


Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (XXXI cycle)

Autonomous Navigation for Mobile Robots in Crowded Environments

Challenges with ground and aerial robots

Stefano Primatesta
* * * * * *

Supervisor
Prof. Alessandro Rizzo

Doctoral Examination Committee:
Prof. Gianluca Ippoliti, Referee, Università Politecnica delle Marche, Italy
Prof. Kimon Valavanis, Referee, University of Denver, USA
Prof. Marina Indri, Politecnico di Torino, Italy
Prof. Bartolomeo Montrucchio, Politecnico di Torino, Italy
Prof. Domenico Prattichizzo, Università di Siena, Italy

Politecnico di Torino
July 11, 2019

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Stefano Primatesta
Turin, July 11, 2019

Summary

The large diffusion of service robots in our life requires safe and efficient autonomous navigation in human environments. In particular, navigation in a crowded environment is a challenge, because the robot motion could compromise human safety.

This Ph.D. dissertation focuses on two different scenarios: with aerial and ground robots, respectively.

The first scenario focuses on safe navigation for Unmanned Aerial Systems (UASs) in urban areas. It is a critical scenario, because an impact of the UAS on the ground may cause casualties. To solve this problem, a novel approach is proposed, where a risk-based map and a risk-aware path planning strategy are used to determine a safe flight mission.

The risk-based map quantifies the risk of flying over an urban area, defining areas where the flight is allowed or not, because of no-fly zones or high risk. The risk is defined with a probabilistic risk assessment approach and by combining several layers with information about population density, sheltering factor, obstacles at the flight altitude and coverage of the mobile network used to connect the UAS with the ground.

Hence, a risk-aware path planning searches for a minimum risk path to reach a desired target position in the map and considering the risk-based map. In particular, the proposed risk-aware path planning strategy consists of two phases. First, an offline path planning searches for the globally optimal path based on a static risk-based map. Then, an online path planning updates the path according to a dynamic risk-based map. In this thesis, two different risk-aware path planning strategies are proposed: *(i)* using riskA* and Borderland algorithms, able to provide an offline and an online path planning, respectively; and *(ii)* with riskRRT^X, a path planning and re-planning algorithm used to perform both offline and online phases.

A simulation of a flight operation over a city is performed, demonstrating how the proposed approach is able to compute and maintain a safe flight mission, even in densely populated areas.

The definition and the implementation of a safe navigation for UASs in urban areas is the main contribution of this thesis. In particular, the risk-based map is a novel tool used to plan safe flight missions. Hence, both path planner algorithms are novel solutions, specifically designed to work with the risk-based map and minimize the risk to the population on the ground.

The proposed approach relies on a Cloud-based framework that enables intelligent navigation of UASs in urban environments, as well as manage and coordinate a fleet of UASs in the low altitude airspace. Thanks to mobile technologies, the UAS is connected with the ground with unprecedented opportunities, enabling Cloud technologies to be used with UASs. The proposed framework is another contribution of this thesis, designed to propose a reference architecture for autonomous UASs in urban areas.

On the other hand, the second scenario focuses on the autonomous navigation of ground robots in crowded environments. Unlike aerial robots, ground robots operate directly in an environment occupied by humans, requiring a safe and comfortable motion among people. The aim is to implement safe autonomous navigation to offer service robotics applications. For this purpose, a Cloud-based architecture for generic service robotics applications is presented. Thanks to Cloud technologies and the concept of Cloud Robotics, most of the intelligence resides on the Cloud. Only some essential elements to control the vehicle, manage the hardware and provide safety are installed aboard the robot. Such elements would guarantee the safe accomplishment of basic actions to maintain the safety toward people.

In particular, a dynamic path planner is proposed, able to compute and update a safe and valid path in highly dynamic environments, solving the so-called *freezing robot problem*. The path planner continuously checks, repairs and updates the current path, in order to always have a valid path to be executed by the robot.

Hence, a novel motion controller is presented, called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC). It comprises a Model Predictive Control method combined with the Equilibrium Point approach. PF-MPEPC determines an optimal trajectory of the robot toward the goal pose, avoiding obstacles with a smooth motion. Particle Filters in the prediction phase take into account uncertainties, such as measurement noise and disturbances.

Both the dynamic path planner and the motion controller are contributions of this thesis. The aim is the implementation of autonomous navigation strategies for service robots. For this purpose, two service robotics applications are described: *(i)* the robot Courier, a service robot in a workspace, which welcomes and escorts new visitors to the desired venue in an office environment, and *(ii)* Virgil, a service robot in a museum. Both service robotics applications demonstrate the effectiveness of the proposed Cloud-based architecture for service robots.

Acknowledgements

The work presented in this thesis would not been possible without the support of many people. It's hard to thank every people who helped me in the last three years.

Firstly, I would like to express my sincere gratitude to my supervisor prof. Alessandro Rizzo for the continuous support of my Ph.D. study and related research. Without his guidance and constant feedback this Ph.D would not have been achievable. I would like to thank also prof. Basilio Bona, my supervisor in the first part of my Ph.D., for giving me the opportunity to start this experience and for passing on his passion for Robotics to me. I would like to thank also prof. Giorgio Guglieri fot its support and help in the development of parts of this dissertation.

Many thanks to Marco Gaspardone, Roberto Antonini and Gian Piero Fici from TIM for their help and support during this period.

I would like to thank my fellow lab-mates for making the laboratory a happy and comfortable place to face daily challenges. Thanks to all people who worked and interacted with me during the Ph.D. at the DAUIN, DET and DIMEAS departments.

I would like to thank also prof. Anders la Cour-Harbo and all people I met at the Aalborg University, who supported me during the visiting period in Aalborg, Denmark.

Last but not the least, I would like to thank all the people who gave me moral support, encouragement and motivation to accomplish my personal goals. My girlfriend Eleonora who supported me and always provided cheerful encouragement every single day. My parents and my brother Maurizio who supported me all these years with their caring love and affection. All my friends for providing support and friendship that I needed.

To Eleonora,
and to my loving parents

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last decade, the number of robots acting in contact with people is growing and their role in our ordinary life is increasingly important.

Until a few years ago, the use of robots was exclusive for industrial applications, due to their ability to perform repetitive tasks with high precision and their high costs. Today, robots are used in a wide variety of applications, such as in the military field, in space exploration, search and rescue operations and in precision surgery, to name a few [186]. Moreover, trends and future directions of robotics suggest that robots will be increasingly used by changing the world in more things than one could possibly imagine [80].

Thanks to technological evolution, robots are getting popular in our daily life. The main reason is the technology development in the field of Service Robotics [196]. As defined in ISO 8373 [49], a service robot is "a robot that performs useful tasks to humans or equipment excluding industrial automation applications".

Service Robotics is an emergent field in robotics. Its aim is developing robotic applications to offer a service to humans. Unlike industrial robots that generally work in structured environments to perform accurate repetitive tasks at high speed, service robots operate in human environments and, often, they collaborate with humans. However, operating in a human environment is very complex, because it is a dynamic and unstructured environment. As a consequence, the development of reliable and efficient service robotics applications is a challenge [28]

The presence of humans in the operational environment is one of the major problems in Service Robotics. Service robots need to operate with particular attention to human safety. Moreover, in our risk-aware and risk-averse society, any potential risk to people is unacceptable.

Nowadays, Service Robotics is already applied to offer both professional and personal services. Professional services help in the execution of professional tasks in workplaces. Professional service robots can interact with people, for instance into escort people in an airport [199], to manage a warehouse logistic [4] and to perform precision surgery [18]. Other robots are used in defense, as well as to perform dangerous

tasks [220]. In the last years, service robots are widely used also in agriculture [20].

Professional service robotics involve also aerial robots [26, 129]. Unmanned aerial systems are gaining momentum in the last years, due to their low cost and flexibility to operate in a large variety of applications, such as monitoring, surveillance, package delivery, to name a few [203]. Moreover, due to their versatility of flight operations, unmanned aircraft will be involved in the concept of *smart city* [134]. Even if unmanned aerial systems don't interact directly with people, they are a hazard for public safety, because a crash of a vehicle on the ground may cause casualties, especially in urban areas [39].

On the other hand, personal service robots provide assistance to people providing domestic tasks or assisting them at home. One of the most popular personal robot is the vacuum cleaner robot, already diffused in our homes [64]. Personal service robotics has already applied in several applications, such as assistive technologies [130], to help elderly people [47] and for social assistance purpose [181].

The latest mobile technologies, such as 4G [38] and, in near future, 5G [8], offer a huge opportunity to connect the robot on Internet, without any additional infrastructure, opening service robotics to the Cloud Robotics paradigm [82]. Cloud Robotics is an emergent field in robotics, in which the robot becomes a simple agent connected to the Cloud. Then, the robot has access to Cloud technologies, such as Cloud Computing, Cloud Storage and Big Data, as well as the Collective Robot Learning, i.e. robots are able to share their knowledge to other agents [102]. Roughly speaking, the brain of the robot moves on the Cloud, providing high computational resources.

In literature, Cloud Robotics is already applied to service robots [12]. The most popular applications are: *(i)* the Google self-driving car, called Waymo [209], and *(ii)* the Kiva System robot for warehouse logistic [4].

In order to support Cloud-based service robotics applications, some Platform as a Service (PaaS) frameworks are proposed recently. In [206] a Cloud Robotics infrastructure called RoboEarth is introduced, based on the Cloud engine Rapyuta [85]. A similar framework is the RObotics in CONcert (ROCON) [171], a Cloud-based multi-robot framework based on the Robot Operating System (ROS).

1.1 Thesis Contribution

The aim of this thesis is to study a safe and autonomous navigation approach for mobile robots to provide service robotics applications. In particular, two different scenarios are considered:

- Unmanned Aerial Systems in urban areas;
- Ground robot navigation in crowded environments.

These scenarios are very diverse. However, they commonly provide autonomous navigation in crowded areas, where human safety is the main goal.

Urban areas are a critical scenario to perform flight operations because the population on the ground can be involved in a possible crash of the aircraft. In this thesis, we propose safe navigation for Unmanned Aerial Systems (UASs) in urban areas, taking into account the risk to the population when the aircraft flies over an inhabited area.

The proposed approach relies on a Cloud-based framework that enables intelligent navigation for UASs, as well as manage and coordinate a fleet of UASs in the low altitude airspace. Thanks to the newest mobile technologies, the UAS (or a fleet) is connected with the ground with unprecedented opportunities, enabling the use of Cloud technologies with unmanned aircraft. Our intention is to implement the use of a connected UAS in Beyond Visual Line-of-Sight (BVLOS) able to perform autonomous flying.

In particular, our work focuses on a risk-aware mission planning approach that uses a risk-based map to assess the risk to the population. The risk-based map is a two-dimensional map that quantifies the risk over urban areas, defining in which locations the flight is allowed or not, because of the presence of obstacles at the flight altitude, no-fly zones or high-risk areas, as well as determining the coverage of the mobile network.

Then, the risk-based map is used to determine a safe flight mission minimizing the risk. The risk-aware path planner consists of two phases: offline and online path planning. The offline phase searches for the minimum risk path based on static information of the risk-based map, while the online one updates and adapts the offline path to changes in the dynamic map. Two different mission planning strategies are presented, based on a common risk-aware path planning approach. The first method relies on riskA* and Borderland algorithms, while the second one uses the riskRRT^X algorithm.

The combination of risk-based map and risk-aware path planning allows a safe flight mission to be defined. Simulation results corroborate the proposed approach.

Safe navigation of UASs in urban areas is the main contribution of this thesis. In particular, the proposed risk-aware path planning based on a risk-based map is a novelty. Compared with methods ath the state-of-the-art, the proposed approach uses a probabilistic risk assessment method estimating the probabilistic impact area and considering different behaviors of the aircraft during the descent. This is a novel and promising tool used to quantify the risk of UASs over urban areas. Hence, risk-aware path planning algorithms are specifically designed to compute a safe path in the risk-based map, minimizing the risk to the population on the ground.

The second scenario discussed in the thesis concerns the autonomous navigation of ground robots in crowded environments. In this scenario, the robot directly operates in a human environment moving among people. The robot needs to navigate safely, avoiding collisions with people and reaching the desired target position with a smooth and safe motion.

For this purpose, a Cloud-based architecture for a generic service robotics application is presented. This architecture is distributed between the Cloud and the robot. Most of the intelligence resides on the Cloud, able to provide high computational resources, while on-board the robot there are only some essential elements to control the vehicle and to guarantee a safe motion.

A dynamic path planner is proposed, able to compute and maintain always a valid path in high dynamic environments, as well as solving the *freezing robot problem*, i.e., when the robot stops the navigation because of a dynamic environment, waiting for new instructions to reach the desired target.

Then, a novel motion control for mobile robots is presented, called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC). It is a Model Predictive Control method using the Equilibrium Point Control approach. Particle filters in the prediction step take into consideration uncertainties, such as the measurement noise and other disturbances.

The dynamic path planner and the motion controller are the second main contribution of this work. In particular, they are proposed to implementing autonomous navigation for mobile ground robots. For this purpose, two service robotics applications are described: Virgil, a museum robot, and the robot Courier, a robot in a workspace.
Virgil is a service robot in a museum, able to navigate autonomously and to stream a real-time video to remote users.
The robot Courier is a robot in a workspace that welcomes new visitors and escorts them to the desired office.

1.2 About this Ph.D thesis

This work is the result of three years Ph.D. activities in the field of *Autonomous Robots* and *Cloud Robotics* at the Politecnico di Torino and in collaboration with TIM S.p.A., the main telecommunication company in Italy. TIM has an interest in exploring service robotics solutions and Cloud Robotics, in order to create new services and products for end users.

The collaboration between TIM and Politecnico di Torino involves several researchers, professors and students from different fields, such as engineers, designers, lawyers, etc., working together on Service Robotics. This multidisciplinary project does not concern only technological research, but also the social and psychological aspects of robotics, as well as the legal problem about the civil and legal liability of robots.

The research here presented has the goal to study and implements solutions for Service Robotics exploiting the Cloud Robotics paradigm.

1.3 Outline

This document is split into two parts. The first one, called *Safe navigation for Unmanned Aerial Systems*, presents a navigation approach for unmanned aircraft in urban areas, minimizing the risk to the population on the ground.

In Chapter 2 a Cloud-based architecture for intelligent navigation and coordination for Unmanned Aerial Systems (UASs) in urban environments is presented, used as a reference architecture in the first part of the thesis. Chapter 3 introduces a risk-based

map, able to assess the risk of UASs in urban areas. Then, in Chapter 3, two risk-aware path planning strategies are presented, able to compute a safe path, minimizing the risk to the population on ground assessed by the risk-based map. Chapter 5 reports a preliminary simulation, where a safe flight mission is performed in an urban area.

The second part of the thesis, called *Autonomous navigation for ground robots in crowded environments*, describes some techniques of autonomous navigation for service robots. In Chapter 6 a Cloud-based architecture to offer a generic service robotics application is presented. Then, Chapter 7 presents a dynamic path planner for mobile robot navigation in crowded environments, while Chapter 8 introduces a motion control method, called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC). Two service robotics applications are described in Chapter 9: Virgil, a museum robot, and the robot Courier, a robot in a workspace.

Finally, we draw our conclusions in Chapter 10.

Part I

Safe navigation for Unmanned Aerial Systems

Chapter 2

Cloud-based architecture for Intelligent Navigation and Coordination of UASs

This Chapter introduces a Cloud-based architecture for risk-aware intelligent navigation and coordination for Unmanned Aerial Systems (UASs) in urban areas. It is the reference architecture used in the first part of the thesis titled *Safe Navigation for Unmanned Aerial Systems*.

The objective of this Chapter is the definition of a Cloud-based framework to implement intelligent and autonomous UASs, minimizing the risk to the population on the ground. The aim is to propose a reference framework to enable safe flight operations in urban areas. In particular, our architecture proposes the use of UASs connected with the ground using a mobile network connection. Exploiting Cloud technologies, the UAS has more capabilities implementing an intelligent flying performed in Beyond Visual Line-Of-Sight (BVLOS), as well as to coordinate a fleet of UASs in the low altitude airspace.

A preliminary version of the Cloud-based architecture is introduced in [161], while in [160] is extended for coordinate a fleet of UASs.

This Chapter is organized as follows. In Section 2.1 a background description is reported. The proposed Cloud-based architecture is introduced in Section 2.2, describing in general terms each module. Hence, we discuss the proposed framework in Section 2.3.

2.1 Background

Nowadays, UASs are widely used in a wide variety of applications, due to their versatility in performing heterogeneous flight operations and low cost [203]. Thanks to the concept of smart cities, UASs will be involved in urban areas for inspection, monitoring, mapping and package delivery [134, 137]. UASs are the ideal platform to sense the

environment, bringing benefits to IoT (Internet-of-Things) applications [71].

Newest mobile technologies, such as 4G [38] and, in the near future, 5G [8], connect the aircraft with the ground with unprecedented opportunities [214]. The aircraft is connected with the Internet with a reliable, safe and high-performance connection, opening the use of UASs in the field of Cloud Robotics. Thanks to Cloud technologies the UAS has access to unlimited resources, improving its capabilities. The emergence of these technologies enables the development of intelligent and autonomous unmanned aircraft, improving the UAS performances in terms of capability and safety.

In a future scenario, with the extensive use of UASs, unmanned aircraft will fly over urban areas. As a common problem of multi-agent systems [7], the coordination and cooperation among them are necessary to improve performances and safety. Since the Cloud allows resources to be shared between UASs, a Cloud-based framework is able to provide a UAS Traffic Management (UTM).

Anyway, regulations strongly limit the use of UAS. Currently, most of the operations are executed in Visual Line-of-Sight (VLOS) only with the prior authorization from the National Aviation Authorities (NAAs), such as ENAC (Enten Nazionale per l'Aviazione Civile) in Italy and FAA (Federal Aviation Administration) in the United States. Even if, they are opening the possibility to execute flight operations in Beyond Visual Line-of-Sight (BVLOS). In any case, urban areas are a special scenario because of the presence of people on the ground. As a consequence, at present, the flight over cities is strongly restricted.

2.1.1 Mobile network

In the last years, mobile networks have great progress with the development of the fourth generation of cellular mobile communication (4G). It offers a reliable and high-performance connection with wide coverage, especially in urban areas.

Generally, UASs are connected with the ground with a point-to-point radio link using unlicensed radio frequencies. However, there are many issues with this communication. The point-to-point radio link is a short-range connection and requires a direct and uninterrupted line-of-sight signal between a vehicle and an operator. This is a strong limitation, especially in the prevision of BVLOS flight operations. Moreover, often the radio link uses unlicensed frequencies at 2.4 GHz and 5 GHz in sharing with a lot of other applications and users. For example, these frequencies are used by wi-fi computer networks (ISO IEEE 802.11). This implies a lot of interference in the radio link communication.

For this reason, according to [212], the main cause of incidents with UASs is the loss of connection between the aircraft and the pilot, causing the so-called "fly away" event, where the pilot loses the authorization of control the vehicle. Then, a reliable communication system prevents the risk of accidents.

Mobile networks are suitable to provide UAS communication, bringing a lot of benefits [217]. It is a long-range communication with wide bandwidth and safe wireless

connectivity. In fact, a mobile connection is a standard and scalable connectivity, using a licensed cellular mobile network with a secure communication channel. For these reasons, mobile connections enable the BVLOS operational mode, where the aircraft is not constrained to the pilot's position. Moreover, the use of SIM (Subscriber Identity Model) credentials introduces a method of UAS identification in the airspace. Every SIM is identified with the IMSI (International Mobile Subscriber Identity) number, while the user equipment using the IMEI (International Mobile Equipment Identity) number. This allows both aircraft and operator to be identified, solving the identification problem raised by UTM systems.

At present, there are already several works where the UAS is connected using the 4G (or 4.5G). In [165], a connected UAS is used for surveillance, while in [213] for a search and rescue application.

Anyway, in the very near future, the fifth generation (5G) will revolutionize the field of mobile networks. 5G provides a high Quality of Service (QoS), with large bandwidth and low latency, as well as high-density connection [116]. Moreover, new concepts of mobile Internet technologies are introduced, such as network slicing, beamforming and mobile edge computing [176]. Network slicing technology provides different service level agreements to users in the same network, guaranteeing some network functionalities [219]. Beamforming technology allows antennas to be dynamically reconfigured in order to follow and offer an optimal connection to the user equipment [170]. Mobile edge computing moves Cloud resources in the edge of the mobile network, reducing the network delay [83]. Especially mobile edge computing will revolutionize the IoT concept, where the response time of a Cloud server will be about 1 ms [188].

According to [214], 5G will solve several problems in the UAS industry, providing many functionalities, such as the remote control, real-time video streaming in high definition, aircraft identification and positioning.

At present, the mobile network is not designed to support UASs in the low airspace. In fact, the mobile network provides the optimal signal to users on the ground, while the quality of the mobile connection varies with the altitude [204]. Hence, the mobile network should be adapted to cover low airspace. Antennas should be deployed to propagate the signal in the tri-dimensional space, covering the low airspace uniformly. For this purpose, the telecommunication operator TIM has conducted in the city of Turin the first measurement activities in the sky on the mobile network using unmanned aircraft. In collaboration with the Politecnico di Torino, they are working on the DRNet project (Drone Ready Network wireless) to provide the first drone-ready network in Europe [54]. Similarly, also other telecommunication companies are working on the introduction of connected UASs in the airspace, such as Qualcomm [166] and Ericsson [120].

2.1.2 Regulations

Regulations for safe and legal operations of lightweight UASs are provided by the National Aviation Authorities (NAAs), with the aim to minimize the danger to people, property and other vehicles.

At present, each country refers to its NAA, such as ENAC in Italy. Currently, in Italy, most of the flight operations must be executed in VLOS, while only special and particular operations have obtained permission to operate in BVLOS.

However, in 2017 EASA (European Aviation Safety Agency) published a proposal for legislation for UASs in the European airspace [55, 56]. They adopted the JARUS (Joint Authorities for Rulemaking on Unmammed Systems) proposal, in which risk-based rules are defined to provide safe flight operations.

Urban areas are a special and critical case, because of the high population density. In fact, NAAs strongly restrict the flight in these areas, even if some experimental tests have been conducted. Recently, the FAA launched the UAS Integration Pilot Program (IPP), where BVLOS flight operations are tested in ten US cities [60]. However, for an analysis of the worldwide regulation, refer to [27, 192].

