POLITECNICO DI TORINO
Master degree course in Mechatronic Engineering

Master Degree Thesis

Cloud-Based UAVs Traffic Management system

Registration, monitoring and management

Supervisors
prof. Alessandro Rizzo
prof. Giorgio Guglieri

Candidate
Enrico Stabile

Internship Tutor
dott. ing. Stefano Primatesta

Anno accademico 2017-2018

Abstract

«««< HEAD Drones economy has been steadily growing in the last years, together with the interest in this fascinating technology. They can be used in several fields, with new possibilities rising up every day. ======= Drones economy has been steadly growing in the last years, together with the interest in this fascinating technology. They can be used in several fields, with new possibilities rising up every day. »»»> 28b8d3064afd7f68c0f80d818a1e9b90ff2ed385 Great challenge of the coming years is to find a way to make people and unmanned vehicles co-exist in complex urban environment, exploiting all the advantages they can bring us without neglecting operational issues concerning safety and privacy. It means demonstrate that safe unmanned operations can be held in city environments, with tele-operated unmanned vehicles.

Authorities, companies, universities and research groups are working, alone or together, in defining the framework and build up efficient UAV Traffic Management systems. In this direction is working also TIM, Italian main communication service provider, that, together with the Polytechnic of Turin, are supporting active researches in that field. Their intention is to use the already well spread 4G data connection, waiting for the upcoming 5G that promises to be even much faster, for building up a city drones network properly structured and organised, able to guarantee safe and effective UAVs operations in smart city environments.

Aim of project, developed within the TIM JOL in collaboration with the Polytechnic of Turin, is to develop and test some patterns on ROS applying Cloud Robotics paradigm in order to build up a prototype of UAV Traffic Management system, able to manage quad-copters UAV with remote computing granting safe and effective UAV operations. Starting from an analysis of the state of the art, surveys of ongoing projects and an investigation of actual regulations and intentions, we proposed a cloud-based solution for building a UTM system, operating in a smart city scenario, trying to demonstrate the goodness of a cloud based approach for minimizing collisions and general risk for unmanned missions.

Target of the project has been only quad-copter type vehicles, the best fitting for an urban scenario thanks to their dimensions and flight dynamics. Goal is to set up, using the Robot Operating System (ROS), a networked system able to prevent, "on-line", collision between drones flying in the same city airspace, relying on the functionalities provided by the cloud.

The result has been obtained setting up a well organised system, able to get information from each vehicle and make them properly interact, flying safely over the city airspace, after it has been well described in cost maps. Tests have been performed to verify the goodness of the solution, demonstrating the effectiveness of such a system in the conditions we supposed to operate, being ready for future expansions according to the next steps of the project.

Acknowledgements

This work has been the last step of a long, long process, started when I first enrolled university. During this period, several things happened, others changed, so many people entered, in a way or another and for different reasons, in my life. So that, at the end of the story, I see back on the way and can say that yes, it has been a very nice journey.

Studying in Rome before and Turin after, I had the opportunity to touch so many topics and fields, following lectures more or less interesting depending on the case, on my mood, on my diligence, and on the quality of the courses and professors. It definitely fostered my curiosity and ambitions. But I cannot sincerely stop underlying how much university actually gets its importance from everything you learn from outside the university itself, with that being the reason, maybe just the "excuse" for things happening around you.

During this period, I have seen myself finding every time my way to get out of troubles and difficulties, counting on the support of others or just on my own, coming out stronger and more conscious of my strength and potentialities. I have seen myself accepting and winning challenges, doing things that till 2 seconds before I could not even expect from me, braking down the walls and building up new ways. I have discovered more and more about my self, finding out my biggest passions and wishes, what turns me on the most, trying to criticize my past for building up a better present and a brilliant future. I travelled a lot and lived abroad, putting my self in the weirdest and most different situations, meeting hundreds of people, feeding my curiosity, opening my mind and widening my horizons.

All this - every step made so far, every decision taken, every place discovered, every people met, every relation grown - led me to be the man I am now, with all the motivation for keep going in the journey I decided to take long time ago, aimed at an endless improvement of myself, in every way.

And along this journey, together with other important things, I also learned to be thankful, to appreciate what makes me happy, teaches me something, transmits me good things, makes me reflect and breath, hope and smile.

So that I would love to say thank to every one played his role so far, those that have directly brought positive energy in my life, but also those bringing some that, at the first glance, may seem negative, but that turned into a positive force, another brick of the castle.

Starting from where everything starts: family. Thanks to my parents, to whom I owe all the adventures done so far, with to their support that never missed. Even if, too often, "in their way". May you discover again the force of passions, the pleasures of life, that is not meant to be just for work, worries and preoccupations. May you revive the joy of living with someone you decided, with good reasons, to have by your side. May you realize that differences hold, yes, but cannot be just reasons for a fight, and may you stop arguing, pretending, blaming, and start working

for the other, doing each one his part. May you prefer the positive way, instead of another bad word, and understand the importance of creating a positive environment between and around you. That being a couple will restart meaning being a team, building up your happiness, that is so just when shared. To that genius of my brother, always been for me not someone to take the distances from, nor to stay glued behind, but a beacon lighting up the way while walking, actually running, his own. Source of inspiration and wise mentor, I am proud of my big brother, and I wish the best for us. And to the rest of the family, especially my grandmother and grandfather, from which I can feel, day after another, I've got so many aspects of my characters, maybe the half.

Thanks to the old friends, with whom relations of course changed, sometimes getting weaker, sometimes getting even stronger. It is all happening according to the process that holds (or should) for everyone, when you realize what does really matter, which are the things you want to spread and, consequently, which the ones you want to receive from the people you decided to have in your surrounding.

Thanks to the new ones, met in the most different ways, conditions, situations, places. Thanks to the crew of the Polytechnic, for the shared efforts and the fun inside and outside the campus - Three for all: the moment "il tavolone meccatronico" was born; jumping Greco's hour in the park; the birthday song - making this experience that good. Thanks to my lovely housemates, for having transmitted me so much and taught how to live and happily survive being the only man with 3 women under the same roof. Thanks to the people more or less randomly met in streets, buses, parks, courses, salsa classes, radio studio, gym, shared flats, parties, I exchanged something with. Thanks for anything will make me remember this period in Turin as a great one. Special thanks to "the dreamers" I have met along the way, the ones living close and the ones met around the world, with whom a certain relationship exists and resists. Thanks for the fresh breeze blown and everything good brought into my life.

And thanks to the Woman I have the honour to have by my side, randomly met and conquered just spreading everything I want to receive back from life, and that I found in such a beautiful, tiny and strong lady. In her I have met someone as good, positive, smiley, active, smart and ambitious as I want for me and to be, that makes me strive for being the one she would always decide for. In a relative short time, she has already been source of inspiration and strong motivation for so many things, reason to improve and for getting the best out of this life. And I hope she will be so for much longer.

Finally, I want just to say thanks to Life, that is always ready to take the shape we want it to have, and will always bring you back what you give around. Wish for myself to be able to face everything will come next as I did so far and better, working for my Life to be full of happiness and satisfaction, love and friends, great adventures and engaging challenges, shared with the right people. And lived with this smile on my face.

La vita è bella, hay que saberla vivere

Contents

Acronyms

UAV Unmanned Aerial Vehicle

GSM Global System for Mobile communication

GPS Global Positioning System

NCS Networked Control System

UTM UAVs Traffic Management

CCS Cloud Control Station

PTP Predictive Trajectory Planner

MM Map Manager

PP Path Planner

NFA National Flight Authorities

IMU Inertia Measurement Unit

ENAC Ente Nazionale Aviazione Civile

ENAV Ente Nazionale di Assistenza al Volo

Chapter 1

Introduction

1.1 Problem definition and considered framework

In the last years, the interest in drones of every kind (not only quad-rotors, the most common and easy to find around, but also fixed-wings, hexa-copters and so on) has steadily risen up, thanks to the fascinating potentialities of these vehicles, that find applications in so many different fields, as well as to the inner beauty of these flying objects.

That is why drone's economy has seen an exponential growth, arriving nowadays at an estimated value of about 6 billion dollars ([4]), involving different targets, as hobbyists to people using them for their own business, researches or, sadly, military purposes.

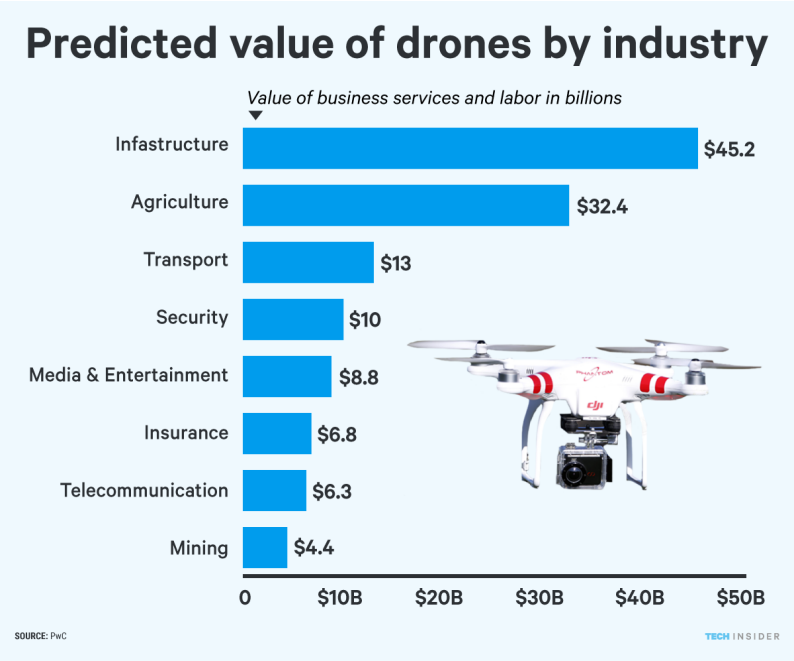


Figure 1.1: UAV sells data expected by the end of 2021, source: BI Intelligence

The growth of their usage has not been accompanied by a proper adaptation of laws and rules, ending in a lack of coherent legislation and thus many limitations imposed, especially when it

comes to cities airspaces, with so many issues and people involved. Despite of some concession, the common feeling tells about so many obstacles to overcome in order to fly a drone, with the legislative apparatus being as usual too slow in adapting to this new technology. In particular, we should underline how the usage of unmanned vehicles is strictly limited by the Visual Line Of Sight (VLOS), denying any drone to be flown too far from the direct operator, blocking not only their recreational but most of all the greatest part of their commercial applications, that would be the strongest motivation in fostering research and development. New technologies, being applied in military fields since a long time, allow to go Beyond the Visual Line Of Sight (BVLOS), but they are still blocked, waiting for proper regulations and procedures able to assure safeness and collision avoidance [6].

However they will be piloted, involving such a huge number of users and vehicles, it follows the need of a well organized system, with clear rules, rights, duty and prohibitions, that meets the needs of users of every kind: who want to fly their drones for hobby, who want to do it for business and who for providing a service to the community. Such an organization process must take into account all the aspects concerning safety, privacy, responsibilities, legal and operative issues.

Figure 1.2: A possible scenario

Unblocking this situation, overcoming every possible issue that makes drones' city networks still too **dangerous** and unreliable, is one of the **new technological challenges**, and may represent a great opportunity for who may be interested in leading this city's airspace revolution. Many people, societies, universities and organisations are already involved in this interesting problem. It led so far to a law-jam, with different solutions proposed among the Countries, with their policy being sometimes too restrictive, some times too relaxed. This is the case of many National Flight Authorities, like ENAV and ENAC in Italy or FAA in US, that can already be considered the result of the cooperation between many States. But since the last years, different entities met in organisations trying to build up standards and common practices for the future UAV Traffic Management (UTM), as it is commonly called - system. This is the case of the European Aviation Safety Agency (EASA) for example, or again the International Civil Aviation Organization (ICAO), a United Nations specialized agency, working together with private and public entities to write a proper legislation and lay the foundation for a well structured UAV system, able to assure, nationally and internationally, safety of drone's network, taking into account every aspect it may involve.

Main preoccupation of this building process is to find the way to assure network's safeness and efficiency concerning technical, ethical, operational and bureaucratic aspects

But in future it is likely there will be a very large number of drones flying daily over cities without any direct human control on board. Thus, it seems anachronistic and not wise at all thinking about a restrictive policy of strict limitations imposed by taxes, one-use-only licenses involving very long processes and every kind of short-term policy adopted so far.

The main elements a working UTM system should count on may be listed as:

- A well defined and structured architecture, able to maintain everything under control and collect all the relative and sensible data, making it always possible to monitor and identify every flying object, linking it to its legal responsible.
- A proper and standardized risk assessment, through which evaluate the safeness of vehicles, missions and operations in a clear metric, so that a certain level of safeness can be always assured, according to local legislation.
- All the techniques and procedures needed to make drones flying autonomously, along a defined and optimal path, in a smooth and safe way, avoiding with generic obstacles, either fixed or moving.

Nowadays, with the growth, spread and consolidation of internet technology and then of revolutionary concepts of Internet of Things (IoT) and Cloud Robotics (CR), new possibilities opened for the proliferation of such an organized urban network, where robots and humans can cooperate in a well structured system.

In Italy, the main telecommunication service provider company, TIM, wishing to use its highly spread high speed connection (4G is already consolidate and well working, while the next generation connection, 5G, is likely to enter in action very soon) is financing a research work within the Joint Open Lab at Politecnico di Torino, fostering the development of a cloud-based urban UTM system.

1.2 Thesis contribution

Goal of the project has been to build up a **Cloud-Based UAVs Traffic Management** system, made up thanks to the cooperation of different colleagues within the Joint Open Lab, able to tackle the problems of safety, collision avoidance and user/vehicles registration, identification and monitoring. The proposed solution sounds quite innovative in the way standard issues a UTM must solve have been faced.

The present work wants to be a step forward in the analysis of a cloud-based approach to air traffic management of unmanned vehicles in city airspace, laying the basis for further developments aimed at mixing all the most efficient technologies for creating a well structured and organised city drone network.

The entire project has been thought and developed in **ROS** environment, the Robot Operating System, which is the most used open source operating system when it comes to robot applications, being supported by a huge community. The way it basically works, with many nodes interacting providing services and exchanging messages between each other through topics, fits perfectly in describing the architecture of our system. It is made up of different entities doing each one its own specific task, and exchanging informations and functionalities for creating a robust and efficient network. A general overview of the system is shown in the figure 1.3 here below.

As can be seen from figure 1.3, the main entities of the network are:

- Cloud Control Station, CCS
- Map Manager, MP
- Path Planner, PP

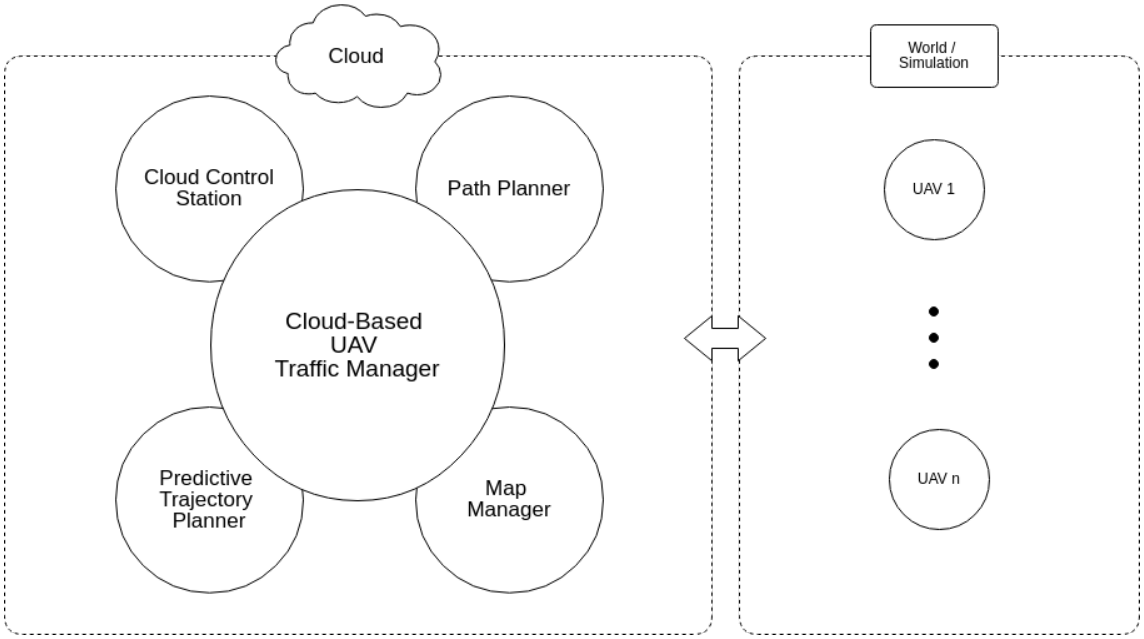


Figure 1.3: Graphical representation at top level of CBUTM's working principle

- Predictive Trajectory Planner, PTP

The final result is then given by the cooperation of these different entities, then ROS nodes, all referring to the same environment. Notice that the whole system has been made possible thanks to the cooperation of 3 colleagues within the JOL, working in parallel exchanging knowledge and efforts.

First of all, a deep based and well structured **risk assessment** has been developed, aimed at creating a standardized metric for evaluating the risk for unmanned vehicles' missions. Having such a clear metric allows to set certain safeness thresholds, imposing a maximum acceptable level of failures-per-hour. This risk analysis ends in the creation of a risk map, on which path planning and collision avoidance can be performed.

Then a proper system architecture has been designed, in order to solve the problem of controlling a drone's network in the most efficient way. Many different approaches have been analysed and confronted, finally choosing for a **distributed cloud based control system**, opting for a **receding-horizon-based control technique**, chosen for its responsiveness and robustness.

Finally, a collision avoidance mechanism has been developed, for managing the traffic of Unmanned Aerial Vehicles in controlled city's airspace.

The present thesis in particular will be mainly focused on presenting the so called "Cloud Control Station" node, a ROS executable performing an equivalent role of the common Ground Control Station in usual Air Traffic Management systems.

The aforementioned CCS is then meant to be the core of the network, able to collect information by drones flying within the network, tracking them and being the bridge between every part of the system, capturing and managing traffic between single nodes. In particular, main functionalities carried out by this node are:

- Perform, through a registration service, all the procedures needed for connecting new drones to the network, identifying the vehicle and its owner, saving and processing all the relevant

informations, keeping an up-to-date list of network users, and establishing the connection between every agent. It is also useful for maintaining the link between virtual and physical entities, for going back to the legal owner of a vehicle at every moment.

- Act as a bridge between the other actors of the system, linking the UAV's ROS interface with the other nodes making up the network and managing the data streams.
- Apply procedures for organising traffic in the city airspace, assigning a specific operational flight altitude and a coherent priority level to every drone, fundamental for the collision avoidance algorithm to work.
- Monitor and track vehicles while flying, detecting whether if two or more UAVs are likely to collide, sending alerts to the interested ones according to their priority level, together with the needed informations for activating the collision avoidance procedure. The way this node has been decided to work, and then how the architecture of CBUTM has been designed, is fundamental for the scalability of the system, able to manage a theoretically infinite number of vehicles.

1.3 Thesis structure

In **chapter2**, all the basic concepts and knowledge involved in the project and useful for a better understanding thereof, are presented. In **chapter 3**, a general overview of what a UTM system is, which problem should be able to solve and how some important aspects have been done or thought to be done so far is analysed. Then, a proposed solution based on the concept of Cloud Robotic will be presented and discussed in **chapter 4**, before showing, in **chapter 5**, the way it has been practically developed, focussing on all the aspects related to the inner organization of the proposed system: starting from the registration of new user/vehicles, to the management of the active one, and the organisation of the traffic in the city airspace. Other relevant topics making up the final result will be analysed in the same chapter 4, but explained in details in other thesis ([48], [52]). This is because the result of the present work must be intended as coming from the cooperation of 3 colleagues, continuously working in parallel exchanging knowledges and efforts. Finally, the practical results obtained simulating such a control system are shown and commented in **chapter 6**, leading to the conclusion about the project and the future works.

Chapter 2

Basics and requirements

2.1 ROS - Robotic Operating System

ROS [17] 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.

It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. It was built from the ground up to encourage collaborative robotics software development.

This last one is for sure the most attractive thing about ROS: it us followed and supported by a huge community, where almost everything is shared with open-source license, and where people from all around the globe can meet, finding what they were missing to build up their application and give their contribute to the community.

Software in ROS is organized in packages. A package might contain ROS executables, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused.

For giving an idea of how ROS basically works, let's analyse its main elements and concepts, which are:

- **Nodes**
  Nodes are independent executables that can be thought, as the name suggests, as nodes of a complex network, able to communicate and interact in different ways. Different nodes can be created or included from outside to perform different actions, separating code and functionalities providing a robuster and organised system. Thanks to two ROS-embedded libraries, they can be coded both in c++ and python.

- **Parameters**
  They represent a great tool of customizing, changing or adapting the behaviour of your ROS application. One can declare many configurable parameters associated with nodes, thanks to which one can change values, names, topics, and everything as been configured as a parameter, allowing, for example, to run a generic code in many different instances, or changing the weight of a function in the simulation session. They are also readable from outside, managed by a specific parameter service, allowing the developer or other nodes to read or change their values.

- **Topics**
  Topics are the lines through which nodes can exchange **messages** in a unidirectional fashion publishing on it or subscribing for reading from it. Each topic is strongly typed by the ROS message type used to publish to it and nodes can only receive messages with a matching type. They have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic. ROS currently supports TCP/IP-based and UDP-based message transport.

- **Services**
  The publish/subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for Remote Procedure Call (RPC) request/reply interactions, which are often required in a distributed system. Here is where services arrive. They are defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Service's calls are blocking, meaning that the node calling that service waits for the service to be done, than goes ahead.

- **Actions**
  Differently from the services, actions are preemptable processes. In any large ROS based system, there are cases when someone would like to send a request to a node to perform some task, and also receive a reply to the request. This can currently be achieved via ROS services. In some cases, however, if the service takes a long time to execute, the user might want the ability to cancel the request during execution or get periodic feedback about how the request is progressing. The actionlib package provides tools to create servers that execute long-running goals that can be preempted. It also provides a client interface in order to send requests to the server.

Another fundamental point of ROS is that easily allows different applications to cooperate and interact, either if their are working in the same machine or in different ones, even far one from the other. This is because ROS is based on the presence of a ROS MASTER, that is the core of the system that give sense and substance to the ROS environment. It provides naming and registration services to the rest of the nodes in the ROS system, tracking publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another within the network. Once these nodes have located each other they communicate with each other peer-to-peer. Great functionality provided by the Robotic Operating System is the possibility to make different machines detect a ROS_MASTER running not only in other machines but also in other networks, knowing its ROS_MASTER_URI and ip address establishing a connection with that. In this way, a machine running ROS can run programs that will be processed by that machine, but will operate within the ROS environment built up on the indicated URI.

A sensible aspect of ROS is the fact of being a Non-Real-Time Operating System. This can be a limitation considering that, speaking about very dynamic objects meant to fly over head of civilians, not being able to assure the respect of deadlines may mean waste of fundamental seconds, then impossibility to act in time for avoiding collision. This considerations must be taken into account during the development process, building up light algorithms not requiring that much of computational efforts to be performed, and continuously check the connection integrity and data losses, including some emergency routines.

2.2 Gazebo

[1]Gazebo is an essential tool for developing robotic's applications. It is free and supported by a great community, making of gazebo one of the primary tools used by the ROS developers. A specific ROS package has been then developed[2], allowing interaction between the simulation and the ROS environment. Gazebo consists of a robust physics engine, high-quality graphics, and convenient programmatic and graphical interface, offering the ability to accurately and efficiently simulate robots in a complex scenario, with everything being customizable according to the needing. By means of proper **world files** is, in fact, possible to describe, all the elements making up the scenario, while with **model files**, is possible to model any kind of robot. This is done in a specific format, called SDF, Simulation Description File, even tough it is still possible to find robot model described in the old URDF (Universal RObot Descriptio File) file, left apart because of the evolving needs of robotics. SDF is a complete description for everything from the world level down to the robot level. It is described using XML, is scalable, self-descriptive and makes it easy to add and modify elements. The **Gazebo server**, the engine of the simulator, parses the description files given to it as input, building up the simulation environment accordingly and simulating dynamics thanks to multiple high-performance physics engines. It is the shown thanks to the **Gazebo graphical client**, that connects to a running gazebo server allowing its visualization. It utilizes OGRE's graphics rendering engine[3], providing realistic rendering of 3D scenarios. Gazebo uses then a number of **Enviroment variables** to locate files, and set up the communications between servers and clients.

Finally, many **plugins** offered within the Gazebo community provide a simple and convenient mechanism to interface and interact with Gazebo, having direct access to Gazebo's API and all its functionalities. They are self-contained routines that can be added or removed from running systems, allowing to include new features like, for example, the simulated GPS sensors used in the present work.

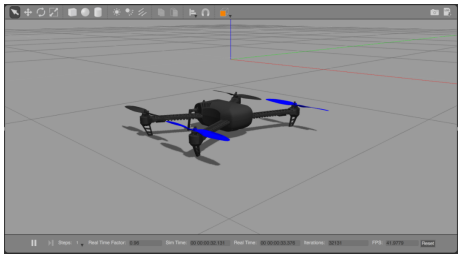


Figure 2.1: Iris drone simulated with its SDF file within the Gazebo environment

2.3 Cloud Robotic

With the term Cloud Robotic is meant an on-line network to which many different robots belong, connected via wired or wireless communication channels, sharing informations and cooperating between each other. The existence of such a cloud, direct consequence of the progresses in **machine**

[1]http://www.gazebosim.org/

[2]http://wiki.ros.org/gazebo

[3]https://www.ogre3d.org/

learning and **IoT** (Internet Of Things), brings unlimited advantages, opening to thousands of applications.

