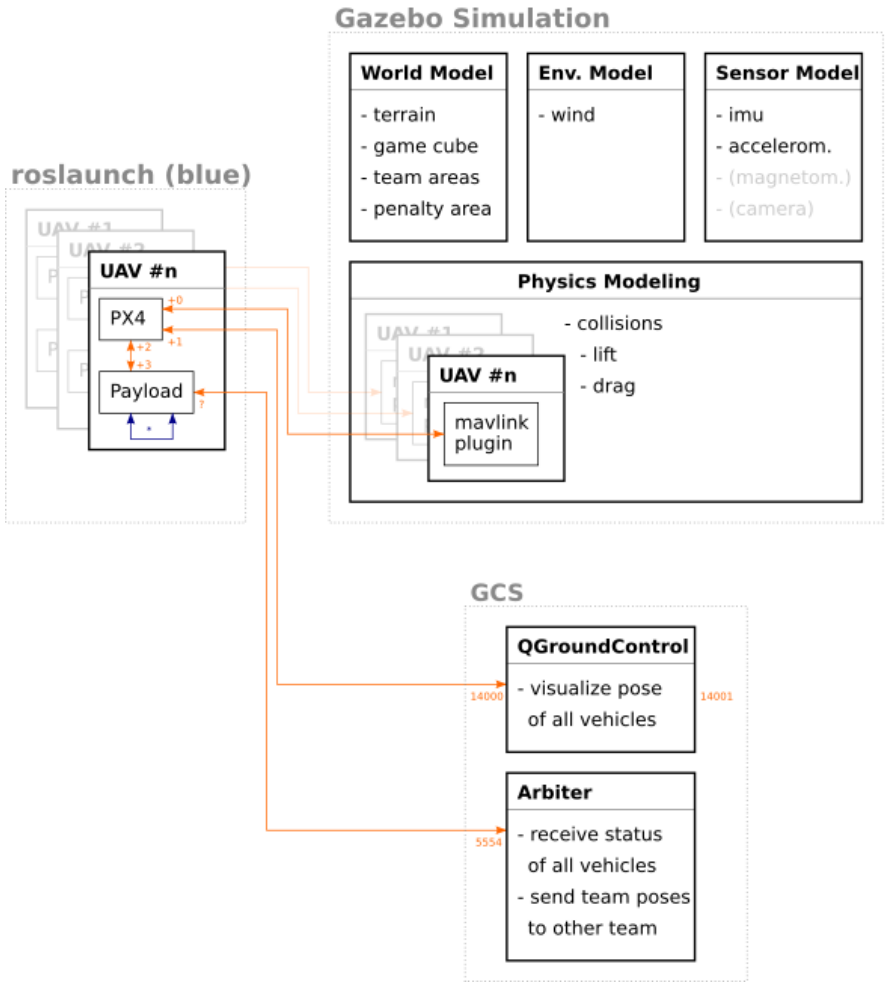


Figure 6.1: Overview of the uctf system architecture

Gazebo simulation environment is set up with certain world and environment models. In that, the original scenario configuration has been removed, and replaced with informations about buildings (position, occupancy and height) coming from the OpenStreetMap Foundation, that provides open-source license .osm files containing these kind of informations.
Drones have been inserted by means of their Unified Robot Description Format (URDF) model, that properly describes their dynamics. This is what has been done from UCTF developers, that used a 3DR Iris quadcopter as simulated drone, together with a scaled down Cessna fixed wing, that has not been used in our work, since for technical characteristics and flight dynamics, does not fit with the purposes of this project. Both these vehicles are equipped with a autopilot software, that, managing all the technical aspects of the flight, allows high-level interaction with the drone. It could be chosen between a PX4 or an ArduPilot-copter, two

of the most common autopilot software in commerce: we decided for the latter for convenience. It is a full-featured, open-source multicopter UAV controller capable of a wide range of flight requirements, which can be programmed through a number of compatible software ground stations. It uses the MAVLink communication protocol, that is used for transmitting commands and informations between vehicles and the control station.
At this point, a Gazebo simulation environment is set up, with certain world and environment models: Gazebo node then, through its plugins, simulates the behavior of imu and GPS sensors, publishing messages about IMU (Inertial Measurement Unit), GPS position and GPS velocity value on specific topics.
This SITL can be spawned in multiple instances, modelling multiple different copters to exist at once, allowing us to run at least three simulated drones on a reasonably powered laptop.

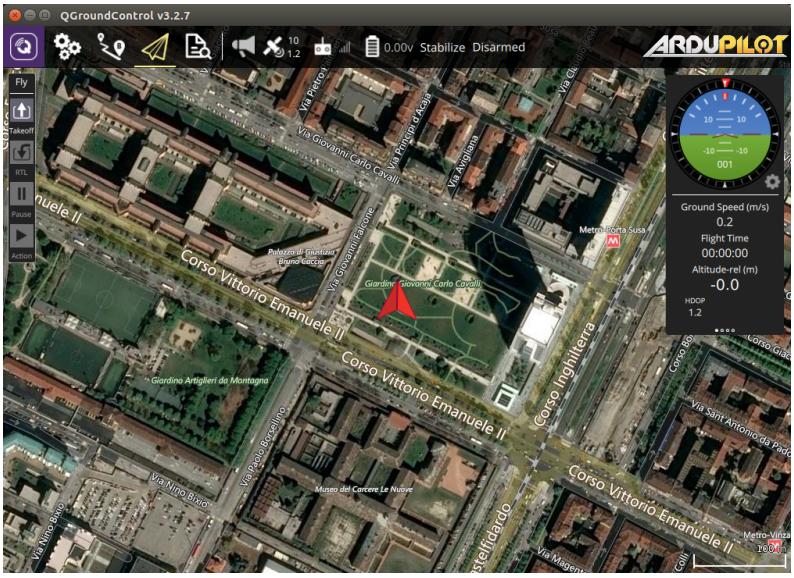


Figure 6.2: QGroundControl graphical user interface

We then used as interface the QGroundControl software (vd. figure 6.2), a powerful Ground Control Station which provides full flight control and mission planning for any MAVLink-enabled PX4- and ArduPilot-powered UAVs. It has been configured according to the Software-In-The-Loop configuration, following the general guidelines, making it read from port 14000, where the simulated drones are communicating via MAVLink using UDP protocol. Its usage has been fundamental for a preliminary understanding of how the autopilot works and how to interact with it, while, once understood the communication's dynamics, it has been put aside, and the system extended to allow communication between our autonomy package and the autopilot, creating proper nodes to send and receive messages and manage

missions.
Last software involved in the simulation environment set-up is Rviz, a 3D visualizer for the Robot Operating System framework. It provides utilities for robotic projects' development and debug phases, allowing to visualize coherently different kind of dynamic quantity defined in ROS (for example the map) and some kind of interaction with it (for example, to indicate the position of the goal by clicking it directly on the map). An example is showed in figure 6.3

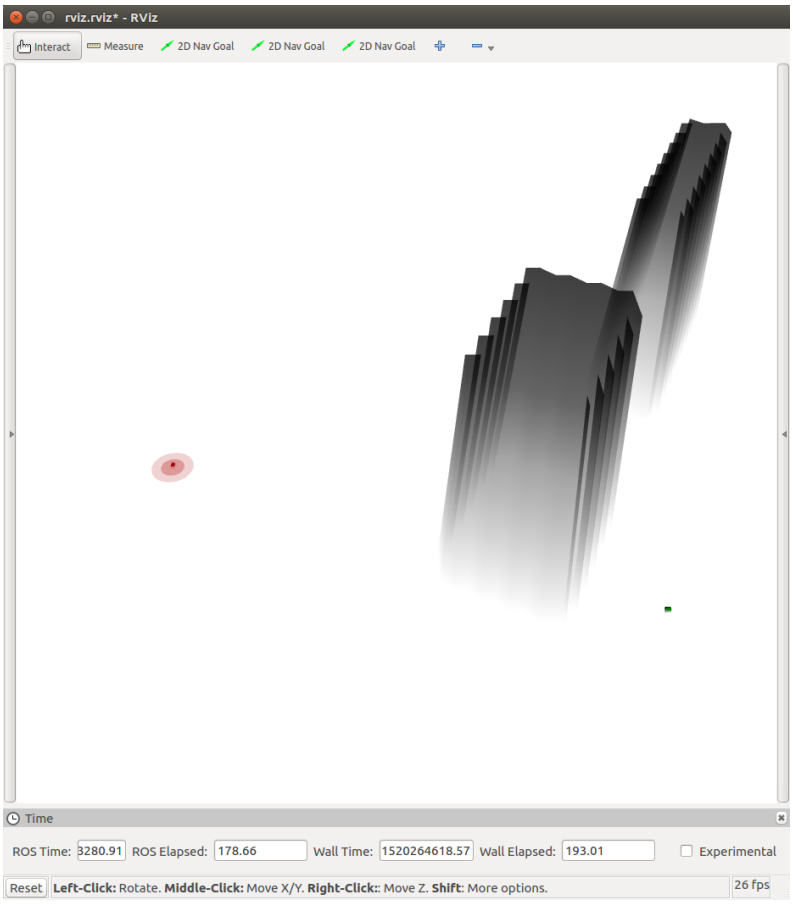


Figure 6.3: Rviz interface. The 3 buttons on top are used to send goals to 3 different UAS

6.2 Distributed architecture

Introducing ROS we said how it is explicitly designed for distributed computing. Here we will see how this possibility can actually be put in place, allowing us at the same time to show a simulative context more coherent with the Cloud infrastructure we designed in this project.