For this reason, EASA in [57] defines a new subcategory of UASs called *A1 - Fly over people* to identify which aircraft are suitable to fly over inhabited areas.

2.1.3 Previous work

In literature, there are several works that propose the use of the Cloud with UASs. However, in some works, the Cloud is used only to implement the ground segment [128, 119], while all navigation capabilities are on-board. Unmanned aircraft are also used for IoT applications [139], where they sense the environment and collect data, supported by the Cloud. In [70] the new term *Internet-of-Drones* is defined.

Other Cloud-based architecture for UASs are proposed in [106, 90], while Cloud-based and on-board architectures are compared in [180]. Anyway, the research in this field is in the early stages.

2.1.4 Current work

In this Chapter, we present a Cloud-based architecture for intelligent navigation and coordination for Unmanned Aerial Systems. The proposed architecture is distributed between the Cloud and UASs, while the aircraft communicates with the Cloud using a mobile Internet connection.

The Cloud plans and executes a safe flight mission, considering the risk of flying over populated areas, avoiding obstacles and no-fly zones, as well as avoiding collisions with other vehicles in the same airspace. Moreover, the Cloud server manages and monitors all vehicles in the same airspace.

The aim of this framework is to enable flight operations of small UASs in the low airspace over urban areas, providing an efficient and safe intelligent flight. In particular, the connected UAS with autonomous capabilities and operating in BVLOS provides the highest mobility [19].

2.2 Cloud-based architecture

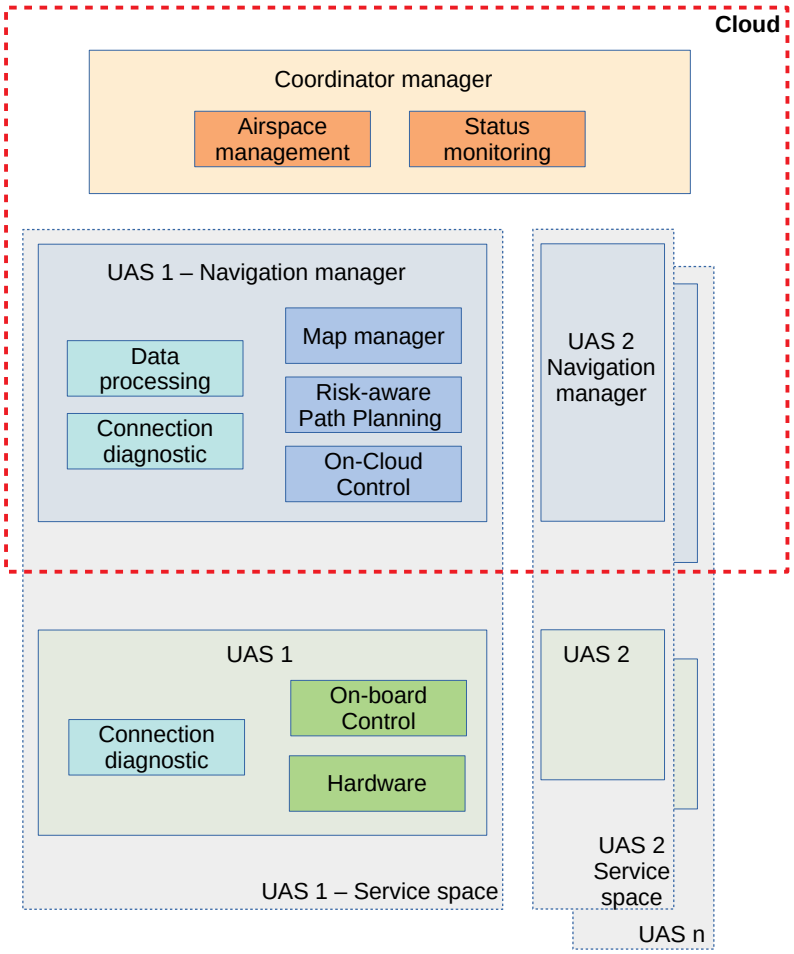


Figure 2.1: The Cloud-based architecture for intelligent navigation and coordination for UASs.

The Cloud-based architecture proposed in this thesis is reported in Figure 2.1, where the framework is distributed between Cloud and UASs.

The main module is the Coordinator Manager that aims to coordinate and monitor a fleet of UASs in the airspace. The Coordinator Manager is a central module able to monitor the status of all vehicles in the airspace and assigns the priority to each aircraft in order to coordinate a fleet of UASs.

The Navigation Manager module manages the autonomous navigation of UASs. According to the architecture, a Navigation Manager exists for each vehicle managed by the Cloud. The Navigation Manager module includes some sub-modules, such as the Map Manager, the Risk-aware Path Planning and the On-Cloud Control, as well as the Data Processing module and the Connection Diagnostic.

The On-board Control, the Connection Diagnostic and the Hardware modules reside on the UAS. The On-board Control aims to control the vehicle when a bad connection with the Cloud occurs, guaranteeing the flight safety in any condition. The Hardware module includes sensors and actuators, as well as the payload equipped on the aircraft. The Hardware receives and actuates control commands from the control system, while transmits data acquired by on-board sensors to all modules in the architecture.

Finally, since the communication between Cloud and UAS is established by a mobile Internet connection, the Connection Diagnostic module continuously monitors the quality of connection. The Connection Diagnostic module resides both on the UAS and on the Cloud, in order to be aware of disconnections from both sides of the communication.

The proposed architecture uses a distributed priority-based approach to solve the coordination problem between UASs. A priority is assigned to each vehicle. Hence, a vehicle needs to avoid only other aircraft with higher priority. The priority is assigned by a central element, i.e., the Coordinator Manager, but the collision-free flight is planned by the Navigation Manager. This approach is preferable to a centralized one because with centralized approaches the complexity of the coordination problem increases with the number of vehicles [7]. Moreover, the distributed approach is more appropriate with Cloud-based systems, thanks to the scalability of the Cloud.

In addition, reactive obstacle avoidance for unexpected obstacles should be performed by the Control system, both on-Cloud and on-board, in order to guarantee additional safety.

2.2.1 Coordination Manager

The Coordinator Manager is the core of the proposed Cloud-based architecture because it manages and monitors a fleet of UASs in the airspace.

When a new flight mission is required, the Coordination Manager defines the *operational area* of the aircraft, considering a starting and target positions defined by the mission. The operational area is used to identify which vehicles operate in the same area, as well as to determine the area evaluated by the Navigation Manager to plan a safe mission.

The Coordination Manager assigns the priority of each UAS, based on the vehicle and mission type. For instance, flight operations for emergencies or public safety have the highest priority. Moreover, the priority assignment is dynamic. If an emergency situation occurs, the highest priority can be assigned to a vehicle, able to reach a safe landing point.

This block monitors the execution of each flight mission. If an aircraft changes its mission or does not respect the time table, the Coordinator Manager notifies other vehicles, which update their flight mission.

The coordination manager has no strict timing requirements. Generally, the definition of the operational area and the priority assignment are tasks executed before the flight mission starts. The only task time-constrained is the monitoring of each UAS. Each vehicle should be monitored with a reasonable fixed frequency. Moreover, if an emergency condition occurs, the Coordinator manager needs to notify vehicles as soon as possible.

2.2.2 Navigation Manager

The Navigation Manager aims to plan and execute a safe flight mission. Since the proposed architecture uses a distributed approach, a Navigation manager is allocated for each aircraft. The Navigation Manager provides many functionalities and it is composed of some sub-modules: a Map Manager, a Risk-aware Path Planning, an On-Cloud Control System, a Data Processing and a Connection Diagnostic.

Map Manager

The Map Manager provides the reference map used by the Navigation Manager. Considering the operational area, the Map Manager computes a risk-based map that quantifies the risk to the population on the ground in the operational area. The risk-based map is a two-dimensional location-based map, in which each element refers to a geo-location with the risk of flying over the associated location.

The risk-based map is generated using several layers:

- **Population density layer:** defines the population density distribution in the map;
- **Sheltering factor layer:** defines the sheltering level of each location in the map;
- **No-fly zones layer:** determines in which locations the flight is not allowed;
- **Obstacles layer:** defines the height of obstacles in the map;
- **Coverage layer:** determines the quality of the mobile network signal in locations of the map.

These layers are combined together using a probabilistic risk assessment approach and considering mission requirements.

Since layers may change in time, the resulting risk-based map is dynamic. For this reason, the Map Manager continuously updates the map. The update rate of the Map Manager depends on the dynamics of the source information. Anyway, all layers refer

to data with a slow dynamic and, as consequence, there is no need to update the risk-based map with high frequency. For instance, assuming the population density layer to be updated every 10 minutes, the risk-based map should be updated with the same rate.

The risk-based map is proposed and explained in detail in Chapter 3.

Risk-aware path planning

The risk-aware path planning aims to compute a safe flight mission, minimizing the risk to the population and avoiding collisions with obstacles and other vehicles, as well as no-fly zones and high-risk areas. The path planning is one of the most important elements of the definition of a mission because it defines the main route of a UAS.

The risk-aware path planning refers to the risk-based map computed by the Map Manager. It consists in two phases: before the beginning of the mission, an offline path planning searches for the optimal path; hence, on online phase updates the offline path according to changes in the dynamic risk-based map.

Generally, the offline path planning is computed when the aircraft is still on the ground, hence, it is not time-constrained. On the opposite, the online path planning updates the path when the aircraft is executing the flight mission. For this reason the on line path planning is time constrained by updating the path as soon as possible.

The risk-aware path planning approach is described in detail in Chapter 4, where two risk-aware path planning strategies are proposed.

On-Cloud Control

The on-Cloud control system aims to control the aircraft providing an optimal control input to the UAS. Using sensor data from the UAS block, the on-Cloud control system determines the motion of the aircraft following the path defined by the risk-aware path planning and avoiding unexpected obstacles.

Generally, the control system is performed on-board the aircraft and, due to on-board resources limitation, complex control techniques are not suitable to be implement. On the contrary, thanks to Cloud Computing, the on-Cloud Control system has access to unlimited resources, enabling the use of complex control strategies.

Anyway, the control loop is closed through a mobile Internet communication. Hence, the on-Cloud control is a Networked Control System (NCS), where the main problem is the network latency between the Cloud and the aircraft.

In literature there are some works about a Cloud-based control of UASs [3] and, generally, for networked systems [183, 123].

Generally, a flight controller requires real-time performances, in order to provide a fixed controller frequency to the aircraft. As a consequence, also the On-Cloud Control requires real-time performances. Unlike traditional flight controllers, the network increases the complexity of the controller because of the network delay and disconnections. For this reason, an on-board controller is always required, to provide safety and

to control the aircraft in emergency condition.

Data Processing

The Data Processing module processes all data from the on-board sensors.

In fact, thanks to Cloud technologies, sensors data can be processed on Cloud with advanced algorithms. For instance, using images from the on-board camera, unexpected obstacles can be detected using artificial vision algorithms [114].

Similarly, using sensor measurements from motors encoders and from IMU (Inertial Measurement Unit), advanced fault detection approaches [124, 77] can be implemented on Cloud, improving the safety of UAS operations.

The timing requirements of this module depends on the task supported by the data processing.

Connection Diagnostic

The Connection Diagnostic module monitors the communication between the Cloud and the aircraft. Since the Cloud has a fundamental role in the proposed architecture, monitoring communication over the mobile network is essential to maintain all functionalities. Moreover, the Connection Diagnostic module is located both on the Cloud (one for each vehicle) and on-board the aircraft, because, in case of disconnection, both Cloud and vehicle need to be aware of it.

The Connection Diagnostic evaluates also the network latency. In fact, in case of weak signal, the delay between the Cloud and the aircraft can be considerable, compromising the on-Cloud control performances. Hence, if the network latency is greater than a reasonable threshold, a disconnection is assumed.

If a disconnection occurs, the aircraft must be able to maintain the safety and the stability of the vehicle. Hence, a sequence of actions should be defined, to recovery the communication with the Cloud or to land in a safe landing point.

2.2.3 Unmanned aerial vehicle

The unmanned aerial vehicle module refers to the aircraft, including both on-board software and hardware.

On-board Control system

The on-board control system aims to control the vehicle when disconnection with the Cloud occurs or the on-Cloud control system is not available.

Generally, the on-board control system uses control algorithms with low computation requirements, due to the limitation of on-board resources. Usually, this block is performed by an on-board professional Autopilot, such as PX4 [132] or Ardupilot [10] to name a few. The most popular control technique implemented by autopilots is the PID

(Proportional-Integrative-Derivative) controller, because of its simplicity and good performances. Moreover, in order to guarantee a safe control system, a Detect-and-Avoid system needs to be included on-board [9, 16].

As already discussed in the On-Cloud Control module, the on-board control requires real-time performances, in order to provide a fixed and reasonable controller frequency to the vehicle.

Hardware

The hardware block refers to all the hardware mounted on-board the aircraft. It includes actuators and sensors, as well as a generic payload.

2.3 Discussion

The proposed Cloud-based architecture defines a reference framework to implement safe and intelligent navigation and coordination for UASs.

In particular, the Cloud-based framework refers to connected UASs using the newest mobile technologies, such as 4G and 5G. The aircraft is connected with the ground with unprecedented opportunities. First at all the aircraft is not constrained to the position of the ground control station, enabling BVLOS flight operations. Moreover, the UAS has access to the Internet, exploiting Cloud technologies. This technological evolution allows autonomous flying to be implemented.

The proposed Cloud-based framework is distributed between the Cloud and aircraft. Most of the functionalities are provided by the Cloud. The Coordinator Manager manages the airspace, defining the operational area and assigning the priority to each vehicle. Moreover, it monitors all vehicles during flight operations. The Navigation Manager, one for each vehicle, plans and executes a safe flight mission. In particular, the combination of risk-based map and risk-aware path planning aims to compute a safe path, minimizing the risk to the population, avoiding obstacles, no-fly zones and high-risk areas. Hence, the on-Cloud Control System controls the aircraft with advanced control techniques, generally not implementable on-board. Moreover, the Data Processing module analyses and processes data from on-board sensors providing advanced additional functionalities, such as real-time object detection and fault detection. In fact, thanks to the Cloud, algorithms with a high computational burden can be executed.

Even if all navigation tasks are provided by the Cloud, an on-board Control System is essential. In fact, if a disconnection with the Cloud occurs, the aircraft uses the on-board controller to maintain the vehicle stable, avoiding unexpected obstacles and to execute emergency operations. For this reason, both on-board the aircraft and on-Cloud, a Connection Diagnostic module is present to monitor the quality of the connection between the UAS and the Cloud.

The proposed Cloud-based architecture aims to define and implement an innovative

flight operation with an autonomous connected aircraft operating in BVLOS. In particular, this thesis focuses on the risk-based map and the risk-aware path planning strategy, i.e. it implements and proposes solutions to the Map Manager and part of the Navigation Manager. Future works will include the implementation of other modules, such as the Coordinator manager and the On-Cloud Control. However, the implementation of the whole Cloud-based architecture is a challenge because requires that all modules work interacting each other and managing a fleet of UASs in the same airspace. Moreover, the use of Cloud technologies increases the implementation complexity, even if they offer unprecedented opportunities to UASs.

The proposed architecture is the reference framework for the first part of this thesis, about the safe navigation of UASs.

Chapter 3

Risk-based map for UAS in urban environments

This Chapter presents the risk-based map introduced in Chapter 1.

The definition of a reference map is one of the most fundamental steps in order to implement autonomous navigation [113]. The map describes the navigation environment and it is usually used to localize the robot and determine the main route of the vehicle. In fact, generally, the map defines the search space used to solve a path planning problem.

The proposed risk-based map is defined inspiring to the work presented in [158], where a ground risk map is introduced to quantify the risk to the population on the ground when an Unmanned Aerial System (UAS) flies over populated areas. In fact, when we talk about unmanned aircraft, safety to people on the ground is the most important problem, because a crash of the aircraft may cause casualties. The proposed risk-based map takes into account several factors, such as the risk to the population, no-fly zones, obstacles and the coverage of the communication system.

The Chapter is organized as follows. In Section 3.1 background information are reported. Section 3.2 describes the risk-based map generation, while in Section 3.3 the probabilistic risk assessment approach used in this work is explained. Results are presented in Section 3.4 and discussed in Section 3.5.

3.1 Background

The large diversity of Unmanned Aerial Systems (UASs) in our society is growing. Thanks to their flexibility and low-cost UASs are used in a wide range of applications, such as package delivery, monitoring, search and rescue, surveillance, to name a few. Their technology is growing and, as a consequence, UASs are more and more intelligent, increasing their capabilities [134].

While UAS technology is developing rapidly, the safety associated with flight operations is not growing equally fast. As a consequence, National Aviation Authorities (NAAs), such as ENAC (Ente Nazionale per l'Aviazione Civile) in Italy, and FAA (Federal Aviation Administration) in the United States, have defined regulations for UAS operations. At present, the flight authorization from NAAs is required to perform flight operations, obtained if a certain level of safety is guaranteed. A risk-based approach is commonly used in aviation and its importance is recognized by EASA (European Aviation Safety Agency) and FAA. For this reason, it is used for the development of a regulatory framework for UASs [53].

EASA is working on a proposal for UAS regulations in the European Airspace [55, 56]. They use the Joint Authorities for Rulemaking on Unmanned Systems (JARUS) proposal, in which three risk-based categories are established: open (low risk), specific (medium risk) and certified (higher risk).

Urban areas are a critical and special scenario, because of the presence of people on the ground. For this reason, generally, NAAs strongly restrict flight operations in these areas. As a consequence, for full exploitation of unmanned aircraft in urban areas, a revision of regulation is mandatory.

Anyway, in order to perform flight operations in cities, it is necessary to assess the risk to the population on the ground. A realistic and detailed risk assessment is one of the major challenges because there are a lot of factors that determine the risk of a UAS mission. In literature, there are several works about risk assessment for unmanned aircraft [207]. Most of them are inspired by manned aviation, where risk assessment approaches are used for decades [59].

In [5], a real-time risk assessment framework is proposed with the intention to provide a real-time safety evaluation during the flight operation. In [208] a Bayesian approach is proposed to perform a risk assessment in order to certificate an UAS according to regulatory requirements. In [31], Barrier Bow Tie Model is used to consolidate existing risk models and for decision-making processes. A dynamic probabilistic risk assessment is presented in [86] to estimate risk, in order to develop a flight control for ground risk mitigation.

In [39], a complete risk assessment and functional requirements for UAS operations are presented, with the aim to define a suitable equivalent level of safety to be achieved by UASs. This approach uses a probabilistic risk assessment method, widely used in literature, such as in [32, 76, 133], where the risk is defined as the probability to have a casualty per flight hours. The method is enhanced in [35], where the probabilistic impact area is estimated evaluating different descent event types, as well as the wind. The probabilistic risk assessment approach is also used to quantify the risk of a particular flight mission [15].

The use of risk maps is a typical approach in risk assessment procedures. Generally, risk maps help to quantify and visualize the risk of a particular event or phenomena. It is a common tool used in several research areas, such as in medicine to estimate areas affected by epidemics [205], to estimate the risk of accidents of nuclear power plants [6],

or used by companies to quantify the risk associated with their business [45].

Anyway, risk maps are not commonly used to quantify the risk associated with UAS operations. In [41], a risk map is used to define the risk of collision with the ground by using the ground orography. In [74], a risk map defines the risk to the population on the ground, in order to define a minimum risk path. Anyway, the above-mentioned maps don't quantify the risk of an entire urban area. Moreover, in [74], the risk is computed considering the method described in [75, 76], where the impact area is computed considering only the dimension of the vehicle.

In this Chapter, we define a new concept of a risk-based map, a tool for risk-informed decision making. The risk-based map quantifies the risk to the population on the ground when the UAS flies over an urban area. The risk is computed considering four descent event types: ballistic descent, uncontrolled glide, parachute descent and the fly-away event. Moreover, it considers UAS parameters and environmental characteristics. The risk factors are also combined with other information, such as no-fly zones, obstacles at the flight altitude and the coverage of the communication system, obtaining the risk-based map.

The risk is determined with a detailed risk analysis. It is based on a probabilistic risk assessment approach commonly used in literature [39, 32, 35]. Moreover, inspiring to [35, 33] different descent event types are taken into account to estimate a probabilistic impact area, as well as using drone specifications and uncertainty on the model.

The proposed risk-based map is designed to be used in the Cloud-based framework introduced in Chapter 1. It is an essential element of our framework because the risk-based map defines the area where the flight is allowed or not, because of high risk, no-fly zones or obstacles. Moreover, the coverage information helps to identify areas where the mobile network is not available or the quality of the signal is not suitable to accomplish the UAS mission.

Moreover, the risk-based map can be used to quantify the risk over an entire urban area. Operators can use the map to plan a safe flight mission. On the contrary, it can be used by NAAs to quantify the risk of a particular flight mission in order to provide permission to fly.

The proposed risk-based map is introduced for the first time in [161], as a component of a Cloud-based architecture for risk-aware intelligent flying for UAS. A preliminary version of the risk-based map is used in [157, 156, 162] in order to define the minimum risk path.

3.2 Risk-based map generation

The risk-based map is a cell-based map, in which each cell has a specific risk value. The risk value is defined as the hourly probability to cause a casualty and it is computed considering drone specifications, environment characteristics, different descent events and the wind effect. In this section, the risk-based map generation is explained in detail.

3.2.1 Risk-based map

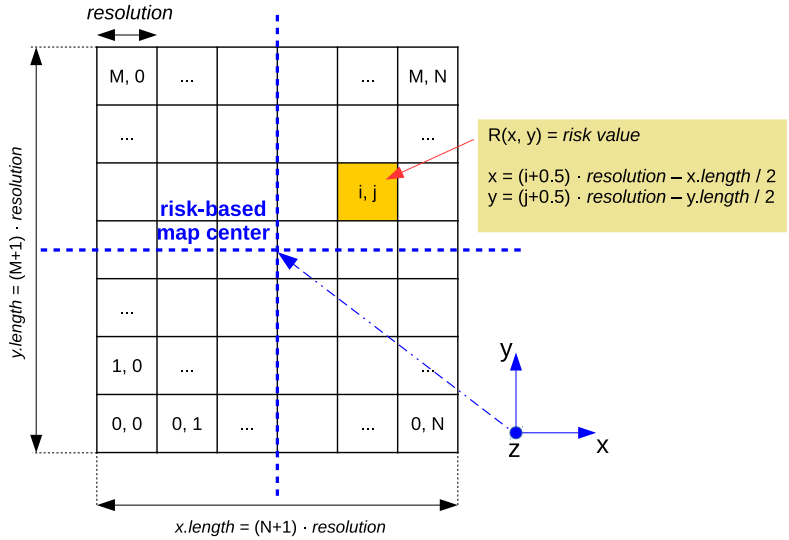


Figure 3.1: Representation of the risk-based map and the associated notation.