The main features of a cloud can be summarized [43] into:

- **Big Data**
- **Collective Robot Learning**
- **Cloud Computing**

The first refers to the **massive amount of data**, both structured and unstructured, that can be collected if many sources share the informations they have, process and get in any possible way, in a **common resource pool**. Having such a big amount of data is the basis for building up a real knowledge of the system one wants to control or interact with. From control system's theory we learnt in fact that the efficiency of a controller is based on its knowledge of the system, the ability to observe its state and the potentiality to predict its future.

And of course, the easiest, fastest and most efficient way to collect such an amount of data to call it "big" is **putting together different sources**, making them cooperate and share in a well structured network, so that one can learn from the data coming from another and vice versa. The only limitation to knowledge will then be the **limited power of on-board computers**, that is even smaller when it comes to drones, not able to process that amount of data and extrapolate all the needed informations. But as nature and history teach us, this limiting issue **can be overcome in a cooperative way**: if the computational power of a single machine is limited, the one resulting from the cooperation of many of them is not, and if high power machines enter in the loop, the game is done. The cloud can than provide the high computational power required by the network to work, while the robot can do the physical job.

Cloud Robotics shifts then demanding processing and data management to the Cloud [36]. Though robots can benefit from various advantages of cloud computing, Cloud Robotics raises critical new questions related to network latency, quality of service, privacy and security. Cloud-based applications can get slow or unavailable due to high-latency responses or network hitch: packet loss or network delay can easily compromise robot's activities such as meeting timing constraints that could cause system failures. On the other hand Cloud Robotics allows robots

2.4 Networked Control System

A Networked Control System (NCS) is a control system wherein the control loops are closed through a communication network. The defining feature of an NCS is that control and feedback signals are exchanged among the system's components in the form of information packages through a network, transmitted via wired or, in our case, wireless communication channels.

According to [30], NCS have four fundamental characteristics:

- **Large-scalability:** multiple control loops can be formed through an underlying communications network so that multiple control objectives can be achieved at the same time
- **Open:** NCS is open to runtime system reconfiguration, i.e., when the NCS is running, new agents can be added, information flow can be dynamically changed, or existing agents can be replaced
- **Time-critical:** the stability and performance of NCS are affected by a time delay in a control loop introduced by the own communication system

- **Safety-critical:** : due to the existence of the computation and communications network, it is harder to achieve a safety-guarantee of NCS and it causes severe consequences once the NCS fails

With the growth of the internet and the birth of Iot and cloud robotic concepts, NCSs attracted the interest of researchers around the globe, finding application in many different fields, and opening to brand new ones.

Main advantages of being networked is the facility to enlarge the system, adding and removing agents, together with the possibility of exploit all the computational power of the cloud, that may include high power computers helping far smaller ones in accomplishing difficult tasks. Along with the advantages, several challenges also emerged. New control strategies, kinematics of the actuators in the systems, reliability and security of communications, bandwidth allocation, development of data communication protocols, corresponding fault detection and fault tolerant control strategies, real-time information collection and efficient processing of sensors data are some of the hottest topics. Moreover, we must consider the introduction of additional time delays in control loops, together with the possibility of packages loss. Depending on the application, time-delays could impose severe degradation on the system performance, and limitation on the performances.

A networked control system consist then of interacting dynamic subsystems, namely agents of the system, which can be:

- **Passive agents:** represent dynamic subsystems without networked control. However, they can communicate their data like current and future states to active agents, or they can be detected by active agents' sensors.
- **Active agents:** are connected using a communications network over which they can exchange data. The exchanged data are used by the agents' controllers to find appropriate inputs to achieve their goals while taking the interaction with other agents into consideration. Additionally, active agents consider passive agents.

An example of NCS with active and passive agents can be a smarter evolution of the one we see every day in our roads, where active agents being vehicles with networked controllers and passive agents being either vehicles without networked controller or road infrastructure such as traffic lights etc.

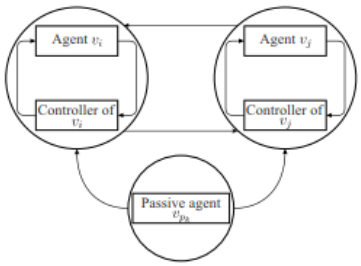


Figure 2.2: example of NCS with active and passive agents modelled as a graph

In an NCS, the network is considered as an independent entity as it imposes restrictions on data exchange and introduces **time delays** in control loops. These restrictions and delays affect the stability and performance of the system. Besides, the network topology is generally time-variant. This leads to the following definition of NCS[26]:

> A NCS is an integration of network and actions, consisting 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.

Based on agents' characteristics, NCS can be classified into **homogeneous and heterogeneous** networks, depending on if the agents have all the same dynamics or at least two of them have different behaviours.

Another important characteristic, leading to a further classification, regards the interaction between agents of the network when it comes to their dynamics. In fact, two or more agents can be **dynamically coupled or uncoupled**, depending on if the states or the inputs of an agent affect its own as well as other agents, or if the coupling takes just place in the control. In the first case, the state equation of the agent dynamically coupled with another of the network will be like the one below, with i being the selected agent and j the dynamically coupled one, while the same for a decoupled agent will not present any direct relation with other agents.

Definition 2.1. $x^i(t+1) = f^i(x^i(t), \mathbf{u}^i(t), x^{i_1}(t), \mathbf{u}^{1_1}(t), ..., x^j(t), \mathbf{u}^j(t), ..., x^{i_{N^i}}(t))$

A fundamental division between different types of NCS regards the way the control action is generated, and, in general, the overall architecture of the networked system itself. Depending on that, the control system can, in fact, be considered

- **Centralized:** if the control action is computed and applied by a central entity of the network, to which all the agents are connected and transmit their information, receiving the control commands obtained by the center in a single optimization problem that takes into consideration all the objective functions and constraints of the agents.

- **Decentralized:** if each agent has its own controller and solves its own optimization problem without communicating. with other agents. Thus, it only considers its own objective function and constraints and no coupling in the optimization problem is done.

- **Distributed:** the system can be considered an hybrid between the two just depicted categories. The agents communicate and each agent has its own controller, which takes other agents into consideration.

And then, from the latter classification can derive a last subdivision between **cooperative** and **non-cooperative** networks, or, better, agents. Cooperation means that each agent, while computing its own control problem, does not only consider its benefits, but also takes other agents' benefits into account.

To summarize, a graphical classification of NCSs is provided in figure: 2.3.

2.5 Receding Horizon Control algorithms

This kind of control algorithms bases its working principle in predicting the future system behaviour by expecting its future states in terms of the control inputs, optimizing the control inputs over a finite discrete-time prediction horizon $H_p \in \mathbb{N}$, for then apply only the first control step, and repeat the process within the same H_p, with the horizon moving one step forward together with the system itself. Here is where the *receding* horizon comes from.

The most powerful and used control algorithm based on that mechanism is known as Model Predictive Control (MPC). The name is due to the fact that for predict the future of the systems, it uses a model of the system itself: of course, the more accurate and precise is the model, the better will be the prediction. MPC solves, at every cycle, an **optimization problem** aimed to find the best sequence of control inputs to reach the goal. An optimization problem is always defined as follows:

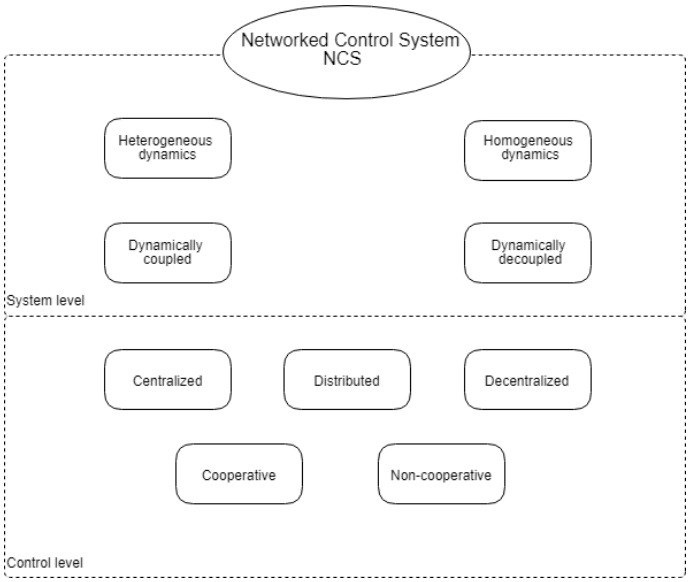


Figure 2.3: Graphical classification of NCSs

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

read as: *the result is the* ***control input*** u *such that the* ***objective function*** J *is the minimum allowable one,* ***subject to constraints*** *due to system's dynamics, constraints on the control input, and constraints on the states.*

The essential elements of an MPC control algorithm are then:

Dynamic model and prediction It utilizes a dynamic model to predict the future system behaviour. There is always a trade-off in choosing an appropriate model. A complex one leads to near optimal control inputs, but at the same time it results in a high computation time of the optimization problem. As MPC is an iterative process and the prediction horizon is limited, it can sustain some model inexactnesses that might be introduced through linearisation or simplifications made during the modelling process. A reasonable rule is to choose the simplest possible model that is accurate enough to make predictions within the defined prediction horizon. Generally, a discrete-time model can be expressed as[26]:

- $x(t+1) = f(x(t), u(t))$
- $y(t) = g(x(t), u(t))$

with states $x \in \mathbb{R}^n$, control inputs $u \in \mathbb{R}^m$, controlled ouputs $y \in \mathbb{R}^p$, states evolution function $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, and output function $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. The model can then be linearized either only around the current operating point or around the trajectory at each operating point in the prediction horizon.

Objective function The objective function of MPC includes the cost to be minimized. It consists of different terms that depend on the chosen MPC algorithm and on the application. The objective function J is defined as

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

where:

$J(\vec{u}(k), \vec{x}(k|k))$ is the objective function

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

$\vec{u}(k)$ 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

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

Constraints One main advantage of MPC is its possibility to respect system constraints in the optimization problem. Constraints on the control inputs often represent physical limits. If the controller does not respect them, the physical system enforces them. In contrast, constraints on the states or the outputs are usually desired constraints. Note that, in NCS, constraints on other agents may be additionally considered. The presence of those constraints, and the facility to include them in an MPC algorithm, makes this control technique an optimal method for dealing with multi-variable constrained control problems in an effective way.

The working principle of an MPC is then explained using figure:2.4. In that a trajectory following problem is being solved. In the upper graph, the current state of the plant is measured, and related point-to-point with the reference ones of the desired trajectory. This is done along the whole prediction horizon, and a proper control input sequence is computed, so that the desired trajectory is reached sooner or later according to the solving criteria. Only the first element of the computed control input sequence is then applied, and the prediction and optimization restarts from a step forward, in the bottom graph. Thus, the optimization problem is solved at every sample time, introducing feedback in the control algorithm.

2.6 Path Plannig

The path planning problem is a really well known task in literature, and many different approaches can be used to solve it, according to the different needs of the robots that has to perform the trajectory. According to a common definition [4]

Definition 2.2. 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.

[4] https://en.wikipedia.org/wiki/MotionPlanning

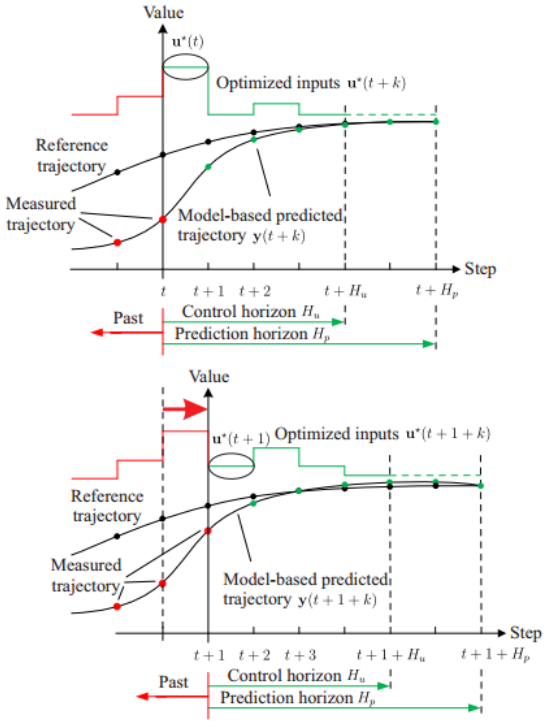


Figure 2.4: Functionality of MPC and the receding horizon principle applied in following a reference trajectory

In other words, given the dynamic of a robot, the environment description and an ordered set of goal states (one or more, doesn't change), the path planner block of the Cloud Based Traffic Manager has to find a sequence of way points on the map that will bring the aircraft from one state to the following, till the last one is reached.
Please notice that, though we are working with Unmanned Aerial Vehicle, thanks to the assumption of not varying the altitude of flight we can treat the path generation task as if it was for a planar robot. Not considering the z-axe obviously have a great simplification impact.
During years, different algorithms were developed to accomplish this tasks. A quite rough but useful classification is:

Deterministic Algorithms Simple to implement, not hard from a computational point of view.

Probabilistic Algorithms Involves high dimensional state space, and its execution time tends to infinite. Practically, if no solution exists it will run forever. [55]

At this point, an high level descriptive introduction to both this classes can be interesting and helpful to understand the choices made by our Path Planning group. Since it's not the main purpose of this thesis to develop new motion planning algorithms, we will not go too much in deep in this section, that will instead better explained by the other members of our research group. Just a brief resume of the most important algorithms and the actual state of the art of our Cloud Based Path Planner will be provided in here, in order to make the reader capable to understand the following sections and especially the Simulations.
In order to properly conclude this introduction to the path planners' world, It's important to underline the main features that we are looking to consider one algorithm better then the others.

As said before, every approach can be considered optimal, according to the context in which it will is used.
Having a look to our environment, its crucial characteristics are:

1. High dimension map. We are going to plan motion over cities (km^2) with a very high resolution. The resulting cost map will be huge and detailed. This implies the need of a light algorithm or the usage of a really great computational power, in order to have a solution in finite time.

2. Changing environment. The obstacles and many of the other risk parameter changes in time, often during the flight. The path planner have to be dynamical, that practically means able to change path in real time.

3. Critical Situation. While running during the flight, the path planner response have to be really fast, in order to prevent collisions. Since we are working with aircraft, a latency in the order of seconds can has catastrophic effects.

4. Drone's Structure. The Path Planner has to consider that the UAV is not a point mass, but instead has its own dynamic and structure. The trajectories have not only to be optimal in theory, but also feasible for the Unmanned System.

Now, let's have a look to the state of the art.

2.6.1 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 [39]. 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 [59]. 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 [**?**][50]. Both of them are similar to D*, but save computational resources and have beter perfomances.

Theta* developed by Nash et al. [44] 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 move 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 [37], 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.

2.6.2 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 [**?**].
Typically, this algorithms works on non convex, 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, always 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 [42]. 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 [47]. 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 [25]. 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.

2.7 Unmanned Aerial Vehicles

2.7.1 Generalities

Unmanned Aerial Vehicles are complex systems made up of many components, both hardware and software, working together for a common result: make it fly. For the purposes of this thesis, we only focused on **quad-copters**, multi-rotor helicopters lifted and propelled by four vertically oriented rotors. They are identical thrusters, 2 spinning clockwise and 2 counter clockwise and arranged so that the resulting torsional moment in null and the vehicle manages to keep the desired orientation. The way they work, or better fly, is depicted in figure:2.5 here below. It clearly shows that every motion of the drone is direct consequence of how each propeller is actuated, then how fast they are singularly spinning, so that the total, resulting thrust is oriented along the desired direction.

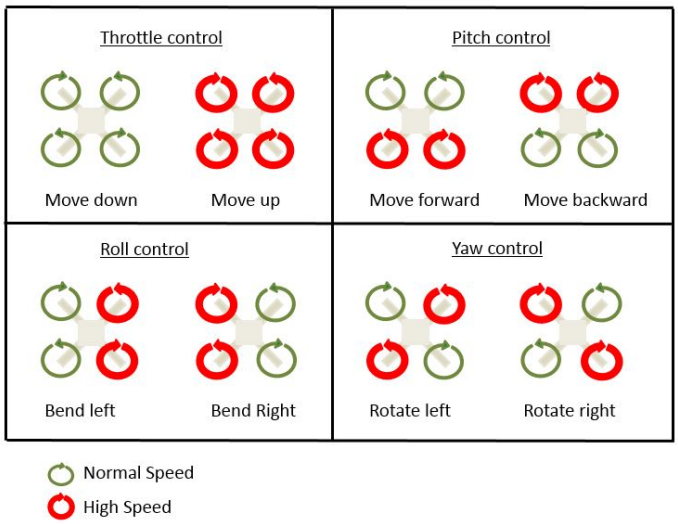


Figure 2.5: Graphical representation of a quad-copter flight principle

A UAV is then a magnificent system where a CPU mounted on an on-board computer gets and processes all the informations coming from sensors and feedback loops, to compute the right control commands to be actuated. It is based on a perfect knowledge of the system physic and dynamics, thanks to which the controller knows how to apply received commands reacting to external disturbances and managing trajectory smoothness and stabilization. Every quad-copter then - like the one used for this project and shown in figure 2.6 - mount a sophisticated software able to manage all the lowest level dynamic-related problems and complications, leaving to the user just the pleasure of flying it. The result is an extremely agile and stable vehicle, able to fly not that long - the average autonomy for commercial medium-size drones is of about 25/30 minutes- but fast enough and being able to stop, almost instantaneously, in a position, just safely hovering, for then move in every direction.

These are for sure the most important characteristics of such a kind of vehicle, that make them totally different from fixed-wings UAVs, plane-type aircraft that have in efficiency, speed and duration of the flight their main advantages. It is mainly thanks to these characteristics that this kind of vehicles fits perfectly with every city-related drones application, where moving within such a complex environment can become too dangerous.

Figure 2.6: An IRIS 3D quad-copter, the model used for the development of this project

When it comes to classification, drones can be divided in many ways, based on physical characteristics, speed, flight altitude and so on. However, there is no a unique standard for classifying UAVs, so that every country has its own classification, that sometimes is not even coherent between vehicles used for different purposes. For the aim of this project, anyway, a preliminary, important classification can be done, regarding the **operational mode**. Depending on how drones are piloted and/or on their level of autonomy, they can be classified ([40]) in:

Remote Controlled The operator, without benefit of video or other sensory feedback, directly controls the vehicle on a continuous basis, from off the vehicle and via a tethered or radio linked control device using visual line-of-sight.

Teleoperated The operator, using video feedback and/or other sensory feedback, either directly controls the vehicle or assigns incremental goals (way-points) on a continuous basis, from off the vehicle and via a tethered or radio linked control device.

Semi-Autonomous The human operator and/or the drone itself plans and conducts a mission. The UAV is capable of autonomous operation in between the human interactions

Fully Autonomous The UAV is expected to accomplish its mission, within a defined scope, without human intervention. Note that a team of UAVs may be fully autonomous while the individual team members may not be due to the needs to coordinate during the execution of team missions.

So far, regarding recreational and commercial drones (which are those this project is related to) the visual line-of-sight has been one of the most limiting criteria for flight permissions. Authorities impose restriction to their operational range, meaning that the vehicle cannot be flown farther than it can be directly seen by its operator. This limitation applies for both autonomous and not vehicles, distorting the meaning of term Unmanned itself, and limiting the potentiality of these vehicles. For the purposes of this project, the attention has been focused on fully-autonomous vehicles, meaning with that vehicle provided of a full featured software on board able to manage everything related to the accomplishment of **unmanned missions**. They consist in the autonomous

computation of geographical trajectories traveled by a Unmanned Systems (US) while performing specific tasks, computed processing only high level commands coming from the operator, managing all the required operations. To do so, a drone should be able to receive commands and messages from far, that a certain **on-board flight controller software** can process and apply. It follows that a specific communication channel and protocol is needed. Among many, for the development of this project the MAVLink protocol has been used.

2.7.2 MAVLink

MAVLink[5][6] is one of the most commonly used **communication protocol** when it comes to Micro Air Vehicles. It is a very lightweight, header-only message marshalling [7] library. Many commercialized drones are already using the MAVLink protocol. Part of the MAVLink ecosystem are, in fact, autopilots like Parrot AR.Drone, ArduPilot, PX4FMU and many others, and sotwares like Andropilot and iDroneCtrl (the most used software applications for mobiles), together with APM Planner, MAVProxy, QGroundControl etc.

It is both used for communication between Ground Control Station (GCS) and Unmanned vehicles, and in the inter-communication of the subsystem within the vehicle.

It works via USB or telemetry (not both at the same time: in that case, USB has the priority), and allows to send a common set of messages, used for transmitting controls, parameters to be set, and a lot of informations like orientation, GPS location, speed etc.

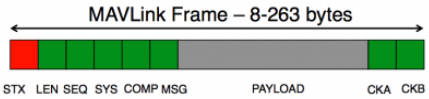


Figure 2.7: MAVLink packet anatomy

As can be seen from 2.7, the MAVLink message is basically a stream of bytes that has been encoded according to certain rules. Each MAVLink packet has a length of 17 bytes, arranged in the following fixed structure:

To summarize:

- Byte 3 and 4 are used by the receiver to transmit the **message ID**, so the source of the message. Any system using MavLink has such a header, and it is used by the interpreter to understand where the message is coming from, and if it can be allowed or not.

- Byte 9 is the **payload**, the actual data we are interested in. It can have variable size, specified in octet 1, range 0-255. Once it has been allowed, the payload data is extracted from the message and put into a packet, a raw data structure based on a "type of information", before being put into an "appropriate data structure" (I.e. attitude, GPS, etc.). The creation of these data structures is done by the MAVLink GUI generator.

[5] http://qgroundcontrol.org/mavlink/start

[6] http://ardupilot.org/copter/docs/common-telemetry-port-setup-for-apm-px4-and-pixhawk.html

[7] In computer science, **marshalling** is the process of transforming the memory representation of an object to a data format suitable for storage or transmission, and it is typically used when data must be moved between different parts of a computer program or from one program to another. Marshalling is similar to serialization, and it simplifies complex communication, using composite objects in order to communicate instead of primitives.

Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, **excluding packet start sign, so bytes 1..(n+6)** Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

Figure 2.8: MAVLink packet description

- Right after the payload, 2 bytes are used for the **checksum**:an error check method used to verify integrity of a transmitted package of data that may be afflicted by alterations during its transmission. It basically consists in adding each bit of the message and transmit the result within the data, so the receiver can perform an integrity check by doing the same sum operation on its own and comparing the results. This sum operation can be done in different ways, leading to different level of robustness.

According to the developers documentation [11], the protocol was totally geared towards transmission speed and safety. Regarding the latter, it allows to check the message content and to detect lost messages still needing only six bytes overhead for each packet. Regarding the communication speed, experimental results are given as in fig 2.9

Link speed	Hardware	Update rate	Payload	Float values
115200 baud	XBee Pro 2.4 GHz	50 Hz	224 bytes	56
115200 baud	XBee Pro 2.4 GHz	100 Hz	109 bytes	27
57600 baud	XBee Pro 2.4 GHz	100 Hz	51 bytes	12
9600 baud	XBee Pro XSC 900	50 Hz	13 bytes	3
9600 baud	XBee Pro XSC 900	20 Hz	42 bytes	10

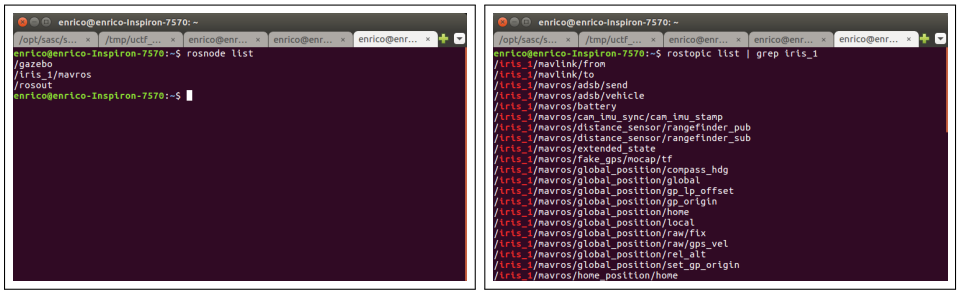
Figure 2.9: MAVLink transmission speed

Interesting works have been developed for transmitting MAVLink messages through 4G data

connection[8]

MAVROS

MAVROS is one of the packages available within the ROS community. This one in particular is aimed to provide a link between MAVLink and ROS, acting like a bidirectional bridge between these 2 *languages*, translating MAVLink messages into ROS messages and vice versa. It creates, in fact, two MAVROS nodes, one for publishing MAVLink stream from autopilot, one for subscribing ROS messages to be sent to the autopilot.



(a) rosnode list command when a mavros node has been launched

(b) rostopic list command when a mavros node is running

It is, then, a package that can be included in a ROS application providing communication drivers for various autopilots with MAVLink communication protocol. It provides a sub-group of the standard MAVLink messages and commands, and can be extended by using plug-ins. A completer description of the package and the set of messages are provided at wiki.ros.org/mavros.

2.7.3 Unmanned Missions

We have already seen how UAVs, 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. With unmanned missions we refer to the set of operations that an unmanned system is able to do on its own for accomplish a mission, from the beginning to the goal. It requires the capability of computing all the needed command to be applied along the whole mission, for travelling a specific geographical trajectory, at the same time optimal an safe.