The way in which the processes are managed by means of the nodes, and the methods of communication between them, analyzed in section 5.1.1, make it possible not only to allocate these processes on different calculation units, but it is also possible to relocate the same processes at run- time to match the available resources in the network.
This last possibility is very interesting as it is suitable for a new emerging technological paradigm: **edge computing**.
This concept basically consists in revising the cloud infrastructures considering the changes that the internet of things implies on the way of perceiving the internet itself. In fact, for years now, the internet has ceased to be a virtual and intangible entity. In the age of IoT, we already see it, Internet is not only present and directly affects many of the activities that each performs every day (appliances, the way we communicate or move or prepare food) but ends up generating the knowledge itself, starting from the information that such devices collect to work.
This in some way shifts the "center of gravity" of internet itself: from large data centers thousands of kilometers away, the measurements and the actuations of internet take place and will take place practically everywhere.
Edge computing is exactly the paradigm for which the computation moves close to the source of the data it collects and close to the actions it determines.
This paradigm is perfectly suited to being associated with 5G, which is "a collective name for technologies and methods that would go into the future networks" [2].
Given the importance of cloud services also on mobile networks, the concept of Mobile Edge Computing (MEC) is recognized as one of the key emerging technologies for 5G networks and it is therefore something that we will deal with in the future.
Distributing the computation with ROS is very easy. It is required that each pair of machine have a complete and bi-directional connectivity on all the port then, assuming that we want to distribute the calculation only on two machines, we proceed to configure them as shown below:

```
Machine1:
$ export ROS_HOSTNAME=Machine1
$ export ROS_MASTER_URI=http://Machine1:11311
$ export ROS_IP=machine_1_ip

Machine2:
$ export ROS_HOSTNAME=Machine1
$ export ROS_MASTER_URI=http://machine_1_ip:11311
$ export ROS_IP=machine_2_ip
```

[2] https://sdn.ieee.org/newsletter/march-2016/mobile-edge-computing-an-important-ingredient-of-5g-networks

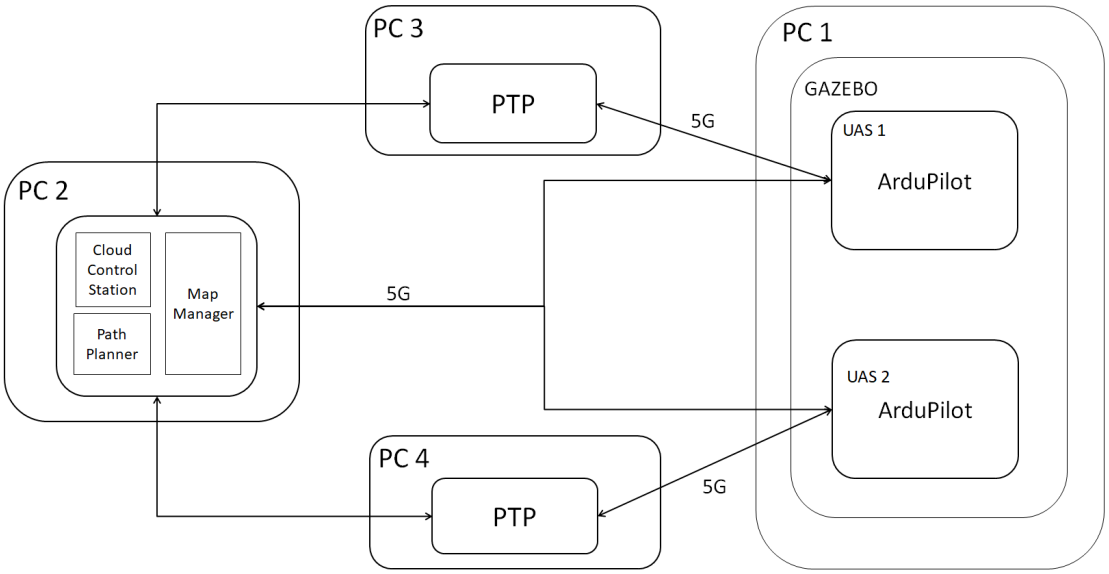


Figure 6.4: Architecture of the distributed simulation environment

Using an architecture like the one in figure 6.4, it is possible to perform simulations more coherent with the real context of application. With such a configuration, in fact, it is possible to experiment the distributed architecture and above all it is possible to introduce a certain latency in the communication between the nodes. Latency was not considered in the course of the project and of the simulations but it is possible to make however some considerations. First we look at the criterion with which we assigned the nodes to the machines. PC 1 simulates the real world: inside we have Gazebo and therefore two simulated drones with ArduPilot on-board. Notice how here we refer to only two UAS, but clearly the discourse can be extended and then actually applied to an arbitrary number of unmanned systems. All connections with PC 1 simulate 5G connections between vehicles and the cloud. The latency specifications of the next generation of mobile network are not yet perfectly known, but we know that they should be in the order of milliseconds. It can be assumed that in general the latency of a network varies according to a normal distribution. Let's see, with the following command, how we can add to PC 1 a normally distributed delay with a mean value of 50ms and a variance of 20ms

```
# tc qdisc change dev wlan0 root netem delay 50ms
      20ms distribution normal
```

These latency values should not affect the functioning of the architecture proposed here. In fact, the frequency of the nodes involved in CBUTM never exceeds 5Hz. PCs 3 and 4, as seen, execute the nodes related to the trajectory following; PC 2 all other CBUTM's nodes.
A consideration of the connections between PC 2 and PCs 3 and 4 can be traced

back to the previous discussion on edge computing. If we assume CBUTM as a centralized entity, PCs 3 and 4 are nothing more than virtual machines within which the nodes are executed. In this case the connection will simulate the connection between the virtual machines, with extremely low latency.
On the other hand, if we assume to apply the paradigm of edge computing, PCs 3 and 4 can be considered as independent units of calculation allocated near the mission area. In this case the latency will depend on the way in which these machines are connected, and we can then use the previous command on these machines too, assuming here also a communication based on 5G.

6.3 Map Manager

6.3.1 Modelling Environment

According to the concepts exposed in previous chapters, the first output that the Risk-Aware Map Manager has to produce its a coherent model of the environment in which the mission takes place. In particular, it has to provide two UAV-independent layers, which will be the same for every drone that will fly (at the same height h, obviously) in this area: "no_fly_zone" and "covered_area". Their role, already explained in chapter 5, is to describe two crucial features of the "world", that for a planar robot (as the drone) is a simple "cut" of the geographical map at the flight's altitude. On one hand, there are all the buildings taller than this value, that for this reasons can be a danger for the Unmanned Aircraft and have to be precisely identified. On the other, all the structures lower than h that can offer a very good sheltering for people on ground, in case of UAV's fall.
De to the high computational cost that such simulation requires, we decided to not consider an entire city, but just a part of it, that can provides some particular features to show the capability of CBUTM. The area we chose is the one contained in the rectangle defined by

$$[minlatitude, maxlatitude] = [45.0645000, 45.0741000]$$

$$[minlongitude, maxlongitude] = [7.6537000, 7.6732000]$$

which is the part of Turin around the San Paolo's skyscraper. In figure 6.5, the view from osmbuildings.com is depicted. Obviously, the choice of this area is not casual, but has some criticisms due to the presence of an over 100 meter building. Starting from the corresponding .osm file, the input of the "Create_Map" service is a point cloud, which can be obtained through specific open source softwares. The result is the one in figure 6.6.