The risk-based map is a two-dimensional location-based map, where each element, called cell, represents a specific location and has a risk value. Cells are square and equidistantly distributed on the map. Hence, the risk-based map is represented as a matrix of dimension $N \times M$ cells, denoted as $\mathbf{R}$. The cell $\mathbf{R}(i, j)$ of the map at the discrete coordinates (i, j) represents a georeferenced location (x, y) defined in the Local NED (North East Down) coordinate system defined in the reference frame placed in the center of the map. With a slight of abuse, in this paper, we use the notation $R(x, y)$ to refer an element $R(i, j)$ that corresponds to a cell centered in the location (x, y). The conversion between the discrete coordinates (i, j) and the position (x, y) is not discussed in this work, even if it is implemented in the risk-based map generation. Figure 3.1 shows the risk-based map representation.

The risk-based map is computed by combining several layers. By definition, each layer is a map with the same characteristics of the risk-based map, according to the map representation of Figure 3.1. Here, the proposed multilayer framework consists of the following layers:

- **Population Density layer**: determines the population density distribution in the area;
- **Obstacles layer**: determines the height of buildings and generic obstacles in the area;
- **Sheltering factor layer**: defines the sheltering factor in the area;

- **No-Fly Zones layer**: identifies areas where the flight is not allowed;
- **Coverage layer**: defines the quality of the signal of the communication system in the area.

These layers are combined according to the architecture of Figure 3.2. The Population Density and the Sheltering Factor layers are used by the risk assessment, as well as the drone and mission specifications and environmental characteristics. The risk assessment procedure computes 4 maps, called *event risk maps*, one for each descent event type. Hence, the Merging layer procedure computes the resulting risk-based map by merging the event risk maps with No-fly Zones, Obstacles and Coverage layers.

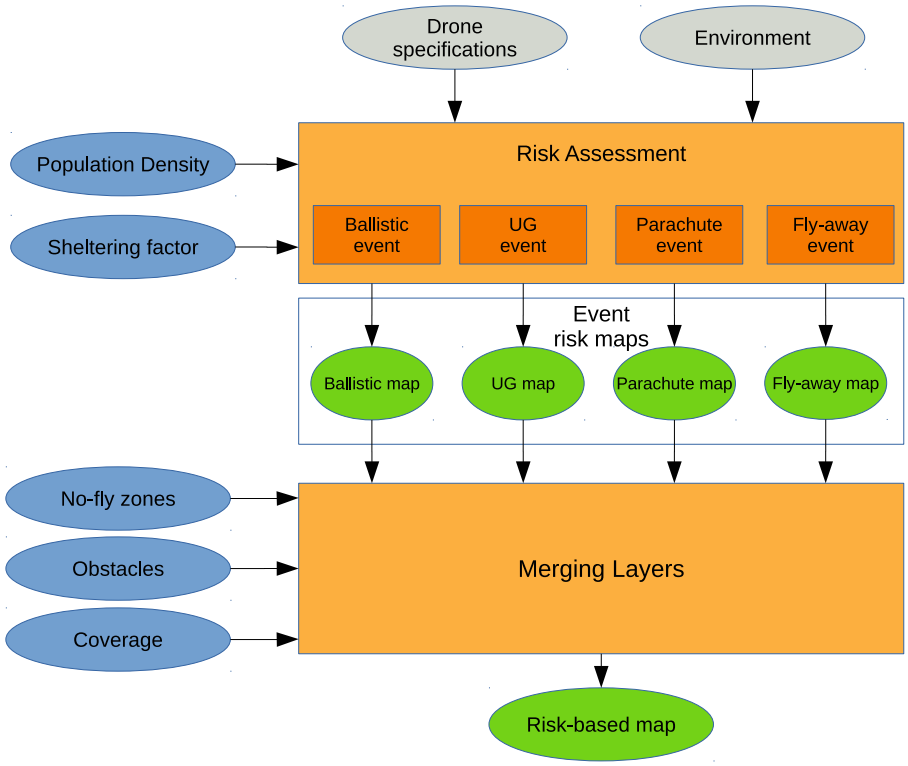


Figure 3.2: The architecture used to generate the risk-based map.

3.2.2 Population Density Layer

The Population Density layer determines the population density distribution in the area related to the risk-based map. The population density is an essential element in the risk assessment procedure because the probability to cause a casualty is proportional to the number of people that can be involved in the crash. Hence, the population density is used to compute the probability of impact with a person, after the uncontrolled crash

of the UAS on the ground. As a consequence, it is important to consider an accurate population density with a reasonable fine resolution.

Generally, the population density distribution is collected by National statistical institutes and by municipalities, by retrieving information from census and cadastral data. Then, the resulting population density distribution is defined considering where people are living. Anyway, during the day, people are not at home but they are at work, school and other places.

The population density of urban areas can be estimated using the mobile phone data [94], a promising method to trace the dynamic distribution of people [44]. Hence, given a realistic estimation of the distribution of the population, a more accurate risk assessment procedure is provided.

As already mentioned, the Population Density layer has the same characteristics of the risk-based map, i.e. it is represented as a matrix $\mathbf{D}$, where each element $\mathbf{D}(x, y)$ is the population density at the location (x, y) expressed in people/m^2.

3.2.3 Obstacles layer

The Obstacles layer determines the height of obstacles in the area interested by the risk-based map, such as buildings and other generic obstacles. The Obstacles layer can be defined considering a tri-dimensional model of the city, often provided by municipalities.

A model of an urban area can be obtained by mapping procedure [143]. In [115], automatic reconstruction of urban areas is presented, using aerial images captured by an on-board camera on an unmanned aircraft.

According to the architecture of Figure 3.2, the Obstacles layer is not considered by the risk assessment, but it is used to determine no-flyable areas because of obstacles at the flight altitude. Hence, the probability of collision with obstacles is not taken into account.

The Obstacles layer is represented as a matrix $\mathbf{O}$ with the same characteristics of the risk-based map. Each cell $\mathbf{O}(x, y)$ has a value defined according to the maximum height of obstacles in the relative area, centered in a location (x, y).

3.2.4 Sheltering Factor layer

The Sheltering Factor layer determines the sheltering factor in the area related to the risk-based map. By definition [39], a sheltering factor is a positive number that defines how people are sheltered by buildings or other obstacles.

It is essential to consider the sheltering factor in the risk assessment because the presence of obstacles in the crash area reduces the probability of casualties. In particular, in urban areas, people are sheltered by buildings, trees, cars and other objects. Hence, ignoring the sheltering factor implies an over-conservative risk assessment.

In literature, the sheltering factor is widely used in the risk assessment procedure, but it is evaluated in different ways. In [35], the sheltering factor is included in the computation of the probability of impact with a person. The sheltering factor is defined in the range from 0 to 1 and reduces the probability of impact a person proportionally. On the contrary, in [39, 15], it is used to determine the probability to cause a fatality, after the impact with a person. In fact, the presence of obstacles in the crash area reduces the kinetic energy at impact, as subsequently, the probability of fatalities.

In this work, the sheltering factor is used to compute the probability of fatality. It is defined as inspiring to [39], where it is an absolute number in the range from 0 to infinite. Anyway, as reported in [75], the sheltering factor can be defined in the range from 0 to 10, with 0 corresponds to an area without shelter and 10 with the maximum one. In fact, with small and lightweight UASs, it is useless to consider high values of the sheltering factor, because the required kinetic energy to cause a fatality is unreachable. For instance, accordingly with Equation (3.12), with a sheltering factor of 10 and a kinetic energy of 250 J (aircraft with a mass of 10 kg and impact velocity of 7 m/s), the probability to cause fatal injuries to a person after the impact is 0.025%. The classification of the sheltering factor is reported in Table 3.1.

Similarly to other layers, the Sheltering Factor layer is defined as a matrix $\mathbf{S}$, where each cell $\mathbf{S}(x, y)$ determines the sheltering factor in a location (x, y).

Table 3.1: Sheltering factor classification

Sheltering	Area
0	No obstacles
2.5	Sparse trees
5	Vehicles and low buildings
7.5	High buildings
10	Industrial buildings

3.2.5 No-fly zone layer

The No-fly Zones layer identifies areas where the flight is not allowed, generally called no-fly zones. No-fly zones are defined by:

- National Aviation Authorities (NAAs), such as ENAC in Italy and FAA in the United States. No-fly zones are determined for safety reasons. Typical no-flight areas are airports, prisons and military areas.
- Security zones where the flight is not allowed because of generic safety reasons. For example, areas with public events.
- Nature sensitive areas, such as National parks, nature reserves or other protected areas.

- Other zones defined by the operator.

Generally, NAAs provide maps where no-fly zones are reported. Some examples are the D-Flight map [37] in Italy provided by ENAV (Ente Nazionale per l'Assistenza di Volo) in collaboration with ENAC, and the Naviair map [142] in Denmark designed by the Danish Transport and Construction Agency.

The No-fly Zones layer is defined as a two-dimensional matrix $\mathbf{F}$, where each element assumes only two values: 0 where the flight is allowed and -1 where the flight is forbidden. Hence, $\mathbf{F}(x, y)$ is defined as

$$\mathbf{F}(x, y) = \begin{cases} -1 & \text{if flight is not allowed,} \\ 0 & \text{if flight is allowed.} \end{cases} \tag{3.1}$$

3.2.6 Coverage layer

The Coverage layer quantifies the quality of the signal of the mobile network in the operational area. It is an essential layer in the proposed architecture of Chapter 2 because the aircraft continuously communicates with the Cloud. The quality of the mobile network signal is defined using the Reference Signal Received Power (RSRP), the Reference Signal Received Quality (RSRQ) and the Signal to Noise Ratio (SNR). As already discussed in Section 2.1.1, the coverage of the mobile network depends on the flight altitude, because, generally, antennas provide the best coverage to people at the ground level. For instance, in the airspace, the signal may be stronger, because the UAS is in Line-of-Sight with antennas. On the contrary, the SNR may be worse, because of the interference between antennas [166, 120]. The coverage map is provided by the telecommunication company that offers a mobile network. Moreover, it can be updated by data collected by UAS in the operational area. In fact, if an area with no signal is detected by a previous mission, the coverage map can be updated with the latest information.

Similarly to other layers, the Coverage map is defined as a two-dimensional matrix $\mathbf{C}$, in which each element $\mathbf{C}(x, y)$ has a value in the range from 0 to 10 according to the coverage level in the low airspace. A value of 0 is defined as locations with no connection, while a value of 10 to locations with the best quality of the signal.

3.3 Risk assessment strategy

The risk assessment is the main procedure of the risk-based map generation because it determines risk values. The risk is defined as the risk to the population on the ground when a UAS flies over a specific area. It is expressed in fatalities *per* flight hours, a typical measurement system used in aviation to assess the risk of a flight operation.

The risk assessment used in this work is based on a common probabilistic risk assessment approach, widely used in literature [39, 76, 32, 35]. The risk is computed as an

hourly probability to have a casualty P_{casualty}, defined as a sequence of three conditional events:

- the loss of control of the vehicle with the uncontrolled crash on the ground;
- the impact with at least one person after the crash on the ground;
- the impact causes fatal injuries to the hit person.

Thus, P_{casualty} is defined as:

$$P_{\text{casualty}}(x, y) = P_{\text{event}} \cdot P_{\text{impact}}(x, y) \cdot P_{\text{fatality}}(x, y), \tag{3.2}$$

with P_{event} the probability that the UAS loses the control of the vehicle with the uncontrolled descent and impact on the ground, assuming independence between failures. P_{impact} is the probability to impact with a person and it depends on the population density and the area exposed to the crash. P_{fatality} is the probability that a hit person suffers fatal injuries and it depends on the kinetic energy at impact and the sheltering factor.

The trigger event of the risk assessment is the uncontrolled descent on the ground of the aircraft. Anyway, the behavior of a vehicle during the descent depends on the failure type. In this work, we assume four descent event types: the ballistic descent, the uncontrolled glide, the parachute descent and the fly-away. Each event has a different value of P_{event}. Based on the event type, the aircraft acts differently, changing the impact area on the ground. For each descent event, a mathematical model is used to compute the ground impact area, assuming the initial condition when the descent begins and drone specifications. The impact area is defined as a two-dimensional probability density function (two-dimensional PDF), estimating the probabilistic impact area in respect to the the UAS position when a descent event starts.

The two-dimensional PDF is represented as a georeferenced matrix relative to UAS position when a descent event begins. Each element of the matrix represents a geolocation $(x_{\text{I}}, y_{\text{I}})$ with the probability that the aircraft impacts inside the represented area.

Since the impact area changes with the descent type, the probability to have a casualty is computed for each descent event. Moreover, the risk is computed for each element of the map centered in the location (x, y). The computation of the risk is executed, for each event, according to the following procedure:

1. Given the mathematical model of the descent event, initial conditions and drone parameters, a two-dimensional PDF is computed. The two-dimensional PDF identifies the probabilistic impact area of the aircraft on the ground relative to the UAS position when a descent event begins, i.e. in a location (x, y).

2. Considering the probabilistic wind, the two-dimensional PDF is modified according to the wind speed and direction.

3. The two-dimensional PDF is used to compute the probabilities P_{impact} and P_{fatality}, by combining the probabilistic impact area with the Population Density layer, Sheltering Factor layer, as well as the estimated impact velocities. Since the two-dimensional PDF is relative to the location (x, y), a generic element of the two-dimensional PDF in $(x_{\text{I}}, y_{\text{I}})$ is combined with elements of layers in $(x + x_{\text{I}}, y + y_{\text{I}})$.

4. The probability P_{casualty} of the element in (x, y) is computed.

Figure 3.3 clarifies how a two-dimensional PDF is combined with layers.

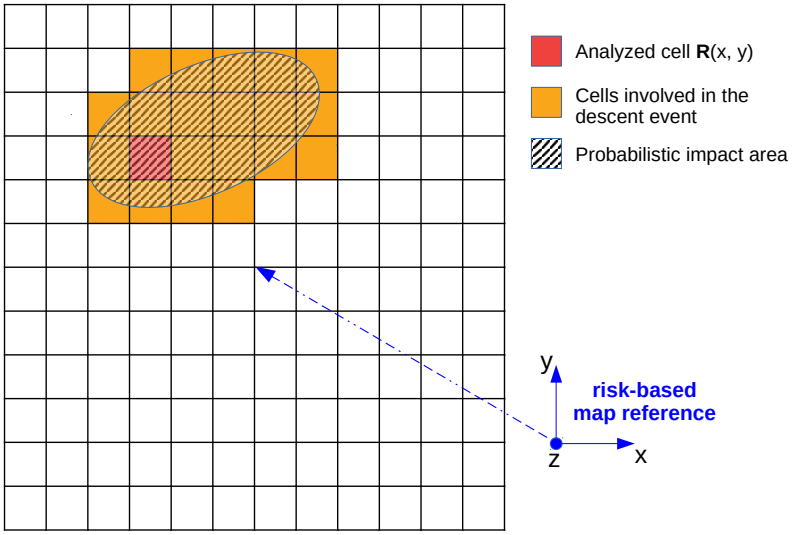


Figure 3.3: Graphical representation of the combination between a probabilistic impact area in the risk-based map. Considering a generic location in (x, y), the risk is computed considering the probabilistic impact area related to the position where a descent event happens, i.e. in (x, y). The risk is computed using information about cells involved by the impact.

The risk is defined assuming a UAS flies over the location (x, y) considering initial conditions, such as flight altitude and velocity. Anyway, the exact velocity and flight direction of the UAS when it executes the flight operation is unknown. Flight directions are assumed to be distributed with uniform distribution. The velocity is chosen according to two scenarios. With the fixed wing configuration an initial horizontal velocity is assumed at the cruise speed with uncertainty modeled with a normal distribution $\text{N}(\mu, \sigma)$, where μ is the mean value and σ is the standard deviation. On the contrary, with multirotor configurations, we assume any velocity between 0 and the maximum one, distributed with a uniform distribution $\text{U}(a, b)$, where a and b define the range of the distribution. The zero velocity condition includes the hovering operational mode.

In the following, each descent event is detailed, as well as each step of the risk assessment procedure. For each descent event type, a two-dimensional PDF is computed

considering two different commercial aircraft: the Heliscope Talon and the DJI Phantom 4. The Heliscope Talon is a fixed wing aircraft, while the DJI Phantom 4 is a quadrotor aircraft. Parameters of both aircraft are reported in Table 3.2.

3.3.1 Ballistic descent event

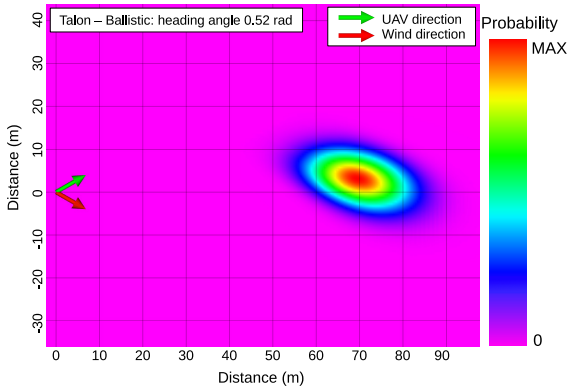


Figure 3.4: A two-dimensional PDF of the ballistic descent event with the Talon aircraft. The UAS flies at an altitude of $N(\mu = 50, \sigma = 5)$ m, with a heading angle of 0.52 rad. The wind has a direction of -0.52 rad and speed $N(10, 2)$ m/s. Parameters of Talon aircraft are reported in Table 3.2.

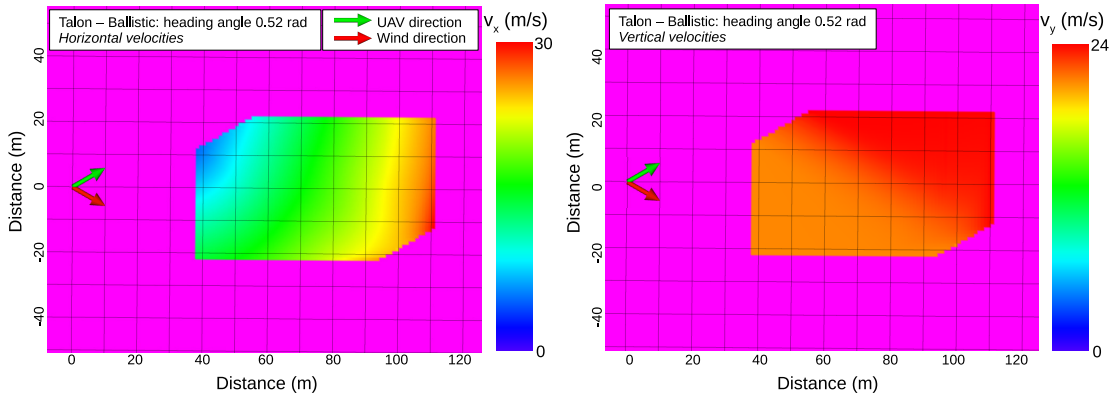


Figure 3.5: Left panel: horizontal velocities of the ballistic descent event with the Talon aircraft. Right panel: vertical velocities. Velocities are computed in the area interested by the ballistic descent event, while velocities are not computed in areas in magenta. Horizontal velocities are in the range between 8.32 m/s and 29.62 m/s, while the vertical velocities are in the range between 21.25 m/s and 23.96 m/s. These examples use the same parameters defined in Figure 3.4.

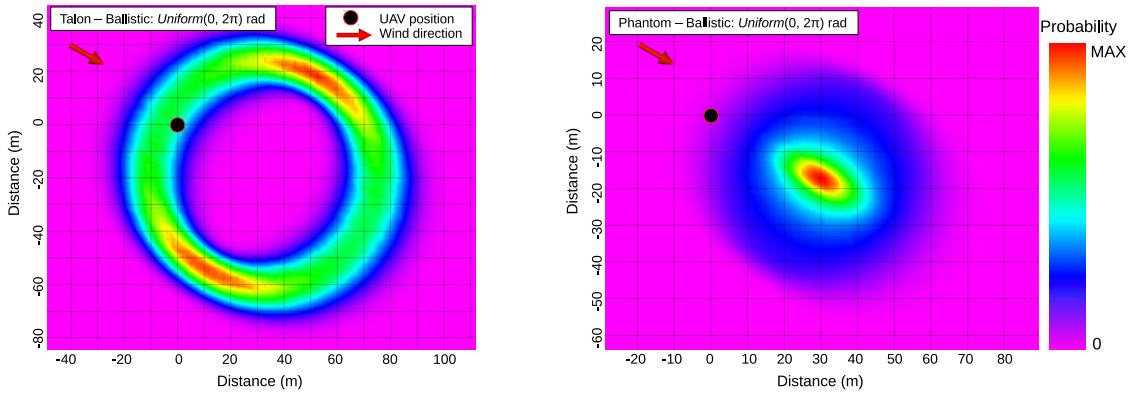


Figure 3.6: Left panel: the two-dimensional PDF of the ballistic descent with the Talon aircraft considering direction $\mathrm{U}(0, 2\pi)$ rad. Right panel: the two-dimensional PDF with the Phantom aircraft considering direction $\mathrm{U}(0, 2\pi)$ rad. In these examples an UAS flies at an altitude of $N(50, 5)$ m. The wind has a direction of -0.52 rad and speed $N(10, 2)$ m/s.