The technologies that allow an unmanned mission originally arise from military researches, but nowadays they are clearly common also in civilian applications, like cargo transport, agriculture, environmental monitoring and 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 humans 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. In short, fully autonomous vehicles and autonomous unmanned missions are

[8]https://wiredcraft.com/blog/drone-copter-uav-4g-network/

going to assume surely an important role in our cities in the next future, even if they are still being blocked by laws and rules (3).

2.7.4 Arducopter

Speaking of unmanned vehicles, the wish arises to see them accomplishing a precise mission doing everything on their own, without being remote piloted but just receiving, at most, mission's specifications. Here is where an **autopilot software** comes. It provides a complete user interface, allowing the operator to send high-level command, like *takeoff*, *land*, fly to a certain *waypoint* and so on. In particular, we came across **ArduPilot** software: it is an open source, unmanned vehicle **Autopilot Software Suite**, originally developed by hobbyists to control model aircraft and rovers and then evolved into full-featured and reliable autopilot used by industry, research organizations and amateurs, running on a wide variety of hardware platforms, becoming one of the most used autopilot software. It is capable of controlling different types of autonomous vehicles, just changing the **firmware** and few settings, providing a large set of general and model-specific features. The version for copters is called **ArduCopter**[9], and its documentation can be found at ArduPilot website.

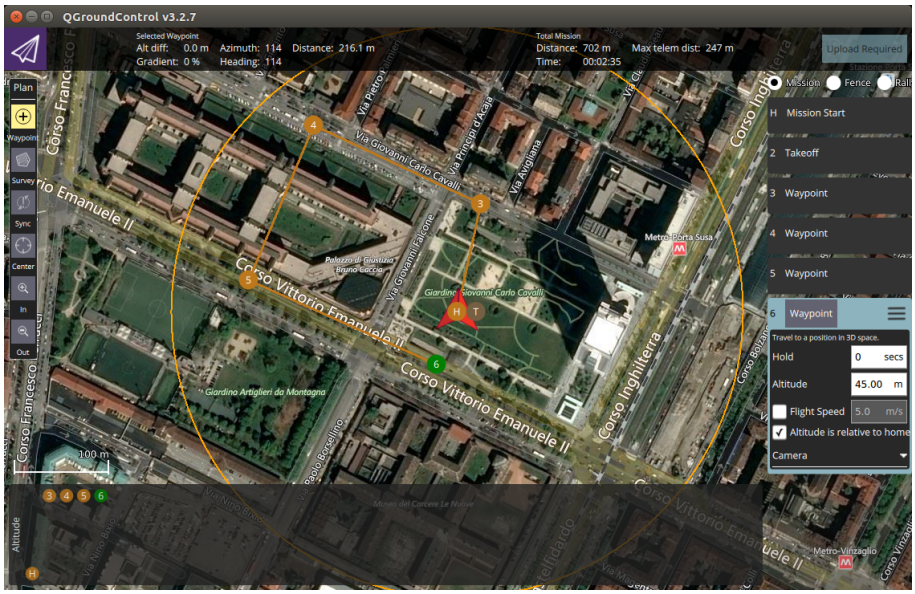


Figure 2.11: QGround mission interface

ArduPilot **adopted a subset of the MAVLink protocol command** and command-parameters that are most relevant and meaningful for each of the vehicles, according to the vehicle type, and defined a certain number of *mav_cmd* messages. Unsupported commands that are sent to a particular autopilot will be simply dropped. If a command or parameter is marked as supported, then it is likely (but not guaranteed) that it will behave as indicated quite accurately. Otherwise, when a command or parameter is not listed (or marked as not supported) then it is extremely likely that it is not supported on ArduPilot. The 3 types of commands that can be used within missions are:

[9] http://ardupilot.org/copter/

- **Navigation** command, used to control the movement of the vehicle, including take-off, moving to and around waypoints, changing altitude, and landing.
- **Do** command used for auxiliary functions that do not affect the vehicle's position (for example, setting the camera trigger distance, or setting a servo value).
- **Condition** command, used to delay DO commands until some condition is met, for example the UAV reaches a certain altitude or distance from the waypoint.

During a mission at most one "Navigation" command and one "Do" or "Condition" command can be running at one time. A typical mission might set a waypoint (NAV command), add a CONDITION command that does not complete until a certain distance from the destination.

2.8 Risk

Final aim of the project is to build up a networked system able to assure that unmanned vehicles can accomplish missions in the city airspace guaranteeing a certain level of safety. Safety is a complex concept, coming from the evaluation of several aspects related to drones flight, ending in the assumption that a certain system can be defined safe if a definite threshold of safety is respected. To do so, a standardized evaluation method is required, that can be found in the opposite of safety: risk. Although there is not yet a commonly accepted definition of risk, speaking about UAVs we can define it based on a widely held concept in the field of traditional avionics:

Definition 2.3. 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.

the term risk is referred to the time frequency in which the drone causes deadly (or very serious) injuries to people on the ground [32]. 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 UAVs 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 modelling the risk, each one with a different level of accuracy. Beyond the differences, there are four main standard criteria that has to be kept into account in every risk model, including ours [38]:

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

The main goal of our analysis is to determine whenever an UAV 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 or more countermeasures in order to increase the safety level of the mission. The latter aspect is usually called *risk management*. According to [38], 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 4.4 the qualitative and quantitative approach to risk we used, entering in details in [48].

We can therefore define the main hazards that may affect unmanned missions, that can be categorized into [32]:

Drone involuntary mobility 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 includes accidents due to in flight collision, and may concern two or more UAVs or fixed obstacles, such as buildings. Typical risk analysis for 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 fundamental in the risk assessment of this project

Early flight termination this category includes all the hazards due to a loss of control of the UAVs flight, and the consequent anticipation of 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.

Evaluating risk 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 [32]. 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 analysing 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. A proper analytical expression could be:

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

Where:

$f_{F,UFE}$ fatality'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 UASs crash 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 [32], 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 UAVs 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 similar 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 2.12 shows just how $P(fatality|exposure)$ varies with varying kinetic energy and sheltering factor.

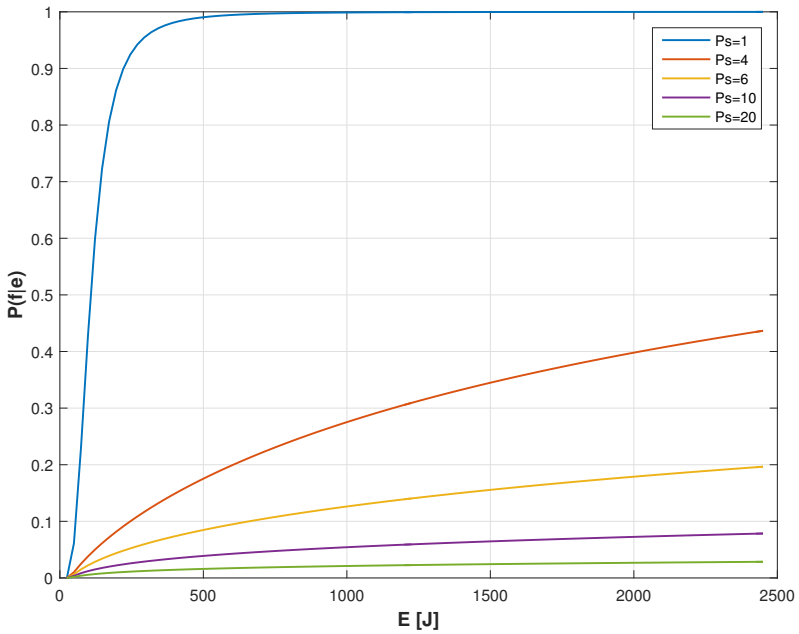


Figure 2.12: $P(fatality|exposure)$ evolution respect to Kinetic Energy and Sheltering P_s

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 [32]:

Definition 2.4. 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, it will be hired $P_S \in [0; 10]$.

2.8.1 Mid-Air Collisions Risk Modelling

Now let's focus on to the analysis of 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 UAVs. According to [32], the frequencies of fatalities due to in flight accidents can be modelled 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 UAVs crash will suffer fatal injuries

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

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 UAVs can have with: one or more other UAV, 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 UAVs 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 analysed. We underline that at this stage we are interested in modelling 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 4.6 Different techniques have been developed to estimate correctly the value of f_{UAV}. According to [61] and [32], 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 [32], [61]. 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 4.6.

2.8.2 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 mid air. 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 mid air. 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.

Chapter 3

UTM - UAVs Traffic Management

With the term Unmanned Aircraft Systems Traffic Manager (UTM) we refer to an ecosystem enabling safe and autonomous UAVs operations in civilian low-altitude airspace. It can be thought as separate but complementary to a classic Air Traffic Management (ATM) used for standard planes and copters, with the difference that the size of the crafts is hundred of times smaller and the density of vehicles, together with their dynamism, dozens of times higher. Air traffic management is about the procedures, technologies and human resources assuring that aircraft are guided safely through the sky and airspace is managed to accommodate the changing needs of air traffic over time. An interesting definition of what an UTM is and should be able to do is given by NASA directly[23]:

> The UTM system would enable safe and efficient low-altitude airspace operations by providing services such as airspace design, corridors, dynamic geofencing, severe weather and wind avoidance, congestion management, terrain avoidance, route planning and re-routing, separation management, sequencing and spacing, and contingency management.
>
> 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 of course requires that only authenticated UAVs could operate in the airspace.

Thanks to a proper UTM then, a co-operative interaction between drones' operators and the National Flight Authority can be established, in order to have a real-time (or near real-time) organization of the airspace and exploit all the advantages coming from these small vehicles while assuring the safety for people on the ground.

Different national flight authorities, together with research groups, universities and companies, are working in that field, with the aim of defining standards, best practices, procedures and everything needed to set up the UTM of the future. Many countries are involved in this topic, working alone according to their specific needs or together in associations like SESAR - Single European Sky ATM Research - or ICAO - International Civil Aviation Organization, a United States specialized agency - to build up international standards and a common policy.

Main examples of agencies and/or collaborations working in that direction are

- U-space: is a project sustained by the EU in cooperation with a private company, that have built up the Single European Sky ATM Research (SESAR) joint undertaking, aimed at the creation of a clear framework for UTM systems in Europe. More at [19]

- Europe is working in parallel in other solutions, coordinated by the European Aviation Safety Agency, that sent out a concept paper [18] where a risk based apoproach to regulation of unmanned aircraft is presented.
- Italy on its own is moving forward in framework definition and UTM development, with the NFA - Ente Nazionale di Assistenza al Volo (ENAV), in cooperation with the civil NFA - Ente Nazionale Aviazione Civile (ENAC) - financing projects in that direction and starting calls for demonstrations [34], looking for industrial partners.
- Nasa, of course, is working in that direction too. Building on its legacy of work in air traffic management for crewed aircraft, NASA is researching prototype technologies for a UAS Traffic Management (UTM) system that could develop airspace integration requirements for enabling safe, efficient low-altitude operations. more can be found navigating in [23]

According to the ICAO ([5]), an UTM framework will include many components facing different tasks, the main of which can be enumerated as follows:

1. Registration system: to allow remote identification and tracking of each UAS.
2. Communications systems: to assure every element of the network is properly. connected, transmitting data and receiving commands, managing problems like real-time concept and data loss.
3. Rules and traffic management: for an effective and organised management, some fixed rules must be defined, imposing some behavioural norms, both ethical and operational. With basic rules, like a priority order to be respected, most of the traffic management issues can be easily solved.
4. Geofencing-like system: to avoid drones to be flown over restricted areas
5. Flight operations' automation service: drone operations management, including flight planning, flight approval, tracking, and interfacing with conventional air traffic control.
6. Collision avoidance System: to make vehicles fly safely within the same airspace, avoiding collision between them as well as with buildings and whatever.
7. Risk assessment: everything needed to measure hazardousness of unmanned operations and build up a metric for minimizing the risk for humans and buildings must be done.

The final system can be seen as a **networked control system** like it has been defined before in section 2.4, with networked drones being active agents of the network.

3.1 UTM - rules, limitations and registration

«««< HEAD UAVs open a very wide range of new issues that modern governments have not previously had to face. By giving the possibility to fly objects into the hands of virtually anyone, large numbers of safety, legal and privacy issues appeared. It necessary follows that a strict, clear legislation is needed. Drones usage is growing in every country, forcing common flight authorities to expand their interests and jurisdiction, or sometimes brand new agencies to born. Many times, national authorities cooperate with private or public companies, research groups and universities, nation ally r internationally, to build up, together, a new ecosystem able to host and manage drones. ======= UAVs open a very wide range of new issues that modern governments have not previously had to face. By giving the possibility to fly objects into the hands of virtually

anyone, large numbers of safety, legal and privacy issues appeared. It necessary follows that a strict, clear legislation is needed. Drones usage is growing in every country, forcing common flight authorities to expand their interests and jurisdiction, or sometimes brand new agencies to born. Many times, national authorities cooperate with private or public companies, research groups and universities, nationally or internationally, to build up, together, a new ecosystem able to host and manage drones. »»»> 28b8d3064afd7f68c0f80d818a1e9b90ff2ed385

In this chapter, an analysis of how issues coming from drone network organisation are being faced nowadays. Underlining that this is a worldwide spread topic, and that different countries are coming out with different ideas, approaches and solutions and that the best one may will come from a perfect mix of many, our focus, for the purposes of this project, will be on Italy and on international organisations involving our country.

3.1.1 Italy - ENAC

«««< HEAD Italy, as every other country, is trying to face the development of drones market on its own too. The national flight authority is ENAC, that, step after step, moved from managing only planes traffic to regulate unmanned vehicles too, creating a brand new exclusive branch of regulation and a subsection on its official website, intended to be the reference point for all the UAVs operators that must declare their vehicle and intentions [16]. ======= Italy, as every other country, is trying to face the development of drones market on its own too. The national flight authority is ENAC, that, step after step, moved from managing only planes traffic to regulate unmanned vehicles too, creating a brand new exclusive branch of regulation and a subsection on its official website, intended to be the reference point for all the UAVs operators that must declare their vehicle and intentions [1]. »»»> 28b8d3064afd7f68c0f80d818a1e9b90ff2ed385

After a though and confused initial period (as in every country because of the news introduced by this technology), a preliminary organisation has been set up, laying the basis for the future Air Traffic Management. Drones have been officially classified depending on their mass on the take off (depending if it is less or more than 25 kg) and on the nature of the operation, divided in critical and non-critical. With non-critical they define those *"specialized operations are VLOS operations which do not overfly, even in case of malfunctions and/or failures, congested or urban areas, and critical infrastructures"*. Prerequisites and certificates slightly vary between the different classes, and an up-to-date documentation can be found in [34], but, in general:

- every operator should be legally identified, passing through a process aimed at obtaining an electronic identity as recognised UAV operator [2]
- specific driving license is required to obtain the authorization to fly, even if in the last period a new class of recreational drones is rising up that do not require a license to be flown;
- Remoted Piloted Aerial Vehicle (RPAS) must be first registered and certified, according to specific standards and satisfying certain requirements, regarding manufacturing and equipment;
- during operations, pilot shall be clearly visible and immediately identifiable by proper means, that include a high visibility vest with the sign “RPA pilot", and that on every vehicles an identification plate must applied;

[1] https://www.enac.gov.it/La_Regolazione_per_la_Sicurezza/Sistemi_Aeromobili_a_Pilotaggio_Remoto_(Droni)/index.html

[2] http://www.quadricottero.com/2018/03/enav-in-italia-la-registrazione-di.html

- for non critical operations, the capability of the RPAS operator to comply with obligations is declared by the operator itself in a form and manner established by the regulation, on its own risk and responsibility, so that a specific authorization is not required;

- before starting a critical operation instead, an operator should apply for and obtain an authorization; critical operations can be carried out when the safety level is adequate for the risk posed by those operations, having regard to the general aviation operations;

- should be equipped with all the necessary requirements for the mission, including those needed for being compliant with common rules and flight operational modes;

- every operator should behave in respect of the common operational modalities defined within the regulation.

Authorities are always on the move for changing and adapting the actual regulation, going in the direction of allowing a sane development of a drone network, compliant with all the needed safety, privacy and regulation requirements.

ENAC is also working with different level of cooperation together with public and private companies, and an interesting example is the one involving the Polytechnic of Turin and TIM.

3.1.2 European Union - EASA

The European Aviation Safety Agency (EASA) is an agency of the European Union (EU) with regulatory and executive tasks in the field of civilian aviation safety. Based in Cologne, Germany, the EASA was created on back in the 2002, and it reached full functionality in 2008, taking over functions of the Joint Aviation Authorities (JAA). European Free Trade Association (EFTA) countries have been granted participation in the agency, that counts now on 32 member states.

Point of reference of the European policy regarding UAVs is the on-line portal[3] of the EASA itself.

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 (low risk): safety is ensured through operational limitations, compliance with industry standards, requirements on certain functionalities, and a minimum set of operational rules. Enforcement shall be ensured by the police.

- **Specific operation** category (medium risk): authorization by National Aviation Authorities (NAAs), possibly assisted by a Qualified Entity (QE) following a risk assessment performed by the operator. A manual of operations shall list the risk mitigation measures.

- **Certified** category (higher risk): requirements comparable to manned aviation ones. Oversight by NAAs (issue of licences and approval of maintenance, operations, training, Air Traffic Management (ATM)/Air Navigation Services (ANS) and aerodrome organisations) and by EASA (design and approval of foreign organisations).

[3] https://www.easa.europa.eu/easa-and-you/civil-drones-rpas/drones-regulatory-framework-background

Based on the market's needs, priority has been given to the development of a regulation for operations in 'open' and 'specific' category. The development of the regulation framework for operations in 'certified' category is planned for 2018 and 2019. Following the publication of a 'Prototype' regulation for the 'open' and 'specific' categories in August 2016, EASA drafted and published on 4 May 2017, NPA 2017-05 with the significant input provided by an expert group. It was complemented by a detailed impact assessment.

During the four months consultation period, more than 3 700 comments from around 215 commenters were received and carefully evaluated. Opinion 01/2018 was published on 6th February 2018, proposing a new European regulation for UAS operations in 'open' and 'specific' category, taking into consideration the developments in the international arena.

The main aspects of the proposed regulation are:

- It provides a framework to safely operate drones while allowing this industry to remain agile, to innovate and continue to grow. The risk posed to people on the ground and to other aircraft as well as privacy, security and data protection issues created by such drones are also taken into account.
- It defines the technical and operational requirements for the drones. Technical requirements refer for example to the remote identification of drones. Operational requirements refer among others to geo-awareness, a system that inform the remote pilot when a drone is entering a prohibited zone. The proposal also addresses the pilots' qualifications. Furthermore, drone operators will have to register themselves, except when they operate drones lighter than 250g.
- It breaks new grounds by combining Product legislation and Aviation legislation. Indeed, design requirements for small drones will be implemented by using the legislation relative to making products available on the market (the well-known CE marking). The standard CE mark will be accompanied by the identification of the class of the drone (from C0 to C4) and by a do's and don'ts consumer information that will be found in all drone boxes. Based on the drone class an operator will know in which area he can operate and what competence is required.
- It allows a high degree of flexibility for EASA Member States; they will be able to define zones in their territory where either drones operations are prohibited or restricted (for example to protect sensitive areas), or where certain requirements are alleviated. For operations posing higher risks, an operational risk assessment will define the requirements that the operator needs to comply before flying the drone.

The proposal also provides special alleviations for people flying model aircraft (also considered to be drones) to recognise the good safety records in aero modelling by identifying three options:

- Member States may issue a special authorisation to model clubs and associations defining deviations from the UAS regulation;
- Operations can be conducted in specific zones designated by Member States; or
- Operations can be conducted in the open category according the operational limitations defined for one of the Subcategory (A3).

In accordance with the proposed new Basic Regulation, for which a political agreement between the Council, the European Commission and the European Parliament was reached on 22 December 2017, the competence of the EU has been extended to cover the regulation of all civil unmanned aircraft systems (UAS), regardless of their maximum take-off masses (MTOMs). It lead to the publication of Opinion 01/2018 [24]. The objective of this opinion is to create a new regulatory framework that defines measures to mitigate the risk of operations in the:

- 'open' category, through a combination of limitations, operational rules, requirements for the competency of the remote pilot, as well as technical requirements for UAS, such that the UAS operator may conduct the operation without prior authorisation by the competent authority, or without submitting a declaration; and
- 'specific' category, through a system that includes a risk assessment being conducted by the UAS operator before starting an operation, or an operator complying with a standard scenario, or an operator holding a certificate with privileges.

Moreover, this Opinion is intended 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.

The proposed regulations will provide flexibility to Member States, mainly by allowing them to create zones within their territories where the use of UAS would be prohibited, limited or, in contrast, facilitated. Pursuant to the new Basic Regulation, market product legislation (CE marking) ensures compliance with the technical requirements for mass-produced UAS operated in the 'open' category. Two acts are proposed that follow different adoption procedures, as defined by the new Basic Regulation: a delegated act that defines the conditions for making UAS available on the market and the conditions for UAS operations conducted by a third-country operator, and an implementing rule that defines the conditions to operate UAS and the conditions for registration. The proposed regulatory framework is expected to increase the level of safety of UAS operations, to harmonise legislation among the EU MSs, and to create an EU market that will reduce the cost of UAS and allow cross-border operations.

EASA then fixed the next steps toward a safe and well defined UAVs network, as shown in the graphical timeline 3.1

Figure 3.1: EASA rulemaking process milestones

Meanwhile, in the end of November 2017 EU negotiators reached a provisional deal on new rules for drones, leaving the door open to several details to be determined later. Representatives of member states, the European Parliament (EP), and the European Commission decided that **a certificate may be required** for some civil uses of drones. The criteria for certification are left open though, and are to be determined through a separate process known as *comitology*. The commission will be tasked with drawing up specific criteria determining when drones and their use should be given a stamp of approval.

The EU parliament published a press release saying that *"EU countries will need to ensure that operators of drones that can cause significant harm to people, i.e. by crashing into them, or present risks to privacy, security or the environment, are registered"* and *"these drones will also need to be individually marked to be easily identified."*

3.1.3 United Nations - ICAO

An interesting project is the one held by the United Nations, which aims at coordinating the work of member States regarding UAVs, collecting ideas, best practices, common rules and differences. the International Civil Aviation Organisation wants, through its community[**?**], provide helpful tools to assist States in realising effective UAVs operational guidance and safe domestic operations. It also contains a database of regulations of the member States, supposed to be an up-to-date source for changing ideas and best-practices.

3.2 Localization and identification

Necessary step for creating an efficient air traffic management system is, of course, the knowledge of every vehicle enrolled. It means not only being able to localize them within the controlled airspace, but also to uniquely identify the vehicle and its legal owner, to which any responsibility will fall on.

With the number of UAVs flying in our cities is (and will keep) increasing fast, and the possibility for virtually anyone to fly its drone, it seems mandatory think about a proper and efficient way to regulate the air traffic and assure the identification of each vehicle.

It follows that, when building a well structured UTM system, some on-board devices shall be asked for being applied, and a precise registration and identification process shall be followed. This would be aimed at collecting data within a centralized and well-informed database, able to link all the relevant data to a physical object flying in the network, its legal owner and its actual operator.

To do so, many technique are already being applied, similarly to what done for other Traffic Management systems like the aviation one, while some others are still in the thinking process, while there is not a unique standard yet, and international associations are on the wait for creating a common regulation.

According to the last updates and directives of the ICAO, in the aim of finding the best way to organise the fly-zones, identify the vehicles and properly adapt to the coming scenario, they started a "call for demonstration". The aim is to find the best fitting identification system, able to allow a formal registration of each UAV and transmit real-time data about the flying drone and its owner/operator, in a way that is as cheap, fast and efficient as possible.

But in general, different solutions for real-time or general localization and identification have been tested and evaluated. They will be analysed in the following.

QR code some UTM require the application of a QR code on the body of the aircraft. It is a passive method, since it is just supposed to be an uniquely way to distinguish different vehicles quickly linking them to all the related informations stored on-line.

License plate similarly to what happens for cars. This plate is composed by a LED array blinking in a unique way identifying the vehicle, a small Ultra High Frequency transponder and a position logger, which acts like a kind of black box. We think the produced logs could be analysed by the Cloud System to monitor potential violations of rules.

Transponder very commonly used devices in aviation, and the name stands for abbreviation of "transmitter responder". It seems they will be required on board of every UAV, so that

even ENAC, for adapting to European regulations, will start requiring forcing every operator to mount such a device on board [15]. Transponders generate, either in response to an interrogation or automatically, a *squawk* signal containing data about the aircraft generalities, its altitude, travel speed and its identification code, to the Ground Control Station system or to other aircraft. Depending on the case, there are many types of transponder used in different ways, and their usage on small vehicles like a UAV, with all the problems related to payload capability, dimensions and power consumption, is being studied. In civil aviation, the information to the ground is commonly provided in response to an interrogation by systems such as Secondary Surveillance Radar (SSR) or multilateralists systems. There are then two main interrogation-reply modes: Mode A/C and Mode S. In response to Mode A interrogations the transponder transmits an identity code for the aircraft in the octal range 0000-7777, with some codes allocated to transmit specific emergency situations. Mode C provides the aircraft's barometric altitude in 100 feet increments. Mode A/C operation has several technical limitations such as its inefficient use of the radio spectrum and the limited number of Mode A codes available. Mode S was developed to overcome the limitations of Mode A/C. In particular, Mode S has over 17 million unique 24-bit aircraft addresses, altitude reports in 25 feet increments and "selective interrogation": unlike traditional SSR stations which elicit multiple replies containing the same information from all aircraft within their range, Mode S makes selective interrogations of each specific aircraft. 'All call' interrogations are also made to identify new aircraft to be interrogated.