Saving it as .obj file, it is finally the proper input for th modeling environment function. Besides this, also other information are needed to compute the layers: height of flight and map's resolution. In this simulation case, we decided to impose

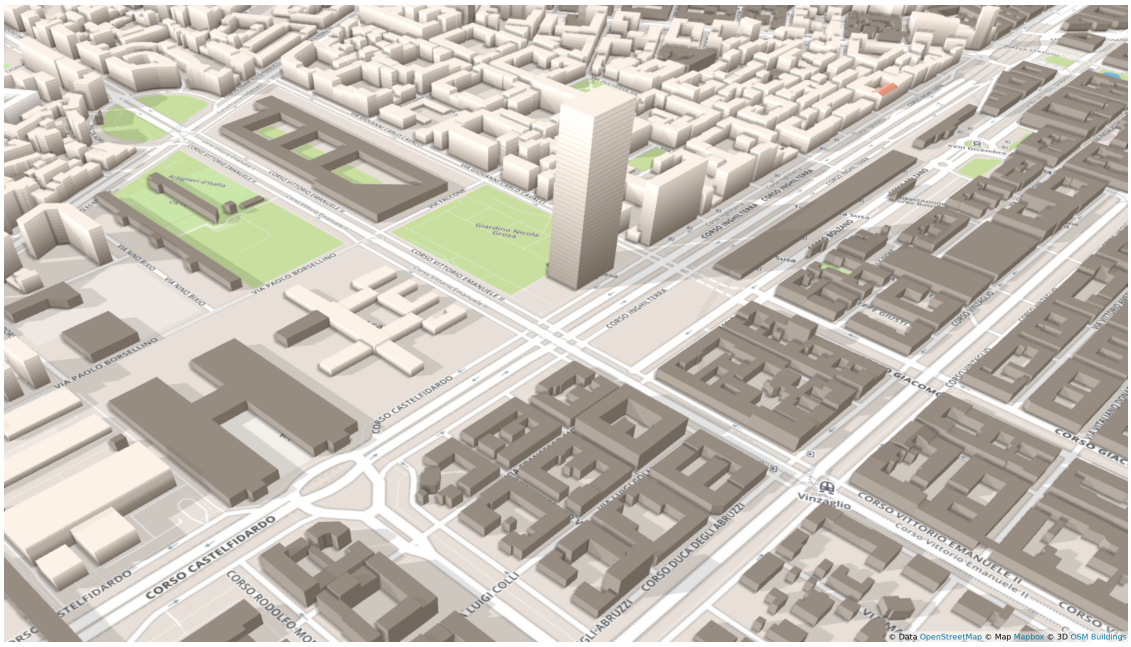


Figure 6.5: Open Street Map Buildings View

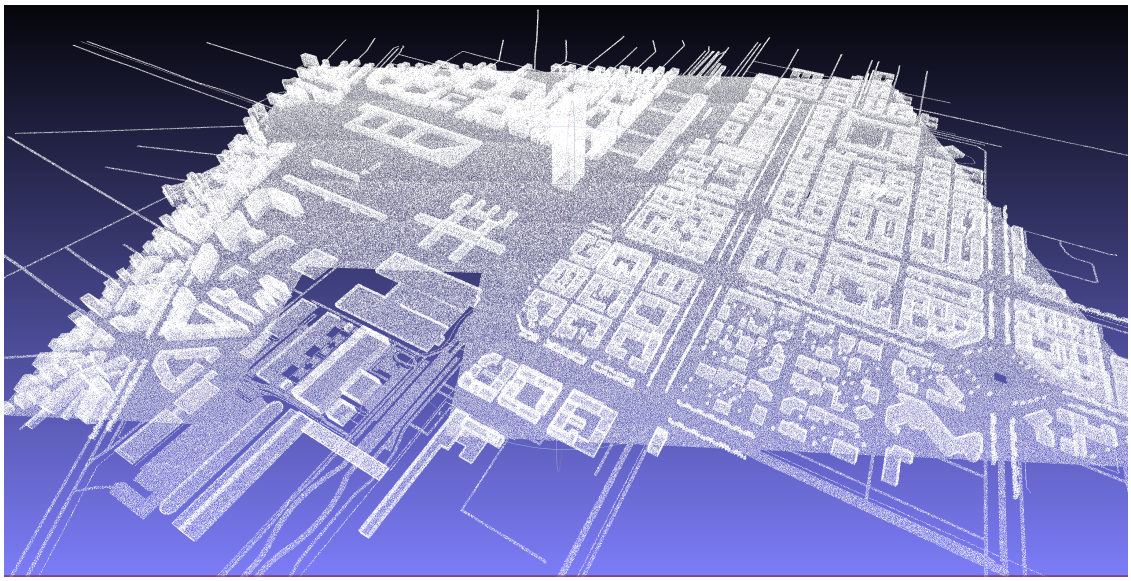


Figure 6.6: Point Cloud Model

$h = 25m$ and a resolution of $25m^2$, which both are in line we real operative situation. Converting the geographical coordinates in pure lengths, we obtain a map of $1535 \times 1070m$, that whit this resolution's value correspond to 307×214 cells. Applying a cut at $h = 25m$ we obtain the first two layers of the map (see figures 6.7 and 6.8). What is interesting of this two layers is their complementarity: merging

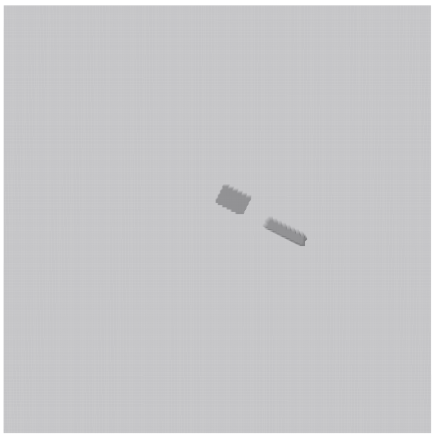


Figure 6.7: No Fly Zone Layer

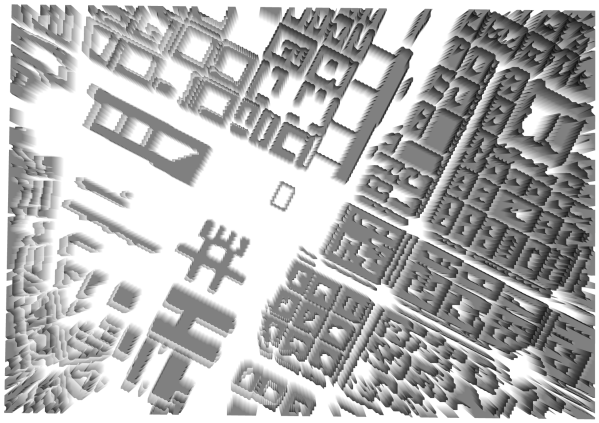


Figure 6.8: Covered Areas Layer

the two figure we obtain the complete representation of the operative environment. In particular, in the first one, only the skyscraper and the building next to it are depicted. As already said in this thesis, it's not very common to find tall palaces in city center. For the same reason, the covered areas layer is very dense, since there are a lot of "small" structures, especially in high densely populated zone as this one.

Two other layers are needed to complete the mapping of the environment: population density and sheltering factor. For what concerns the first, we set a constant value of $0.0073\frac{ab}{m^2}$, that means a flat layer (that for this reason will not be reported here). Finally, the sheltering has been modeled as maximum (10) in the zones corresponding to the covered areas, or a generic mean value (6) in all the other parts. In figure, it has the same shape of figure 6.8, but with different values.
Finally, the environment's model is completed. It is structured in layers and published every 2 seconds on "risk_map_node/risk_map" topic.

6.3.2 UAV-Aware Risk Map

Once the model of the environment is available, is possible to accomplish to the registering procedure advanced by the UAVs. In order to correctly do this, its necessary to merge the information present in the first four layers with the drone's characteristics. As discussed previously in this thesis, three layers are UAV-Aware:

- Signal Layer, which represents the capability of the drone to receive Internet connection, in every area.
- Risk Layer, which represents the risk for the UAV, evaluated with equation 2.6.
- Cost Layer, which merges the first two according to proper weights.