The ballistic descent event happens when an UAS loses most of its lift. Hence, the vehicle descends with a ballistic trajectory, where the vehicle is affected only by gravity and drag force. The mathematical model of the ballistic descent used in this work is inspired by [33], based on the standard second order drag model

$$m\dot{\boldsymbol{v}} = mg - c|\boldsymbol{v}|\boldsymbol{v}, \tag{3.3}$$

with m is the mass of the vehicle, g is the gravitational acceleration, v is the velocity vector of the aircraft and c is a constant that takes into account the drag coefficient, drag area and air density. The drag model is considered in two dimensions, defined in the vertical plane, spanned by the horizontal velocity and the gravity vectors. Assuming $v_{\mathrm{x}} > 0$ and monotonically decreasing, the drag model can be approximated to

$$m\dot{v_{\mathrm{x}}} = -c \max(v_{\mathrm{x}}, v_{\mathrm{y}})v_{\mathrm{x}} \tag{3.4}$$

$$m\dot{v_{\mathrm{y}}} = mg - c|v_{\mathrm{y}}|v_{\mathrm{y}} \tag{3.5}$$

According to [33], the ballistic trajectory is a second order polynomial, considering the aerodynamic properties of the vehicle, as well as probabilistic uncertainties on horizontal and vertical initial velocities and on drag coefficients. Because of our purpose, the model presented in [33] is extended considering also uncertainties on flight altitude and flight direction. The flight altitude is modeled as a normal distribution, in which the mean value is the operational altitude defined by the mission. Since the risk-based map assumes any possible direction, it is modeled as a uniform distribution considering any direction. As a consequence, the resulting probabilistic impact area has a bigger distribution than the original method proposed in [33] due to uncertainties on the flight altitude and direction.

As already discussed, the probabilistic impact area is defined as a two-dimensional PDF estimated with the ballistic descent model. In this work, we use the same model to estimate two bi-dimensional matrices with the estimated vertical and horizontal velocities at impact. Impact velocities are useful to estimate the impact angle and the kinetic energy at impact, used to compute the probabilities P_{impact} and P_{fatality}.

Initially. a two-dimensional PDF is computed in the wind frame. Hence, it is modified by the effect of the wind that changes the drop times of descent. The wind effect an a two-dimensional PDF is detailed in Section 3.3.5.

Figure 3.4 illustrates an example of a two-dimensional PDF of the ballistic descent, considering a specific flight direction and initial velocities. With the same initial conditions, Figure 3.5 reports horizontal and vertical velocities at impact. Finally, Figure 3.6 reports a two-dimensional PDF considering any flight direction with both fixed wing and multirotor configurations.

3.3.2 Uncontrolled glide event

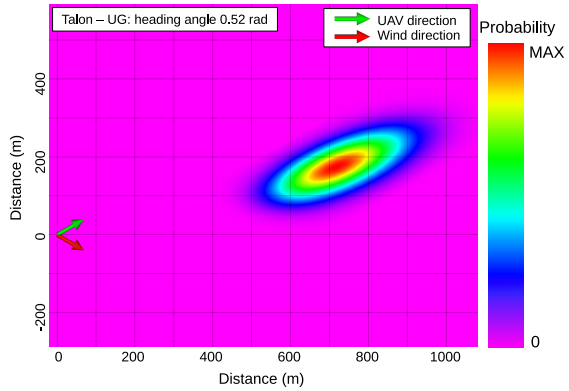


Figure 3.7: The two-dimensional PDF of the uncontrolled glide event with the Talon aircraft. This example uses the same parameters defined in Figure 3.4.

The uncontrolled glide event occurs when the aircraft starts a descent governed by a glide ratio or autorotation descent angle.

With a fixed wing configuration, the event occurs when there is a loss of thrust or power for the flight control surfaces.

With a rotorcraft, this event occurs when there is a loss of thrust on the main rotor. Hence, the vehicle begins an autopiloted autorotation descent. Similar behavior with the multirotor configurations, where a loss of thrust causes an autopiloted descent.

The mathematical model of the uncontrolled glide descent is

$$\text{dist}(h) = \gamma h, \tag{3.6}$$

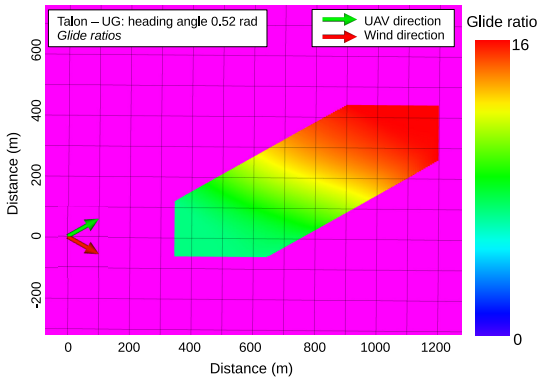


Figure 3.8: The glide ratio distribution with the uncontrolled glide event with the Talon aircraft. The glide ratio is in the range between 8 and 16. In magenta, areas where the glide ratio is not computed. This example uses the same parameters defined in Figure 3.4.

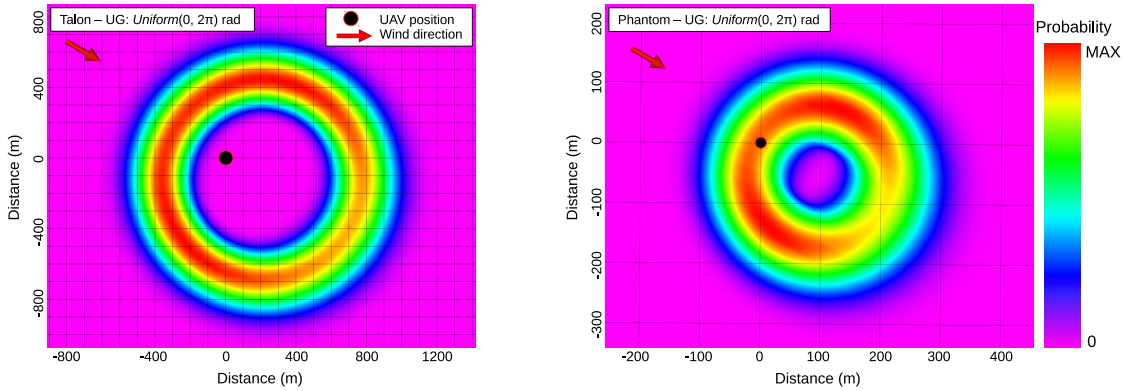


Figure 3.9: Left panel: the two-dimensional PDF of the uncontrolled glide event with the Talon aircraft. Right panel: the two-dimensional PDF with the Phantom aircraft. These examples use the same parameters defined in Figure 3.6.

with dist(h) is the horizontal traveled distance, h is the flight altitude and γ is the glide ratio. By definition, the glide ratio is the ratio between the horizontal and the vertical traveled distances.

Similarly to the uncontrolled glide model used in [35], a two-dimensional PDF is computed with probabilistic assumptions on initial velocities and the glide ratio. Moreover, in this work, we extend the model proposed in [35] by considering uncertainties on the flight altitude and flight direction.

The same model is also used to compute a two-dimensional matrix with stored the dynamics of the estimated glide ratio. This matrix is used to estimate the impact angle of the vehicle on the ground. The impact angle is useful in the computation of the probability P_{impact}.

Similarly to the ballistic descent, the wind effect modifies a two-dimensional PDF

as described in Section 3.3.5.

An example of two-dimensional PDF is illustrated in Figure 3.7, while the associated matrix with glide angles is reported in Figure 3.8. Figure 3.9 shows the two-dimensional PDF considering all directions with both the fixed wing and multirotor configurations.

3.3.3 Parachute descent event

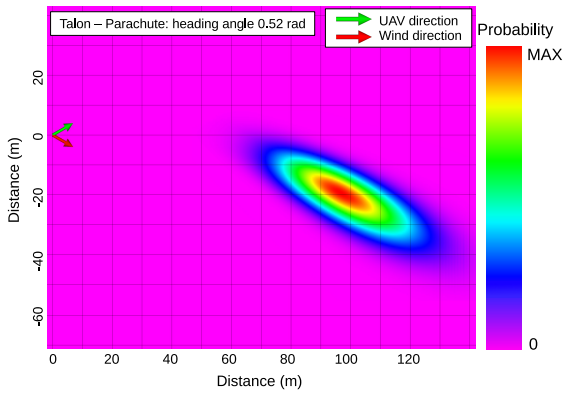


Figure 3.10: The two-dimensional PDF of the parachute descent with the Talon aircraft. This example uses the same parameters defined in Figure 3.4.

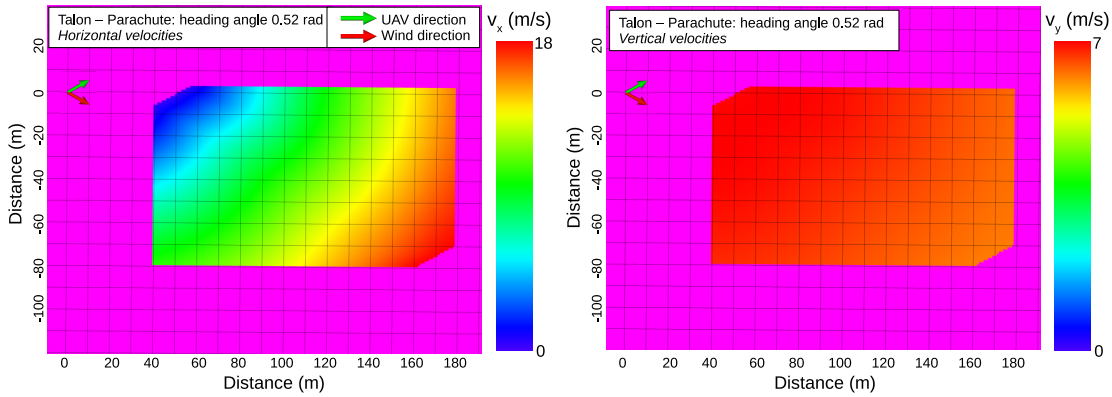


Figure 3.11: Left panel: horizontal velocities of the Talon aircraft with the parachute descent. Right panel: vertical velocities. Horizontal velocities are in the range between 2.74 m/s and 17.65 m/s, while vertical velocities are in the range between 6.15 m/s and 6.82 m/s. In magenta, areas where the impact velocities are not computed. These examples use the same parameters defined in Figure 3.4.

The parachute descent occurs when the UAS starts a descent with a fully deployed parachute. Often, after a failure is detected, motors are turned off and the parachute is deployed. Generally, a short time delay elapses between the failure detection and the

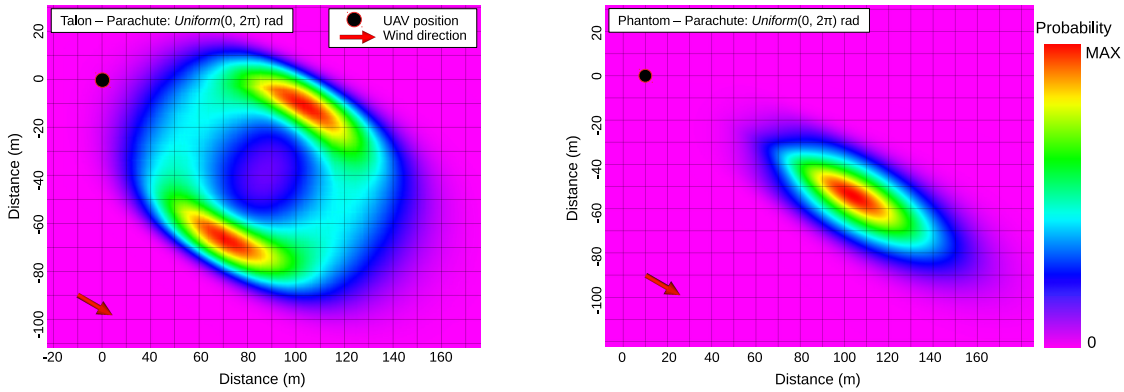


Figure 3.12: Left panel: the two-dimensional PDF of the parachute descent event with the Talon aircraft. Right panel: the two-dimensional PDF with the Phantom aircraft. These examples use the same parameters defined in Figure 3.6.

fully deploy of the parachute. Thus, the descent of the aircraft is governed only by the aerodynamic properties of the parachute.

The mathematical model of the parachute descent used in this work is presented in [35], considering the mass of the vehicle and the physical characteristics of the parachute, such as the parachute area and the parachute drag coefficient. As with previous descent types, in this work, the method proposed in [35] is extended by considering also probabilistic assumptions on the flight altitude and directions. The model is used to compute the probabilistic impact area represented as a two-dimensional PDF. Moreover, the mathematical model of the parachute descent is also used to estimate the vertical and horizontal impact velocities that are stored in two-dimensional matrices. These matrices are useful to compute the probabilities P_{impact} and P_{fatality}.

The parachute reduces drastically the vertical velocity of the vehicle during the descent. As a consequence, the kinetic energy at impact is lower, as well as the probability to cause a fatality. Anyway, the effect of the wind changes both velocity and direction during the descent. In particular, with a strong wind, the resulting horizontal velocity can cause significant kinetic energy at impact. Section 3.3.5 explains how the wind modifies the two-dimensional PDF.

Some examples of the parachute descent are reported in Figures 3.10 and 3.12, as well as the two-dimensional matrices of both horizontal and vertical impact velocities in Figure 3.11.

3.3.4 Fly-away event

The fly-away event occurs when an operator completely loses the control authority of the vehicle. As a consequence, the autopilot maintains the aircraft stable and the UAS may flight in any direction as long as the autonomy of the vehicle is exhausted or until the aircraft crashes on the ground.

The mathematical model of the fly-away event is proposed in [35], where the model is composed of two contributions:

- the probability of impact on the ground decreases linearly with the traveled distance. Hence, the maximum probability is in the initial UAS position.
- the probability of impact is modeled with a normal distribution for distance from the initial UAS position. This contribution assumes the vehicle moves vertically, such as with a *helicopter climb* or spiraling up with a fixed wing aircraft.

The two contributions are linearly combined. In this work, we consider the two scenarios happen with the same probability.

Since it is impossible to estimate how the aircraft terminates the flight, we assume a flight termination with an uncontrolled glide descent.

Unlike other descent event types, the two-dimensional PDF of the fly-away event is not computed in the wind frame, but directly in the local NED frame. More details are explained in Section 3.3.5.

Some examples of the fly-away event are shown in Figure 3.13.

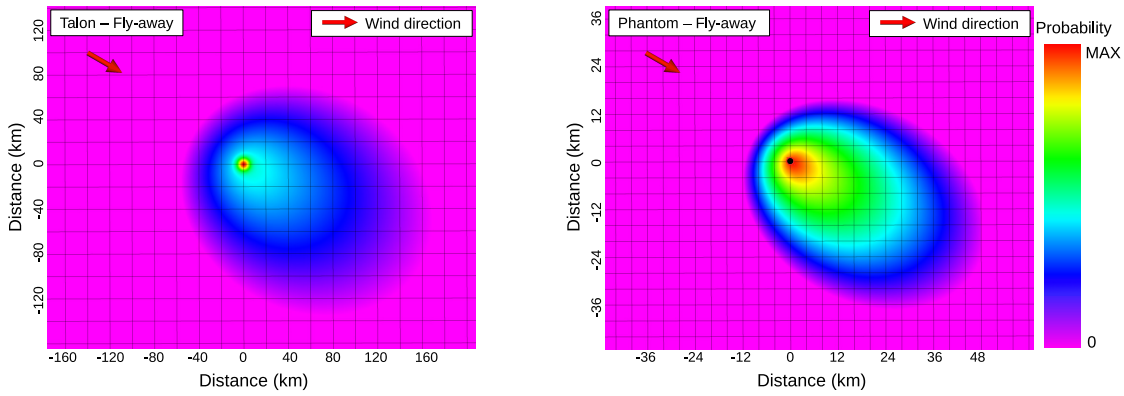


Figure 3.13: Examples of two-dimensional PDF with the fly-away event. Left panel: an example with the Talon aircraft. Right panel: an example with the Phantom aircraft. These examples use the same parameters defined in Figure 3.4.

3.3.5 Wind effect

The effect of the wind during the descent of the aircraft is considerable because it changes the probabilistic impact area.

The wind characteristics, such as wind speed and direction, can be defined using weather reports. Hence, the wind can be defined according to the time and the day scheduled for the flight mission. The wind effect is modeled using the method proposed in [35], where both wind speed and direction are modeled with a normal distribution.

The probabilistic impact areas of the ballistic descent, uncontrolled glide and the fly-away events are computed in the wind frame. These models define a series of drop times, i.e. the time required by the vehicle to reach the ground level, considering the initial conditions. For each drop time, defined as $\{t_k\}_{k=0,\dots,n}$, a georeferenced PDF is determined as a matrix $\mathbf{M}_k$. Each element of $\mathbf{M}_k$ is a geolocation with the probability of the aircraft to impact inside the represented area, assuming a purely vertical drop in the wind frame.

The wind introduces an offset only on drop times because all other parameters are already considered by the descent model. As a consequence, an offset is applied to the georeferenced ground grid of each PDF with a horizontal translation defined as the wind velocity times the drop time.

The probability of each drop time is defined as a set of probabilities $\{p_k\}_{k,\dots,n}$ computed with probabilistic assumptions on initial conditions. Hence, the final two-dimensional PDF is computed with a weighted sum of the PDF matrices as

$$\mathbf{M} = \sum_{k=0}^{n} p_k \mathbf{M}_k, \tag{3.7}$$

where the resulting matrix $\mathbf{M}$ is also georeferenced, taking into account the georeference of each matrix $\mathbf{M}_k$.

Differently, the fly-away event is not computed in the wind reference frame, but directly in the local NED coordinate system with respect to the risk-based map reference frame. With the fly-away, the wind simply modifies the maximum range of the distance traveled by the vehicle.

The two-dimensional PDF of Figures from 3.4 to 3.13 are affected by the wind. The wind effect is visible especially with the ballistic and parachute descent events, where the two-dimensional PDF is clearly deformed by the wind.

3.3.6 Probability of impact a person P_{impact}

The probability P_{impact} is the probability that an UAS impacts a person after an uncontrolled crash on the ground. In literature, it is defined with a simple common approach [39], as

$$P_{\text{impact}}(x, y) = \rho(x, y) \cdot A_{\text{exp}}, \tag{3.8}$$

with $\rho(x, y)$ is the population density in the area represented by the location (x, y) and A_{exp} is the area exposed to the crash, also known as *lethal area*.

One of the most used approach to compute the area exposed to the crash is the method proposed in [189], where the area is computed considering the vehicle characteristics and the dimension of an average human

$$A_{\text{exp}}^{\text{old}}(\theta) = 2(r_{\text{p}} + r_{\text{uav}})\frac{h_{\text{p}}}{\tan(\theta)} + \pi(r_{\text{p}} + r_{\text{uav}})^2 \tag{3.9}$$

with θ is the impact angle. r_p and h_p are the radius and the height of the cylinder defined by the occupation by an average person. r_{uav} is the radius of the sphere determined by the occupation of the vehicle: with a multirotor aircraft is the half of the maximum diameter, while with a fixed wing configuration is equal to the semi-wing span. Anyway, this method was defined for commercial space launch and reentry mission and, with values of glide angle close to zero, the lethal area diverges. Basically, the area is defined as the shadow of the human on the ground according to the impact angle. Therefore, this approach is not suitable for small UASs.

In this work, we use a novel method that we proposed in [162]. Unlike the method of Equation (3.9), the lethal area is the projection of the man-cylinder on a plane normal to the velocity vector of the aircraft, because it considers that the UAS is offensive only for the first hit person. Hence, the proposed method is defined by the following Equation:

$$A_{exp}^{new}(\gamma) = \pi(r_p + r_{uav})^2 \sin(\theta) + 2(r_p + r_{uav})(h_p + r_{uav})\cos(\theta). \tag{3.10}$$

The only non-constant parameter of Equation (3.10) is the impact angle θ. Based on the descent event type, the impact angle assumes different values.

In this work the parameter θ is determined based on the descent event type. With the ballistic and parachute descent events, θ is estimated using the horizontal and vertical impact velocities. In fact, as mentioned in Sections 3.3.1 and 3.3.3, impact velocities are computed and stored in two-dimensional matrices.

With the uncontrolled glide and, as a consequence, with the fly-away event, the impact angle is determined using the glide ratio, also stored in a two-dimensional matrix.

The probability of impact with a person is computed considering the probabilistic impact area of each descent event, used to determine the expected values of the population density. Hence, based on Equation (3.8), we obtain the following formulation

$$P_{impact}(x, y) = \sum_{x,y} \text{PDF} \cdot \mathbf{D}(x, y) \cdot A_{exp}(\theta(x, y)), \tag{3.11}$$

with $\mathbf{D}$ the Population Density layer. $A_{exp}(\theta)$ and $\mathbf{D}$ are independent variables.

The expected value is computed by a combination between the two-dimensional PDF and the Population Density layer, both defined as two-dimensional matrices and according to the approach illustrated in Figure 3.3.

3.3.7 Probability of fatality $P_{fatality}$

The probability of fatality is the probability that the impact with a person causes fatal injuries. It is not easy to estimate it, because the impact may occur in several ways. It depends on which part of the human body is hit by the vehicle, as well as from which part of the aircraft. For instance, a spinning propeller may cause more serious consequences than other parts of a multirotor aircraft.

In some works, the probability of fatality is set equal to 1 [5, 32], i.e., any impact causes a fatality. Anyway, the resulting risk is over conservative, especially with small

UASs. A study about the fatality rate considering a wide variety of aircraft type is proposed in [34], in order to define the mass threshold to consider the aircraft as a "harmless" one.

The definition of this probability is a challenge and, in literature, several methods are proposed [11]. Widely used approaches are the Blunt Criterion (BC) [17, 30] and the Viscous Criterion (VC) [92], based on the energy absorbed during the impact. Anyway, they are not suitable with low impact velocities.

In this work, we use the method proposed in [39], where the probability of fatality is computed using the kinetic energy at impact and the sheltering factor. In particular, it is essential to consider the sheltering factor, because the presence of obstacles in the impact area reduces the kinetic energy at impact, and, as a consequence, the probability to cause fatal injuries. Hence

$$P_{\text{fatality}}(x,y) = \frac{1-k}{1-2k+\sqrt{\frac{\alpha}{\beta}}\left[\frac{\beta}{\mathsf{E}[E_{\text{imp}}(x,y)]}\right]^{\frac{3}{\mathsf{E}[\mathbf{S}(x,y)]}}}, \tag{3.12}$$

with $k = \min[1, (\frac{\beta}{\mathsf{E}[E_{\text{imp}}(x,y)]}^{\frac{3}{\mathsf{E}[\mathbf{S}(x,y)]}})]$. In Equation (3.12), the α parameter is the impact energy needed to obtain a fatality probability of 50% when the sheltering factor is equal to 6, and the β parameter is the impact energy to cause a fatality when the sheltering factor goes to zero. According to [169], the fatality threshold can be defined as $\beta = 34$ J. $\mathbf{S}$ is the Sheltering Factor layer and E_{imp} is the kinetic energy at impact.