ADS-B this is a mode of operations of a certain kind of transponders. An Automatic Dependent Surveillance - Broadcast capable transponder is able to "broadcast" information to ground stations and other aircraft without interrogation, but in a continuous and automatic way

Sim localization Subscriber Identification Module (SIM) is an integrated circuit that is intended to securely store the International Mobile Subscriber Identity (IMSI) number and its related key, which are used to uniquely and internationally identify and authenticate subscribers on mobile networked devices. Since we are speaking of NCS made of vehicles connected via 4G/5G, it is logical think about the presence of SIM on board, and they can be localized and identified in the same way it happens daily with phones.

Many methodologies are applied by different national authorities, and the best one may come from a mix of them, finding the most effective way to identify vehicles.

3.3 Communication and control

In order to build up a city UTM system able to coordinate vehicles and, when needed, pilot them efficiently, a good and stable communication link is needed between the Control Station (CS, normally referred to as Ground Control Station GCS in ATM systems, and as Cloud Control Station CCS in our project) and the single drones.

So far, the remote control of unmanned vehicles for what regards civil and commercial usage has been mainly conducted from the ground by means of a controller, connected with the UAV through different wavelengths signals. The most commonly used frequency is the 2.4 GHz, which is the same frequency that wireless computer networks work in. Direct consequence is that they may suffer interferences by other common devices, that may seriously compromise the flight operations, with several incidents happened in dense housing areas reporting the loss of control over the flying objects. This is for sure a huge problem in developing a smart city drone network, together with the fact that they can even interfere with their own on-board systems. Drones with embedded camera, for example, show this issue, with controller and recording system working at same or

near the same frequency. In order to avoid the entangling of frequencies in the same band, in some cases the 5.8 GHz frequency is used as well. A worldwide spread drone model using this trick is the DJI Phantom model. These 2 radio frequencies are often used simultaneously in the same drone in order to transmit control and video signals using two different channels. Then, when the 2.4 GHz is used to fly, the 5.8 GHz RF is used for the FPV (First Person View), or video transmission anyway.

Both of these radio frequencies are considered Line of Sight (LoS) in that they do not work when any barriers are in-between the user and the quad-copter. Current rules and best practice are then imposing drone to be flown not any farther than the operator can directly see it, according to the Visual Line Of Sight (VLOS) rule.

Even if this limitation has been successfully overcome in the military field, where unmanned vehicles can be piloted far **Beyond the Visual Line Of Sight (BVLOS)**, it holds for what regards civil and commercial usage, blocking the greatest part of UAVs applications. This is mainly due to the unreliability of such a remote piloting mode, that even adds more difficulties in censusing and controlling the operators. But also to the unreliability of such technologies adapted to civil vehicles flying in complex city environments, and the dependence on communication link integrity and speed.

The challenge is than to find a way to allow BVLOS missions in city environment, and here is where a fast and reliable next generation mobile network arrives. We can already count in fact on the 4G connection - fourth generation of broadband cellular network technology - well spread over advanced cities' environment, that is the framework in which a UTM should work. Nowadays, 4G is a quite sold reality, assuring stable connection with a measured average download rate speed of $40\,mbps$ for what regards 4G-LTE Advanced[2], with a mean latency of $45\,ms$. The next generation - 5G - is about to come, with a theoretical latency of only $1\,ms$. It fosters every kind of research in many fields, opening up thousands of new possibilities, especially regarding Cloud Robotic and IoT applications.

A technique that is being spread in the last period and is getting more and more attention regarding drones remote controlling consists in exploiting 4G connection for establish a communication link between the on-board autopilot software (Pixhawk among others) by coupling it with a micro-computer like Raspberry [4] or Arduino, able to connect interface MAVLink with external world and communicate via 4G [5].

[4] https://wiredcraft.com/blog/drone-copter-uav-4g-network/

[5] https://www.youtube.com/watch?v=DGAB34fJQFc

Chapter 4

CBUTM

In chapter 3 we have seen what an Air traffic Manager is, what are the tasks it has to deal with and finally how these tasks has been accomplished (or thought to be accomplished) according to the literature.

In this chapter, the way we decided to implement a traffic management system will be described, giving to the reader an overall view of our system and how it works, then showing the strategy we decided to adopt regarding every issue making up the problem, in order to reach the prefixed goal.

Notice that the whole project is meant to be the result of the cooperation between different colleagues within the JOL - Joint Open Lab - and then it must be seen as a whole. The results of the work will then be shown in its totality among the three thesis, with different level of details according to the main subject of the work of each of us.

4.1 Overview

The proposed **Cloud-Based UAVs Traffic Management** system, namely CBUTM, is an **air traffic management** solution exploiting the advantages offered by the Cloud Robotic paradigm. Goal of the project was the creation of an organized networked system for managing a well structured low-altitude city airspace, assuring at the same time safe and effective UAVs operations. The final idea is to provide a unique platform where users, costumers and authorities can meet and cooperate, and where rules, procedures, rights and duties can be clear and well defined.

As also analysed in section 3, we identified the main issues a proper UTM system should be able to solve, summarising them as:

- **Registration**
- **Risk assessment**
- **Path planning**
- **Path following**
- **Monitoring**
- **Collision avoidance**

How we decided to face each of these issues - of course from the cloud side, leaving the bureaucratic stuff to whom it may concern - will be explained and analysed in the following sections, while the way we implemented our system will be shown in details in three different thesis. It is due

to the way we worked on this project, laying more than the basis for building up a Cloud-Based UTM in ROS environment.

The working principle of the Robot 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. A sketch representing the overall architecture of the system is shown in the figure 4.1

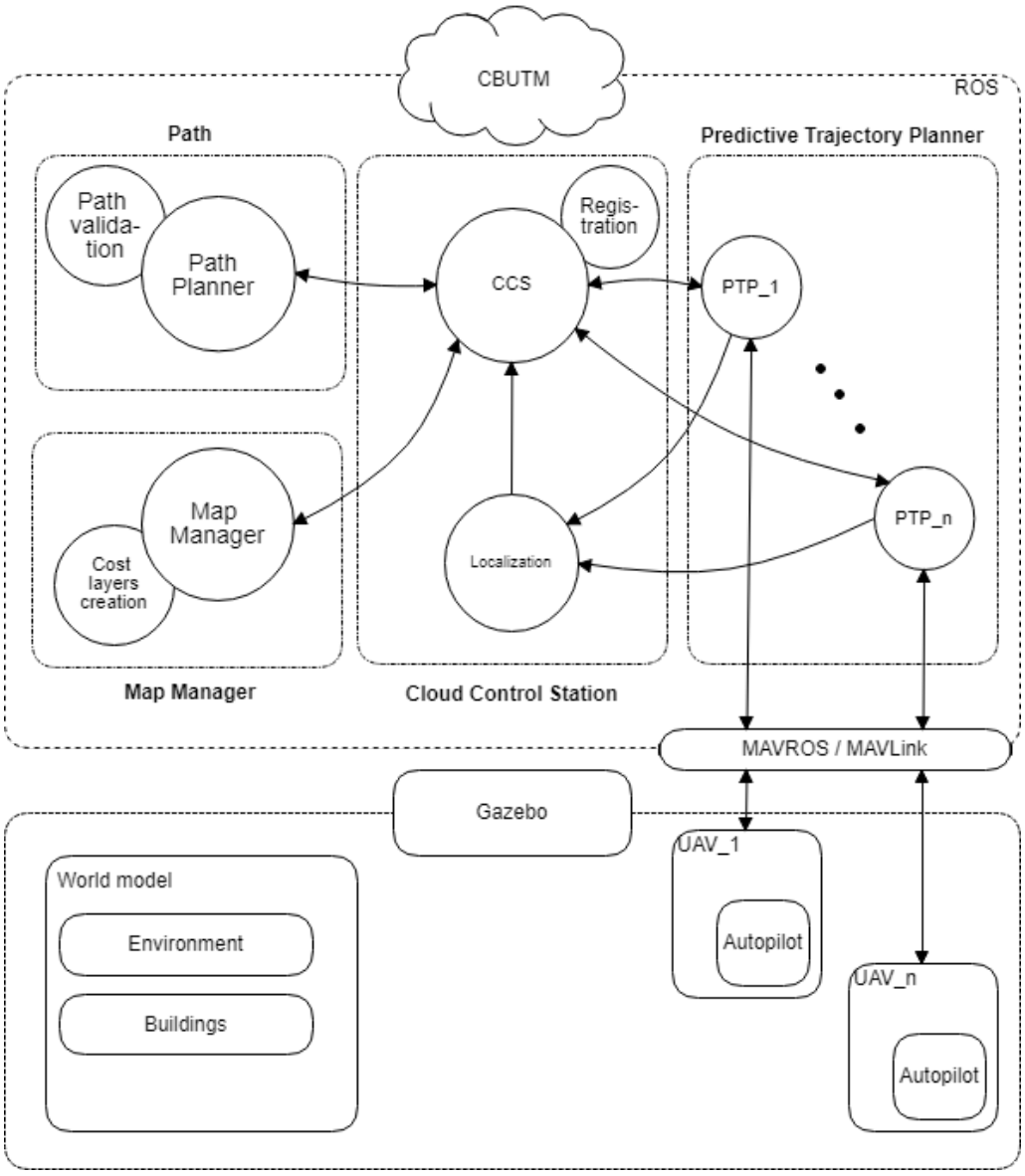


Figure 4.1: Graphical representation of CBUTM working principle

As can be seen from figure 4.1, our Cloud-based UAVs Traffic Management system can be thought as the cooperation of 4 main entities, with specific roles and functionalities, 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. They can be enumerated as:

Map Manager, MM aimed at the creation and management of cost layers and risk map, applying the rules of a standardized risk assessment

Cloud Control Station, CCS meant to be the equivalent of a Ground Control Station: core of the system, is aimed at catching and collecting informations of all the agents acting within the network, putting in contact different elements and managing data streams

Path Planner, PP receiving initial and goal position, it applies a specific path planning algorithm for computing optimal paths on the risk map generated by the MM

Predictive Trajectory Planner, PTP actually a group of similar nodes, images of physical drones flying in the city airspace, aimed at being their in-cloud networked controllers directly linked to the vehicles.

These four entities work in the cloud, interacting with drones flying either in real world or in a gazebo simulated environment.

Important for the aim of the project has been a well-defined and standardized **risk assessment**, described in 4.4 and deep in details in [48]. Thanks to that, risk and safety requirements of unmanned missions can be coherently measured, so that any others considerations can be done accordingly to that, assuring safe and effective UAVs operations in cities environments. On the basis of this risk assessment, a **risk map** has been built up, where a certain cost - representing the risk for a vehicle to fly over that specific area - is assigned to each block of a grid, computed as superimposition for different kind of data (4.4). It gives then the possibility to compute **optimal paths** for every unmanned mission within the city map, as will be seen in 4.5. Critical phase of the project has been then the choice for the design of the system's **architecture**, in order to make it as light, reactive and effective as possible. A whole structure has been then built up in the ROS environment, managing **data streams and connections** within the different elements of the network, and setting up some **procedures** for keeping it organised and operative, able to adapt to actual traffic conditions **tracking** and managing the active vehicles. This will be the main focus of this thesis, and the working principle of CBUTM will be explained in the following sections (especially in those directly regarding the CCS -4.2, 4.3 and 4.7), entering in detail for what regards the implementation of the core in 5. Once a proper network has been set up, final important step has been to be able to pilot the enrolled vehicles, remote-controlling them from the cloud and coordinating their motion for avoiding collisions. This part will be analysed in 4.6 and 4.8, where the results will be introduced, while another thesis [52] will enter more in details about how we implemented a collision avoidance system.

But before entering in details of how every elements of the system works, we would have a look to the flow-chart in figure 4.2, for having a quick understanding of how CBUTM operates. The steps involved in a standard mission process are shown, from the moment of its definition to the goal.

They can be summarised as follows:

1. Supposing a user wants to fly his drone in a city's airspace in which CBUTM is working, he should first ask for its **drone's cloud image** to be created. It means that for each vehicle flying within the network, a specific **ROS node** is created within the cloud, launched with the proper parameters accordingly with its characteristics. This node runs, as we justified considering different system architecture we could have used (2.4), in the cloud, and is directly linked to the real drone by means of the MAVROS/MAVLink bridge we spoke about in **??**.

2. First thing the newly created node will do is calling the **registration service**. It is provided by a specific node, the Cloud Control Station, the functionalities of which will be summarised and explained in details in the following sections. During this process, vehicle's informations

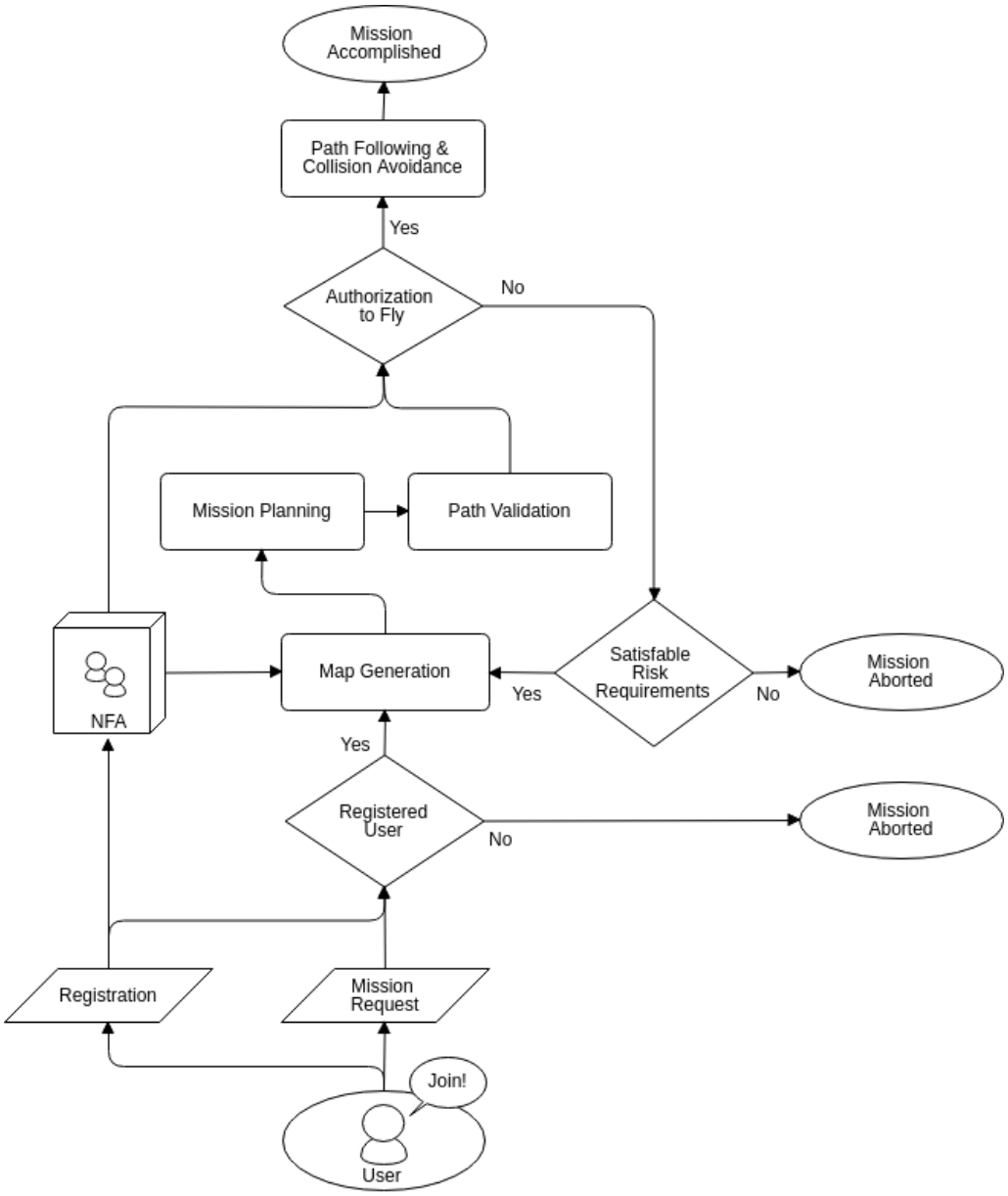


Figure 4.2: CBUTM operational flow chart

are collected within specific database structures, and analysed while forwarded to the National Flight Authorities (NFA). Aim of this service is to set up the structure needed for the drone to be monitored and safely flown within the controlled airspace.

3. Fundamental step is the creation, for each vehicle, of a specific **UAV risk layer**. This is done by the **Map Manager node**, that provides a special service called by the CCS during the registration service routine, which is performed for every drone. This layer is tailored on every vehicle, accordingly to its characteristics and parameters. Together with the other

layers (ether static or dynamic) already stored in the network they make up the final **risk map** the controller of the drone will refer to.

4. Once the drone has been properly inserted in the network, it can proceed asking for its mission. The starting and goal position are then transferred to the **Path Planner node**, which will use the specific drone risk map for computing an **optimal path**, looking for the trade-off between safety and efficiency, with the former being, of course, privileged.

5. A redundant safety check has been then introduced, aimed to validate the computed path according to the safety requirements imposed by the local NFA. The results of this check can be 2: either the path is accepted as it is, or it is rejected because of unacceptable hazard level. In this case, the path validator can, if possible, modify some parameters of the risk map to force the path planner in finding a better solution, or discard the mission as a whole.

6. Once the computed path for a mission has been accepted (either at the first attempt or after a re-computation), the drone obtains its authorization to fly, and the mission can start. It means that the drone controller, running on the cloud, sends to the autopilot the proper command inputs to follow the prefixed path, while a collision avoidance routine assure the safeness of the flight according to the procedures that will be analysed next

The way a user joins the network consists in some preliminary steps that must be performed by the user itself, according to the regulations imposed by the authorities. It should include the declaration of the legal owner, the specifications of the vehicles (dimensions, type, model, weight) and all the details defining the mission he wants to fly. When the application has been accomplished, a customized launcher (figure 5.1) should be created, that can be run directly in cloud, specific for that mission. How the system will use these parameters for organising the airspace accordingly will be analysed in the next session, and shown in details in 5.

It invokes the execution of the Predictive Trajectory Planner node, that will be the one triggering all the in-cloud operations introduced here above, and explained in details in the next sections.

Everything regarding the operations directly performed by the Cloud Control Station node will be introduced here below, and deeply analysed in chapter 5.

4.2 Rules and organisation

In a complex system with many agents working at the same moments, a set of fixed rules is clearly needed, so that all the operational decisions taken for traffic management purposes will be coherent with a general guide lines, and most of the issues will be easily solved thanks to some standardized procedures. Agreeing with ideas shared by other relevant project in the same field (NASA and EASA first), we thought that a basic and efficient division of the city airspace based on different operational altitude is very likely to be applied in future UTM systems. In this sense, a subdivision of the airspace similar to the one shown in figure 4.3 can be considered. This kind of division allow a better organization of the airspace, with certain layers dedicated to specific vehicles or missions, and allowing the UTM to use this division for organising the traffic, assigning to each drone a certain altitude within the range of altitudes reserved for that specific kind of mission.

Focus of this project has been on the layer depicted in green in figure 4.3. According to the same airspace height division, it can be thought to be divided in different layers too, where UAVs can be placed according to their characteristics and mission. In section 5 can be seen how a certain **priority level** will be assigned to every drone, and in a similar way an **operational altitude**, that is than used for organising the airspace and managing the traffic accordingly.

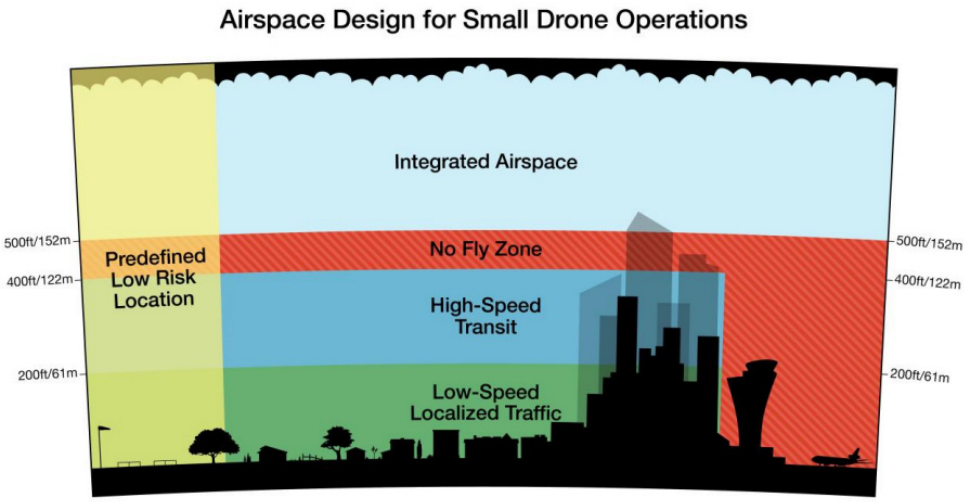


Figure 4.3: Example of subdivision of the city airspace in different layers.

Target of the project has only been quad-copter drones that will autonomously fly over the city being tele-operated by CBUTM itself. To them, a certain range of altitude will be assigned, that is likely to be the green and part of the blue zones in figure 4.3, sorted, within these big layers, in several sub-layers, each one reserved for specific usage. The lowest one will be assigned to highest priority UAVs, performing special missions, like the one used by authorities for safety, monitoring and rescue missions. To commercial drones, that are the main target this project is referred to, an higher airspace level is reserved. For what regards remote piloted vehicles, to which the same cloud-based collision algorithm cannot be applied due to the impossibility of predicting their local, actual trajectory and operator's intentions, a lower airspace layer is assigned, dividing autonomous from piloted vehicles.

The relative altitudes of these layers are not mentioned on purpose, being this decision totally dependent on NFA's future decisions. We can just state that the operational altitude of the tests performed during the development of this project has been $25\,metres$, where a maximum cruise speed of $4\,metres$ has been imposed. For general guidelines regarding city airspace management, one can have a look at chapter 3, and directly on NFAs' websites.

Finally, we decided to build up our system so that at each drone flying within the controlled airspace is assigned a flight altitude, and, within that aerial layer, a certain priority level, so that between two close vehicles that are likely to collide, will always be the lowest priority one moving out its original path, allowing the other to accomplish its mission as established. This priority based organisation allows solving many circumstances, avoiding situations in which both the vehicles move, and, in case the communication between them is not direct - as supposed in our operational context - they do it in the same direction, like in front of a mirror, leading to incomprehension that may cause waste of time and then collision.

Within the network, each enrolled UAV will have its own cloud image managing the connection between the vehicle in real world and the ROS environment. This node is supposed to be launched after a mandatory registration performed on an official web portal, according to what imposed by the authorities.

CBUTM is supposed to be able to manage an arbitrary number of vehicles at the same time. We started with a static structure where just 3 vehicles are controlled, for allowing tests and simulations

even on modest powered laptops. It has been then expanded, for managing a maximum number of vehicles, due to airspace "saturation" and/or the available computational power, taking into account that the whole architecture and approach has been thought to be as light as possible, dividing the big, complex optimization problem in several small ones. Once reached the maximum manageable number of vehicles, the CCS node block the registration of new vehicles. The Cloud Control Station will then initialize N indexed structures of $ros :: Publishers/Subscribers$, as analysed in section 4.3.

4.3 Registration

CBUTM is meant to be a control system closed through the cloud, able to manage and coordinate a high number of vehicles, and assure certain rules, requirements and procedures are respected. To do so, a perfect knowledge of vehicles and operators joining the network is required, so that everything going on within the control system can be tracked, and the system is aware that everything is connected and working.

First of all, it follows that should be possible at every moment to check the data of each vehicle together with their position and speed, being able to go back to its operator. As we saw in chapter 3, different countries are trying to standardize the control process for this new type of aircraft, trying when possible to organise common international rules. Local authorities are already trying to make a census of drones - with many difficulties coming from the not clearness and application of the rules - requiring some bureaucratic steps. These include the declaration of vehicle possession, the application for the approval of its technical characteristics and for obtaining the authorization to fly (section 4.3). This is a mandatory process, a legacy check that must be done, in the same way it is done in traffic management system of every other kind (planes, cars, boats etc.). It follows the need for our system to not pretending to substitute it but be compliant with that. We then thought to organise our CBUTM in a way that this fundamental, preliminary step collecting data is always performed at the beginning of every instance, provided by a central entity, the **Cloud Control Station node**. It is the core of the system, that, collecting informations and managing data streams between different nodes and their interconnections, performs a fundamental redirecting function, needed by the system to work properly.

This preliminary procedure is performed during the registration service, that is needed for practically insert a new drone within the system and make it fly under its control. This service is called by the just spawn UAV-node, that has been launched after the creation of a specific script (as supposed in section 4.2) containing all its relevant informations. The CCS node then sets up a **quick access database**, meant to be always up-to-date with the current informations coming from the active drones. Some of these information are static, like the vehicle type, its legal owner, its weight, dimensions and so on, other dynamic, like the position - which is updated at every cycle according to the working frequency of the system - or its **priority level**. The computation of the priority level is another of the steps involved in the registration service. Having a look at figure 5.1, it can be noticed that every PTP instance is drone-specific, launched with some parameters set according to the case. Depending on them, a coherent priority level and flight altitude is assigned. For now, the parameters we decided to use for classify different vehicles are 5:

- aircraft type: depending on if the vehicle is a quad-copter (1 - this has been the target of the project), a generic multi-copter (2), or a fixed-wing like vehicle (3),
- owner category: users are classifiable according to the category they belong to. It means that common civil flying their drone will be assigned to category 1; operators using UAVs for commercial purposes will be assigned to 2; finally, operators belonging to private companies and flying UAVs for them, will be assigned to 2; finally, operators from the public

administration entities or working for public safety will be in category 3.