In this simulation, the drones are supposed to have a length equal to 0.7 meters and a weight of 0.5 kilograms.
For what concerns the first layer, it is a generic measure of the Quality of Service of the Internet Connection, and is used just as example to show the potentiality of the cost layer. In any case, during this simulation we assume every drone has on board a 5G compliant antenna, to receive TIM's signal. Furthermore, in order to have some criticism, we supposed to have an optimal signal reception, except for a circular area, as in figure 6.9. This is not strictly true, in particular in urban environment where this kind of infrastructure are very powerful. However, it's useful for a better result of the simulation. According to the metric, the layer's value is always 100 except for the cone, where it goes down till 0.
Beside, also the Risk Layer of the drone can be computed. According to what we said in chapter 3, it follows the rule (for every cell):

if no_fly_zone **then**
  $Risk \leftarrow upperbound$
else
  $Risk \leftarrow (N \times P(f|e) \times \lambda)$
end if

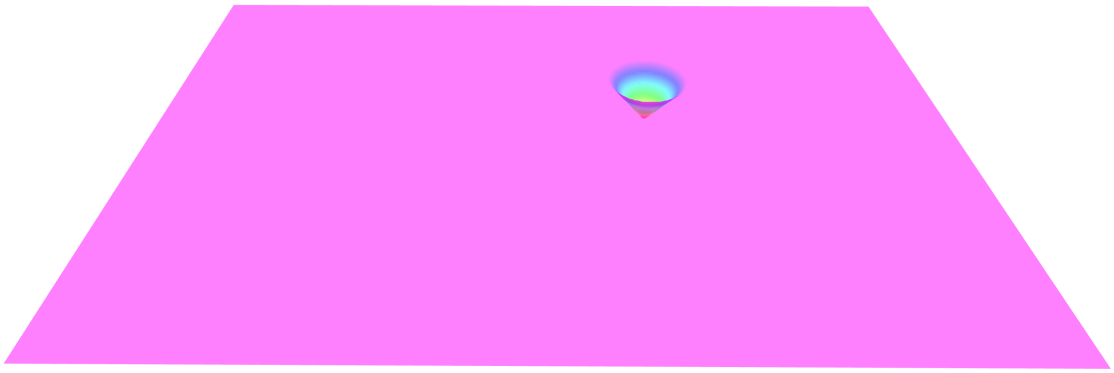


Figure 6.9: Signal QoS Layer

In this case, as for the rest of the work, we impose $upperbound = f_{max} = 10^{-5} \frac{victims}{h}$. This layer is probably the most important one, since the risk assessment is a crucial procedure in our framework. It merges all the information present in the other layers, and the results have to be consistent wit real empirical results. Finally, normalizing the risk in range [0,100], we obtain the risk map in figure 6.10.
Three kind of areas emerges clearly from the figure:

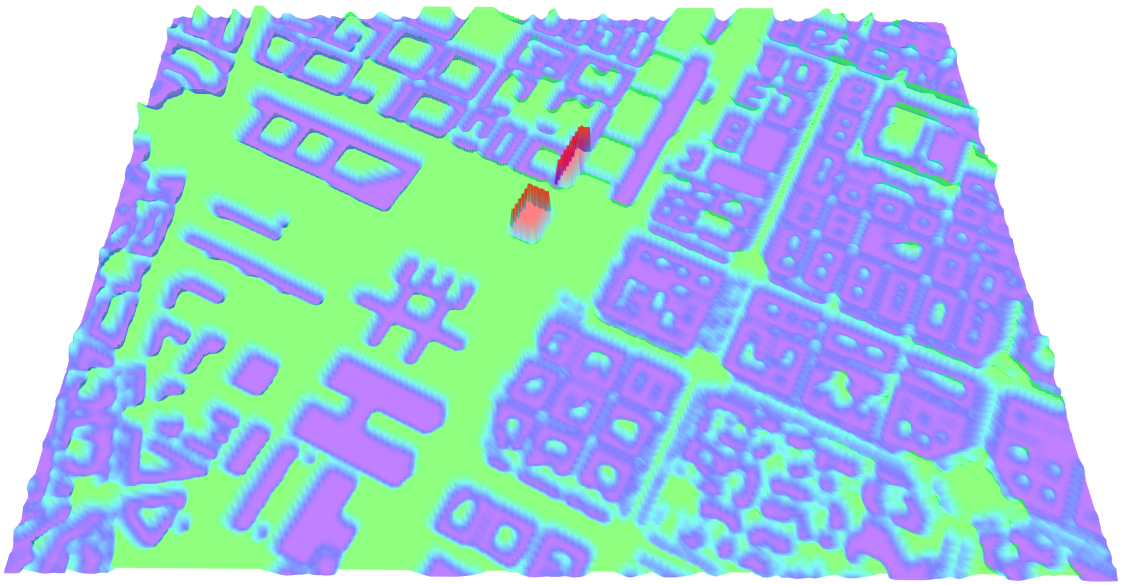


Figure 6.10: Risk Layer

- Areas with the lowest value of sheltering (6), which has a green color in figure. Their risk is equal to 7×10^{-6}, which normalized is 54.

- Areas with the maximum sheltering possible (10). They have a purple color ad the lowest risk $f_F = 5 \times 10^{-6}$, that normalized become 19
- No-Fly Zone, which are red and have the higher risk $f_F = 1 \times 10^{-5}$ that corresponds to 100.

Obviously, many other values are present in the risk maps, but in a rough view they can be summarized in this three categories.
Since both risk and signal layer are in range [0,100], they can be summed in a final layer, called Cost Layer. According to the theoretical concept exposed about it, this last map can be seen as a weighted sum of the other two layers, obtained using coefficients α and $\beta = 1 - \alpha$. In particular, when $\alpha = 1$, only the risk is considered, and the resulting map is the same as in figure 6.10. Instead, when $\alpha = 0$, only signal's QoS affects the cost, and the layer turns out to be the opposite of the one in figure 6.9. An example with $\alpha = 0.5$ is provided in figure 6.11.

In particular, it's possible to notice the no-fly zones and the area of no signal

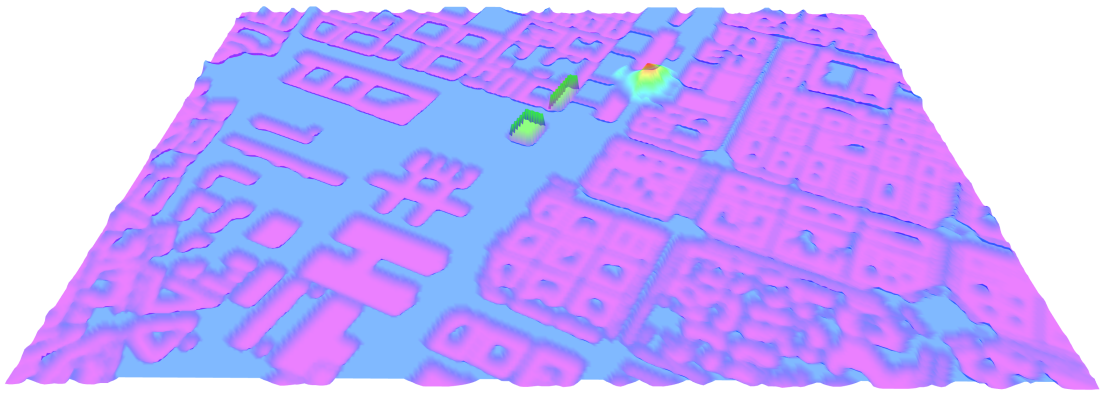


Figure 6.11: Cost Layer

coverage, which in this case is no more an "hole" but instead a cone. This behavior is due to the fact that a low level of signal corresponds to an high cost.
Finally, the Risk-Aware Map Manager ends its work, providing a set of layers common for every drone, and three different layers specific for each aircraft. Upon the last one, the Path Planner will evaluate the better trajectory to perform.