Similarly to the computation of the probability P_{impact}, a two-dimensional PDF is used to compute expected values of the sheltering factor and kinetic energy, defined with the notation E. Hence

$$\mathsf{E}[\mathbf{S}(x,y)] = \sum_{x,y} \text{PDF} \cdot \mathbf{S}(x,y), \tag{3.13}$$

$$\mathsf{E}[E_{\text{imp}}(x,y)] = \sum_{x,y} \text{PDF} \cdot E_{\text{imp}}(x,y), \tag{3.14}$$

The expected value of the sheltering factor $\mathsf{E}[\mathbf{S}(x,y)]$ is computed using the probabilistic impact area and the associated values in the Sheltering Factor layer $\mathbf{S}$

Similarly, the expected value of the kinetic energy at impact is computed considering a two-dimensional PDF, where the kinetic energy associated to the area in the location (x, y) is computed using impact velocity, based on the descent event type

$$E_{\text{imp}}(x,y) = \frac{1}{2} m \cdot v_{\text{imp}}(x,y)^2, \tag{3.15}$$

with m is the mass of the vehicle and v_{imp} is the impact velocity. With the ballistic and parachute events, the impact velocity is computed using the horizontal end vertical velocities stored in two-dimensional matrices. With the uncontrolled glide and fly-away, the glide speed is used.

3.3.8 Merging layer procedure

The merging layer is the last procedure of the risk-based map generation. According to the architecture of Figure 3.2, after the risk assessment, the merging layer combines event risk maps with all other layers, i.e. the Obstacles, No-fly Zones and the Coverage layers.

Since all layers have the same dimension and geometric characteristics, this procedure performs an element wise combination. Hence:

$$\mathbf{R}(x,y) = \begin{cases} -1 & \text{if } \big(\mathbf{F}(x,y) = -1\big) \vee \big(\mathbf{O}(x,y) \geq h\big) \vee \big(\mathbf{C}(x,y) < c^*\big), \\ P_{\text{casualty}}(x,y) & \text{otherwise.} \end{cases} \tag{3.16}$$

Each element of the risk-based map $\mathbf{R}(x,y)$ has a value of -1 if the flight is forbidden because of three conditions:

- the element corresponds to an area in a no-fly zone.
- at least an obstacle at the flight altitude h is in the area.
- the quality of mobile communication in the area is lower than the required quality of signal c^*.

Both the thresholds h and c^* are defined by mission requirements. The risk associate to all other elements is defined as the sum of risk levels of each descent event type

$$P_{\text{casualty}}(x,y) = P^{\text{bal}}_{\text{casualty}}(x,y) + P^{\text{ug}}_{\text{casualty}}(x,y) + P^{\text{par}}_{\text{casualty}}(x,y) + P^{\text{fa}}_{\text{casualty}}(x,y) \tag{3.17}$$

The probabilities may be simply added because we assume the independence between failures and descent events.

After this procedure, a risk-based map is generated.

3.4 Results and discussion

3.4.1 Implementation

The risk-based map generation is implemented in C++ as an executable process in the Robot Operating System (ROS) [167], also called ROS node.

The risk-based map and all layers are generated using the Grid Map library [61], a C++ library ROS-compatible able to manage two-dimensional grid maps with multiple data layers. The mathematical model of each descent event and the associated two-dimensional PDF are computed using the OpenCV library [23], able to provide fast matrix computation. Thanks to the compatibility between the Grid Map library and OpenCV, a two-dimensional PDF, defined as an OpenCV matrix, is simply combined with Grid Map layers, also defined as matrices.

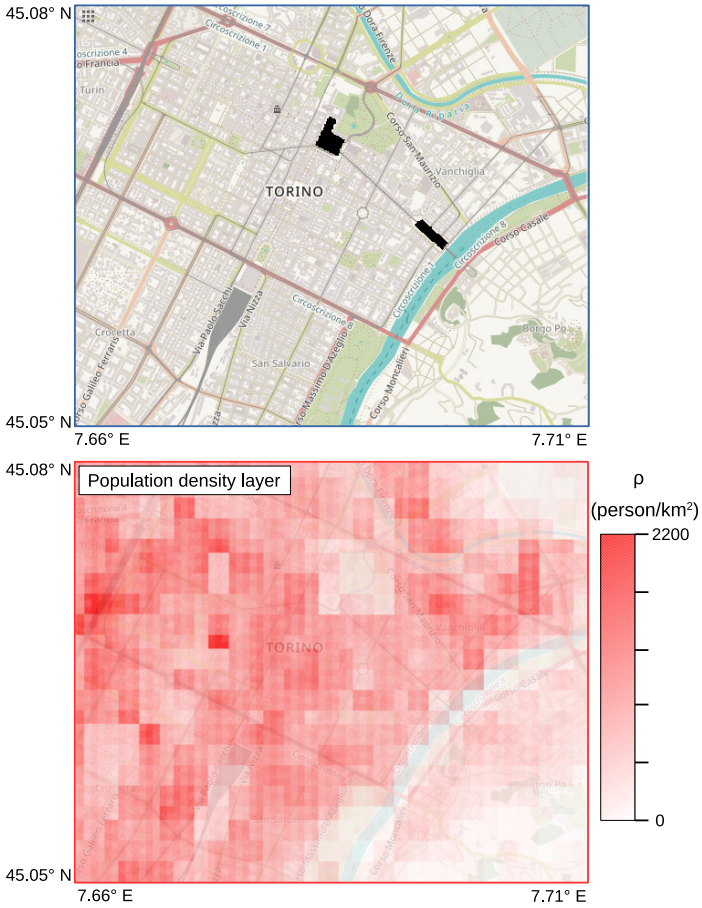


Figure 3.14: Top panel: the portion of the city of Turin from OpenStreetMap [148] used as example. The area comprises the city center and it has a dimension of about 4.05 × 3.52 km. Black areas are fictitious no-fly zones defined in correspondence of the two main city center squares. Bottom panel: the fictitious Population Density layer used as example with a resolution of 50 × 50 m.

The risk-based map computed in this Section considers a portion of the city of Turin (Italy) illustrated in Figure 3.14.

Turin is an interesting city to test our framework because of the high population density with an average value of 6939 people/km^2 [89]. In Figure 3.14 is illustrated a realistic (but not real) Population Density layer of the city used to generate the risk-based map.

The Obstacles layer is defined using the model of the city from OpenStreetMap (OSM) [148]. OSM is an open-source map of the world. Thanks to the OSM data, the tridimensional model of the city is extracted using the OSM2World tool [149]. Thus, the Obstacles layer is generated. Figure 3.15 reports the Obstacles layer used in this work.

The Sheltering Factor layer used in this work is determined using the Obstacles

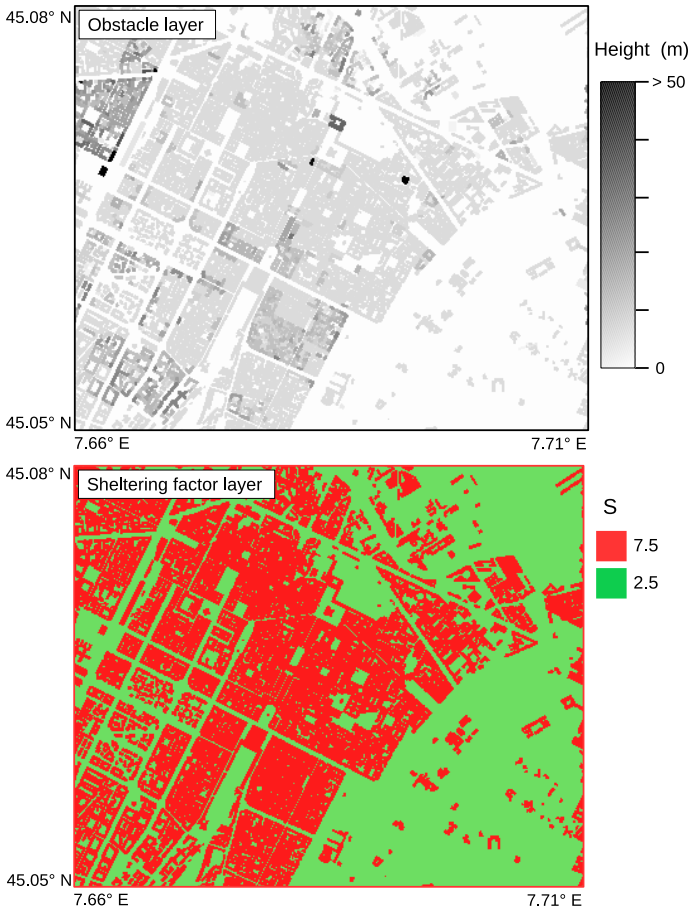


Figure 3.15: Top panel: the Obstacle layer of Turin city center with a resolution of 10 × 10 m. The data are obtained from OSM [148]. Bottom panel: the Sheltering Factor layer with the same resolution of 10 × 10 m.

layer. Each element is defined with a simple method: a sheltering factor equal to 7.5 is associated to areas occupied by buildings, while all other areas have a value of 2.5. In fact, if we have no information about the sheltering factor, an average value is defined. 2.5 is a reasonable value of sheltering factor because people are generally sheltered by cars, trees and other generic obstacles. The Sheltering Factor layer is shown in Figure 3.15.

According to the D-Flight map [37], in the tested area of Turin, there are no areas where the flight is forbidden. Anyway, we assume two fictitious no-fly zones over the two main city center squares. In Figure 3.14, no-fly zones are highlighted on the map.

The Coverage layer used in this work is reported in Figure 3.16. It is a realistic map (but not real), where the quality of the signal depends on the distribution of antennas in the city.

In the following paragraphs, the risk-based map is computed taking into account

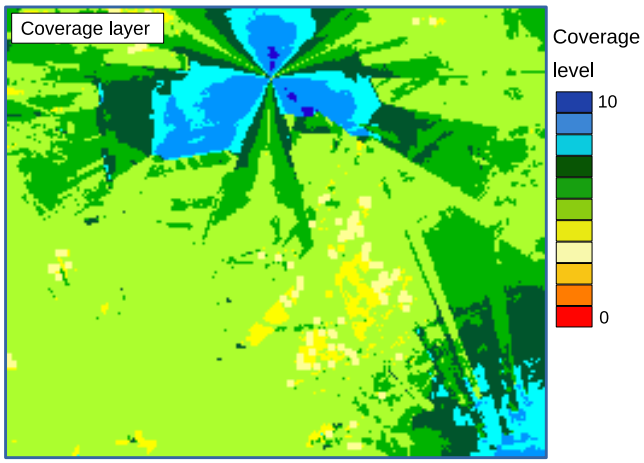


Figure 3.16: An exemplificative Coverage layer in a portion of the city of Turin. By courtesy of TIM: map created with a proprietary mobile planning tool.

several aircraft, which differ in configuration type, mass and dimension. Vehicles and the parameters are reported in Table 3.2.

In order to compare the risk associated with each vehicle, we define the same event probabilities for each vehicle, as determined in Table 3.3.

Table 3.2: Parameters of the aircraft used as example.

Specific	**Talon**	**DJI Phantom 4**	**DJI Inspire 2**	**Parrot Disco**	**DJI Mavic**	**Parrot Bebop**	**ADPM EVO**
Type	Fixed wing	Quadrotor	Quadrotor	Fixed wing	Quadrotor	Quadrotor	Quadrotor
Mass (kg)	3.75	1.4	4.25	0.75	0.7	0.5	0.3
Front area (m^2)	0.1	0.02	0.04	0.07	0.02	0.02	0.01
Radius (m)	0.88	0.2	0.4	0.575	0.2	0.25	0.115
Maximum flight time	1.25 h	20 min	25 min	45 min	20 min	25 min	10 min
Drag coefficient at ballistic descent	N(0.9, 0.2)	N(0.7, 0.2)	N(0.7, 0.2)	N(0.9, 0.2)	N(0.7, 0.2)	N(0.7, 0.2)	N(0.7, 0.2)
Initial horizontal speed (m/s)	N(18, 2.5)	U(0, 15)	U(0, 20)	N(15, 2.5)	U(0, 15)	U(0, 15)	U(0, 9.7)
Initial vertical speed (m/s)	N(0, 1)	N(0, 1)	N(0, 1)	N(0, 1)	N(0, 1)	N(0, 1)	N(0, 1)
Glide speed (m/s)	16	7.5	10.0	12.0	7.5	7.5	4.85
Glide ratio	N(12, 2)	N(2.7, 0.8)	N(2.7, 0.8)	N(12, 2)	N(2.7, 0.8)	N(2.7, 0.8)	N(2.7, 0.8)
Drag coefficient at parachute descent	N(1.3, 0.2)	N(0.9, 0.2)	N(1.3, 0.2)	N(1.3, 0.2)	N(0.5, 0.2)	N(0.3, 0.2)	-
Parachute area (m^2)	1	0.5	1	0.5	0.5	0.5	-
Parachute deployment time (s)	2	2	2	2	2	2	-

Table 3.3: The event probabilities.

	Ballistic	**UG**	**Parachute**	**Fly-away**
Event probability (per flight hour)	1/200	1/200	1/100	1/250

3.4.2 Talon aircraft

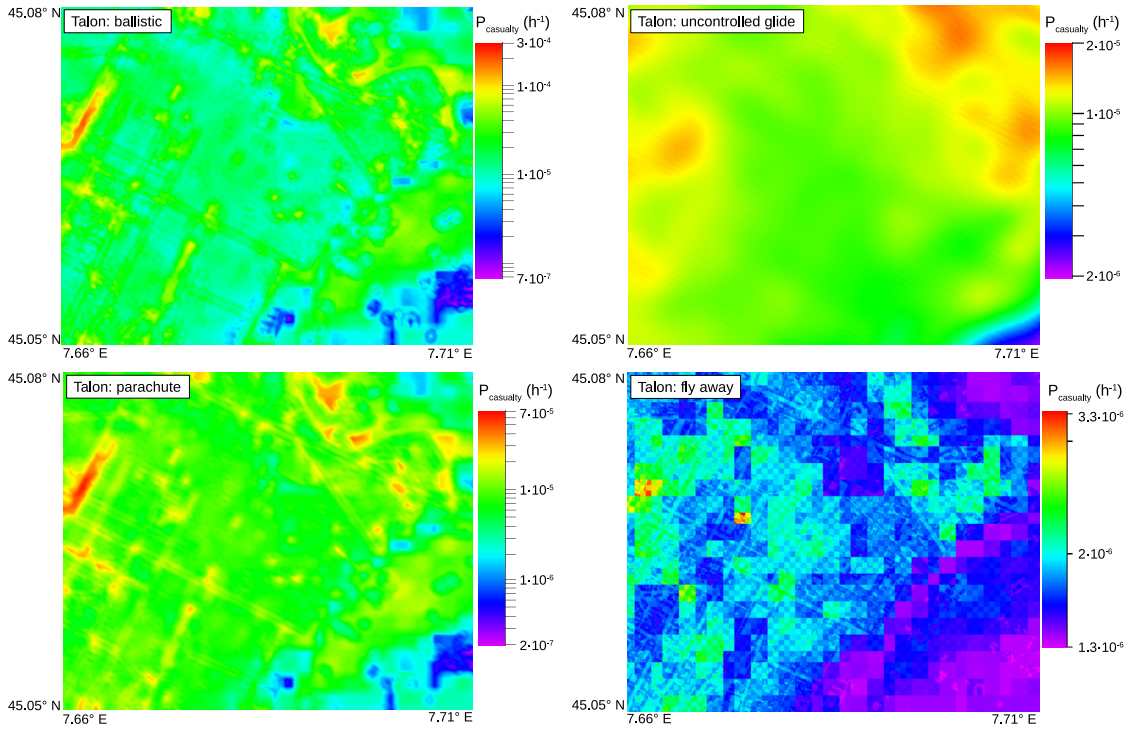


Figure 3.17: Event risk maps (from top to bottom: ballistic descent, uncontrolled glide, parachute descent and fly-away events) with the Talon aircraft. In all maps the flight altitude is $N(50, 5)$ m, wind has direction of -0.52 rad and speed $N(5, 1)$ m/s. The parameters of Talon aircraft are reported in Table 3.2.

The first vehicle taken into account is the Talon aircraft, a fixed wing aircraft. Using parameters of Table 3.2, the risk-based map and the event risk maps are computed. Resulting maps are shown in Figures 3.17 and 3.18, while the minimum, maximum and average risk values are reported in Table 3.4. In particular, the risk-based map of Figure 3.18 is computed by merging the four event risk maps of Figure 3.17 and including no-fly zones, obstacles at the flight altitude of 50 m and areas with a coverage level lower than 4.

According to maps of Figures 3.17 and 3.18, risk values are strongly affected by the population density. In fact, the highest risk values are concentrated in areas with greater population density, while the south-east area of the map is poorly populated and there is less risk.

With the Talon aircraft, predominant risk values are caused by the ballistic event and the uncontrolled glide. The main reason is the mass of the vehicle that causes high kinetic energy at impact, as well as the high cruise speed assumed as initial velocity. Moreover, fixed wing aircraft are characterized by high glide speed, very dangerous

for people on the ground. Risk values are reduced by the use of the parachute, but the resulting risk is still considerable, because of the mass of the vehicle and the wind. In fact, the parachute reduces the vertical velocity at impact, but the horizontal one is affected by the wind. The risk associated with the fly-away is also considerable, even if it is reduced by a wide probabilistic impact area that involves poorly populated areas.

In order to ensure an appropriate level of safety, the risk of a flight operation should be lower than a maximum acceptable risk. According to [39, 74], a suitable Equivalent Level of Safety (ELOS) is $1 \cdot 10^{-6}$ h^{-1}. Hence, referring to the risk values of Table 3.4, the Talon aircraft is not suitable to perform flight operations in urban areas. In fact, the risk is always greater than the ELOS.

The risk-based map can be used to plan a safe flight mission. In order to demonstrate it, we compute a minimum risk path using the risk-based map of Figure 3.18. The minimum risk path is computed by the Optimal Rapidly-exploring Random Tree (RRT*) algorithm [97] with the minimization of risk values [162]. The resulting path is reported in Figure 3.18, while Figure 3.19 illustrates the evolution of the risk along the path. This test confirms that the Talon aircraft is not appropriate to operate in this urban area.

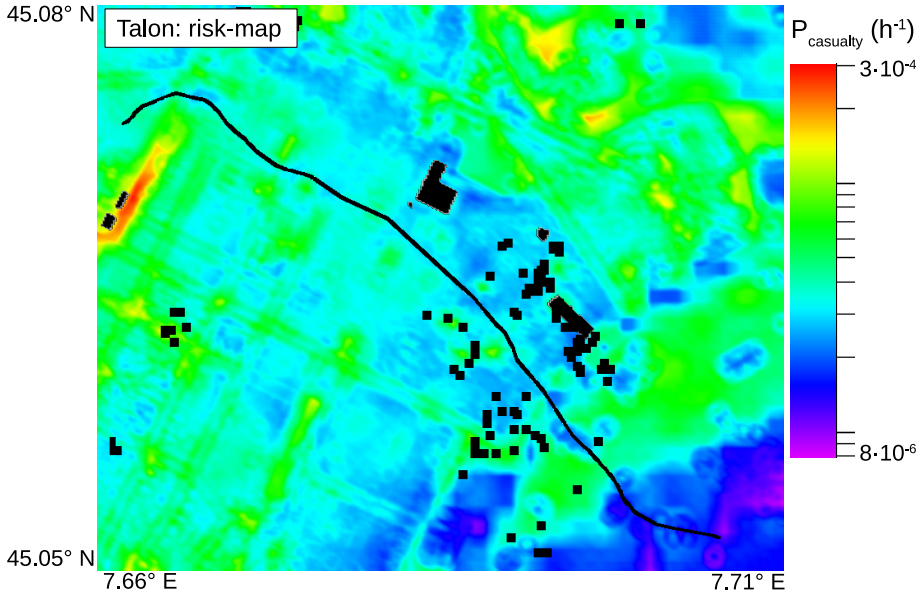


Figure 3.18: The risk-based map of the Talon aircraft with the same parameters and flight conditions of Figure 3.17. The map includes areas where the flight is not allowed because of no-fly zones, obstacles at the flight altitude of $N(50, 5)$ m and a coverage level lower than 4. On the risk map is reported the minimum risk path computed to demonstrate the potential of the risk map.

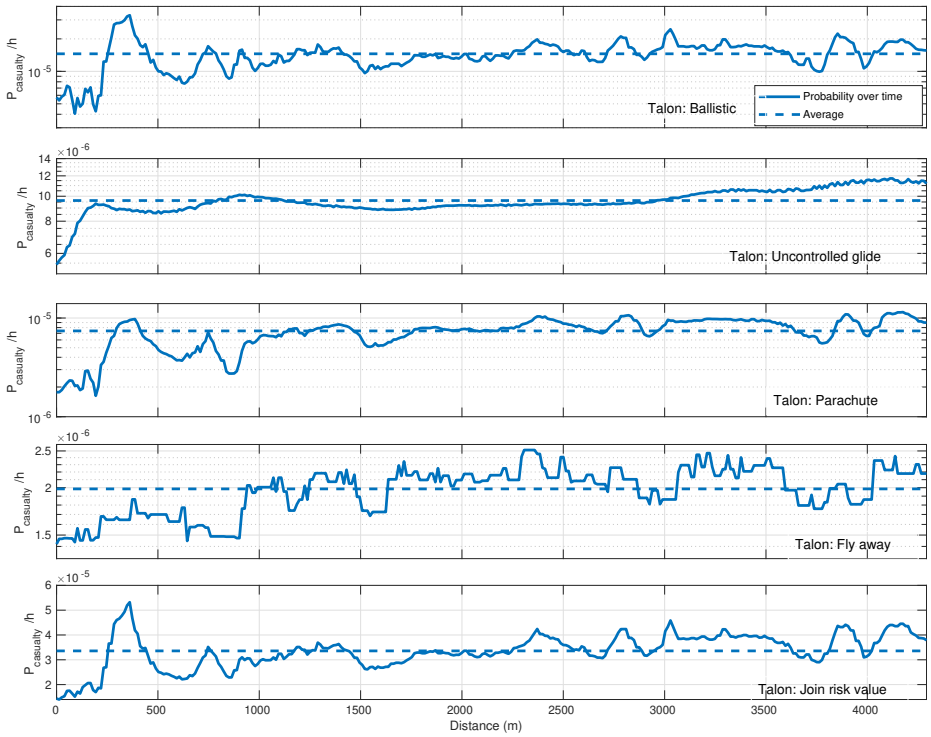


Figure 3.19: The distribution of the probability of casualty $P_{casualty}$ along the minimum risk path with the Talon aircraft. Probabilities of each descent event type and of the risk map are reported.