- dimensions (diameter, in *cm*) and weight (in grams) of the aircraft, that of course influence the hazardousness of its operations and the difficulty of manoeuvring them, leading us to decide for giving higher priority to heavier vehicles with bigger inertia.
- mission category: depending on the purposes of the mission itself, they can be classified in recreational (1), commercial (2), public interest and safety (3)

This is, for the moment, a static parameter, just assigned at the beginning of the mission, but it is likely to become dynamically, adapting accordingly to parameters like the actual inertia of the vehicle.

Another important prerogative of a working UTM is the capability of knowing, in every moment, the status of its agents. It means, it should be detect whether if drones are active and their ROS node are working, or if some instance has terminated, either cleanly or by crashing, relaunching the node when needed or refreshing the list. For this reason, as soon as a registration request is performed by a drone, an **heartbeat check** procedure has been developed between the CCS and drones cloud images - the ROS node directly connected to the vehicle by the MAVROS bridge - so that the former can notice when the other one stops working. This mechanism allows the system to be aware of which drones are active at the moment, having a responsive recovery tool for managing disconnection issues.

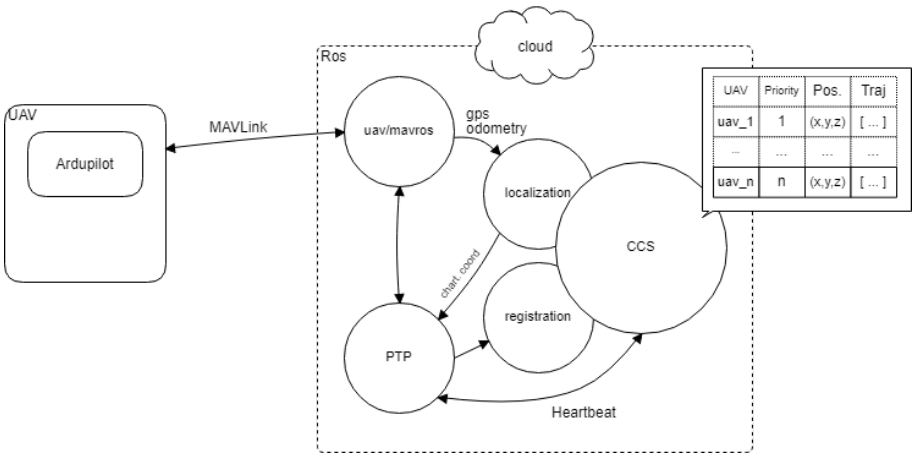


Figure 4.4: Graphical representation of the CCS-PTP interaction

Summarising, the operations performed by the CCS regarding the registration routine are:

- collection of all the relevant data of the vehicle and its operator, stored in a database
- identification of the vehicle and of the nature of its missions, assigning a coherent priority level and a flight altitude
- initialization of an heartbeat check mechanism for disconnection awareness
- redirection of the needed data for the creation of a drone-specific risk layer
- assignation of a priority number, sorting the active drone list according to their priority level

All these procedures will be deeply analysed in chapter 5.

4.4 Risk assessment and map generation

Basis of the whole project, in which safety is the first goal to reach, is a well-defined and standardized risk assessment, described deep in details in [48]. Aircraft operate in National Air Space (NAS), which are controlled by governments worldwide, and each one can impose different ideas or restrictions. Considering that the future framework in which drones will fly is still in the design process (3), an important characteristic of CBUTM is its capability to adapt to different conditions and legislations, maintaining its working principle. Changing some structural parameter will change the behaviour of the system, but not its working principle, being always able to assure the minimum required safety level. For this reason, we think CBUTM can be intended as a meeting point where different national flight authorities can speak the same language, which is the metric and the basic working principle, while in the details it can be customized for the needing of each different local authority according to their usage, conditions, restrictions and, in general, legislation.

Goal of the project is to assure the safeness of the system, so that drones can be flown within the city airspace minimizing **failures**, that would mean injuries, fatalities or, in general, accidents involving buildings, people and whatever. Failures can derive from different causes (battery failure, connection lacks, bad weather etc.), and safeness becomes such a difficult parameter to quantify. According to the literature (more in 2.8), risk is defined as the **number of fatalities per hour**, and safeness can be only assured when this value doesn't go over a certain threshold.

With CBUTM, following the direction of other important researches in this field (more in [48]), we propose a **standardized** way for computing the risk for a drone to fly over a determined area. This risk computation is based on the knowledge of the city environment. It means that a precise and parametrized knowledge of the scenario is needed.

We had then a look at ROS community, searching for a package helping us in this direction, finding one perfectly fitting with out purpose. The **grid_map**[9] package is a C++ **library** with ROS interface to manage two-dimensional grid maps with multiple data layers, designed for mobile robotic mapping to store data of any kind. The library supports multiple data layers and is for example applicable to elevation, variance, colour, surface normal, occupancy etc. To each **cell** of the grid is assigned a certain value, building up a **layer** as an **array of float-type values**, each one representing the cost of an $xbyx$ cell depending on the desired **resolution**. All grid map data are stored as data-types from *eigen*, a popular C++ linear algebra library, that allows easy and fast tools for data manipulation and proper **iterators** allow fast access for reading from each cell of the map. The underlying data storage is implemented as two-dimensional circular buffer, which allows for non-destructive and computationally efficient shifting of the map position. This is important in applications where the map is constantly repositioned as the robot moves through the environment. The final **cost map** is then the result of the superimposition of many **layers**, each one carrying **data** of a certain kind.

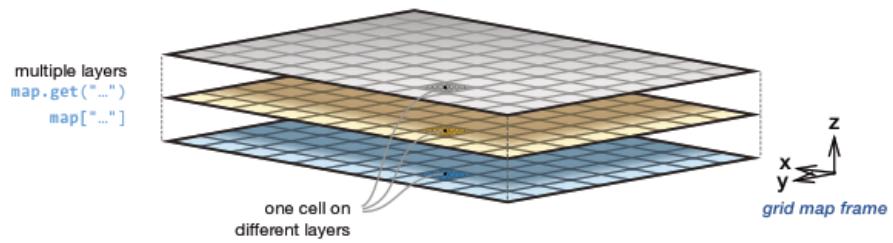


Figure 4.5: The grid map working concept

Some layers are **static** and fixed for every drone and mission, like the ones containing informations about buildings height and occupancy, or relevant forbidden areas commonly labelled as *no fly zones*, like the ones in the neighbourhoods of an airport or a military area. Others are **dynamic**, but still in **common** between all the vehicles, like the weather forecasts, population density measurements or whatever. These layers are supposed to be refreshed at a certain frequency, so that CBUTM can count on an up-to-date and coherent geographical map of the urban environment where the control takes place. 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 can be merged with the real-time ones coming from the flying drones: finally, the result is a dynamic map of the city. Finally, other layers are **drone-specific**: in CBUTM, a **Map Manager** node exists, that is in charge of creating one layer for every drone, representing the risk layer for that specific vehicle, based on its physical and technical characteristics. The creation of this last layer just happen at the registration phase for each drone, called by the CCS providing the registration service to the specific vehicle. The superimposition of all these layers, each one weighted in a proper way according to the considerations done in [48], makes up the final cost map, function of geography, drone's parameter and mission standard objectives. The creation of layers and the management of this map is deeply analysed in [48]. The final result, function of both geography and UAVs constructive parameters, is shown in figure 4.6.

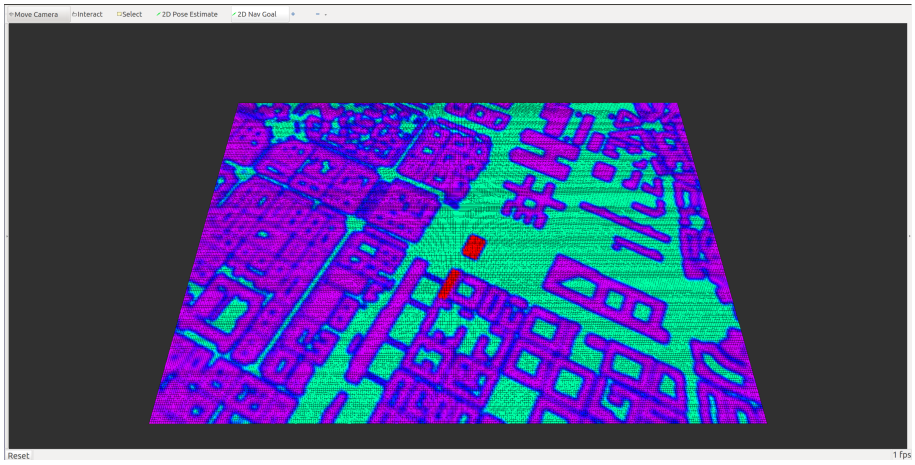
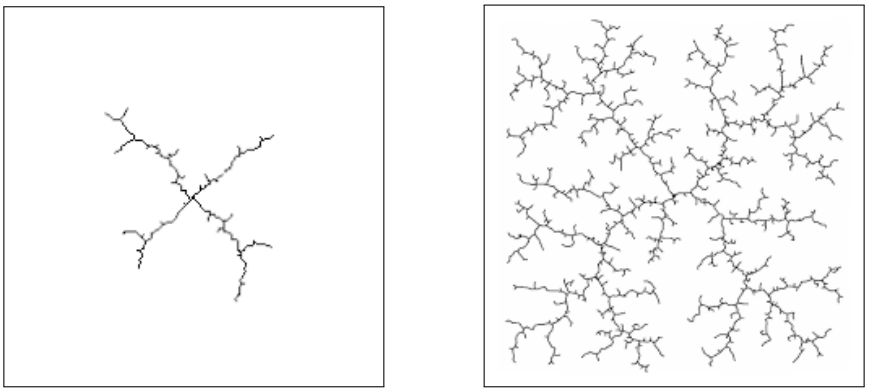


Figure 4.6: Final grid map created by the superimposition of different layers, with different weighting factors

4.5 Path planning and validation

The path planner algorithm used in this project is, among many different ones analysed in 2.6, an $RRT*$. It is a probabilistic algorithm derived from the classical **Rapidly Exploring Random Tree** (RRT) technique. 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 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 unresearched 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, 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 when the random sample is too far than the allowed is to substitute it with a new state, at the maximum distance along the line that connects the sample to the tree.
The tree then, starting from the initial state (the starting position of the UAV) keeps **growing** adding random sample. This iterative procedure ends as soon as the tree contains a node in the goal state region. In figure 4.7 a typical evolution of RRT's tree is depicted.



(a) (b)

Figure 4.7: RRT's tree evolution in time

The **advantages** of this approach are many. First of all, it is simpler and easier than many other to implement [27]. 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 [54]. The main **drawback** instead is due to the need of saving 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 [54]. The mechanism are obviously quite similar, 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 4.8 the differences between the two approaches are shown: it seems evident that the tree built with RRT* (right side) is more ordered than the one on the left, 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.

The map coming from the Map Manager becomes the input of the **Mission Planner**, whose final aim it's dual: on one side, it must find the **optimal path** (cheapest within the feasible ones) for the drone to follow, on the other it must evaluate if the just computed trajectory is compliant with the safety standard imposed by authorities, **validating** the path. This two different operations, **path planning** and **path validation**, co-operate in the **map manager block**, since the output of the former - a sequence of n way-points that the vehicle shall follow for accomplishing its mission - is the input of the latter. At the end of the process, the possible situation are then three:

- the trajectory satisfies mission requirements and safety bounds, so that the drone can be

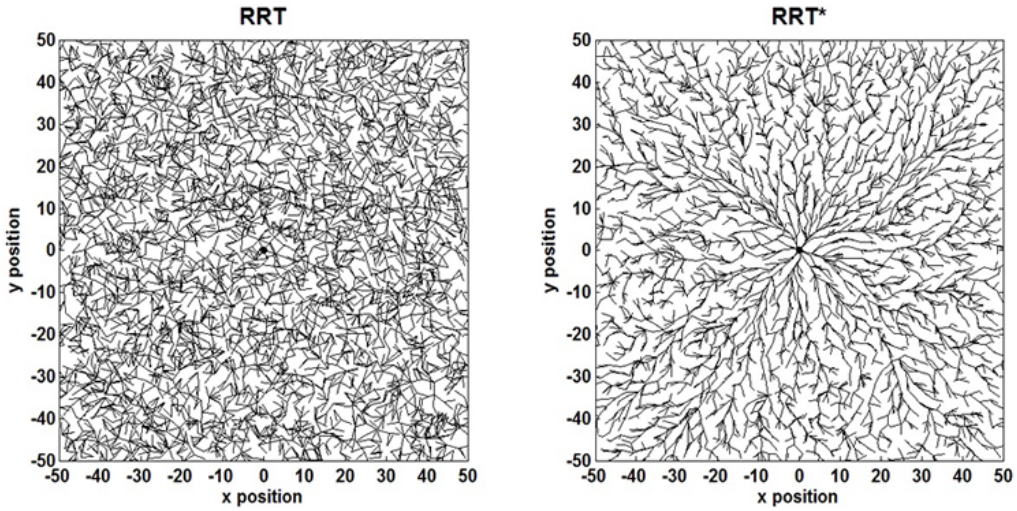


Figure 4.8: Differences between RRT's tree and RRT*'s one

authorized to fly

- the chosen trajectory does not met some requirements, but it's still possible to change the weights of the cost map and re-calculate another path to find a proper trade-off between optimality and safety, computing another trajectory
- mission requirements are impossible to be satisfied: the only solution in this case is to abort the mission, that's infeasible.

4.6 Path following

After the mission has been identified and an optimal, safe path has been computed, what comes next is to make the drone physically fly along the n **way-points** of the path, that must be reached in ordered sequence. As said in 2.7.4, most of the drones actually in commerce are provided with an on-board autopilot software, offering an high level user interface that makes everything far easier. Considering then for our network only drones mounting a MAVLink ready autopilot software (see 2.7.2 for an overview of what MAVLink is and its ecosystem), we made our application able to send drone, by means of a MAVROS bridge, all the needed kind of **command** inputs. A complete reference of which kind of messages can be sent to the autopilot and which are the commands we can use, are shown in the developer's official documentation [11]. In that, one can see that three different ways to drive the drone are provided, depending on if one wants to drive it sending a **velocity vector** to be applied, precise **position coordinates** to be reached or a complete **mission** to be performed. We then tried all of these ways, evaluating pros and cons of each approach, looking for the best fitting our purposes. At the end of the work, we can state that:

- Sending a velocity vector to apply is the less convenient way to fly the drone. This is because the actual velocity of the vehicle, that can be detected by reading the relative topic of the MAVROS node, is just the physical result of a very complex control algorithm that takes place within the autopilot, completely hidden at our eyes. It takes into account so many aspects of the vehicle dynamics, that, because of their complexity, is even useless to try to model, also because this work is already so well done by the autopilot, and do it a second

time, in addition to the poor results, would just mean more computational complexity and time required. The results have been, in fact, many problems in driving the vehicle properly, with the required smoothness and precision.

- Sending a new position to reach makes possible to fly the drone directly to the desired position, leading to the goal, giving less possibility to control the way it does it tough. The vehicle, guided by its autopilot, reaches the point with high precision, but it does it approaching as a "parking problem" - aimed at stopping in that position - managing its motion consequently. The way the MAVLink protocol itself is designed (2.7.2), limits the possibility to send, at the same time, 2 commands to be performed, so that a certain position is reached and a specific velocity vector is applied. The results were drones flying at a maximum velocity of $2.5m/s$, too slow for our purposes.

- Sending a completely defined mission, finally, means using the autopilot potentials to the fullest. As said, in fact, it provides an high-level user interface, that is used by software like QGroundControl - providing full flight control and mission planning for any MAVLink enabled drone - for easily and efficiently fly the vehicle. With mission definition we mean a sequence of way-points to be reached in a certain order, in a precise way and with a defined velocity, ending in a smooth trajectory. This is what we were looking for, and the best way for interact with the autopilot tool.

Based on these considerations, we built up our flight controller node so that it looks for the best way to reach the sequence of way-points, sending it as a properly mission command. The autopilot is then able to elaborate it, managing the rest of the work (thrusters regulation, weight balance, drift control etc.). In that way, we consider the drone as a black box, that we just know how to interact with in order to obtain the desired results, ignoring everything happens inside. To exploit the functionalities offered by the autopilot, we made our code communicate with it for sending a mission as input for the drone with a specific $push\_mission(trajectory)$ method, quickly shown in 1.

Algorithm 1 Mission management

```
procedure PUSH_MISSION(trajectory)                    ▷ trajectory is output of opt.prob
    for int i = 0; i<trajectory.size(), i++ do
        traj_wp = meter_to_gps(trajectory(i))                ▷ i-th pose meter->gps
        mission_wp.x_lat = traj_wp.latitude;
        mission_wp.y_long = traj_wp.longitude;
        mission_wp.z_alt = traj_wp.z_alt;
        mission_wp_list.push_back(mission_wp);               ▷ store i-th wp in a buffer
    end for
    mission_push_msg.request.waypoints = mission_wp_list;    ▷ then into srv message
    mission_push_client.call(mission_push_msg);                        ▷ call the service
    mission_cmd_client.call(MISSION_START);
end procedure
```

4.7 Monitoring

When deciding the best architecture for our system, we ended, as previously said, in a **distributed non-cooperative priority-based NCS**. It has been decided for avoiding optimization problems to get too big, compromising the reactivity of the system and the scalability of our UTM solution.

This is because if every drone, solving its own optimization problem, must be aware of every vehicle connected in the same network (considering, for assumption, only blind drones, just relying on data coming from the cloud), the complexity of such a problem would increase exponentially with the number of vehicles. Not to mention what would happen in a centralized NCS, where a single machine should be able to solve the whole problem, even dividing it in smaller ones, alone. We decided then to build up our system in a way that the **CCS is the only one caring for all the connected vehicles**, collecting the actual position of each of them. It does so receiving the GPS position from each UAV node, that are then translated into Cartesian coordinate referred to the map of the city used for the risk computation. Informations about the position of each vehicle are collected, and used for a distance check. It just consists in a simple routine, iteratively checking the distance between the active drones, sending alert to the right one when the safety distance of $50metres$ is not respected. To do so, an $n \times n$ **lower triangular matrix** is built up, with n being the number of active drones, sorted according to their priority level as seen in 4.3. When the distance computed in element (i, j) is lower than the allowed, an alert is sent to the i-th drone node, containing the name and the index in the priority list for connecting to the j-th vehicles, which as higher priority, "activating" the subscriptions to the relative topics. In this way, the controller node supposed to compute a deviation in its trajectory caused by the presence of an higher priority vehicle knows exactly which topic subscribe for getting just the needed information.

This mechanism is quickly shown in figure 4.9 in the following section, in which the process just sketched here above and the elements involved are graphically shown.

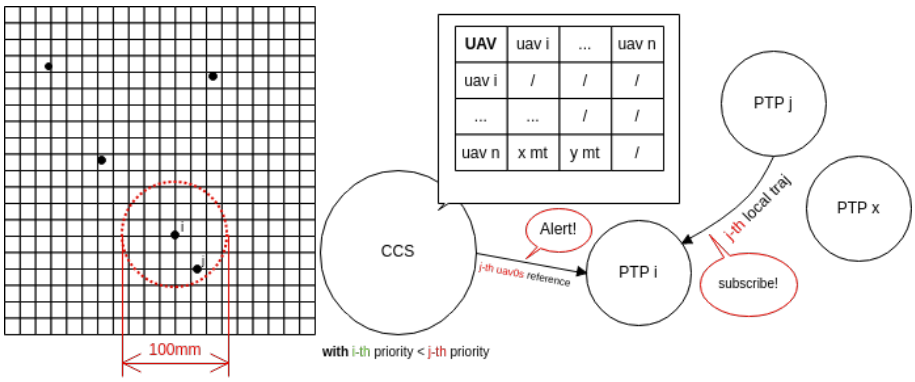


Figure 4.9: Graphical representation of the monitoring and advertising process

In this way, the CCS can be thought working like a router, managing connections between different nodes, that should be able to access just the really needed informations.

4.8 Collision avoidance

Main functionality of an UTM system is to avoid collisions between UAVs while flying in the controlled airspace, that is for sure one of the most critical aspects of such a control system. A collision may happen between two vehicles or with an object, either stationary (like buildings, trees etc.) or moving (like other not controller vehicles or any other moving object not included in the network). Each UAV should be able to handle the collision avoidance autonomously, detecting obstacles and change the route as well as the altitude by itself [35]. As first thing, we can state that any collision avoidance system can be split in two phases: sensing&detection, and reaction. The first one includes all the techniques through which the drone gets informations about the

surrounding environment. It may clearly require sensing capabilities, made of several on-board sensors. Restricted autonomy and dimensions limit the possibility of having the required wide variety of sensors, and another blocking parameter is given by the computational power of the on-board processors. Developing a Cloud-Based UTM which counts on a direct wireless connection with the vehicles, we can instead think to exploit to the fullest the potentialities of this powerful technology, relying just on the data flowing within the network and not coming from sensors for an effective collision avoidance. In this way we can even state that no particular sensing capabilities would be required, increasing autonomy and decreasing equipment costs for each single flight. Finally, speaking about a huge network involving several vehicles sharing the same airspace, it logically follows how making vehicle cooperate in a smart way would represent a desirable solution, so that everything needed for avoid collisions is stored and manage in an efficient way. Here is where the potential of a Networked Control System comes.

When an UAV is flown for an unmanned mission, it receives a path that is the solution of a global optimization problem, done all along the way from the start to the goal, considering all the stationary obstacles in the map. This is done by any conventional path-planning algorithm, that it takes its time to perform such a complex computation. One can think that the same (or similar) algorithm can be applied also to compute changes to the old trajectories, dynamically. In this case, we would speak about a **re-planning** approach to the collision avoidance problem [57][41], where every time a feasible new trajectory is re-planned *ahead of time*, predicting the future position of the controlled vehicles and of those it has to avoid. The slowness of a standard path planner and the exponentially growing complexity of such a scenario forces to search for another solution to be as reactive as required by UAVs dynamics. A faster and more efficient solution must be found, applicable in short time and able to assure safety in the actual neighbourhood of the vehicle. This is referred to be a **reactive** approach.

Of course there are several circumstances in which this mechanism shows its weaknesses, for example when:

- connection breaks down (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 in cloud all data and use those data to extract knowledge that vehicles can inherit, it can often happen that this is not sufficient for collision avoidance, especially in real dynamical and complex environments like cities.

For these reasons some sensing capabilities are obviously required on board, and it is even not admissible think to not include some sensors on-board, but for the purposes of this project we will just consider the cloud approach at this fundamental problem. Our cloud system is made up of active and passive agents, with the formers being all the registered UAVs flying in the controlled airspace, and the latters being any obstacle (either fixed or mobile) and other not registered flying object of any nature. 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, according to the defined protocol.

We have then that CBUTM has to prevent that the real trajectories being flown by two or more UAVs under its control intersect in a dangerous manner, acting directly on the networked agents for changing their trajectories. This decision should be the less invasive - meaning that the changes respect the original path should not be big - and quick, leading to a overall optimal result.

Because of the slowness and the difficult scalability of a re-planning approach, we decided to start developing a **reactive on-line 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.

In literature, many are the architectures and algorithms commonly used to manage collision avoidance and in robotic motion in a complex environment ([46],[28], [56],[41], [31], [60] and many others). But a work in particular [26] caught our attention, treating a similar problem: focus of the project was the creation of a centralized control system which, knowing the objects to be controlled and their positions, aims at controlling the involved vehicles for avoiding collisions. The networked system is meant to have a central entity with whom each vehicle has to refer with, communicating its parameters (position, velocity, mass of payload etc.) and receiving instruction to accomplish its task. The cloud system has then not only an up-to-date state of the involved agents, but also the capability to modify their trajectory. In that work the computational time required for solving a complex, global optimization problem is appointed as the major drawback. As suggested by the same author, distributing the problem makes it lighter and easier to solve, and the system more robust at the same time [26].

Among different architecture design possibilities (that have been analysed in section 2.4), we opted for a **distributed non-cooperative priority-based networked control system**. According to that, every drone is supposed to have its own, independent controller, that uses data collected in a shared resource pool to reach the safety goal. Such an architecture can lead, compared to different strategies (section 2.4), to the reduction and simplification of the optimization problem that every controller must solve on its own. In that way, in fact, the huge problem optimizing, at the same time, the trajectory of every vehicle within the network can be practically divided into many sub-problems, in which every controller solves independently an optimization problem taking into account, of just the neighbours vehicle, only the higher priority ones. It leads to:

- Less computational power required: since a unique and complex optimization problem is decomposed into smaller ones.
- Scalability: since increasing the number of involved vehicles the complexity of the controller does not proportionally increase.
- Robustness to failures: since with a centralized approach there would be a single point of failure for the system, while in this way a failure of one can be solved by the optimization of another.

It means that every controller should be able to quickly access to only the required information, shared by a central entity which collects them, like the proposed CCS is. In order to have a clearer idea of this architecture, one can have a look at the sketch in figure 4.10 here below, where $vehicle\_i$ and j get too close to each other, and the CCS, after having noticed that, alerts $vehicle\_i$ sending the extremes for getting the local trajectory of $vehicle\_j$, with higher priority.