6.4 Path Planning and Validation

The map, understood as a collection of layer, is published on topic "/risk_map_node/risk_map" with a frequency of 0.5 Hz. Every time a new layer

is available, it is simply added above the others. In particular, when the Cost Layer of a drone is completed, it is possible for the CBUTM to start the mission planning phase. In this part, we suppose to have a known start position, and to receive the goal of the mission as Rviz input. Each UAV registered to the cloud traffic manager can provide its own end position, different from the other ones and on its specific layer.
Main aim of the Path Planner is to choose the less costly path, from point A to B, and then to validate it through the Path Validation procedure already described in previous section. Since for now we are not considering collisions with other aircrafts, its possible to show the capabilities of this two algorithms just on a single drone, since it will have the same behavior on the others. In figure 6.12 a typical

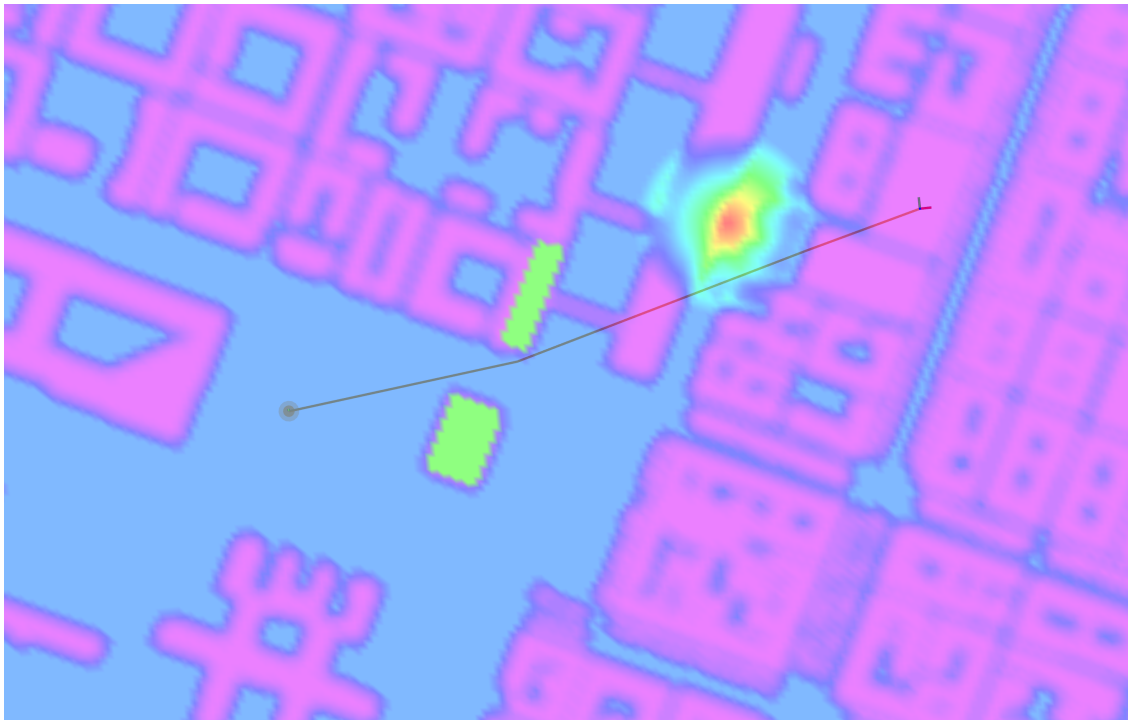


Figure 6.12: Path Planning Example

path planning situation is shown, including also some criticisms due to the presence of the skyscraper. The circle in red its the Unmanned Aerial System in its starting pose, while the three axis frame is its final pose. In between, both the no-fly zones and the "signal hole" are present, and easily identifiable.
What emerges its the Path Planner capability in working also in very small spaces: for example, the drone's trajectory is very close to the buildings but never hit them. Since we are also considering an inflation area around them, we are almost sure that no collision will happen. Furthermore, the feasibility of the path for what concerns the aircraft's dimensions it's guaranteed by the cells area, which is big enough to

allow any kind of maneuvers.
Once the path has been found, the Path Validation Service is called.
The first step to perform is the path decoding, where some useful information are extracted. For this particular simulation, we are assuming a constant cruise speed for the drone equal to $3m/s$ ($10km/h$). Founding out all the cells that belong to the trajectory, its overall length can be measured and is equal to $1008m$. From this value, also the time length of the mission can be calculated, as: $\tau_{miss} = \frac{length}{speed} = 336s$.
At this point, the signal's QOS validation can starts.
As we seen previously, once the path's cells are known this is a quite simple operation, since the validator only has to verify that each cell has a QOS value bigger than the lower-bound, which in this case is set to $S_{min} = 50$. From figure 6.12, we can see that the path covers for almost all the time areas with signal's QOS maximum value. Zooming around the hole, it can be verified that a little part of it has a lower value, which however is never below 50. For this reason, the signal quality validation can be considered successfully completed.
Finally, a risk-aware validation have to be performed. First of all, it have to check the presence along the path of no-fly zone: as it's possible to see in figure, none of them is crossed during the flight. Furthermore, it have to ensure that the predicted number of victims of the trajectory N_{traj} is not greater than the maximum acceptable N_{Max}. Since:

$$N_{Max} = f_{F,Max} \times \tau_{miss} = 0.00336 \geq N_{traj} = 0.001239 \tag{6.1}$$

The risk validation procedure is successfully completed.
Since the path is valid from all the point of views, the drone is authorized to start the mission, setting to true the boolean flag $authorization\_to\_fly$.

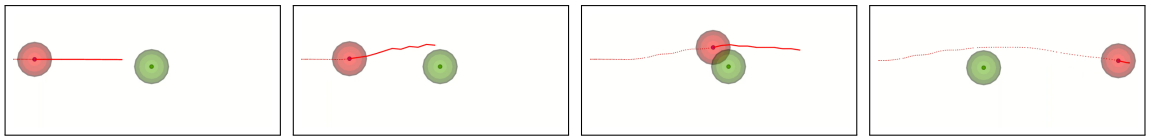
6.5 Path Following and Collision Avoidance

A set of tests and simulations has been performed to analyse the behaviour of the cloud-based traffic management during the flight of the aircraft, in order to validate our approach. For this reason, CBUTM has been seen in action in several operational situations, and the obtained results will be here presented. Since our collision avoidance algorithm is priority-based, meaning that it is only and always the lower priority drone avoiding the higher priority one, for an easier understanding the following colours convention will be applied:

- red: lowest priority level drone
- orange: middle priority level drone
- green: highest priority level drone

The following sequences of screen-shots come from Rviz, where vehicles are represented with a circle (a $geometry\_msgs :: PoinStamped$), surrounded by their own safety area. This is the forbidden area around every drone, where other drones are not allowed to fly, and it is to be considered as a safety distance marker of 10 metres. Output of the collision avoidance algorithm is the local trajectory the autopilot will follow, that is shown with a thicker line coming out of the drone, while the dashed line the UAV leaves behind is its odometry trace. Buildings and fixed obstacles are represented in a certain colour scale on the map: in the sequences below, the map used is built on the no_fly_zone layer.

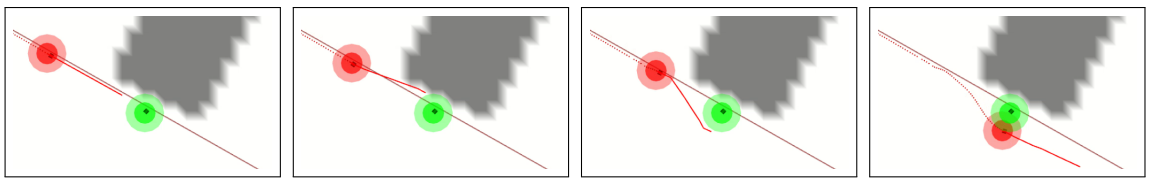
The first test is a basic one, where 2 unmanned vehicles, regularly registered within the network, are flying in the same area, too close one to the other. In figure 6.13, the first test sequence is shown. In this case, only one drone is flying - the *red* one - while the other, with higher priority level, is standing in its position, hovering. As supposed, the red drone is the one forced to move from its original path, as soon as the other drone is detected in figure 6.13b. The *red* drone starts slightly moving out of its path, going around the *green* one, and finally going back to its original path, and reaching its goal.



(a) (b) (c) (d)

Figure 6.13: Test 1: 2 UAVs, only one moving

The same situation in the neighbourhood of a $noflyzone$ represented by a building is shown in figure 6.14. In this case, the *red* US, when looking for a way to escape from collision, tries to go left-ward first, being that one the shortest way to she goal (6.14b). When the presence of the building is detected (6.14b), it immediately changes direction, overcoming the *green* obstacle going right (6.14c), and then coming back to the original path.