3.4.3 Phantom 4 aircraft

The Phantom 4 is a quadrotor aircraft. The associated risk-based map and event risk maps are reported in Figure 3.20, computed using the parameters of Table 3.2. Minimum, maximum and average risk values are reported in Table 3.4.

With the Phantom 4 aircraft, the risk is significantly lower than the risk obtained with the Talon aircraft. The first reason is the lower mass of the vehicle, as well as the lower cruise velocity that involves less kinetic energy at impact. Moreover, due to the quadrotor configuration, the glide speed is considerably lower. As a consequence, the risk values of the uncontrolled glide and fly-away events are drastically reduced.

Predominant risk values are determined by the ballistic descent event, while the risk associated with the parachute descent is null. This happens because the kinetic energy at impact is lower than the threshold determined by Equation (3.12).

According to risk values in Table 3.4, also the Phantom 4 aircraft is not suitable to fly over urban areas, because the average risk is greater than the ELOS threshold. However, there are few areas with acceptable risk, i.e. areas with low population density.

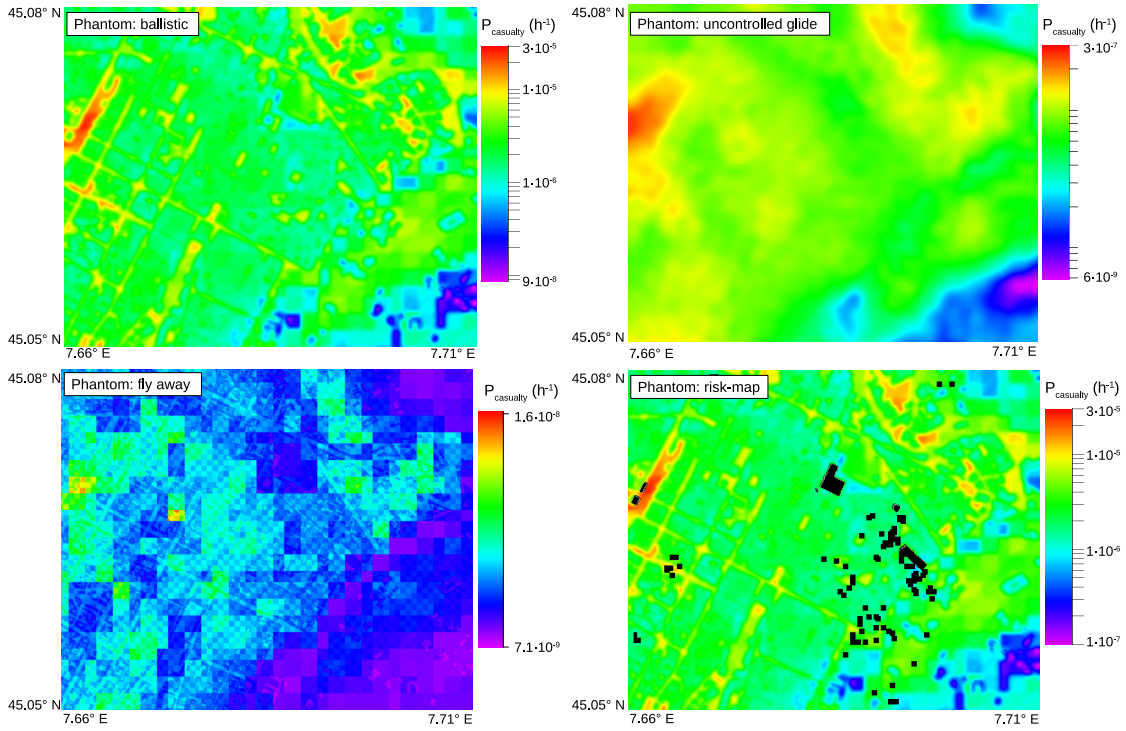


Figure 3.20: Event risk maps (from top to bottom: ballistic descent, uncontrolled glide and fly-away events) and the risk map with the Phantom aircraft. The parachute descent is not reported because the risk is null for all elements in the map. In all examples, the flight altitude is $N(50, 5)$ m and wind has direction of -0.52 rad and speed $N(5, 1)$ m/s. The parameters of Phantom aircraft are reported in Table 3.2.

3.4.4 Other vehicles

Several risk-based maps are created considering other vehicles, in order to define which vehicles are suitable to operate in urban environments. Vehicles are: the DJI Inspire, the Parrot Disco, the DJI Mavic, the Parrot Bebop and the ADPM Evo. These vehicles have different characteristics, such as mass and maximum velocity. Parameters are reported in Table 3.2.

The DJI Inspire is a professional multirotor aircraft, widely used in a wide variety of applications, because of its versatility and payload capacity. Due to the vehicle mass and dimensions, this aircraft is not suitable for urban areas, because risk values are always greater than the ELOS.

The Parrot Disco is a lightweight fixed wing aircraft. However, even if the vehicle has a low mass, the resulting risk is considerable because of the configuration of the vehicle that involves high cruise velocity, glide speed and, as a consequence, high kinetic energy at impact. For this reason, generally, fixed wing aircraft are not suitable to operate in urban areas.

The DJI Mavic and the Parrot Bebop are quadrotor aircraft. With both vehicles, the resulting risk is determined only by the ballistic descent event. In fact with both parachute and uncontrolled glide events, the kinetic energy at impact is lower than the fatality threshold, due to the low mass. Similarly, with the fly-away event that terminates with an uncontrolled glide event. With both vehicles, the risk is acceptable in most areas, even if there are few areas where the risk is greater than the ELOS, i.e. areas with the maximum population density.

The last vehicle is the ADPM EVO, a lightweight aircraft considered harmless by ENAC. Due to the small size, it is very difficult to equip the vehicle with a parachute. As a consequence, with the ADPM EVO, we increase the probability of ballistic descent by incorporating the probability of parachute descent generally considered by other vehicles. Hence, $P_{\text{event}}^{\text{bal}} = 3/200\ \text{h}^{-1}$. Even if the probability of the ballistic event is three times greater than other vehicles, the resulting risk is acceptable for the tested urban area. Only a few areas have a risk greater than the ELOS threshold.

In order to demonstrate it, we compute a minimum risk path considering the ADPM EVO aircraft. Both risk-based map and minimum risk path are shown in Figure 3.21, while in Figure 3.22 the evolution of the risk along the path is illustrated. The minimum risk is also compared with the line-of-sight (LOS) path, i.e. the path performed by default by an autopilot without any path planning procedure. The minimum risk path is longer than the LOS path, but, the resulting risk is lower. Moreover, the average risk of the path is about $6 \cdot 10^{-7}\ \text{h}^{-1}$, lower than the ELOS threshold.

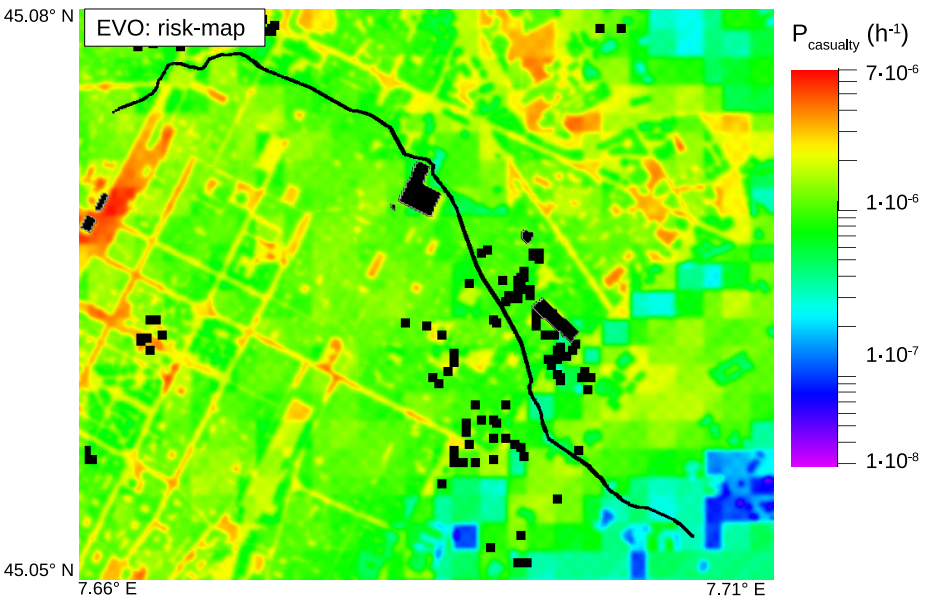


Figure 3.21: The risk-based map with the Bebop aircraft. On the risk map is illustrated the minimum risk path.

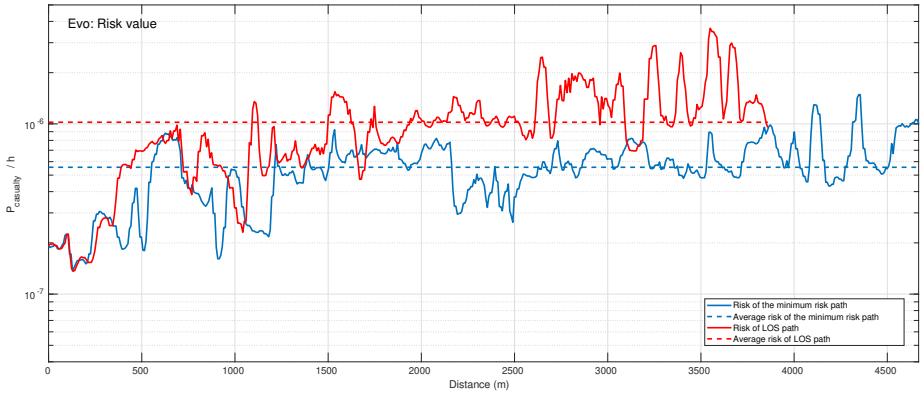


Figure 3.22: The distribution of the probability of casualty along the minimum risk path of Figure 3.21 with the ADPM EVO aircraft.

Table 3.4: Probabilities of casualty per flight hours for all risk-based maps.

Vehicle		**Ballistic**	**Uncontrolled glide**	**Parachute**	**Fly-away**	**Final risk**
Talon	min	$7.820 \cdot 10^{-7}$	$2.392 \cdot 10^{-6}$	$2.894 \cdot 10^{-7}$	$1.302 \cdot 10^{-6}$	$8.497 \cdot 10^{-6}$
	max	$2.042 \cdot 10^{-4}$	$1.638 \cdot 10^{-5}$	$6.346 \cdot 10^{-5}$	$3.593 \cdot 10^{-6}$	$2.784 \cdot 10^{-4}$
	av	$2.161 \cdot 10^{-5}$	$1.056 \cdot 10^{-5}$	$9.164 \cdot 10^{-6}$	$1.962 \cdot 10^{-6}$	$4.329 \cdot 10^{-5}$
Phantom	min	$9.439 \cdot 10^{-8}$	$6.518 \cdot 10^{-9}$	0	$7.105 \cdot 10^{-9}$	$1.106 \cdot 10^{-7}$
	max	$2.631 \cdot 10^{-5}$	$2.565 \cdot 10^{-7}$	0	$1.535 \cdot 10^{-8}$	$2.653 \cdot 10^{-5}$
	av	$2.977 \cdot 10^{-6}$	$8.705 \cdot 10^{-8}$	0	$9.961 \cdot 10^{-9}$	$3.074 \cdot 10^{-6}$
Inspire	min	$4.479 \cdot 10^{-7}$	$2.823 \cdot 10^{-7}$	$1.369 \cdot 10^{-7}$	$3.518 \cdot 10^{-7}$	$1.369 \cdot 10^{-6}$
	max	$1.107 \cdot 10^{-4}$	$1.030 \cdot 10^{-5}$	$3.430 \cdot 10^{-5}$	$5.330 \cdot 10^{-7}$	$1.498 \cdot 10^{-4}$
	av	$1.201 \cdot 10^{-5}$	$3.045 \cdot 10^{-6}$	$4.660 \cdot 10^{-6}$	$4.159 \cdot 10^{-7}$	$2.013 \cdot 10^{-5}$
Disco	min	$2.942 \cdot 10^{-8}$	$1.011 \cdot 10^{-7}$	0	$5.409 \cdot 10^{-8}$	$3.494 \cdot 10^{-7}$
	max	$8.715 \cdot 10^{-6}$	$9.977 \cdot 10^{-7}$	0	$1.103 \cdot 10^{-7}$	$9.679 \cdot 10^{-6}$
	av	$1.326 \cdot 10^{-6}$	$6.703 \cdot 10^{-7}$	0	$7.051 \cdot 10^{-8}$	$2.067 \cdot 10^{-6}$
Mavic	min	$4.758 \cdot 10^{-8}$	0	0	0	$4.758 \cdot 10^{-8}$
	max	$1.093 \cdot 10^{-5}$	0	0	0	$1.093 \cdot 10^{-5}$
	av	$1.413 \cdot 10^{-6}$	0	0	0	$1.413 \cdot 10^{-6}$
Bebop	min	$2.599 \cdot 10^{-8}$	0	0	0	$2.599 \cdot 10^{-8}$
	max	$6.853 \cdot 10^{-6}$	0	0	0	$6.853 \cdot 10^{-6}$
	av	$9.653 \cdot 10^{-7}$	0	0	0	$9.653 \cdot 10^{-7}$
EVO	min	$1.957 \cdot 10^{-8}$	0	-	0	$1.957 \cdot 10^{-8}$
	max	$6.482 \cdot 10^{-6}$	0	-	0	$6.482 \cdot 10^{-6}$
	av	$9.771 \cdot 10^{-7}$	0	-	0	$9.771 \cdot 10^{-7}$

3.5 Discussion

In this Chapter, we introduce a risk-based map to quantify the risk of UASs over urban areas. The presented results show how the risk-based map is able to assess the risk associated with a particular aircraft, considering its parameters and environmental characteristics, as well as the wind.

Risk values of the risk-based map are computed with a probabilistic risk assessment approach, taking into account four different descent event types, drone parameters and uncertainties. The risk value of each element of the map $\mathbf{R}(x, y)$ assumes that a failure and the associated descent event occurs in (x, y). Hence, the risk is computed considering a probabilistic impact area. Differently, risk maps computed in [74, 157] assume that the vehicle impacts directly in the analyzed element of the map, without considering the UAS behavior during the descent.

The risk-based map is a promising tool to ensure risk-informed decision making. As reported by results, the proposed risk-based map is able to define in which areas the flight is allowed or not, because of obstacles, no-fly zones or high-risk areas. Moreover, it is able to identify which vehicles are suitable to operate in a particular urban area. As reported in Table 3.4, only lightweight aircraft imply a risk lower than the ELOS threshold. Generally, fixed wing aircraft are not appropriate to operate over inhabited areas because they involve high kinetic energy at impact.

The risk-based map can be used by NAAs as a tool to assess the risk of flight operations in order to provide permission to fly. In order to meet regulation requirements, the risk-based map can be computed assuming the worst flight conditions, such as a maximum flight altitude and cruise speed, as well as pessimistic wind-conditions. The resulting risk-based map may be conservative, but it guarantees that the ELOS requirement is satisfied. Moreover, according to [36], the probabilistic risk assessment is in agreement with the SORA (Specific Operations Risk Assessment) approach proposed by JARUS and adopted by EASA.

As demonstrated in results, the risk-based map can be used to compute the minimum risk path. In fact, using a risk-aware path planning, the minimum risk path can be computed. Hence, we are able to quantify the average risk of the path, determining if it guarantees the ELOS requirement. The combination of risk-based map and risk-aware path planning allows a safe and optimal path to be planned and executed.

Chapter 4

Risk-aware path planning strategies for UASs in urban environments

This Chapter describes two different risk-aware path planning strategies for Unmanned Aerial Systems in urban environments. The risk-aware path planning is one of the modules defined in the Cloud-based architecture of Chapter 2.

The risk-aware path planning aims to compute an effective path minimizing the risk to the population on the ground, in order to define safe flight operations in urban areas. The risk is quantified by the risk-based map introduced in Chapter 3. The risk-aware path planning consists into two phases: first, an offline path planning searches for the optimal path considering a static risk-based map, then, using the dynamic risk-based map, an online path planning updates and adapts the offline path according to dynamically arising conditions.

In this Chapter, two different strategies are proposed, using different path planning algorithms.

In the first approach, a risk-aware path planning is performed using the riskA* and the Borderland algorithms. RiskA* is based on the well-known A* algorithm and solves an offline path planning problem. The online one is solved by the Borderland algorithm, which uses the *check and repair* approach to rapidly update the offline path.

The second strategy uses riskRRT^{x}, a path planning and re-planning algorithm suitable to perform both the offline and online phases.

The Chapter is organized as follows. In Section 4.1 some background information are reported. In Section 4.2, the risk-aware path planning method assumed in this Chapter is explained. Section 4.3 describes the first risk-aware path planning strategy, based on riskA* and Borderland algorithms, while Section 4.4 introduces the riskRRT^{X} algorithm, used to implement another risk-aware path planning strategy. Finally, we discuss the proposed strategies in Section 4.5.

4.1 Background

The extensive use of Unmanned Aerial Systems (UASs) has induced the rapid growth of related research areas. Because of their success, the technology of UASs is growing and, at present, intelligent and autonomous aircraft are already implemented [87].

Path planning is one of the most important elements in autonomous agents, by defining the route to reach the desired destination, satisfying some optimal criterion [72]. The path planning problem has been widely studied in the last years. One of the first path planning algorithms is the Dijkstra algorithm [48], presented in the late 50s. Graph search algorithms are widely used to solve the path planning problem. One of the most popular is A* [79], which includes a heuristic component in the cost function. In literature, there are many works based on A* for dynamic [191] and anytime planning [117, 118], and in high dimensional environments [88]. A famous A* variant is the Theta* algorithm [141], that solves the problem of path constraining to grid edges.

Other popular path planning techniques comprise sample-based approaches, which explore the search space with a sampling scheme. The most popular sample-based algorithms are the Probabilistic Roadmaps (PRM) [100] and the Rapidly-exploring Random Trees (RRT) [112]. Especially, RRT is widely used in literature and many RRT-based algorithms are developed to perform optimal [97] and anytime [99] path planning, as well as with kinodynamic constraints [98].

In literature, there are many path planning algorithms specifically dedicated to unmanned aircraft based on sample-based algorithms [122], evolutionary algorithms [146] and reinforcement learning approaches [218]. In [46] the authors propose an energy-efficient path planning, while in [216] the path planning optimizes the mission considering mobile recharging stations. Often, path planning for unmanned aircraft is based on an explicit 3D description of the environment [42, 136].

When an autonomous vehicle operates in an inhabited and uncertain environment, the risk should be considered in order to compute a safe path. Static and dynamic threats are taken into account in [211], where a dynamic path planning for UAS is proposed. In [41, 210], risk maps are used to compute an optimal and minimum risk path. Risk-aware path planning is a common problem in robotics, concerning also mobile robots [63] and AUVs (Autonomous Underwater Vehicles) [154].

Generally, risk-aware path planning considers the risk emerging from the aircraft point of view, taking into account the risk of collision with other vehicles and obstacles [182, 211]. Anyway, when the UAS flies over inhabited areas, the risk should consider the risk to the population on the ground. In [177], a risk-based path planning is proposed, minimizing the trade off between the risk to the population and flight time. In [74], a risk-aware path planning based on A*, called RA*, computes the minimum risk path. Similarly, in [101] the bidirectional RRT (bi-RRT) is used to compute a safe path. Risk-aware path planning is also used in emergency landing [187], in order to define a safe landing location.

In this Chapter, we propose a risk-aware path planning strategy in order to compute a safe path, considering both static and dynamic risk factors for the population on the ground along the route. The proposed approach refers to a risk-based map that quantifies the risk to the population of an urban area. The risk-aware path planning strategy consists in two phases: first, the offline path planning computes the effective path considering the static risk, then, based on changes in the dynamic risk-based map, the online path planning adapts and updates the offline path.

The risk-aware path planning strategy is implemented with two different methods. A first approach uses the riskA* and the Borderland algorithms as offline and online path planning, respectively. This approach is introduced in [156, 157].

A second approach uses the riskRRT^X algorithm, able to perform both phases.

The proposed risk-aware path planning is introduced for the first time in [161], where a Cloud-based framework for risk-aware intelligent flying for UAS is presented.

4.2 Risk-aware path planning

The proposed risk-aware path planning approach searches for the optimal path accounting for both static and dynamic risk factors for the population on the ground along the route. The input is a risk-based map, a dynamical two-dimensional location-based map, where each location has a specific value, called risk cost that quantifies the risk of flying over that location [158]. The risk-based map is generated as described in Chapter 3. The proposed approach consists of two phases: offline and online path planning.

The offline path planning solves an optimal path planning problem. Given the starting and final positions and the risk-based map, the offline path planning searches for a globally optimal path, minimizing the risk costs defined by the risk-based map, avoiding obstacles and no-fly zones. The offline path is defined before the mission starts and, in general, the aircraft is still on the ground, then, it is not time constrained.

The online path planning adapts and updates the offline path, according to changes in the dynamic risk-based map. In fact, during the mission execution, the risk-based map changes because of obstacles or other risk factors appear. Hence, the online path planning updates the path with a *check and repair* approach. Unlike the offline path planning, this phase is time-critical, because the aircraft is executing the mission and needs to react promptly to dynamically changing conditions.

After the path planning procedure, a fast path smoothing based on Dubins curves [51] is applied to the obtained path, in order to transform the path in a flyable one. Hence, the resulting path is handed over to the UAS, able to execute the flight mission.

The architecture of the proposed approach is illustrated in Figure 4.1.