Since the goal of CBUTM is to provide a reactive collision avoidance system, being at the same time efficient and light enough to be solved with limited computational power in limited time, we found in a **receding horizon based control algorithm** a suitable approach, as also sustained by many works in the same or similar field ([26][53]). As deeplier analysed in section 2.5, it is a very effective algorithm, able to fulfil speed requirements and to deal with the insertions of constraints. The usage of a model (from where it comes the common label of Model Predictive Control algorithm) of the plant makes it able to predict its future behaviours and find the optimal control to be applied. The way it works, analysed in section 2.5, make it possible to find the right trade-off between preciseness of the model (then complexity of the algorithm) and computation efforts and time (then efficiency and responsiveness), assuring great results even with a modestly precise model.

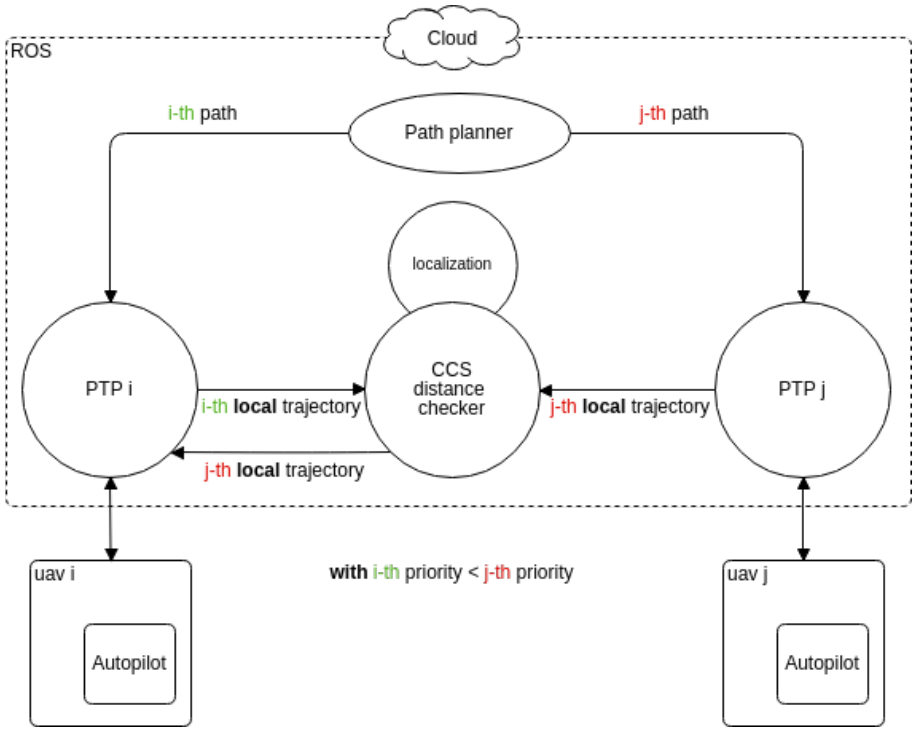


Figure 4.10: Graphical representation of how a vehicle controller node gets aware of the presence and data of another one to be avoided

The way we decided to approach the control issue is to solve, then, an optimization problem at every cycle, with a fixed frequency, defined as follows:

$$\begin{aligned} \vec{U}^*(k) = \min_{\vec{U}(k)} \quad & J(\vec{U}(k), \vec{x}(k|k)) \\ \text{subject to} \quad & 1)\ \vec{x}(k+1) = f(\vec{x}(k), \vec{u}(k)) \\ & 2)\ \vec{U}(k) \in \mathbb{R} \\ & 3)\ \vec{x}(k+1|k) \in \mathbb{X} \end{aligned}$$

where J is the objective function, defined as:

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

with:

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

$\vec{u}(k)$ control input at time k

H_P prediction horizon

c per-stage weighting function

Φ weight for the last state of prediction

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

and

1) are constraints on the state due to system dynamic, imposed by the model

2) are constrains on the control input

3) are the constraints on the state imposed by other agents

This last element is where the coupling between different vehicles within the network would happens, remembering that every agent considers, in addition to its own objective function and constraints, and only the coupling objectives and constraints with close, higher priority agents. It would mean the insertion of dynamic constraints in the optimization problem, exponentially increasing its complexity. It would make the problem a non-convex one, excluding part of the state space from the feasible set, with all the difficulties that derive. To solve this problem in a faster and easier way, we decided that the coupling to take place as an additional cost to the infeasible path, that will lead the optimization problem in finding a better trajectory for avoiding collision. The final scheme of the collision avoidance algorithm we implemented in our package is shown in figure 4.11 here below.

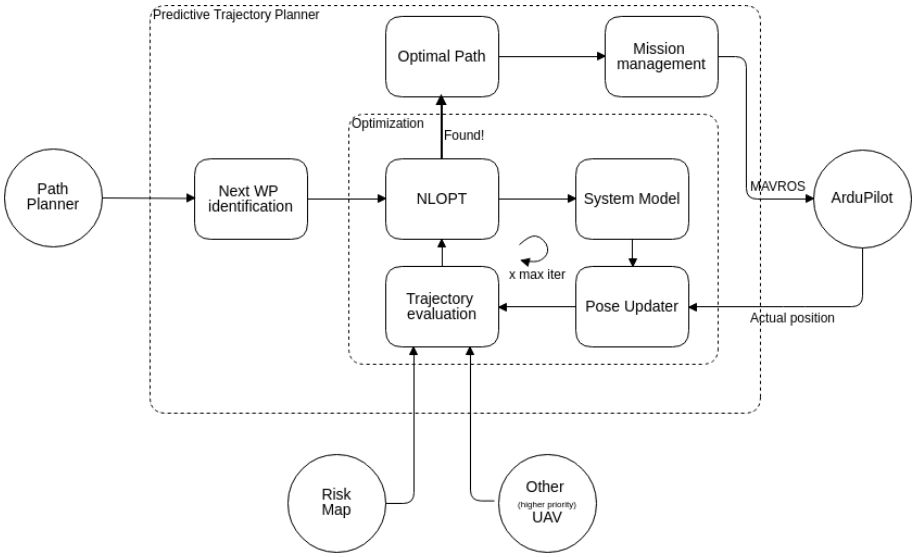


Figure 4.11: A block scheme of the collision avoidance procedure

The process shown here above is aimed at finding the best trajectory, composed as a succession of poses reached within the *prediction horizon* adding, to the previous one, a random spatial offset within the range [0,4] m in every direction, computed as the distance the vehicle would at maximum travel flying at the cruise speed of $4\,m/s$ in a prediction step of $1\,second$. Using a free/open-source library for non-linear optimization (NLopt[1]), we set up two nested optimization routines aimed at finding the best trajectory to reach the next way-point, selected and isolated from the total path. The first optimization is global: it tries, iteratively (according to the $max\,iter$ value in

[1]https://nlopt.readthedocs.io/en/latest/

figure, set to 8000 for the first, 6000 for the second optimization) random values all around the space. The second is local instead, just looking around the solution of the first one, effectively enhancing the quality of the optimization. Goal of the optimization problem is to minimize the objective function J, that is actually a cost function, measuring the cost of every tried trajectory. This cost is computed weighting every pose of the trajectory considering the distance from the next way-point (selected and isolated from the total path), the distance from the higher priority vehicles (received subscribing the relative topics) if lower than the allowed ($10\,m$), and the cost of the respective cell on the risk map. An extra cost is assigned to the last pose of the trajectory, for pulling the solver in reaching the goal faster.

$$J = Q * distance\_cost^2 + R * safety\_cost^2 + F * last\_pose\_cost^2 + M * map\_cost \tag{4.1}$$

The control vector $\vec{U}(k)$ is then the just computed optimal trajectory, that is sent to the autopilot as a proper MAVLink message, according to what seen in 4.6. The way it has been practically developed will be analysed deeper in details in [52].

Chapter 5

Implementation

In this chapter, the way the core of the system, the CCS node, has been built up will be shown. This is the node in charge of connecting all the other nodes and managing the data streams, so that all the procedures CBUTM is based on can take place. It then provides services and functionalities that are vital for the system itself.

5.1 Context and assumptions

During the development of the project, some assumptions and simplifications have been done, so that we could manage to simulate the result even on modest powered laptop and we could proceed gradually, enhancing step after step the code and then the final result.

First of all, as previously said, we only considered quad-copter type aircraft, in particular Iris model by 3DRobotics, flying at a cruise speed of $4\,m/s$. The vehicles are then supposed to have no sensors on-board, relying completely on information shared within the cloud.

The considered motions are performed in a 2D space, so that every drone can just refer to the planar risk map relative to its flight altitude, which as been set at $25\,metres$. The scalability of the system, at first static for testing and simulation purposes, as been then adapted for being ready to manage an a-priori unknown number of vehicles.

The whole registration service analysed here below is not supposed to be an alternative to every registration, declaration and whichever procedure required by National Flight Authorities (NFA). It is, instead, meant to be consequent to that, aimed at inserting a certified vehicle within the networked control system. Just then it can be properly managed according to CBUTM working principle, coordinating its flight operations with the other users of the network.

5.2 Cloud Control Station node

The Cloud Control Station is the ROS node meant to take the place of the classic Ground Control Station used in common Air Traffic Management systems. It is then thought to be the *central* entity of the network, in charge of holding relevant informations and linking the other elements of CBUTM, managing the communication streams within the system.

It's job starts as soon as a user wants to join the network. When the user requires, according to the procedure described in section 4.1, the PTP node relative to its unmanned vehicle to be launched, it needs first to be included in the working network. It is done trough the registration procedure, a specific service provided by the CCS. The UAV-specific node calls the aforementioned service through a service **client**, using the registration.srv message defined as shown in 5.1.

Through this message, it transmits all its relevant informations (contained in the custom defined message shown in *Listing*5.1) to the Control node.

Listing 5.1: registration.srv message type definition definition

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

Within the registration service routine the applicant is identified, and all the relevant informations are collected.

Some parameters (dimensions of the vehicles, weight and type of the aircraft), are transmitted to the Map Manager node, that creates a **UAV-aware risk layer**, tailored on drone itself, as explained in [48]

Other informations (dimensions, weight, mission type, owner category and mission category) are used to compute a coherent **priority level**. As previously shown in figure 5.1 and analysed in section 4.2, the priority level is computed according to some parameters defining the vehicle, the operator and the purpose it is flown for. This launch file, with the described parameters, is shown in figure:5.1 here below.

```
<launch>
  <node pkg="uavs" type="uavs_node" name="iris_1" output="screen">
    <param name="uav_name" type="str" value="iris_1" />
    <param name="owner_cat" type="int" value="1" />
    <param name="uav_type" type="int" value="1" />
    <param name="dimensions" type="double" value="500.0" />
    <param name="weight" type="double" value="350.0" />
    <param name="mission_cat" type="int" value="1" />
  </node>
</launch>
```

Figure 5.1: The launch file of a UAV-node, specific for every drone, user and mission

These parameters are given to the node through its own launcher, that is obtained after an application has been completed by the user, where everything is checked according to the criteria imposed by the NFA (section 4.2). These are assigned according to the classification described in section 4.3- To each of these parameters, a weighting factor is assigned, used for evaluating a coherent priority level that is growing proportionally to the number given to each parameter, so that the combination will give a priority level as a $float$-data type according to which drones will be sorted and managed. An operative flight altitude would be also assigned to the specific mission. In our tests, its has been fixed at $25\,mt$, and the priority level has been supposed to be static and assigned at the beginning of the process. In a future moment, it is likely to become a dynamic parameter, used for a better traffic management and for avoiding blocks that may happen with a static priority level.

Finally, all the relevant data are stored in a quick-access **database**, built up using $std :: map$ and $std :: set$ class objects, associative containers that store elements formed by a combination of a **key** value, used to uniquely identify the elements, and a **value**, storing the content associated

to the relative key. In such data containers, elements are always sorted following a specific strict weak ordering criterion indicated by its internal comparison object. All the inserted elements are given a certain position according to the fixed order. The container uses an allocator object to dynamically handle its storage needs, assuring a high rate access - logarithmically proportional to the number of elements - which can be a limiting problem when its dimensions exceed. Even if generally slower than unordered_set containers to access individual elements by their key, they allow the direct iteration on subsets based on their order, useful in many performed computations.

This strict order makes than possible to quickly perform procedures within the CCS node, for checking the distances and alerting the lower priority ones, so that they can be connected to the higher priority drones' topic, where they can get their trajectories from, to be avoided, with the CCS acting like a router connecting and disconnecting different nodes.

The insertion of the UAV in the network is followed by the creation of a direct link between the CCS and the UAV node, aimed to the awareness of drone's disconnection, with the former checking if the latter's process has terminated, either cleanly or by crashing. It is done through the presence of a specific topic - */utm/heartbeat* - in which every active UAV node periodically publishes its id reference, that the CCS subscribes, representing its **heartbeat**. Every time it has been received, an specifically initialized timer is restarted with a certain time-out period of *at least 5 seconds* - considering the time the UAV node is blocked while calling the path planner service, during which the node is pending and does not publish -, so that when the heart stop beating a time-out callback is run. This routine recognises the crash and updates the list, removing the UAV from the active list and $turningoff$ every subscribers related to it.

Before sending back to the PTP node the response of the registration service call containing the authorization to fly, the computed priority level is used to build up a specific list, fundamental for organising all the interconnections between every drone flying over the controlled airspace. Each drone is inserted in that **list of the active drones, sorted according their priority level**, and that is refreshed every time a new drone joins the network. The case in which a user joins the network is shown in fig:5.2, where the resulting log coming out from the CCS node is presented.

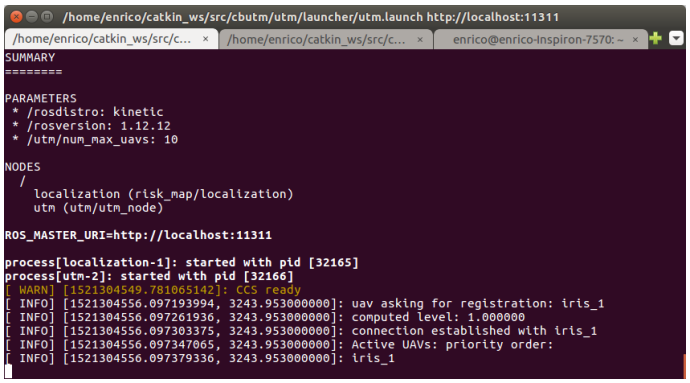


Figure 5.2: Log coming out from CCS node, when the first drone calls the registration service

From there we can see the steps involved in a single registration routine, where the vehicle is identified, all the operations for linking it to the network (the actual registration of the node in the ROS environment) are performed (giving a positive result if properly done), and a priority level is assigned as a double-type data. It then continue with the arming operations, done sending to the autopilot a proper MAVLink command through the MAVROS node. Transparent to the user is the creation of the drone-specific risk layer, managed and stored by the MM node. The priority sorted list is then published by the CCS, and subscribed by every PTP node. From that subscription, every UAV-node will get informed about its actual priority number in the list of the

active drones. The case of the first vehicle joining the network is shown in fig:5.3. After all the initialization procedure has been accomplished, the PTP declares the vehicle ready to fly, and, receiving the priority list from the CCS, gets informed about its own priority number, that in this case is the highest.

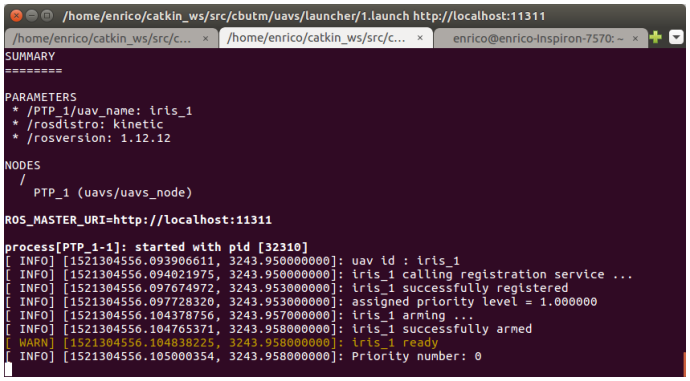


Figure 5.3: Log coming out from PTP_1 node, related to drone $iris\_1$, when it is the first drone logging in

Depending on the position occupied in that list, every PTP node will refer to a **coherently-indexed set of topics** for publishing its position and trajectory, and for receiving specific communications coming from the CCS, fundamental for the traffic management. It means that vehicle $iris_1$, occupying position 0 in the priority list, will receive/publish messages from/to topics that will be indexed as element 0 of arrays of topics. It is useful to direct the stream of data within the network, since, being priority based, in that way results easy to link every drone to the actual role played within the network, and quick to properly direct messages using vectors of $ros :: Publisher$ and $ros :: Subscribers$, indexed simply referring to the specific index in the priority list.

Every time a new aircraft joins the network, this list is refreshed, and published again by the CCS with the due modifications. When a second vehicle joins the network with an higher priority than the first one, what happens, seen from the CCS point of view, is shown in fig: 5.4.

Subscribing the priority list topic, every drone node will notice the change, looking for its name into the list and adapting to the new order as can be seen in figure 5.5 here below, changing publisher/subscriber address . In particular, from 5.5a can be noticed that $iris_1$, to which was previously assigned priority number 0, has become now number 1, leaving the precedence to $iris_2$.

A pseudo-code of the whole registration service routine is shown in algorithm 2.

The **heartbeat check** introduced before is then used for realising whether if the connection with the PTP node, running the path following and collision avoidance procedures, is working or the instance has been terminated, either cleanly or by failure. This is needed for checking the health of the network, and being updated with the status of the system. In ROS, it may happen that a process dies and nodes stop working. In that case, a specific recovery routine shall be developed, aimed at relaunching the dead node as quick as possible. Whenever a PTP node is shut down on purpose, instead, the system must be only aware of that, updating the list of the active drones. The UAV insertion-removal from the list allows the system to know which and how many drones are flying in every moment, having an always up-to-date priority list.

Finally, a **localization node** is present, aimed at translate the **GPS position** sent by the the vehicle by the MAVLink protocol (and translated in ROS message by the MAVROS package) for localize the vehicle within the risk map generated by the Map Manager. Every active drone sends, via its MAVROS instance, its GPS position specified using the WGS-84 reference ellipsoid. It is translated in Cartesian coordinates, linking it with the GPS position of the origin of the map used

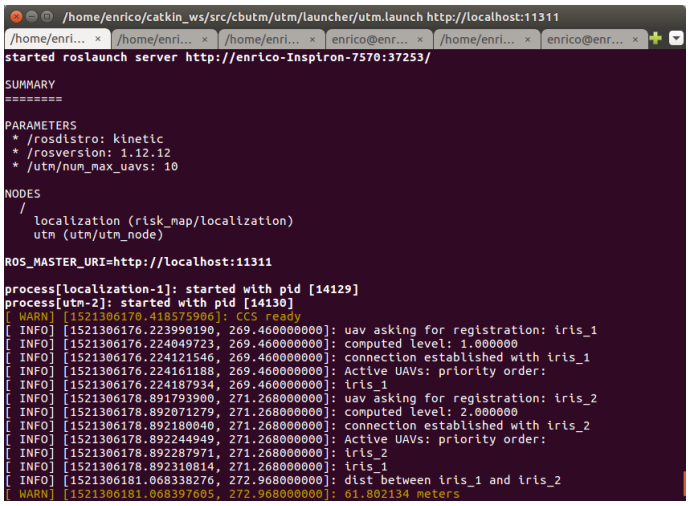
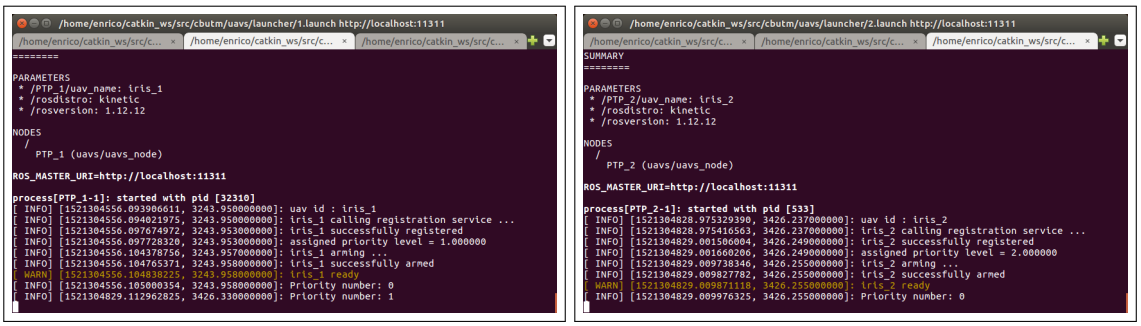


Figure 5.4: Log coming out from CCS node, when a second drone calls the registration service. The priority list is then updated according to the result of the priority computation



(a) (b)

Figure 5.5: Logs of PTP nodes 1 and 2, after that $iris_2$, with priority level higher than $iris_1$, logs in

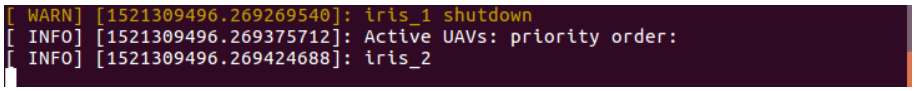


Figure 5.6: Log from CCS when a node (relative to $iris\_2$) shut-down has been detected, and the new priority list published

for computing the risk. This information is then stored into an up-to-date $std :: map$ object as a value linked to the correspondent UAV's name. The precision of a GPS based localization system is, of course, not infinite, but limited at an average value of $5\,m$ for what regards commercial devices [8]. Since in our testing operations we have not used real drones, we just refer to simulated vehicles, with on-board GPS sensors simulated by means of gazebo plug-ins. An interesting work on testing the GPS precision is performed in [36].

All procedures described above make possible to build up an efficient **tracking** system. It is used to monitor the position of every active drone, and check the distance between all the registered

Algorithm 2 Registration service

```
procedure REGISTRATION_SERVICE(req, res)                ▷ custom .srv message defined
    uav_id ← req
    compute UAV risk layer

    init_heartbeat_check(uav_id.uav_name)

    procedure COMPUTE_PRIORITY(uav_id)
        check mission type and vehicle specs
        check airspace population
        assign fly zone
        assign priority level
    end procedure

    update_list(uav_id.uav_name)
end procedure
```

vehicle flying in the city airspace. CCS is in fact meant to be the only node aware of the position of every vehicle within the network, and one of its functionalities is to **detect when two or more drones are likely to collide**.

A cyclic check is performed, computing the distance between the active drones only, stored in a triangular $n \times n$ matrix, where n is the number of connected vehicles. When the distance between two or more drones starts being lower than the allowed (it has been set to be $50\,metres$), the CCS sends the alert, calling the attention of the involved UAV-nodes according to their priority level: the lower priority one receives the id of the other one, together with its topic address within the network, so that it can subscribe the relative topic and read its trajectory, avoiding the collision. The pseudo-code of the just depicted procedure is shown in 3, according to the graphical description previously provided in figure: 4.9.

Algorithm 3 Collision alert

```
procedure CHECK_DISTANCE                                ▷ N × N upper triangular matrix
    compute distance between uav_i and uav_j
    if distance < 50 mt then
        check priorities
        send alert to lower priority one
    end if
end procedure
```

Every drone will then start checking the distance with the higher priority ones, computing trajectories for respecting the safety distance, deleting the drone from the list when out of danger, shutting down the subscriptions.

Summarising, every drone is assigned to a prefixed communication structure, labelled according the actual priority level. It means that, being the system able to manage N vehicles at a time, it will provide N parametrized structures. The CCS keeps the knowledge of which physical UAV is assigned to which priority level. Through these indexed communication structures every drone will publish its local trajectory and read the trajectory of only the drones that are close to it and have a higher priority, without caring of anything else than that. It allows to split the total, complex optimization problem in many small ones, providing large scalability to the system.

Chapter 6

Simulations and results

6.1 Software-In-The-Loop configuration

Simulation, testing and debugging have been performed in **Gazebo 7**, the robot simulation software we have seen in section 2.2, running on **ROS Kinetic** (section 2.1) distribution. 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 like they were actually flying in the real world. In that, **UCTF**[22] has been 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* match, that can be played both in the real world and in the simulator. Diagram 6.1 shows all the components originally involved and how they are connected with each other. It consists of a simulator (Gazebo), an autopilot software (PX4 in figure) mounted on a real or simulated vehicle, and a mission management interface (GCS), linked together for building up, according to what provided by 3D Robotics, a Software-In-The-Loop (SITL) environment [20].

Starting from the provided configurations, some modifications have been applied in the JOL, in order to adapt it to our case. From figure 6.1, it can be seen that a Gazebo simulation environment is set up with certain **world and environment models** (section 2.2). In that, the original scenario configuration has been removed, and replaced with informations about buildings (position, occupancy and height) coming from the **OpenStreetMapFoundation**, that provides open-source license .osm files containing these kind of informations.

Drones have been inserted by means of their **URDF model** (section 2.2), describing their physics and dynamics. This is what has been done from UCTF developers, who used a **3DR Iris quadcopter** [1] as simulated drone, together with a scaled down Cessna fixed wing, that has not been used in our work, being not compatible with the purposes of our project. These vehicles are equipped with an **autopilot** software, that, managing all the technical aspects of the flight, allows high-level interaction with the drone. It could be chosen between PX4 or ArduCopter, 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 (section 2.7.2), that is used for transmitting commands and informations between vehicles and the control station. Gazebo node then, through its plug-ins (section 2.2), simulates the behaviour of Inertial Measurements Unit (IMU) and GPS sensors, publishing messages about IMU, GPS position and GPS velocity values on specific topics. This SITL configuration can be spawned in multiple instances, modelling different copters to exist at once, allowing us to run at least three simulated drones on a reasonably powered laptop.