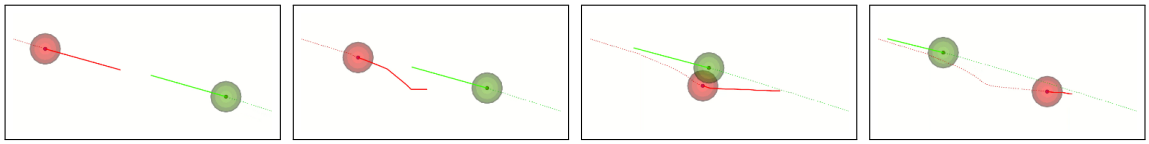


(a) (b) (c) (d)

Figure 6.14: Test 2: 2 drones, only 1 moving. Presence of a fixed obstacle

From now on, at least two vehicles will be in motion at the same time. The first, basic case involves two drones flying one against the other, shown in the sequence

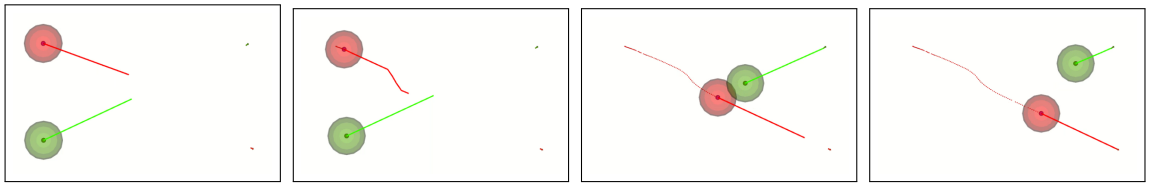
in figure 6.15. The time requirements of this case are a bit more tight with respect to the previous one, since both the drones are flying at their cruise speed of about $4m/s$. The *red* drone then, once detected the *green* one in figure 6.15b, must be faster than before in changing trajectory. This is done in the optimal way shown in figure 6.15c, so that both the vehicles can safely accomplish their mission according to their priorities.



(a) (b) (c) (d)

Figure 6.15: Test 3: 2 drones, both moving. Presence of a fixed obstacle

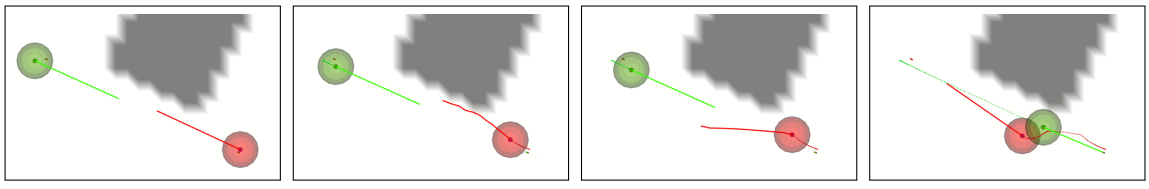
A slightly different circumstance is presented in the sequence depicted in figure 6.16 , where the two vehicles are crossing their paths in a diagonal way.



(a) (b) (c) (d)

Figure 6.16: Test 3: 2 drones, both moving. Presence of a fixed obstacle

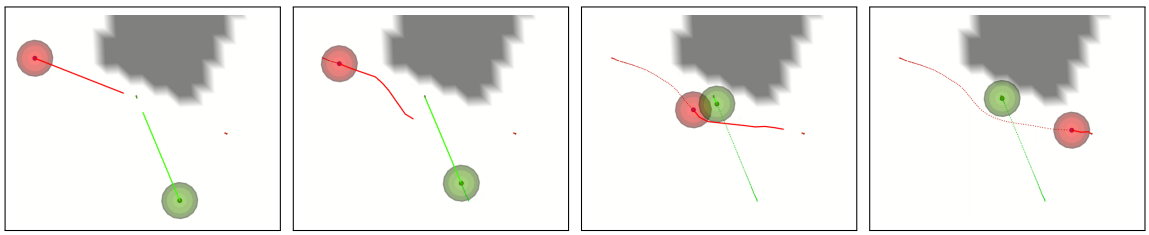
A bit more complex is the situation presented in 6.17, where both the drone are completing their own mission, and their paths cross in front of a building. The *green* one, having an higher priority level, keeps going in its direction, will the *red* one must again fly away from its path. The operation is not performed along the shortest way because of the building, detected in figure 6.17b, forcing the vehicle to go in the other direction, as can be seen in figure 6.17c.



(a) (b) (c) (d)

Figure 6.17: Test 5: 2 drones, both moving one toward the other, in presence of a fixed obstacle

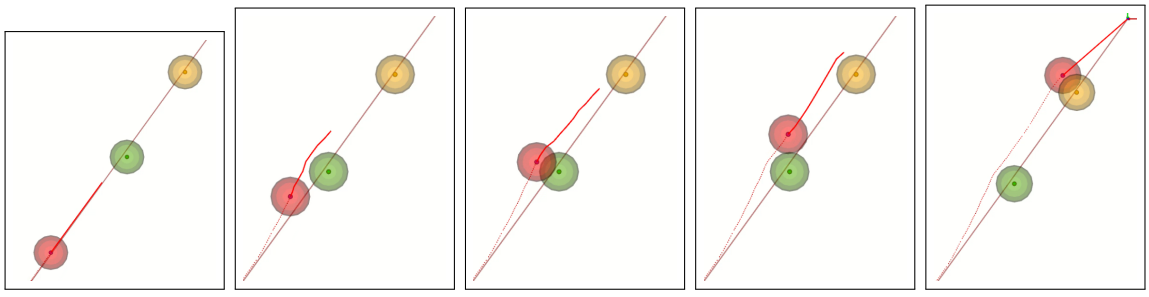
Sequence of figure 6.18 shows a similar situation, with different dynamics: here, the *green* vehicle is approaching the building, and its mission consists in hovering on a fixed position in front of it. In the same moment, the *red* drone is coming from the left-hand side, passing next to the building without stopping by. When it detects the higher priority UAV arriving, the Predictive Trajectory Planner (PTP) computes the optimal deviation from its original path, predicting to go all around the safety area of the *green* one (6.18b).



(a) (b) (c) (d)

Figure 6.18: Test 6: 2 drones, both moving, crossing their path, in presence of a fixed obstacle

Adding one more vehicle, the situation gets a bit more tricky, but same well managed by the CBUTM system. In figure 6.19, the *red* vehicle coming from below finds two standing vehicles on its path. When the first one has been detected in figure6.19b, the vehicle starts moving left to avoid it. Once it has been overcome, while trying to get back on its original path (6.19c), the *red* drone comes across the second higher priority vehicle, and changes again its trajectory to avoid that one too, as can be seen in figure 6.19d.

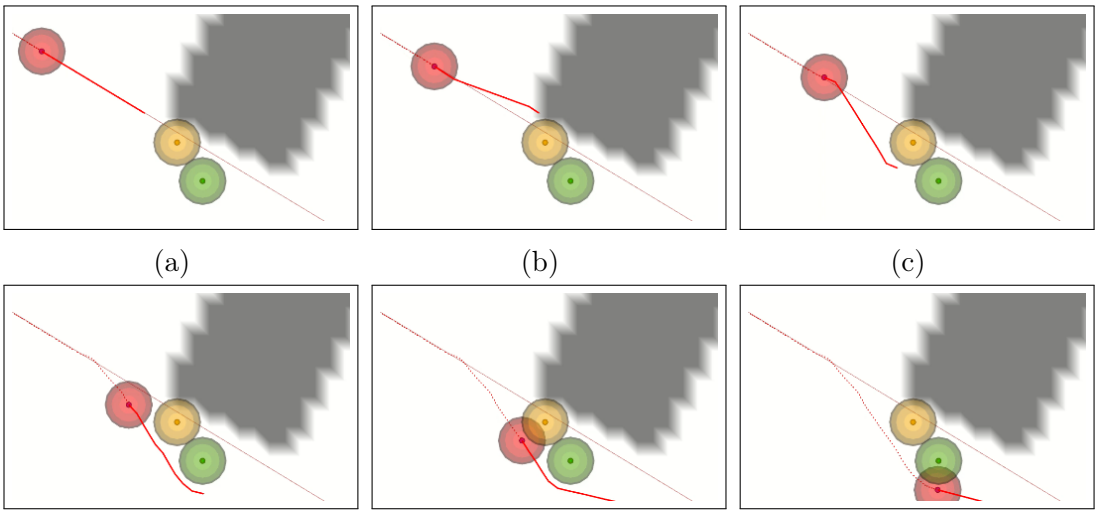


(a) (b) (c) (d) (e)

Figure 6.19: Test 7: 3 drones, one moving

Similar situation, with the only one moving drone meting the other two standing, is shown next (figure 6.20) this time in presence of a fixed obstacle. The same considerations done so far can be applied here.

In the next tests, both the *red* and the *yellow* drones are moving, while the green is standing in its position. A first case is represented in the sequence depicted in

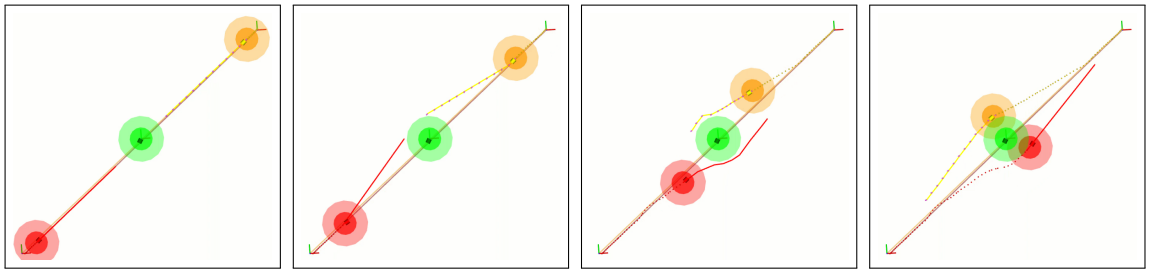


(a) (b) (c)

(d) (e) (f)

Figure 6.20: Test 8: 3 drones, one moving, in presence of an obstacle

figure 6.21, where the *red* and *yellow* vehicles are moving one against the other, while the *green* one is hovering in the middle.