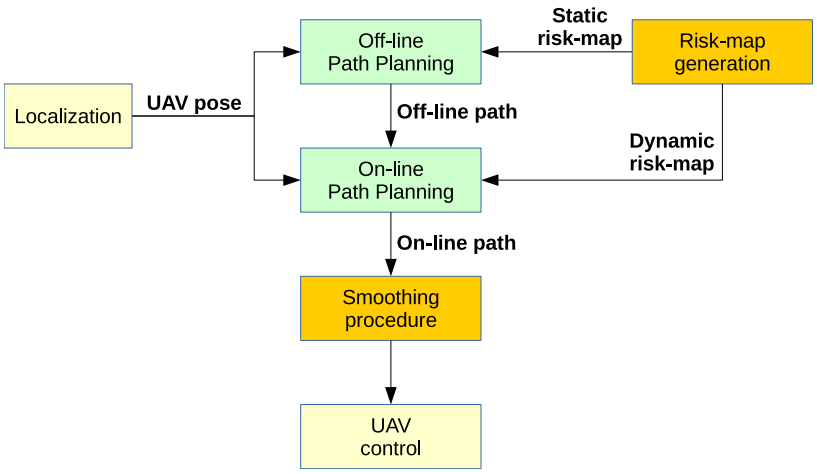


Figure 4.1: The main architecture of the proposed risk-aware path planning approach.

4.2.1 Risk-based map

The risk-based map is described in detail in Chapter 3, explaining the generation process, as well as the probabilistic risk assessment approach. Anyway, in this section, some preliminary concepts about the risk-based map are defined, in order to understand better the proposed solution, as well as the mathematical notation used in the rest of the Chapter.

In path planning problems, the map has an essential role because it defines the search space, i.e., the space with all possible configurations where the path planning algorithm seeks for the solution [113].

The risk-based map is a two-dimensional location-based map that quantifies the risk of flying over a specific area. Hence, the map is defined as a 2D matrix $\mathbf{R}$ of a $n \times m$ locations, where each element of the map $\mathbf{R}(p_{i,j})$ has a risk value that quantifies the risk level associated to the location $p_{i,j}$. Notice that, hereby we refer to a location $p_{i,j}$, but it corresponds to the location (x, y) of the risk-based map of Chapter 3. This notation has been changed because in the following sections we refer to states x of the search space X, a common notation used to describe the path planning problem [113].

The risk value is determined with a probabilistic risk assessment approach, in which the risk is defined as the hourly probability to have a casualty, a classic measure system used in aviation. The risk-based map is computed assuming the flight at fixed altitude.

4.2.2 Problem formulation

Offline path planning

Let $C \subseteq \mathbb{R}^2$ be a continuous search space of a path planning problem. As in many other applications [140], C is discretized into a discrete space X, on which the risk-based map will be constructed considering the map dimension and resolution. Each state $x \in X$ is a discrete location in the discrete search space. With a slight abuse of notation, here and henceforth we will refer to x as a state of search space, location of the risk-based map $\mathbf{R}$ or a node of a search grid graph.

The obstacle region $X_{\text{obs}} \subseteq X$ is the set of locations in which flight is forbidden because of obstacles at the flight altitude, no fly zones or with no coverage. The set $X_{\text{free}} = X \setminus X_{\text{obs}}$ contains the remaining navigable locations.

The initial and final locations are denoted as $x_{\text{start}}, x_{\text{goal}} \in X_{\text{free}}$. Let Σ be the set of all paths, where a single path σ is a sequence of connected locations x in the search space X. The path planning algorithm searches for an optimal path σ^* from x_{start} to x_{goal} in X_{free} that minimizes a given cost function $f : \Sigma \to \mathbb{R} \geqslant 0$. Hence, the optimal path is the solution of the following program:

$$\begin{aligned} & \sigma^* = \arg\min_{\sigma \in \Sigma} f\big(\sigma(t)\big) \\ \text{subject to} \quad & \sigma(0) = x_{\text{start}} \\ & \sigma(1) = x_{\text{goal}} \\ & \forall t \in [0,1]\,,\ \sigma(t) \in X_{\text{free}}. \end{aligned} \tag{4.1}$$

Online path planning

The online path planning is based on a *check and repair* approach [62], in which the path is dynamically adapted in accordance with a dynamic risk-based map.

Recalling the notation of the offline path planning, the offline path σ is a sequence of locations $x \in X_{\text{free}}$ from x_{start} to x_{goal}. Considering the dynamic risk-based map, the search space $X(k)$ changes at each discrete-time step k, then the path is considered as a sequence of states $x(k)$. At each time step, the differential search space $X_{\text{diff}}(k)$ can be defined as follows:

$$X_{\text{diff}}(k) = X(k) - X(k-1), \tag{4.2}$$

The *check* routine verifies if $x_n(k-1) \in X_{\text{diff}}(k),\ \forall\ x(k) \in \sigma$, i.e., it verifies which part of the path has to be updated because of a change in the risk-based map.

The *repair* routine tries to adjust the path with a fast algorithm, in order to tackle the dynamic risk-based map.

4.3 RiskA* and Borderland

In this section, we present the first risk-aware path planning approach based on the combination of riskA* and Borderland algorithms. Each algorithm is described in detail, then, simulation results corroborate the proposed approach.

4.3.1 RiskA* algorithm

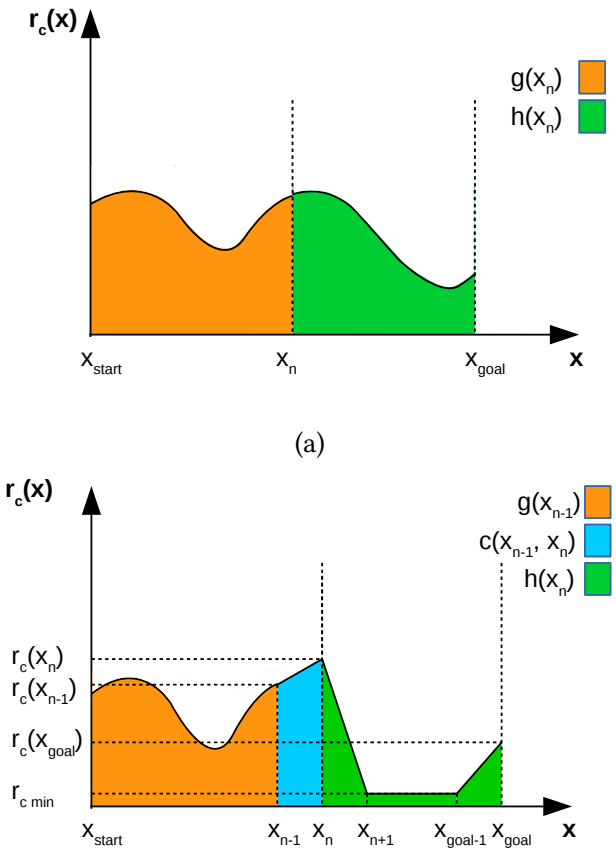


Figure 4.2: Graphical representation of the cost function $f(x)$. Given a generic state x_n, in (a), the cost function is composed by the motion cost $g(x_n)$ and the heuristic cost $h(x_n)$. Similarly, in (b), the incremental step defined in Equations (4.6), (4.7), (4.8), (4.9) is illustrated.

The riskA* algorithm is based on the well-known A* [79]. The input of riskA* is a two-dimensional grid graph **R**, where each element corresponds to a graph node, and each portion of the path between two adjacent nodes corresponds to a graph edge.

Similarly to A*, the output of the algorithm is a back-pointer path, i.e. a sequence of nodes starting from the goal and tracing back to the start node.

In the same way of A*, riskA* searches for the best solution in the graph minimizing the cost function $f(x)$

$$f(x) = g(x) + k \cdot h(x), \tag{4.3}$$

with $g(x)$ is the effective motion cost of the path from the start node x_{start} and the node x, $h(x)$ is the heuristic function, i.e., the estimated motion cost from the node x and the goal node x_{goal} and the constant k is the adjustment variable. Then, $f(x)$ is the estimated motion cost of the path from x_{start} to x_{goal} passing through the node x.

Differently from the traditional A*, riskA* computes a cost function $f(x)$ considering risk costs along the path. Given a generic node x_{n}, the function $g(x_{\text{n}})$ is defined as the integral of the risk cost between the starting node x_{start} and the node x_{n}

$$g(x_{\text{n}}) = \int_{x_{\text{start}}}^{x_{\text{n}}} r_{\text{c}}(x) dx, \tag{4.4}$$

with $r_{\text{c}}(x)$ is the risk cost function that takes values in $0 < r_{\text{c}}(x) \leq 1$, based on risk levels in the risk-based map. Each element of the risk-based map $\mathbf{R}(p_{i,j})$ has a risk level expressed in fatalities *per* flight hours. Hence, the risk cost is determined in the range between 0 and 1, where 0 corresponds to an area with no risk, while 1 to an area with the maximum risk, i.e., a no flyable area. The maximum acceptable risk is defined according to the Equivalent Level of Safety (ELOS) required by National aviation agencies. According to [39, 74], a suitable and reasonable value for small UASs is $1 \cdot 10^{-6} h^{-1}$. Anyway, the risk cost is never assumed equal to zero because we cannot say that no one will be involved in the crash. As a consequence, the motion cost is never equal to zero.

Similarly, the heuristic function is the integral of the risk cost between the state x_{n} and the goal node x_{goal}

$$h(x_{\text{n}}) = \int_{x_{\text{n}}}^{x_{\text{goal}}} r_{\text{c}}(x) dx, \tag{4.5}$$

Figure 4.2a reports the cost function $f(x)$ computed at the generic node x_{n}. $g(x)$ is the motion cost between the starting node and the node x_{n}, while $h(x)$ is the estimated motion cost until the goal node. Thus, $g(x)$ and $h(x)$ are complementary along the path.

The riskA* algorithm is defined by Algorithm 1. The inputs are the initial node x_{start} and the final node x_{goal} and the grid graph $\mathbf{R}$ related to the risk-based map. Equally to A*, two data structures are used: the Open set O and the Closed set C.

The Open set contains the currently discovered nodes waiting to be evaluated. It is a priority queue, where elements are ordered according to the estimated cost $f(x)$, such that a node with a lower motion cost is evaluated before a node with a higher cost.

On the contrary, the Closed set is the set of nodes already processed or invalid.

The riskA* algorithm uses the same logic of the original A*, in which the main routine explores the search space until the Open set is empty or the goal node is reached.

Algorithm 1 riskA* algorithm

procedure RISKASTARSEARCH($x_{\text{start}}, x_{\text{goal}}, \mathbf{R}$)
  Add x_{start} to O
  Add all invalid nodes $x_{\text{invalid}} \in \mathbf{R}$ to C
  repeat
    Pick x_{best} from O with $f(x_{\text{best}}) \leqslant f(x), \forall x \in O$
    if $\exists$ multiple x_{best} **then**
      Pick the x_{best} with lower $r_{\text{c}}(x_{\text{best}})$
    end if
    Remove x_{best} from O and add to C
    if $x_{\text{best}} = x_{\text{goal}}$ **then**
      return ReconstructPath($x_{\text{goal}}, x_{\text{start}}$)
    end if
    Expand x_{best}: for all $x_{\text{adj}} \in \text{Near}(x_{\text{best}})$ and $x_{\text{adj}} \notin C$
    if $x_{\text{adj}} \notin O$ **then**
      Add x_{adj} to O
    else if $g(x_{\text{best}}) + c(x_{\text{best}}, x_{\text{adj}}) < g(x_{\text{adj}})$ **then**
      Update x_{adj}'s backpointer to point to x_{best}
    end if
  until O is empty
  return ReconstructPath($x_{\text{goal}}, x_{\text{start}}$)
end procedure

Important variants are in lines 6, 7, where if multiple x_{best} nodes with the same cost exist, the algorithm picks the node with lower risk cost.

The riskA* algorithm creates a search tree, which, by definition, has no cycles.

When the algorithm evaluates a node x_{n}, the cost $f(x_{\text{n}})$ is computed using the formulation of Equations (4.3), (4.4) and (4.5). Practically, because of the discrete grid map, the integral is computed using an incremental and approximate method.

The function $g(x_{\text{n}})$ is the sum of the motion cost at the previous node $g(x_{\text{n}-1})$ and $c(x_{\text{n}-1}, x_{\text{n}})$, i.e., the trapezoidal area described by the motion cost from the node $x_{\text{n}-1}$ and x_{n}. Hence

$$\begin{aligned} g(x_{\text{n}}) &= \int_{x_{\text{start}}}^{x_{\text{n}-1}} r_{\text{c}}(x)dx + \int_{x_{\text{n}-1}}^{x_{\text{n}}} r_{\text{c}}(x)dx \\ &= g(x_{\text{n}-1}) + c(x_{\text{n}-1}, x_{\text{n}}), \end{aligned} \tag{4.6}$$

with

$$c(x_{\text{n}-1}, x_{\text{n}}) = \frac{r_{\text{c}}(x_{\text{n}-1}) + r_{\text{c}}(x_{\text{n}})}{2}\text{dist}(x_{\text{n}-1}, x_{\text{n}}), \tag{4.7}$$

with $\text{dist}(x_{\text{n}-1}, x_{\text{n}})$ the Euclidean distance between two adjacent nodes.

Similarly, the function $h(x_{\text{n}})$ is computed considering the area described by the estimated motion cost from the node x_{n} and x_{goal}. By definition, if the heuristic function used by A*-based algorithm is admissible, the algorithm is able to seek for the optimal solution. A heuristic function $h(x_{\text{n}})$ is admissible if $h(x_{\text{n}}) \leq h^*(x_{\text{n}})$ for all nodes in the grid graph, and with $h^*(x_{\text{n}})$ being the effective optimal motion cost from x_{n} to the goal node x_{goal}. If the heuristic is not admissible, the algorithm may overestimate the motion cost to reach the goal, overlooking nodes that would lead to the optimal solution. As a consequence, in order to have an admissible heuristic function, the estimated motion cost is computed considering the minimum risk cost between nodes x_{n} and x_{goal}. In fact, during the computation of the heuristic function, we are not able to define the risk cost to reach the goal node. Hence, the heuristic is computed as

$$\begin{aligned} h(x_{\text{n}}) = &\frac{r_{\text{c}}(x_{\text{n}}) + r_{\text{c min}}}{2}\text{dist}(x_{\text{n}}, x_{\text{n}+1}) + \text{dist}(x_{\text{n}+1}, x_{\text{goal}-1})r_{\text{c min}} + \\ &+ \frac{r_{\text{c}}(x_{\text{goal}}) + r_{\text{c min}}}{2}\text{dist}(x_{\text{goal}-1}, x_{\text{goal}}), \end{aligned} \tag{4.8}$$

where $r_{\text{c min}} > 0$ is the minimum value assumed by the risk-cost function. Assuming $\text{dist}(x_{\text{n}}, x_{\text{n}+1}) = \text{dist}(x_{\text{goal}-1}, x_{\text{goal}}) = \text{dist}_{\text{min}}$, with dist_{min} the minimum distance between two adjacent nodes, the heuristic function becomes:

$$h(x_{\text{n}}) = \frac{r_{\text{c}}(x_{\text{n}}) + r_{\text{c}}(x_{\text{goal}})}{2}\text{dist}_{\text{min}} + (\text{dist}(x_{\text{n}}, x_{\text{goal}}) - \text{dist}_{\text{min}})r_{\text{c min}} \tag{4.9}$$

Figure 4.2b illustrates how both $g(x)$ and $h(x)$ are computed with the incremental and approximate step.

Algorithm 2 Post-Optimization algorithm

```
1:  procedure PostOptimization(σ)
2:      j = 1, k = 3
3:      while j ≠ length() do
4:          Interpolate(LOS(x_j, x_k))
5:          if cost(LOS(x_j, x_k)) ⩽ cost(segment(x_j, x_k)) then
6:              Replace segment(x_j, x_k) ∈ σ with LOS(x_j, x_k)
7:              Update k
8:              k = k + 1
9:          else
10:             j = j + 1
11:         end if
12:     end while
13:     return
14: end procedure
```

Since riskA* is an ad-hoc variant of the well-known A* algorithm, the time complexity of riskA* is the same of A*. Generally, the time complexity of A* depends on the heuristic. In the worst case, the number of nodes expanded increases exponentially with the depth of the solution (d), i.e. the shortest path between the start and goal nodes. Hence, the time complexity is $\mathcal{O}(b^d)$, with d is the branching factor. However, according to [215], the complexity of A* on a grid is $\mathcal{O}(d^2)$, and it does not depend on the grid type.

Generally, A*-based algorithms seek for the optimal path in the graph. Anyway, due to the discrete grid map, the solution of A*-based algorithms may not be the optimal one in the continuous space. In fact, the path is constrained to turn angle multiple of 45°. For this reason, a post-optimization phase is executed after the riskA* algorithm execution.

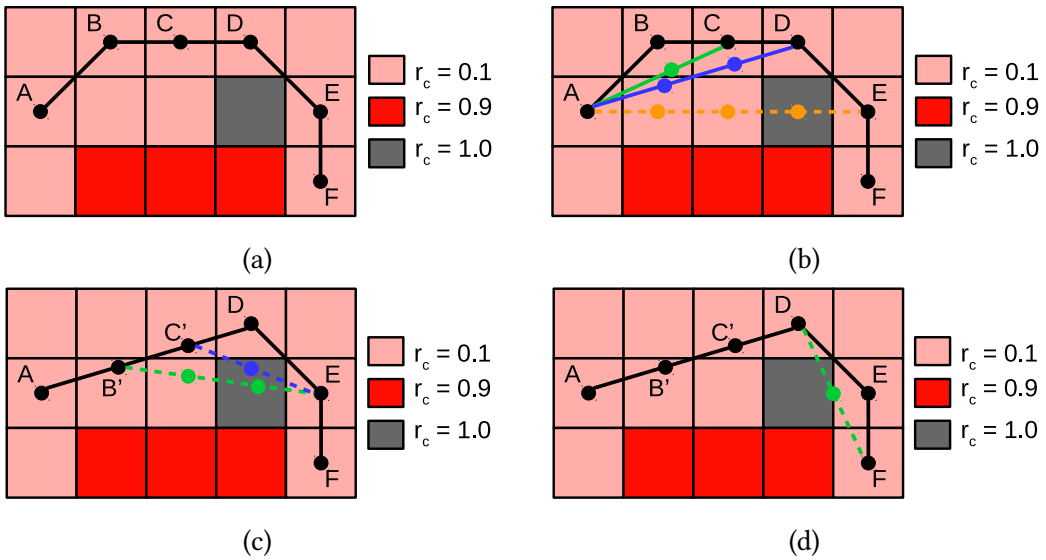


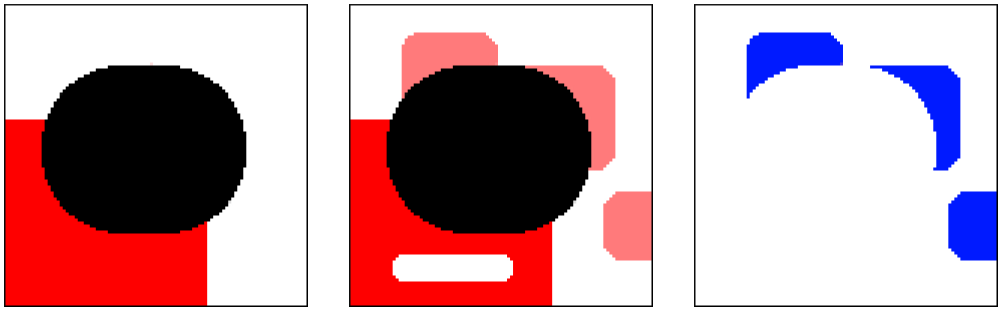
Figure 4.3: Simple example of the Post-Optimization procedure. In (a), the path is computed with riskA* as a sequence of nodes from A to F. In (b), the Post-Optimization procedure searches for LOS(·) segments that improve the path. Starting from node A, it considers at first the LOS(A, C), then the LOS(A, D). On the contrary, it discards the LOS(A, E) because it crosses a high risk area. In (c), the path is updated with the segment A-B'-C'-D, with B' and C' being the interpolated nodes of the LOS(A, D). Then, the Post-Optimization procedure discards the LOS(B′, E) and the LOS(C′, E), as well as the LOS(D, F) in (d).

The post-optimization procedure is reported in Algorithm 2. The input is the path σ computed by riskA*, while the output is the post-optimized path. The post-optimization algorithm explores the path with an iterative procedure (lines 3 to 12). Taking into account two nodes $x_j, x_k \in \sigma$, the algorithm verifies if the line-of-sight segment LOS(x_j, x_k) is valid and improves the motion cost of the path. The LOS(·) segment connects two nodes x_j and x_k with a straight line in the continuous space, inserting additional nodes

using linear interpolation (line 4), with an interpolation step comparable with the grid map resolution. If the LOS(x_j, x_k) segment has a lower motion cost than the motion cost of path from x_j to x_k, the algorithm replaces the old segment, denoted as (x_j, x_k), with the LOS(x_j, x_k) and updates the path (line 6). The motion cost cost($\cdot$) of the LOS($\cdot$) segment is computed using the method described in Equations (4.6) and (4.9). After the segment is replaced, the k index needs to be updated with respect to the updated path (line 7). The main iterative procedure of the post-optimization continues until the x_{goal} is reached. Figure 4.3 illustrates a simple example of the post-optimization procedure.

The Post-Optimization procedure aims to optimize the path locally since the global optimization is guaranteed by the riskA* algorithm.

4.3.2 Borderland algorithm



(a) (b) (c)

Figure 4.4: Example of the risk-based map in which the risk areas are identified: white areas are with minimum risk-cost, black areas are with maximum risk-cost, and shade of red areas are with middle cost, in which darker red areas involving more risk than bright red ones. In (a), the risk-based map at time $k-1$. In (b), the risk-based map at time k. In (c), the differential risk-based map defined according to Equation (4.10).

The Borderland algorithm is based on Bug algorithms [125] applied to grid graphs and with a generic motion cost. Bug-based algorithms are widely used to solve online path planning in complex and dynamic environments [25, 96].

The main idea of the Borderland algorithm is to detect which portions of the path are involved by changes in the dynamic risk-based map. Thus, the algorithm follows the contour of each involved risk area and circumnavigates it, in order to adapt the path minimizing the combination of risk costs and path length.