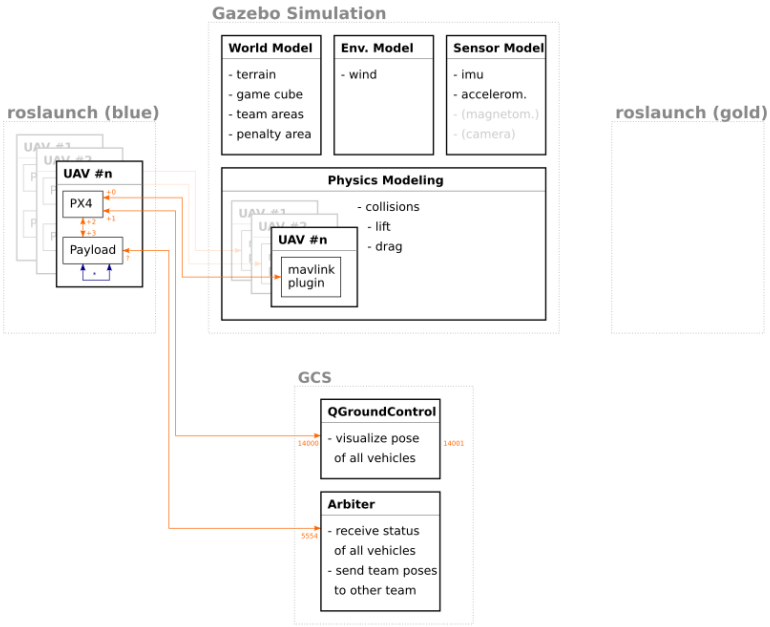


Figure 6.1: Overview of the uctf system architecture

Finally, drones are spawned in an arbitrary GPS position - that, changing some lines in a configuration file, we decided to be the garden next to an high skyscraper close to the polytechnic - together with their related **MAVROS** (section 2.7.2) node instances, representing the **ROS/-MAVLink** bridge, managing the communication between ROS and simulated environments. The relative graph is shown in figure 6.2.

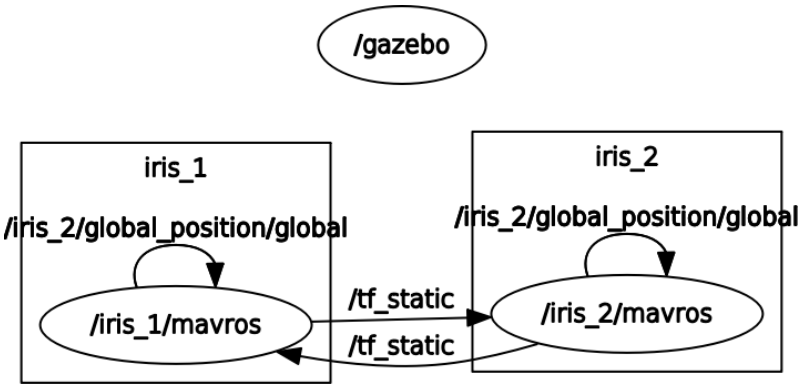


Figure 6.2: rqt graph of the ROS system when gazebo is running and 2 drones are spawned. Note that the position topics are closed on the same node because no subscribers have been created at that moment. The subscriber will be the localization node of our package

We then used as interface the **QGroundControl**[14] software, a powerful Ground Control Station providing full flight control and mission planning for any MAVLink-enabled PX4- and ArduPilot-powered UAVs (refer to figure 6.3). 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. 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 direct communication between our autonomous package and the autopilot, creating proper nodes to send and receive messages and manage missions.

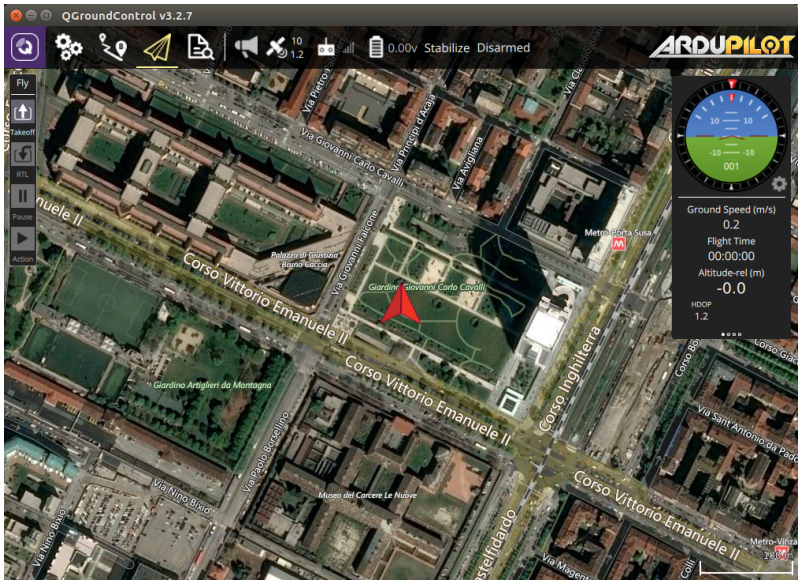


Figure 6.3: A screenshot of the QGroundControl software interface

Thanks to that, a 2-ways communication link is established between the vehicle and ROS, as suggested by the first 2 topics shown in the first listing here below, where can be seen that the ROS messages are translated into MAVLink ones and vice-versa. Relevant informations (i.e. position, altitude, battery status etc.) are shared on it, publishing or subscribing topics and commands/services (i.e. arming, take-off, landing and so on) can be sent/required to the ArduPilot by means of service clients.

Listing 6.1: Topics opened at the creation of the iris/mavros node, as soon as it is spawned

```
/iris_1/mavlink/from
/iris_1/mavlink/to
/iris_1/mavros/battery
/iris_1/mavros/global_position/global
/iris_1/mavros/mission/reached
/iris_1/mavros/mission/waypoints
/iris_1/mavros/setpoint_position/global
/iris_1/mavros/setpoint_velocity/cmd_vel
...
```

Listing 6.2: partial result of a rosservice list command, showing some of the command that can be used by sending specific messages on these topics

```
/iris_1/mavros/cmd/arming
```

```
/iris_1/mavros/cmd/command
/iris_1/mavros/cmd/takeoff
/iris_1/mavros/cmd/land
/iris_1/mavros/mission/push
...
```

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 them by reading specific topics, and even sending inputs - like the goal pose, sent as a geometry_msgs/PoseStamped message published by Rviz.

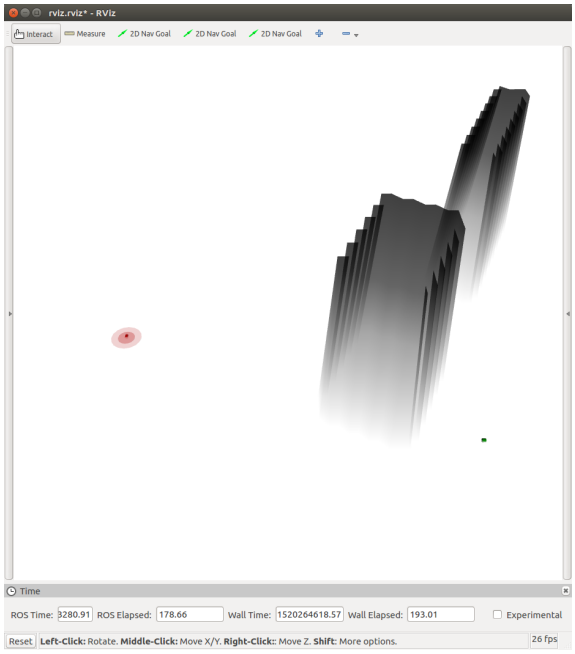


Figure 6.4: Screenshot of rviz with a spawned drone. The 3 buttons on top are used to send goals to 3 UAVs in 3 different topics

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 2.1, 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 kilometres away, the measurements and the actuations of internet take place and will take place practically everywhere.
Edge computing paradigm is the innovative concept that will boost the Cloud usage in many application fields, where the computation moves close to the source of data, augmenting efficiency and reactivity. 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" [1]. 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, with different machines connected with the same ROS_MASTER:

```
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
```

Using an architecture like the one in figure 6.5, 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 UAV, 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 practically known, but, according to the theory, it should be around $1\,ms$.
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

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

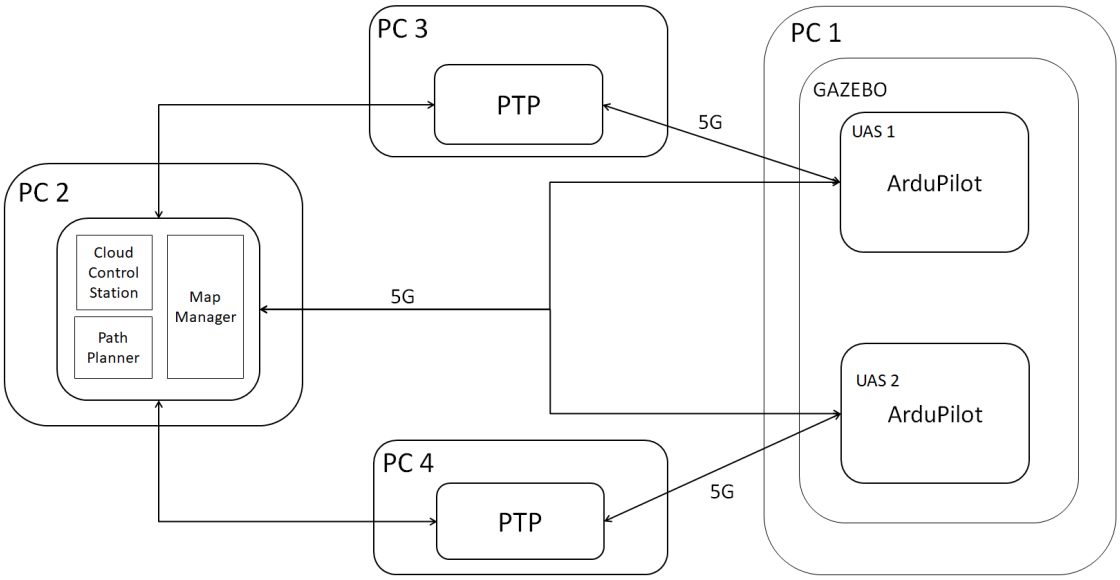


Figure 6.5: Architecture of the distributed simulation environment

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 Modelling Environment

First result, basis of the rest of the project, is the creation of a coherent and well defined risk map able to model the environment in which the unmanned missions will take place. As seen in section 4.4, the total risk map is created by the superimposition of many layers. In particular, it has to provide two UAV-independent layers, which will be the same for every drone that will fly (within the same airspace at the same height h): no_fly_zone and $covered\_area$. The former reports the buildings appearing from the flight altitude on, danger fixed obstacles for the Unmanned Aircraft. the latter represents all the structures which are lower than h, that may be seen as a very good sheltering for people on ground, in case of UAVs failures.
Due to the high computational cost, that obviously increase with the size of the map, we decided to limit the controlled area within a rectangle, big enough to test the goodness of CBUTM working principle.
The area we chose is the one contained in the rectangle defined by $[minlatitude, maxlatitude] = [45.0645000, 45.0741000]$ and $[minlongitude, maxlongitude] = [7.6537000, 7.6732000]$, which is the part of Turin around the San Paolo's skyscraper that, being an over 100 metres high building, introduces a criticism in the simulations. In figure 6.6, the data about buildings height taken

from OpenStreetMap[12] are shown. Starting from the corresponding .osm file, the input of the

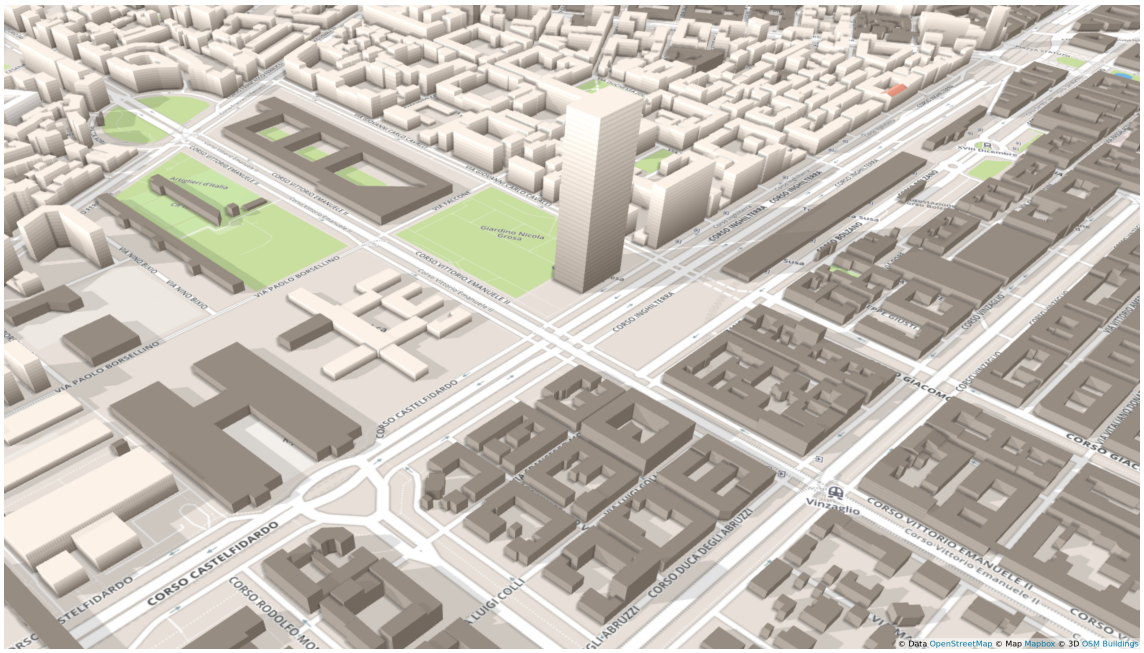


Figure 6.6: Open Street Map Buildings View

Create_Map service is a **point cloud**, which can be obtained through specific open source softwares. The result is the one in figure 6.7.

Saving it as .obj file, it is finally the proper input for keep modelling the environment, creating

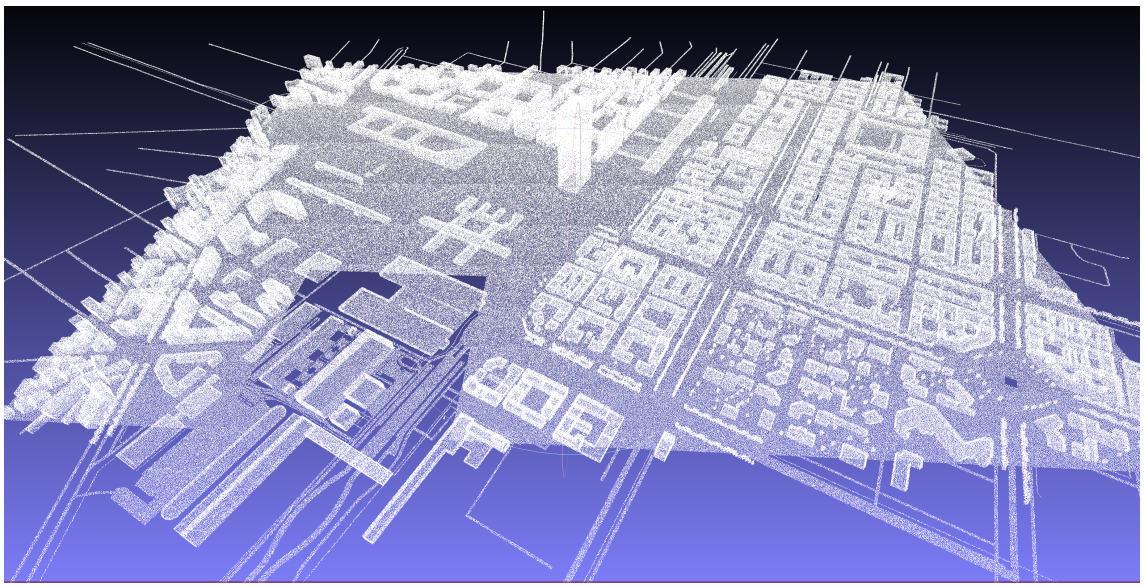


Figure 6.7: Point Cloud Model

the map. 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 $h = 25m$ and a **resolution of** $\mathbf{25}m^2$, 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 fig 6.8 and 6.9).

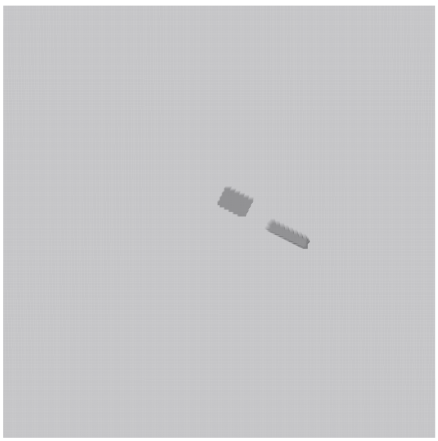


Figure 6.8: No Fly Zone Layer

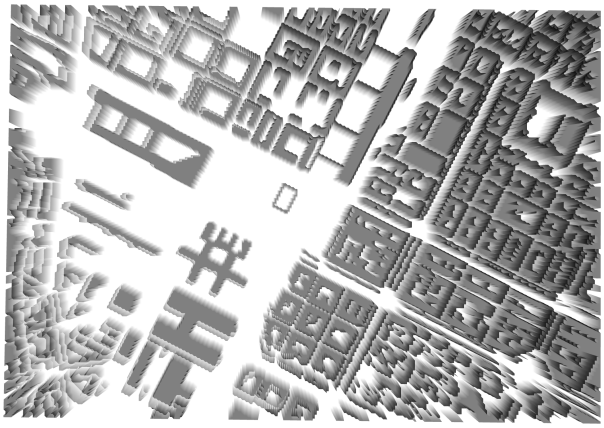


Figure 6.9: Covered Areas Layer

These two layers are clearly complementarity: merging 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 centre. 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 modelled as maximum (10) in the zones corresponding to the covered areas, or a generic mean value (6) in all the other parts. Finally, the environment's model is completed. It is structured in layers and published every 2 seconds on "risk_map_node/risk_map" topic.

6.4 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 6.10. This is not strictly true, in particular in

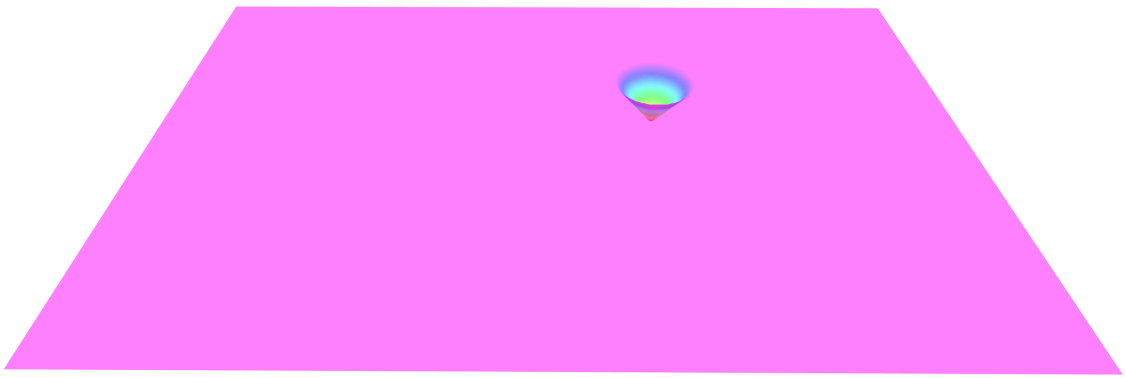


Figure 6.10: Signal QoS Layer

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, assigning a value for every cell according to the rule:

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

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 6.11.
Three kind of areas emerges clearly from the figure:

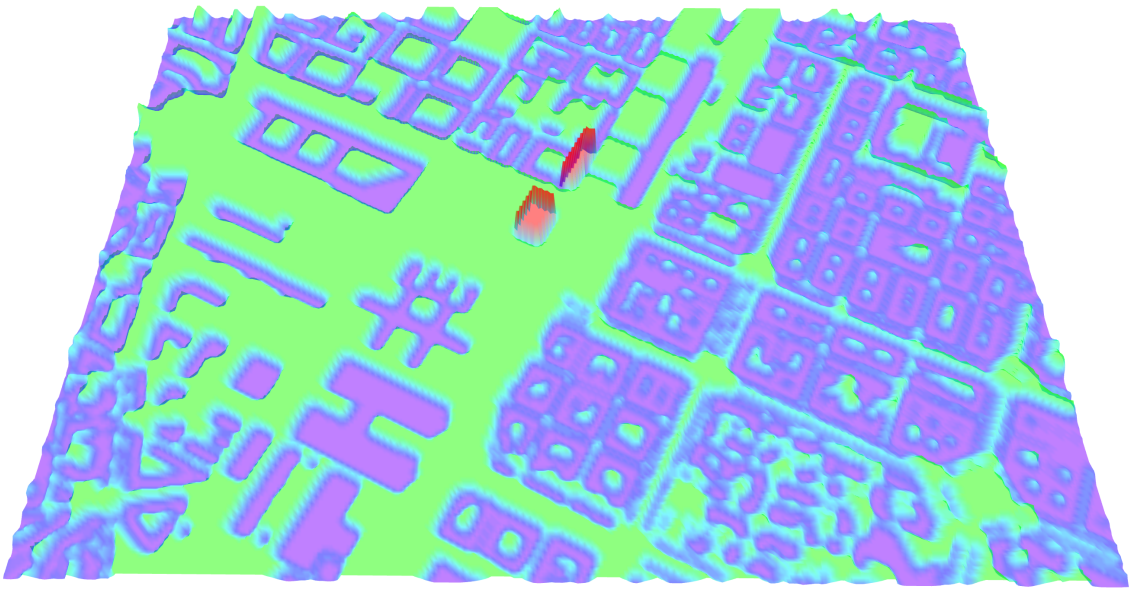


Figure 6.11: Risk Layer

- 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 4.6. 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 6.10. An example with $\alpha = 0.5$ is provided in figure 6.12.
In particular, it's possible to notice the no-fly zones and the area of no signal coverage, which in

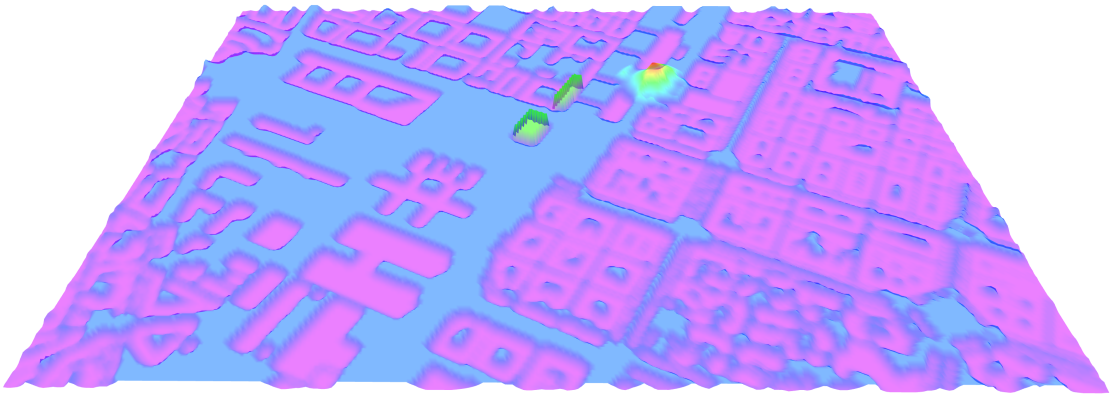


Figure 6.12: Cost Layer

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.

6.5 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 6.13 a typical 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

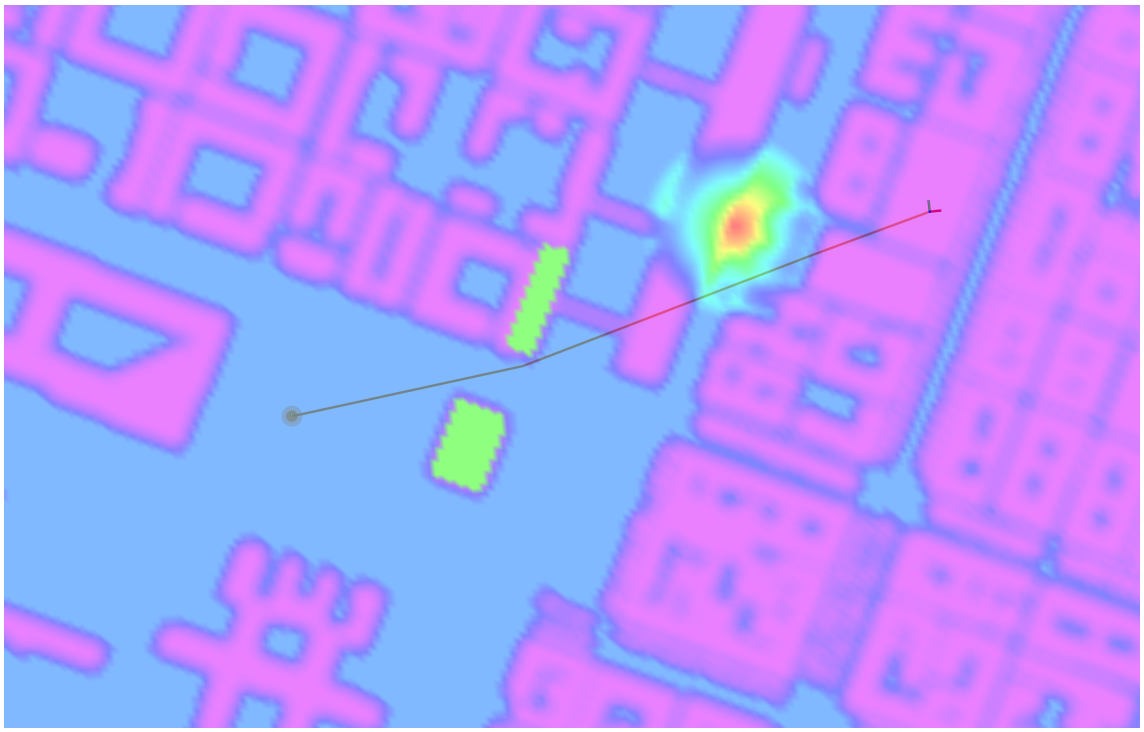


Figure 6.13: Path Planning Example

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 fig:6.13, 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 $authorization\_to\_fly$.