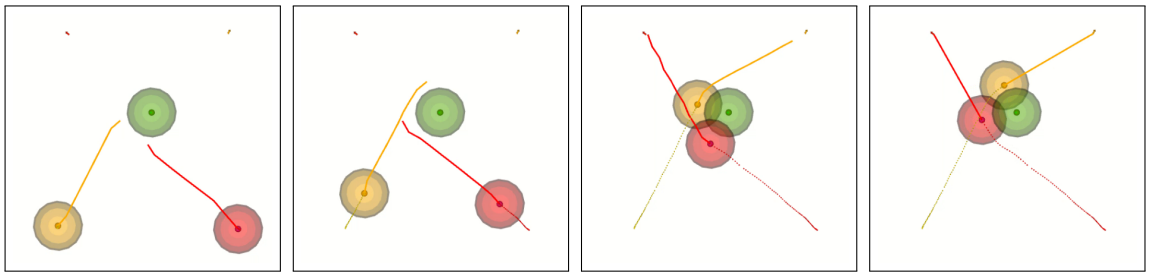


(a) (b) (c) (d)

Figure 6.21: Test 9: 3 drones, two moving, 1 standing in the middle

The first two vehicles are flying in different directions (6.22a), and both will meet someone with higher priority on their path, forcing them to change trajectory (6.22b). While doing it, the *red* will also come across the *yellow*, being forced to find another way. It solves the situation computing the path in a way that some way-points are in the same position or close, resulting in a deceleration of the vehicle, as can be noticed looking at its trajectory projection, that becomes shorter than usual (6.22c). After the *yellow* vehicle will be passed by, the way will be free for the *red* one, that keeps going, knowing the path is safe.

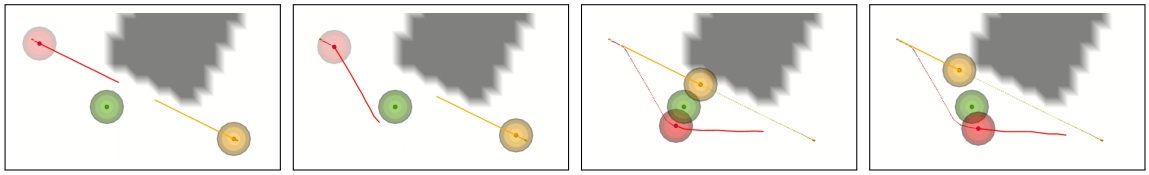
Again, we tested this intersection between three vehicles putting them close to a building. The result is shown in sequence of figures 6.23. The two vehicles with lower priority are flying in opposite direction (6.23a), until they will get caught



(a) (b) (c) (d)

Figure 6.22: Test 10: 3 drones, two moving in different directions, one hovering

between the highest priority one and a fixed obstacle. When pointing toward the space in between the *green* drone and the building, not large enough for both of them, the *yellow* has the precedence, while the *red* one searches for another way, finding at its right-hand side (6.23b). They keep then flying their missions.

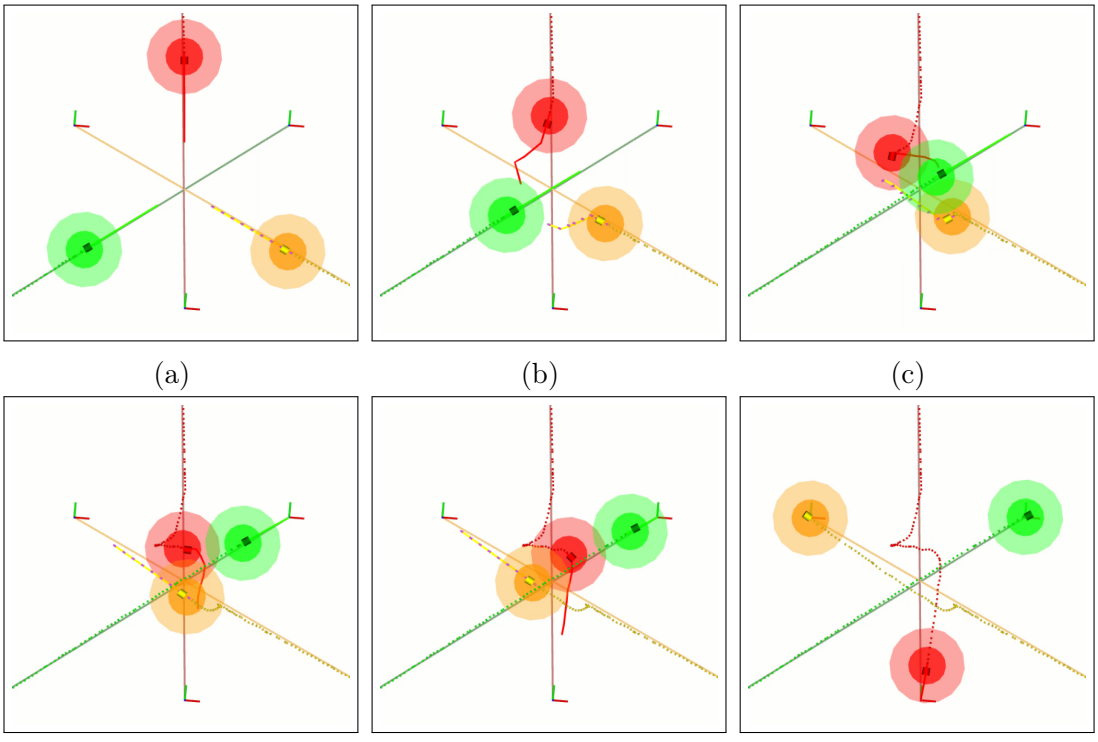


(a) (b) (c) (d)

Figure 6.23: Test 11: 3 drones, two moving, 1 standing, in presence of a fixed obstacle

Last interesting test is shown in the sequence of figure 6.24. Two vehicles with lower priority meet the *green* one, and start computing a different trajectory, that would bring them to move in the same direction (6.24b). When the *red* one detects the *yellow*, it understands it must re-change the plan, and so it does, after tending towards the opposite direction to avoid both the *yellow* and the *green* vehicle (6.24c). In this operation, the predicted trajectory of the *red* drone is made up of way-points which are closer one to each other, resulting in a deceleration of the motion, that allows it to fly just behind the *green* one maintaining the safety distance (6.24d). The priority order is then respected during the crossing, and the safety distance maintained, allowing all the drones to reach their goal with a small deviation.

That's said, the simulations end. In this chapter we showed the software environment we used, and then the all the most important functionalities of the Cloud-Based Traffic Management, treating not only the Risk-Aware Map Manager, but also the Path Planner, the Path Validator and the Collision Avoidance algorithms. What emerges is that the framework is already operative, and it is able to correctly manage not only an ideal operation, but also criticisms during flight.