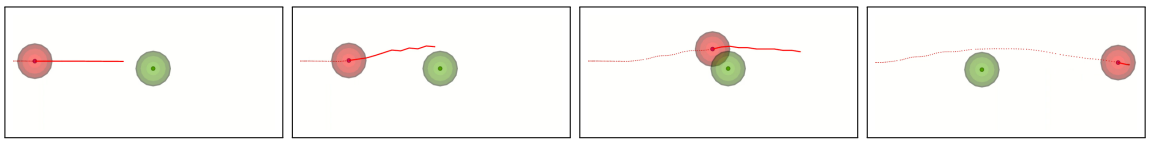
6.6 Collision avoidance

A set of tests and simulations has been performed to analyse the behaviour of the system and validate our approach. To this aim, 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 is it 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 here below, the showed map is built with the $noflyzone$ layer.

The first tested scenario 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 fig **??**, the first test sequence is shown. In that 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 needing to move from its original path, as soon as the other drone is detected in 6.14b. The *red* drone starts slightly moving out of its path, going around the *green* one, for then going back to its original path, and reach its goal.

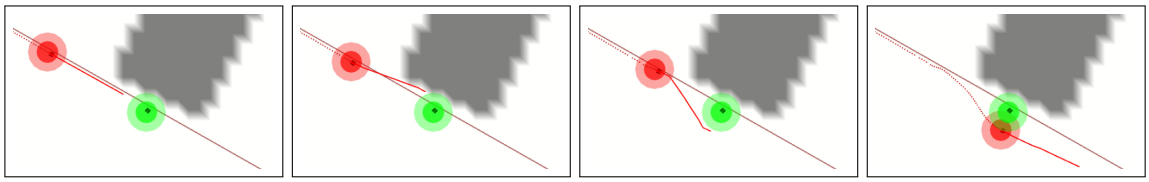


(a) (b) (c) (d)

Figure 6.14: Test 1: 2 UAVs, only one moving

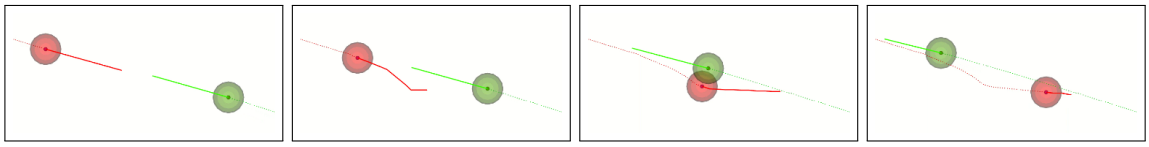
The same situation in the neighbourhood of a $noflyzone$ represented by a building is shown in fig:6.15. In this case, the *red* drone, when looking for where to escape from collision, tries to go leftward first, being that one the shortest way to she goal (fig:6.15b). When the presence of the building is detected (fig:6.15b), it immediately changes direction, overcoming the *green* obstacle deviating on its right-hand side (fig:6.15c), for then going back on the original path.

From now on, at least 2 vehicles will be in motion at the same time. The first, basic case involves 2 drones flying one against the other, shown in the sequence in fig:6.16. 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* in 6.16b, must be faster than before in changing trajectory. This is done in the optimal way shown in 6.16c, so that both the vehicles can safely accomplish their mission according to their priorities.



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

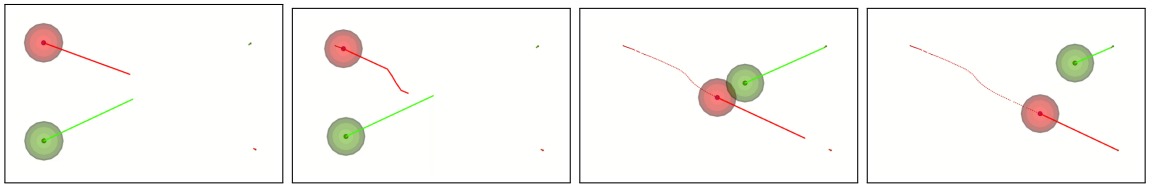
Figure 6.15: Test 2: 2 drones, only 1 moving. Presence of a fixed obstacle



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

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

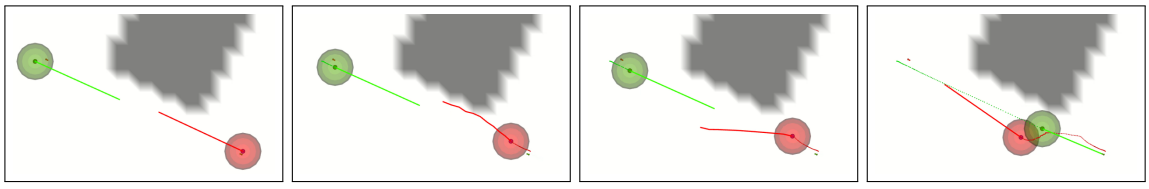
A slightly different circumstance is presented in the sequence 6.17 here below, where the two vehicles are crossing their paths in a diagonal way.



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

Figure 6.17: Test 3: 2 drones, both moving. Presence of a fixed obstacle

A bit more complex is the situation presented in 6.18, 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 6.18b, forcing the vehicle to go in the other direction, as can be seen from fig 6.18c.

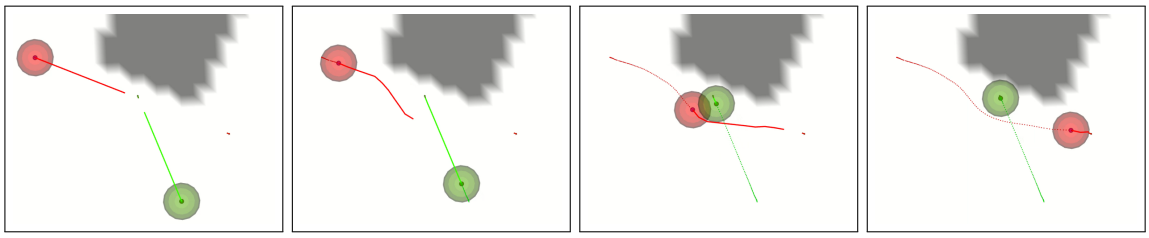


(a) (b) (c) (d)

Figure 6.18: Test 5: 2 drones, both moving one toward the other, in presence of a fixed obstacle

Sequence 6.19 shows a similar situation, with different dynamics: here, the *green* vehicle is approaching their building, and its mission consists in hovering on a fixed position in front of the building. In the same moment, the *red* drone is coming from the left-hand side, passing next to

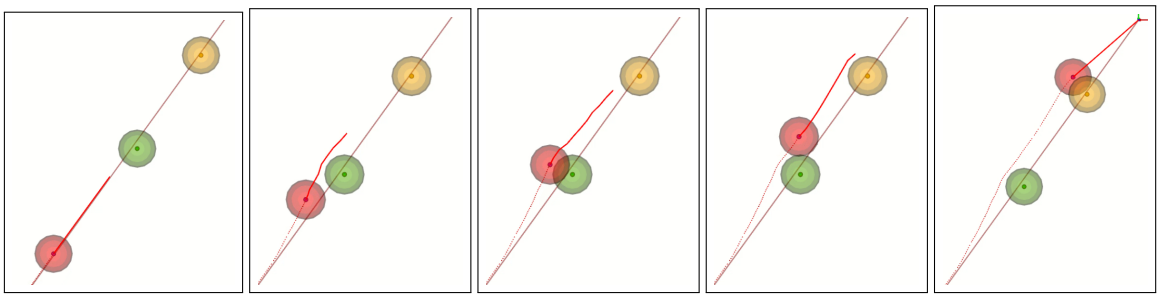
the building without stopping by. When it detects the higher priority UAV arriving, the PTP computes the optimal deviation from its original path, predicting to go all around the safety area of the *green* one (6.19b).



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

Figure 6.19: 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 fig:6.20, the *red* vehicle coming from below finds two standing vehicles on its path. When the first one has been detected in 6.20b, the vehicle starts moving left for avoiding it. Once it has been overcome, while trying to get back on its original path (6.20c), the *red* drone comes across the second higher priority vehicle, and changes again its trajectory for avoiding that one too, as can be seen in 6.20d.



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

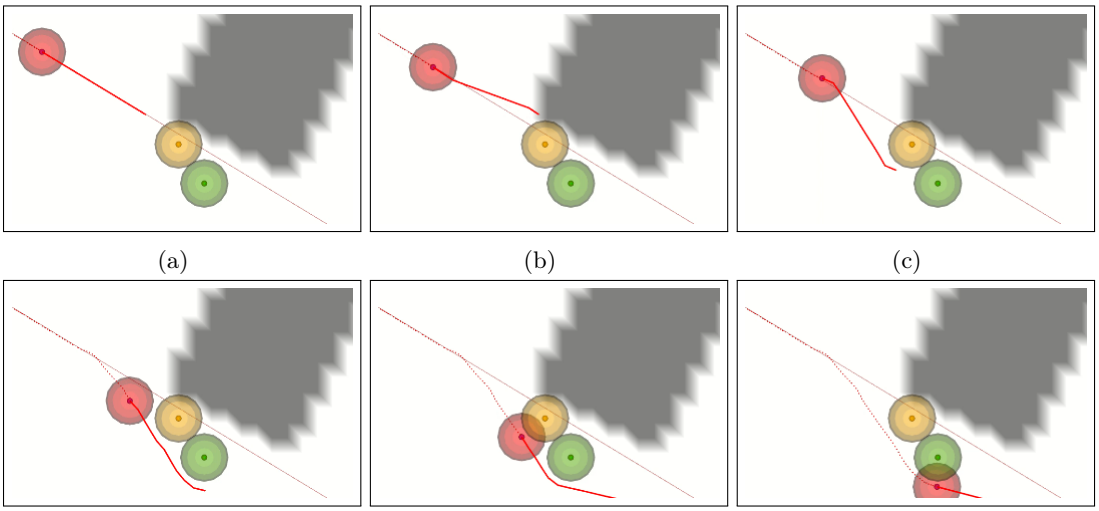
Figure 6.20: Test 7: 3 drones, one moving

Similar situation, with the only one moving drone meting the other two standing, is shown next (fig:6.21) 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 6.22, where the *red* and *yellow* vehicles are moving one against the other, while the *green* one is hovering in the middle.

The first 2 vehicles are flying in different directions (6.23a), and both will meet someone with higher priority on their path, forcing them to change trajectory (6.23b). 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. Since after few moments it detects the *yellow* vehicle will be passed by and the way will be free, it keeps going, knowing it is safe.

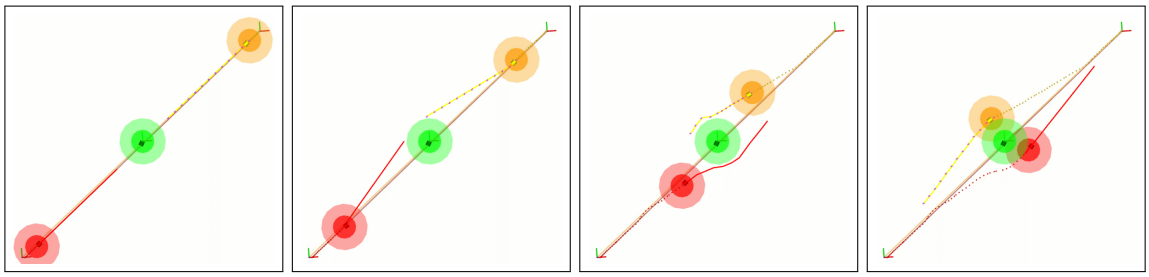
Again, we tested this intersection between 3 vehicles putting them close to a building. The result is shown in sequence 6.24. The two vehicles with lower priority are flying in opposite direction (6.24a), until they will get caught between the highest priority one and a fixed obstacle. When



(a) (b) (c)

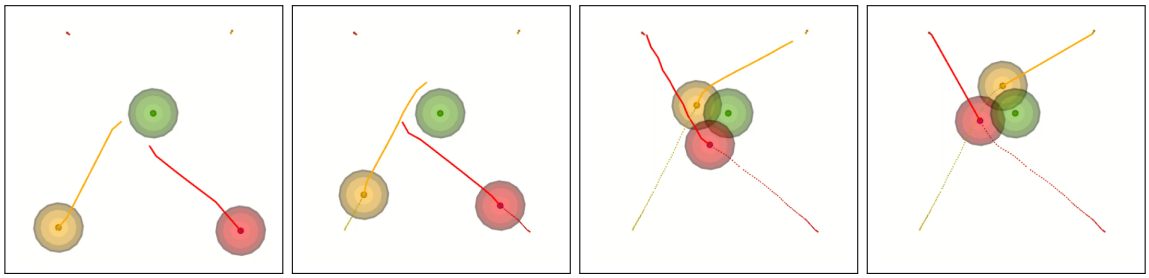
(d) (e) (f)

Figure 6.21: Test 8: 3 drones, one moving, in presence of an obstacle



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

Figure 6.22: Test 9: 3 drones, two moving, 1 standing in the middle

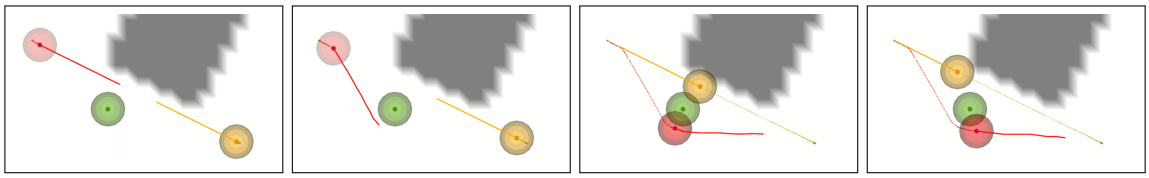


(a) (b) (c) (d)

Figure 6.23: Test 10: 3 drones, two moving in different directions, one hovering

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 (6.24b). They keep then flying their missions.

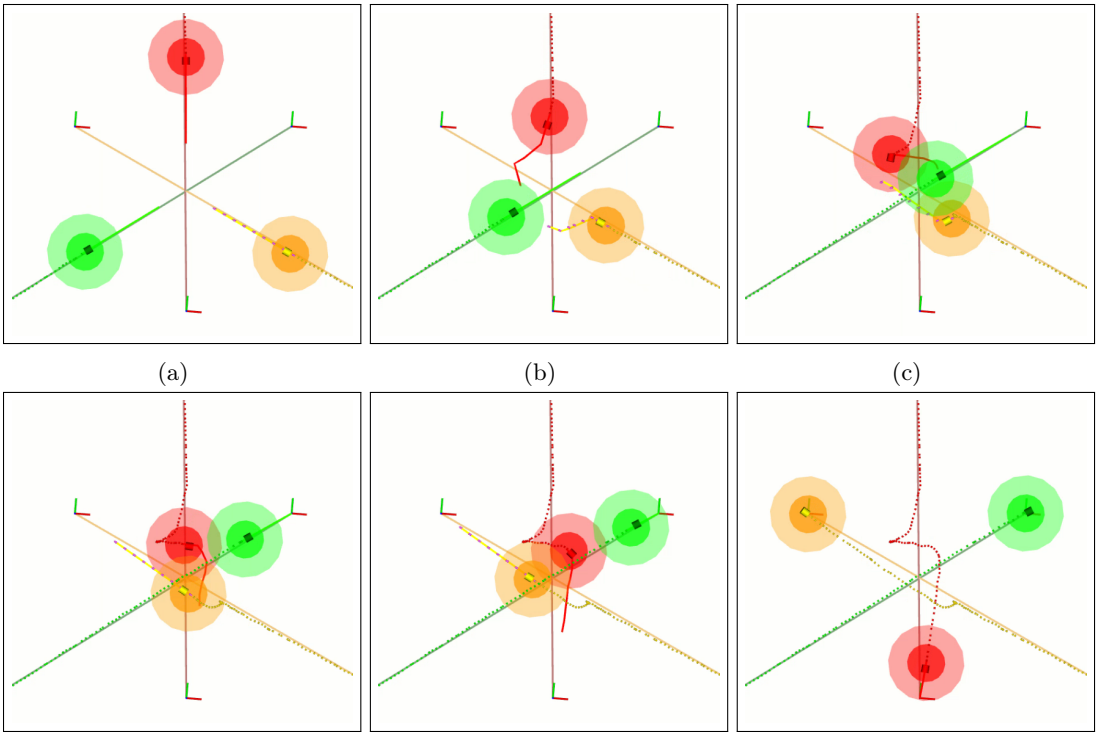
Last interesting test is shown in the sequence 6.25. When the 2 vehicles with lower priority



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

Figure 6.24: Test 11: 3 drones, two moving, 1 standing, in presence of a fixed obstacle

meet the *green* one, they start computing a different trajectory, that would bring them to move in the same direction (6.25b). 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 for avoiding both the *yellow* and the *green* vehicle (6.25c). 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 (6.25d). 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.



(a) (b) (c)

(d) (e) (f)

Figure 6.25: Test 12: 3 drones moving in different directions

Chapter 7

Conclusions

Aim of the thesis was to demonstrate the goodness of a UTM system built as a distributed cloud-based control system developed in the Robot Operating System (ROS). Main results of the project have been: the creation of a standardized metric for measure the risk of unmanned missions, that assure unmanned mission respect the safety requirements imposed by authorities; the development of a well structured network able to manage several drones, collecting their informations and coordinating their operations; the application of a receding horizon based control algorithm for fast and reactive collision avoidance procedures.

In particular, main focus of this thesis, part of the overall project conducted in the Joint Open Lab (JOL) with other colleagues, has been to build up a structured ecosystem, able to manage efficiently data collection, their stream within the network, and to be adaptable to traffic conditions, allowing the management of a huge number of vehicles. Main preoccupation has been the creation of a well defined structure and set of procedures, so that future developments can be easily integrated within the present architecture. The obtained results have shown the goodness of the proposed solutions, validating the approach that, with further developments, can play a role in the creation of a future UTM system.

This project is meant to be an interesting step forward for the development of a working UAV Traffic Management system, stressing the attention on this innovative approach of the collision avoidance problem, based on the Cloud Robotic paradigm. Future works will include the improvement of the proposed solution, and other steps toward a coherent adaptation to the future regulations that will be imposed by the authorities. Other interesting improvements will regard the accessibility of the proposed solution, allowing users to easily interact with it, facilitating the procedures needed for joining it. CBUTM has been already designed thinking that every user, after a mandatory application on an official portal, will be able to download a script to be run on a drone mounting ROS, or make it run on a in-cloud machine, when needed. In this way his vehicle, linked to the ROS node defined within the script, can be regularly inserted in the network, and it can proceed with mission definition, obtaining the authorization and fly.

An interesting test bench will be Harwdare-In-The-Loop simulations, aimed at analysing the efficiency of the system when the 4/5G connection is not just simulated, and how a system like this may react in a real environment.

Bibliography

[1] 3dr - iris; https://3dr.com/support/iris/.
[2] 4g connection speed; https://www.4g.co.uk/how-fast-is-4g/.
[3] Arducopter; http://ardupilot.org/copter/docs/introduction.html.
[4] Commercial unmanned aerial vehicle (uav) market analysis – industry trends, companies and what you should know.
[5] Development of uas regulation; https://www.icao.int/safety/ua/uastoolkit/pages/narrative-regulation.aspx.
[6] Droni oltre l'orizzonte: perché è ancora una chimera il volo bvlos, che farebbe davvero esplodere il mercato?
[7] Enav avvia selezione partner industriale per la gestione dei droni, https://www.enav.it/sites/public/it/media/comunicati/selezione-partner-induistriale-droni.html.
[8] gps - accurancy; https://www.gps.gov/systems/gps/performance/accuracy/.
[9] grid map package; https://github.com/ethz-asl/grid-map.
[10] Mavlink, http://qgroundcontrol.org/mavlink/start.
[11] Mavlink, https://github.com/mavlink/mavlink.
[12] Openstreetmap, https://www.openstreetmap.org/map=16/48.1495/11.5457.
[13] Px4 - the professional autopilot; http://px4.io/.
[14] Qgroundcontrol - ground control station for px4 and ardupilot uavs; http://qgroundcontrol.com/.
[15] Quadricottero news: Enac a dronitaly; http://www.quadricottero.com/2018/03/enac-dronitaly-bozza-regolamento-bvlos.html.
[16] Regolazione per la sicurezza - sapr; https://www.enac.gov.it/la-regolazione-per-la-sicurezza/sistemi-aeromobili-a-pilotaggio-remoto-(droni)/index.html.
[17] Ros introduction, http://wiki.ros.org/ros/introduction.
[18] Sconcept of operations for drones, https://www.easa.europa.eu/sites/default/files/dfu/204696-easa-concept-drone-brochure-web.pdf.
[19] Sesar: U-space, https://www.sesarju.eu/u-space.
[20] Setting up sitl on linux; http://ardupilot.org/dev/docs/setting-up-sitl-on-linux.html.
[21] Uas traffic management (utm) - rtt plan; https://www.faa.gov/uas/research/utm/media/faa-nasa-uas-traffic-management-research-plan.pdf.
[22] Uctf - unmanned capture the flag; https://github.com/osrf/uctf.
[23] Unmanned aircraft system (uas) traffic management (utm); https://utm.arc.nasa.gov/index.shtml.
[24] European Aviation Safety Agency. Opinion no 01/2018.
[25] Vittorio Maniezzo Alberto Colorni, Marco Dorigo. Distributed optimization by ant colonies.
[26] Bassam Alrifaee. Networked model predictive control for vehicle collision avoidance, 05 2017.
[27] Abdullah Bade Atikah Janis. Path planning algorithm in complex environment: A survey.

[28] 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.

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

[30] Heemels Maurice Vejdemo-Johansson Mikael Bemporad, Alberto. Networked control systems.

[31] Andrew Cunningham, Victoria Wu, Saad Biaz, and David Jones. Decentralized collision avoidance framework for unmanned aerial vehicles. 2013.

[32] Konstantinos Dalamagkidis. *On Integrating Unmanned Aircraft Systems into the National Airspace System.*

[33] N. Dousse, G. Heitz, F. Schill, and D. Floreano. Human-comfortable collision-free navigation for personal aerial vehicles. *IEEE Robotics and Automation Letters*, 2(1):358–365, Jan 2017.

[34] ENAC. Remotely piloted aerial vehicles - regulation.

[35] 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.

[36] Sara Giammusso. Cloud robotics in real time applications.

[37] Gianluca Ristorto Giorgio Guglieri, Alessandro Lombardi. Operation oriented path planning strategies for rpas.

[38] RANGE SAFETY GROUP. Common risk criteria standards for national test ranges.

[39] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[40] Hui-Min Huang. Autonomy levels for unmanned systems (alfus) framework. volume i: Terminology.

[41] 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.

[42] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, Aug 1996.

[43] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, April 2015.

[44] Sven Koenig Kenny Daniel, Alex Nash. Theta*: Any-angle path planning on grids.

[45] Busra Koken. Cloud robotic platforms.

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

[47] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

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

[49] Spyros Maniatopoulos, Dimitra Panagou, and Kostas J. Kyriakopoulos. Model predictive control for the navigation of a nonholonomic vehicle with field-of-view constraints. *2013 American Control Conference*, pages 3967–3972, 2013.

[50] Hojat Allah Hasanvand Masoud Nosrati, Ronak Karimi. Investigation of the * (star) search algorithms: Characteristics, methods and approaches.

[51] 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.

[52] Francesco Polia. Cloud-based uavs traffic management system: Trajectory tracking and collision avoidance.

[53] 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.

[54] E. Frazzoli S. Karaman. Sampling-based algorithms for optimal motion path planning. *The International Journal of Robotic Research*, 2011.

[55] Stefano Scheggi and Sarthak Misra. *An experimental comparison of path planning techniques applied to micro-sized magnetic agents*, pages 1–6. IEEE, 7 2016.

[56] 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.

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

[58] Anthony Stentz. The focussed d* algorithm for real-time replanning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1652–1659, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[59] Anthony (Tony) Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)*, volume 4, pages 3310 – 3317, May 1994.

[60] Csaba Viragh, Mate Nagy, Carlos Gershenson, and Gabor Vasarhelyi. Self-organized uav traffic in realistic environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1645–1652, Piscataway, NJ, 2016. IEEE.

[61] Roland Weibel and R John Hansman. Safety considerations for operation of different classes of uavs in the nas.