(a) (b) (c)

(d) (e) (f)

Figure 6.24: Test 12: 3 drones moving in different directions

Chapter 7

Conclusions and Future Works

The goal of this work and of the project was the design and development of a framework that would allow the creation of a structured airspace, i.e. a platform able to regulate and manage the access and use of this airspace for the execution of commercial unmmanned missions, with particular reference to urban environments and focusing on the management of the risk that such missions may involve for people on the ground.
Starting from the identification of the concepts involved, we have gone on defining the problems raised and the techniques and methodologies that can be used to treat them, often getting what is available in the literature of the sector.
In particular, we saw how the risk itself can first be defined, then treated. This led to the definition of a standardized and coherent metric for risk analysis, which is among the main advantages of the platform shown here. We then moved on to the methodologies that can be used to monitor and control the airspace itself and, especially in this work, to tackle the problems related to possible mid-air collisions between unmanned vehicles. Based on the assumptions made at this stage of the project, it was decided to design and develop a responsive system of trajectory following that, based on receding horizon control strategies, could deal with the problem in a decentralized manner in order to improve scalability and robustness of the solution.
An essential part was the implementation of the framework itself, which led us to know and use the potentialities offered by ROS. As we expected, after the initial learning phase in which we invested time to get hold of the tool, the use of this platform allowed and accelerated the entire development.
Once the system was implemented, it was possible to test the design results through simulations that have been reported in this thesis. The simulations conducted finally validated the proposed framework, as they have shown how it is able to: evaluate and guarantee a certain level of safety for the missions; identify, monitor and manage the vehicles present in the framework; avoiding the risks due to collisions between vehicles while achieving the missions themselves. The system seems

to adequately manage traffic even when the original trajectories of multiple vehicles intersect in a dangerous manner. In this phase we also had the opportunity to directly experience the advantages of the modular and distributed architecture on which ROS is based, and this allows us to underline how it is possible to continue the development of the framework in a systematic way, adapting it to the perpetual changes due to the real context as well as to some emerging principles of the cloud paradigm.
Finally, we identified the next steps to continue the development and improve the proposed framework. Specifically, the risk analysis, which converges as seen in the definition of a risk map, can explicitly consider time and other information, such as wind speed, which can be easily grouped and managed by a cloud system. The monitoring mechanisms, already dynamic and expandable, can be lightened to actually work with an arbitrary number of vehicles. This dynamism can also directly involve the collision avoidance system, to which a mechanism based on a planning approach can be associated for redundancy. Further possible improvements concern the overcoming of the 2D assumption during navigation, which would allow more freedom for the necessary maneuvers. The latter could then follow standardized behaviors, in order to make them more predictable and therefore more easily managed by the vehicles involved, always in the distributed logic presented here. The experimentation with real vehicles, missed in this phase of the project, would certainly lead to being able to consider and face all those aspects that reality itself entails.

Bibliography

[1] Faa - unmanned aircraft system traffic management (utm): https://www.faa.gov/uas/research/utm/.

[2] Grid map package on ros: http://wiki.ros.org/grid-map.

[3] Nasa utm reference page - `https://www.faa.gov/uas/research/utm`.

[4] Sesar-ju blueprint: https://www.sesarju.eu/u-space-blueprint.

[5] Warsaw declaration, https://ec.europa.eu/transport/sites/transport/files/drones-warsaw-declaration.pdf.

[6] Bassam Alrifaee, Dirk Abel, and Christoph Ament. Networked model predictive control for vehicle collision avoidance. Technical report, Lehrstuhl und Institut für Regelungstechnik, 2017.

[7] Bassam Alrifaee, Masoumeh Ghanbarpour Mamaghani, and Dirk Abel. Centralized non-convex model predictive control for cooperative collision avoidance of networked vehicles. In *Intelligent Control (ISIC), 2014 IEEE International Symposium on*, pages 1583–1588. IEEE, 2014.

[8] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1917–1922. IEEE, 2012.

[9] Swee Balachandran, César Munoz, and Maria Consiglio. Implicitly coordinated detect and avoid capability for safe autonomous operation of small uas. In *AIAA Aviation 2017 Conference*, 2017.

[10] Alberto Bemporad, Maurice Heemels, and Mikael Johansson. *Networked control systems*, volume 406. Springer, 2010.

[11] Ros contributors. Mavros offial page: http://wiki.ros.org/mavros.

[12] ROS contributors. Ros wiki , http://wiki.ros.org.

[13] Wikipedia contributors. Motion planning — Wikipedia, the free encyclopedia. [Online; accessed Feb-2018].

[14] Wikipedia contributors. Randomized algorithm - Wikipedia, the free encyclopedia. [Online; accessed Feb-2018].

[15] Range Commanders Council. Standard 321-07 "common risk criteria standards for national test ranges: Supplement". *USA Dept. of Defense*, 2007.

[16] Konstantinos Dalamagkidis, Kimon P Valavanis, and Les A Piegl. *On integrating unmanned aircraft systems into the national airspace system: issues, challenges, operational restrictions, certification, and recommendations*, volume 54. springer science & Business Media, 2011.

[17] Aislan Gomide Foina, Clemens Krainer, and Raja Sengupta. An unmanned aerial traffic management solution for cities using an air parcel model. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1295–1300. IEEE, 2015.

[18] Aislan Gomide Foina, Raja Sengupta, Patrick Lerchi, Zhilong Liu, and Clemens Krainer. Drones in smart cities: Overcoming barriers through air traffic control research. In *Research, Education and Development of Unmanned Aerial Systems (RED-UAS), 2015 Workshop on*, pages 351–359. IEEE, 2015.

[19] Sara Giammusso. Cloud robotics in real time application.

[20] Giorgio Guglieri, Alessandro Lombardi, and Gianluca Ristorto. Operation oriented path planning strategies for rpas. *AMERICAN JOURNAL OF SCIENCE AND TECHNOLOGY*, 2(6):1–8, 2015.

[21] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[22] Steven G. Johnson. The nlopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt.

[23] Mina Kamel, Javier Alonso-Mora, Roland Siegwart, and Juan Nieto. Nonlinear model predictive control for multi-micro aerial vehicle robust collision avoidance. *arXiv preprint arXiv:1703.01164*, 2017.

[24] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[25] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[26] Alex Kushleyev, Daniel Mellinger, Caitlin Powers, and Vijay Kumar. Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300, 2013.

[27] Yoshiaki Kuwata and Jonathan P How. Cooperative distributed robust trajectory optimization using receding horizon milp. *IEEE Transactions on Control Systems Technology*, 19(2):423–431, 2011.

[28] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[29] Andrea Lorenzini. Cloud-based uavs traffic management system: a risk-aware map manager.

[30] Spyros Maniatopoulos, Dimitra Panagou, and Kostas J Kyriakopoulos. Model

predictive control for the navigation of a nonholonomic vehicle with field-of-view constraints. In *American Control Conference (ACC), 2013*, pages 3967–3972. IEEE, 2013.

[31] ACMDV Maniezzo. Distributed optimization by ant colonies. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, page 134. Mit Press, 1992.

[32] Anusha Mujumdar and Radhakant Padhi. Evolving philosophies on autonomous obstacle/collision avoidance of unmanned aerial vehicles. *Journal of Aerospace Computing, Information, and Communication*, 8(2):17–41, 2011.

[33] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta^*: any-angle path planning on grids. In *AAAI*, volume 7, pages 1177–1183, 2007.

[34] Masoud Nosrati, Ronak Karimi, and Hojat Allah Hasanvand. Investigation of the*(star) search algorithms: Characteristics, methods and approaches. *World Applied Programming*, 2(4):251–256, 2012.

[35] Stefano Primatesta and Basilio Bona. Motion control of mobile robots with particle filter model predictive equilibrium point control. In *Autonomous Robot Systems and Competitions (ICARSC), 2017 IEEE International Conference on*, pages 11–16. IEEE, 2017.

[36] Earle Robotics. Ros introduction.

[37] Stefano Scheggi and Sarthak Misra. An experimental comparison of path planning techniques applied to micro-sized magnetic agents. In *Manipulation, Automation and Robotics at Small Scales (MARSS), International Conference on*, pages 1–6. IEEE, 2016.

[38] SESARJU. U-space web site, http://www.sesarju.eu/u-space.

[39] David H Shim, Hoam Chung, and S Shankar Sastry. Conflict-free navigation in unknown urban environments. *IEEE Robotics & Automation Magazine*, 13(3):27–33, 2006.

[40] David H Shim, H Jin Kim, and Shankar Sastry. Decentralized nonlinear model predictive control of multiple flying robots. In *Decision and control, 2003. Proceedings. 42nd IEEE conference on*, volume 4, pages 3621–3626. IEEE, 2003.

[41] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[42] Enrico Stabile. Cloud-based uass traffic management: Registration, identification and monitoring.

[43] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[44] Anthony Stentz et al. The focussed d^* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[45] Paul Trodden and Arthur Richards. Cooperative distributed mpc of linear systems with coupled constraints. *Automatica*, 49(2):479–487, 2013.

[46] Megha Vora and TT Mirnalinee. From optimization to clustering: A swarm intelligence approach. In *Handbook of research on artificial intelligence techniques and algorithms*, pages 594–619. IGI Global, 2015.

[47] Chengcheng Wang, Xiaofeng Liu, Xianqiang Yang, Fang Hu, Aimin Jiang, and Chenguang Yang. Trajectory tracking of an omni-directional wheeled mobile robot using a model predictive control strategy. *Applied Sciences*, 8(2):231, 2018.

[48] Roland Weibel and R John Hansman. Safety considerations for operation of different classes of uavs in the nas. In *AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum*, page 6244, 2004.