The Borderland algorithm aims to repair the path only when it is necessary, i.e., when there are portions of the path involved by areas with increased risk costs. For this reason, a differential grid map $\mathbf{R}_{diff}$ related to the differential search space X_{diff} is

defined

$$\mathbf{R}_{\text{diff}}(x_{\text{n}}(k)) = \begin{cases} 1 & \text{if } \Delta r_{\text{c}}(x_{\text{n}}(k)) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

with

$$\Delta r_{\text{c}}(x_{\text{n}}(k)) = r_{\text{c}}(x_{\text{n}}(k)) - r_{\text{c}}(x_{\text{n}}(k-1)) \tag{4.11}$$

The differential grid map $\mathbf{R}_{\text{diff}}$ has the same dimension and resolution of the grid map $\mathbf{R}$. An example of the grid map at k and $k-1$, and the corresponding differential grid map are illustrated in Figure 4.4.

The Borderland algorithm is described in Algorithm 3. The inputs of the algorithm are: *(i)* the current position of the aircraft at time k, *(ii)* the last path computed at time $k-1$, *(iii)* the grid graph $\mathbf{R}$ at time k, related to the search space X, and *(iv)* the grid map $\mathbf{R}_{\text{diff}}$ at time k related to the differential space X_{diff}. The algorithm adapts the path from the current position of the aircraft because the previous portion of the path is already executed.

First, the algorithm verifies if each element x of the path σ is involved in the differential grid map and, in the affirmative case, it adds x to the differential set D (lines 2 to 5). The differential set contains all the nodes of the path σ involved by the differential grid map $\mathbf{R}_{\text{diff}}$. Then, Borderland detects path segments $S[x_{\text{a}}, x_{\text{b}}]$ as a sequence of adjacent locations in the set D (line 7). Segments denoted with $S[x_{\text{a}}, x_{\text{b}}]$ are the segments of the path that need to be repaired.

If the current position of the aircraft is in the segment $S[x_{\text{a}}, x_{\text{b}}]$, the algorithm searches for an escape location x_{esc} outside the area involved by the differential grid map. Hence, it searches for an alternative segment σ_{seg}, passing through x_{esc} (lines 9 to 11). Figure 4.5a illustrates this scenario.

Otherwise, for each segment $S[x_{\text{a}}, x_{\text{b}}]$, the algorithm seeks for an alternative segment σ_{seg}, by circumnavigating the differential area in $\mathbf{R}_{\text{diff}}$ (line 13). This is the most common scenario, reported in Figure 4.5b.

If the Borderland is not able to circumnavigate the area and the involved differential area has a risk cost lower than 1 (i.e., it is a flyable area), the algorithm *reduces* the differential area in $\mathbf{R}_{\text{diff}}$, until a valid segment σ_{seg} exists (lines 15, 16). This scenario is illustrated in Figure 4.6.

Otherwise, if the involved differential area is a no-flyable area, the algorithm cannot find σ_{seg} in the differential area. Hence, it seeks for an alternative segment in the grid map $\mathbf{R}$, by circumnavigating no-fly zones (lines 17, 18).

If an alternative path σ_{seg} exists, the algorithm compares the motion cost of the new segment with the old one $S[x_{\text{a}}, x_{\text{b}}]$. Then, if the new segment has a greater cost, it discards it (lines from 22 to 25). Figure 4.5c exemplifies this scenario. Otherwise, if a solution does not exist, the algorithm is not able to solve the online path planning problem and reports it (line 27).

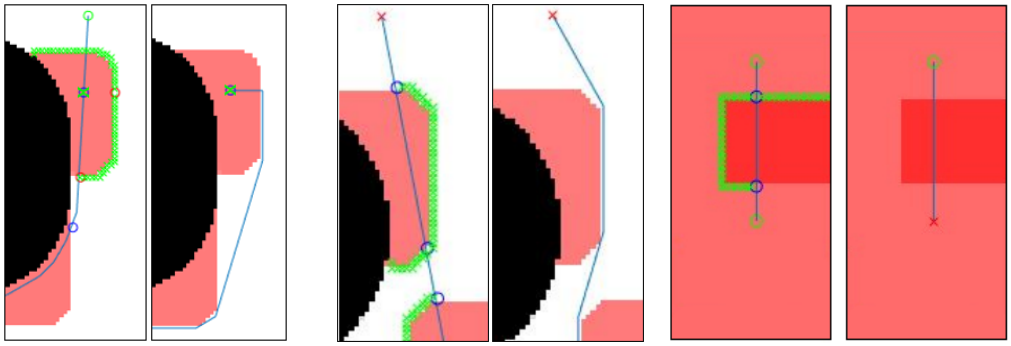
Once all segments are analyzed, the path is updated with new segments σ_{seg} (line 31). Finally, a post-optimization procedure is applied to the new path σ_{new} (line 32),

Algorithm 3 Borderland algorithm

```
1:  procedure BorderlandSearch(x_pos, σ, R, R_diff)
2:      for each x ∈ σ do
3:          if R_diff(x) > 0 then
4:              Add x to D
5:          end if
6:      end for
7:      Detect segments S[x_a, x_b] as a sequence of x ∈ D
8:      for each S[x_a, x_b] do
9:          if x_a ∈ S[x_a, x_b] = x_pos then
10:             Search nearest x_esc with R_diff(x_esc) = 0
11:             Search for σ_seg[x_a, x_b] through x_esc
12:         else
13:             σ_seg = Circumnavigate area(R_diff > 0)
14:             if ∄ σ_seg then
15:                 if r_c(area(R_diff > 0)) < 1 then
16:                     Reduce area(R_diff > 0) until ∃ σ_seg
17:                 else if r_c(area(R_diff > 0)) = 1 then
18:                     Search σ_seg[x_a, x_b] in R
19:                 end if
20:             end if
21:         end if
22:         if ∃ σ_seg then
23:             if cost(σ_seg) > cost(S[x_a, x_b]) then
24:                 Discard σ_seg
25:             end if
26:         else
27:             ∄ solution
28:             return
29:         end if
30:     end for
31:     Reconstruct Path σ_new with σ and every σ_seg
32:     Simplify σ_new
33:     return σ_new
34: end procedure
```

using the procedure of Algorithm 2.

The time complexity of the Borderland algorithm depends on the specific scenario, because the number of explored nodes is function of the updated areas and their dimension. In the ordinary scenario, the path is updated with the Circumnavigate area(·) function. Hence, the complexity is proportional to the distance of the not updated path σ and the perimeter of each updated area with high-risk cost. Otherwise, when the Reduce area(·) function is applied, the complexity increases because, in the worst case, the algorithm explores all nodes of the updated area. If the updated area is not valid, i.e. with a cost equal to 1, the algorithm searches the solution by circumnavigating no-fly zones in the map. As a consequence the complexity depends on the distance of the not updated path and the perimeter of each no-fly area.



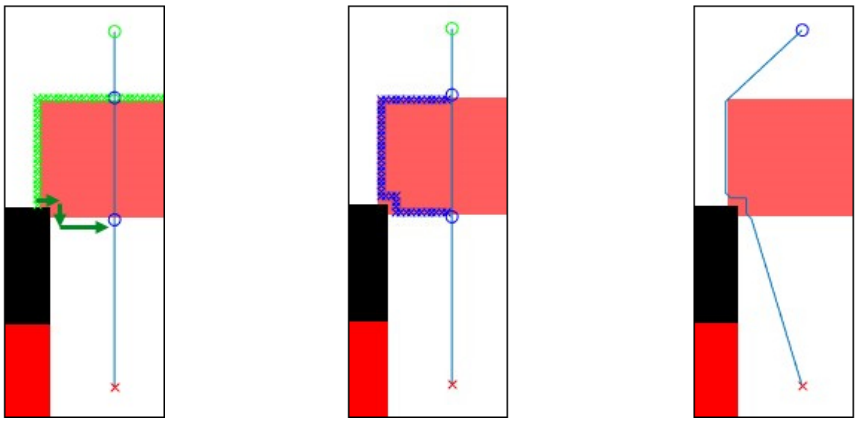
(a) (b) (c)

Figure 4.5: Examples of Borderland scenarios. After the update of the risk-based map, in (a), the current position is in a high-risk area. Thus, an escape route is computed, finding an alternative path with lower cost. In (b), a common scenario, whereby the algorithm circumnavigates the risk area with a path with a lower motion cost. In (c), the algorithm tries to circumnavigate the risk-area. The alternative path has a greater cost than the original one, then, the route doesn't change.

4.3.3 Path Smoothing using Dubins Curves

After the path planning procedure, the resulting path is not suitable to be performed by an aircraft, because of kinetic constraints. Hence, a smoothing procedure is performed, in order to achieve a flyable path. Due to their simplicity and performances, Dubins curves are a suitable solution.

Dubins curves are introduced in [51] and they refer to the shortest path between two poses in the two-dimensional space considering a constant radius curvature. Given the state of the aircraft $q = (q_x, q_y, q_\theta)$ and assuming a constant speed, the differential



(a) (b) (c)

Figure 4.6: Example of the Borderland scenario. After the update of the risk-based map, in (a), the path crosses an area with a high-risk cost. The algorithm searches for an alternative path. As there is no solution, the algorithm searches for the solution in the differential map by *reducing* the involved area, until an alternative path is found (b). In (c), the final solution.

equation of Dubins curves are:

$$\dot{q_x} = cos(q_\theta) \tag{4.12}$$

$$\dot{q_y} = sin(q_\theta) \tag{4.13}$$

$$\dot{q_\theta} = u \tag{4.14}$$

where u is normalized in the range between -1 and 1, considering the maximum curvature of the aircraft. The shortest path between two poses can be expressed as a combination of no more than three motion primitives [51]. Hence, only three values of u are defined $u \in \{-1, 0, 1\}$. The value $u = 0$ describes a straight motion (S), $u = -1$ the right (R) turn, while $u = 1$ the left (L) turn. As a consequence, only six combination of curves exist:

$$\{LRL, RLR, LSL, LSR, RSL, RSR\} \tag{4.15}$$

Often, Dubins curves are used directly in the path planning phase, in which the path is defined considering the curvature radius of the vehicle [78, 121]. Sometimes, this latter approach is preferable, because the path planning directly computes the optimal flyable path. In fact, the smoothing procedure can compromise the optimality of the path. On the contrary, the path planning problem is more complex, increasing the computation time.

Anyway, we perform the smoothing procedure after the path planning phase for two reasons: *(i)* since the resolution of the risk-based map (i.e. the search space) is compared with the curvature radius of the vehicle, the path remains optimal after the

post-smoothing procedure; *(ii)* the smoothing is performed in very short time, suitable for the online path planning phase.

Figure 4.7 illustrates the Path Smoothing procedure using Dubins curves considering different curvature radius.

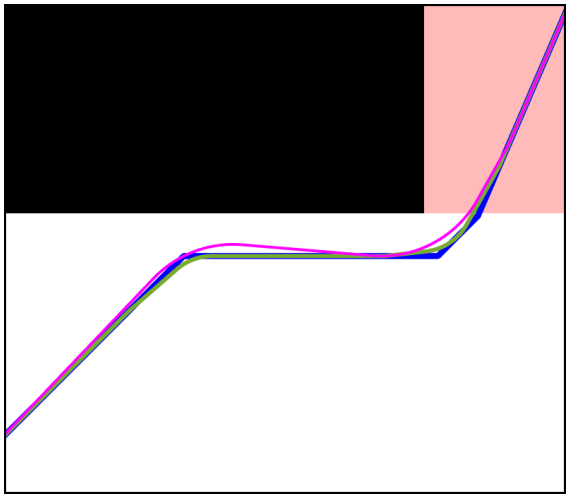


Figure 4.7: Example of the Path Smoothing procedure using Dubins curves. In blue the path before the smoothing procedure. In green the smoothed path with a curvature radius of 10 m, while in magenta the path with a curvature radius of 20 m.

4.3.4 Simulation results

In this section, simulation results are reported and discussed. Simulations are obtained through Matlab using a laptop with a 2-core with 1.9 GHz CPU.

The risk-based map used in simulations is illustrated in Figure 4.8. It corresponds to a Torino's (Italy) neighborhood. The risk-based map has 126×76 cells, in which each element is a square cell with dimensions 5×5 m. The risk-based map is defined by coded colors: white areas are with the minimum risk cost, while black areas are no-flyable because of the presence of obstacles at the flight altitude, no-fly zones or high-risk areas, i.e. with a risk greater than the ELOS threshold. Shade of red areas have middle costs, in which darker areas have higher risk costs than the lighter ones. The risk-based map used in this section is not represented as the risk-based map of Chapter 3. The main reason is a different application used to carry out the simulations. In fact, the risk-based map of Chapter 3 is generated using the ROS framework, while simulations here reported are performed using Matlab. Anyway, the risk-based map can be easily adapted to be used with the riskA* and Borderland algorithms in Matlab by normalizing risk costs in the range from 0 to 1 according to the ELOS requirement.

Both riskA* and Borderland algorithms are implemented as Matlab functions.

(a) (b)

Figure 4.8: Risk-based map related to the Torino's neighborhood. In (a), the urban area from Google Maps. In (b), the realistic risk-based map at 20 m of altitude. Black pixels describe the occupied areas ($\mathbf{r_c} = 1$), while in shade of red areas are with other risk-costs ($0 < \mathbf{r_c} < 1$), where darker red areas have a greater risk-cost than bright red ones.

Table 4.1: Results of riskA* algorithm with different values of k.

Map	k	solve time [s]	path length [m]	cost	average risk-cost
Map 1 126 × 76 cells (500 simulations)	0.0	0.3010	333.7345	107.1005	0.3220
	0.5	0.2623	333.7345	107.1005	0.3220
	0.75	0.2432	333.7345	107.1005	0.3220
	1.0	0.2256	333.7525	107.1010	0.3219
	1.25	0.2066	333.7525	107.1030	0.3220
	1.5	0.1864	333.9410	107.1150	0.3218
	2.0	0.1472	334.2295	107.1945	0.3219
	2.5	0.1126	334.2990	107.5485	0.3227
	3.0	0.0814	333.7855	108.2410	0.3252
Map 2 339 × 131 cells (200 simulations)	0.0	3.3292	769.4265	220.5930	0.2864
	0.5	2.8039	769.4265	220.5930	0.2864
	0.75	2.5165	769.4265	220.5930	0.2864
	1.0	2.3007	769.5300	220.5940	0.2863
	1.25	2.1237	769.5300	220.6045	0.2864
	1.5	1.9884	769.7800	220.6225	0.2864
	2.0	1.6323	771.2785	220.7955	0.2865
	2.5	1.1328	773.3660	222.0745	0.2872
	3.0	0.6551	774.1210	223.5875	0.2888

Regarding the riskA* algorithm, a Monte Carlo simulation is performed in order to determine the best value of the k parameter. In particular, we consider a set of k values and two maps. 500 independent simulations are executed using the risk-based

map of Figure 4.8, while 200 independent simulations with the map of Figure 4.11. With both maps, simulations are randomized with respect to the start and goal positions, obtaining the results reported in Table 4.1. According to the results of Table 4.1, the k parameter has an effect on the time required to compute the solution and on the path characteristics, such as path length, average risk cost and resulting cost of the path. The cost of the path is the motion cost computed according to Equations (4.6) and (4.9). High values of k provide fast solution time at the expense of path optimality, while we have the opposite effect with low values of k. In particular, with $k = 0$, the heuristic function is not evaluated and the resulting behavior is the same as the Dijkstra algorithm. Moreover, this test demonstrates how the computational time increases with the map size.

With both maps, we identify the value of $k = 0.75$ as the best one to provide a good trade off between optimality of the path and computational time.

The riskA* algorithm is compared with the original A* and RA*. RA* is a risk-aware path planning algorithm proposed by Guglieri et al. in [74]. Similarly to riskA*, RA* is based on the well known A* with the minimization of the risk to the population. However, the cost function of RA* uses risk costs as an additive factor. Instead, the original A* optimizes the path length.

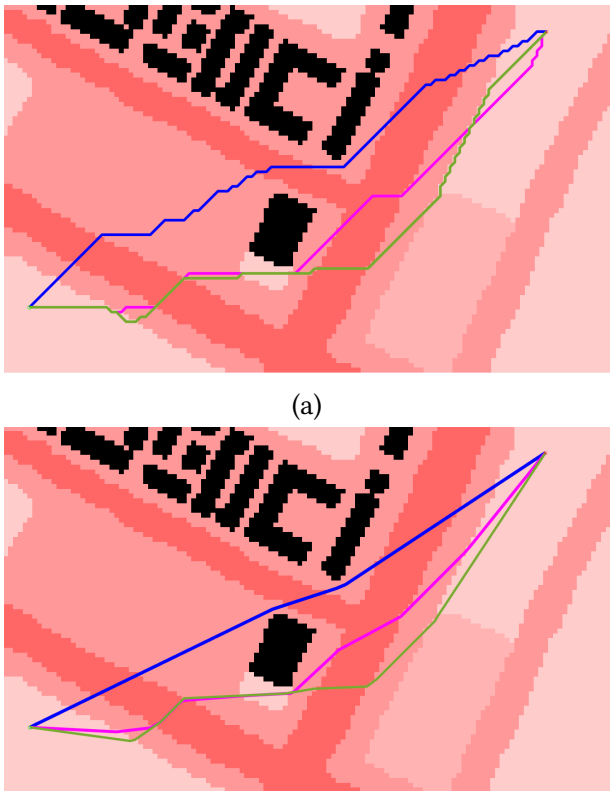
The comparison between these algorithms is illustrated in Figure 4.9, while numerical results are reported in Table 4.2. RiskA* computes a longer path, but with a lower average risk cost, as well as the resulting motion cost.

Table 4.2: Numerical results of the simulation depicted in Figure 4.9. The percentage values refer to the values of the A* algorithm.

Algorithm	**solve time [s]**	**path length [m]**	**cost**	**average risk-cost**
A*	0.5621	653.0510	254.0945	0.3883
PO A*	0.5637	608.5530 (-6.81%)	234.6380 (-7.65%)	0.3834 (-1.26%)
RA*	0.4684	668.9085 (+2.42%)	209.9570 (-17.37%)	0.3070 (-20.94%)
PO RA*	0.4692	649.4940 (-0.54%)	202.9355 (-20.13%)	0.3098 (-20.21%)
riskA*	0.5446	700.6245 (+7.2848)	196.0440 (-22.85%)	0.2790 (-28.14%)
PO riskA*	0.5468	674.4790 (+3.28%)	189.9475 (-25.25%)	0.2781 (-28.38%)

In order to validate the proposed algorithm, a Monte Carlo simulation with 500 independent executions is performed, comparing riskA*, RA* and A*. Table 4.3 reports the numerical results. It is demonstrated how riskA* is able to find an optimal path by trading off path length and risk costs. Compared with A* the path is longer (+4.15%), but the resulting average risk cost is lower (−13.09%). Instead, RA* computes a shorter path, but with higher average risk cost.

Numerical results of Tables 4.2 and 4.3 report also the path characteristics after



(a)

(b)

Figure 4.9: Path planning with A* (in blue), RA* (in magenta) and riskA* (in green). In (a), only the path planning algorithm is executed, while, in (b), the Post-Optimization procedure improves the path.

the post-optimization (PO A*, PO RA* and PO riskA*). Figure 4.9b illustrates the post-optimized paths, compared with the original paths in Figure 4.9a. According to Tables 4.2 and 4.3, the post-optimization procedure reduces both path length and motion cost, maintaining a comparable average risk cost. Moreover, the post-optimization slightly increases the computation time of the solution.

Regarding the online path planning, the Borderland algorithm is implemented. A complete simulation scenario is shown in Figure 4.10, where the Borderland algorithm continuously updates the path according to the dynamic risk-based map and path smoothing with Dubins curves is applied. The re-planning phase is executed twice: in Figure 4.10b the path is still valid, but the Borderland algorithm computes a path with a lower risk (Figure 4.10c); in Figure 4.10d the map changes and the path is invalid, hence, the path is updated in Figure 4.10e. Numerical results are reported in Table 4.4. Figure 4.10 compares also the path updated with the Borderland algorithm with the optimal path obtained with A*. Moreover, Figure 4.10f shows a detail of the smoothed path.

Table 4.3: Comparison of A*, RA* and riskA*. The numerical results are the average values of 500 simulations.

Algorithm	solve time [s]	path length [m]	cost	average risk-cost
A*	0.0554	376.9025	140.5340	0.3705
PO A*	0.0755	356.6225 (-5.38%)	133.1715 (-5.23%)	0.3707 (+0.05%)
RA*	0.1964	381.8215 (+1.31%)	128.1155 (-8.84%)	0.3329 (-10.15%)
PO RA*	0.2066	367.5745 (-2.47%)	123.2700 (-12.28%)	0.3341 (-9.82%)
riskA*	0.2812	392.5540 (+4.15%)	125.5625 (-10.65%)	0.3220 (-13.09%)
PO riskA*	0.2948	375.1550 (-0.46%)	120.0565 (-14.57%)	0.3207 (-13.41%)

Table 4.4: Results of online path planning.

Map ID		solve time [s]	path length [m]	cost	average risk-cost
1	PO riskA*	0.8634	674.4790	189.9475	0.2781
2	previous path		497.3965	165.7020	0.3336
	PO riskA*	0.7902	530.0510	157.9735	0.2986
	Borderland	0.1932 (-75.55%)	506.5655 (-4.43%)	159.3680 (+0.88%)	0.3159 (+5.79%)
3	previous path		336.5660	Invalid	Invalid
	PO riskA*	0.2985	351.5975	98.2045	0.2839
	Borderland	0.1211 (-59.43%)	348.9715 (-0.74%)	98.4210 (+0.22%)	0.2856 (+0.60%)

According to the numerical results of Table 4.4, the Borderland algorithm is able to adapt and update the path, maintaining a safe and valid route. The path is not the optimal one (instead of riskA*), but the resulting path has acceptable characteristics comparable with riskA*. However, the processing time is significantly less than the time required by riskA*.

The main advantage of the Borderland algorithm is the fast adaptation of the path. In fact, Borderland searches for an alternative path exploring the space locally. On the contrary of riskA* that visits the graph globally seeking for the optimal solution. For this reason, the advantage of the Borderland algorithm is more visible with high dimensional maps. To demonstrate it, in Figure 4.11, we execute a simulation with a map of 339 × 131 cells. According to Table 4.5, the Borderland is faster than riskA*. The solve time reported includes the time required to perform the re-planning, as well as the post-optimization and the post smoothing with Dubins curves.

4.4 RiskRRT^X

In this Section, we present the second risk-aware path planning strategy based on the RRT^X algorithm, proposed in [150] by Otte et al..

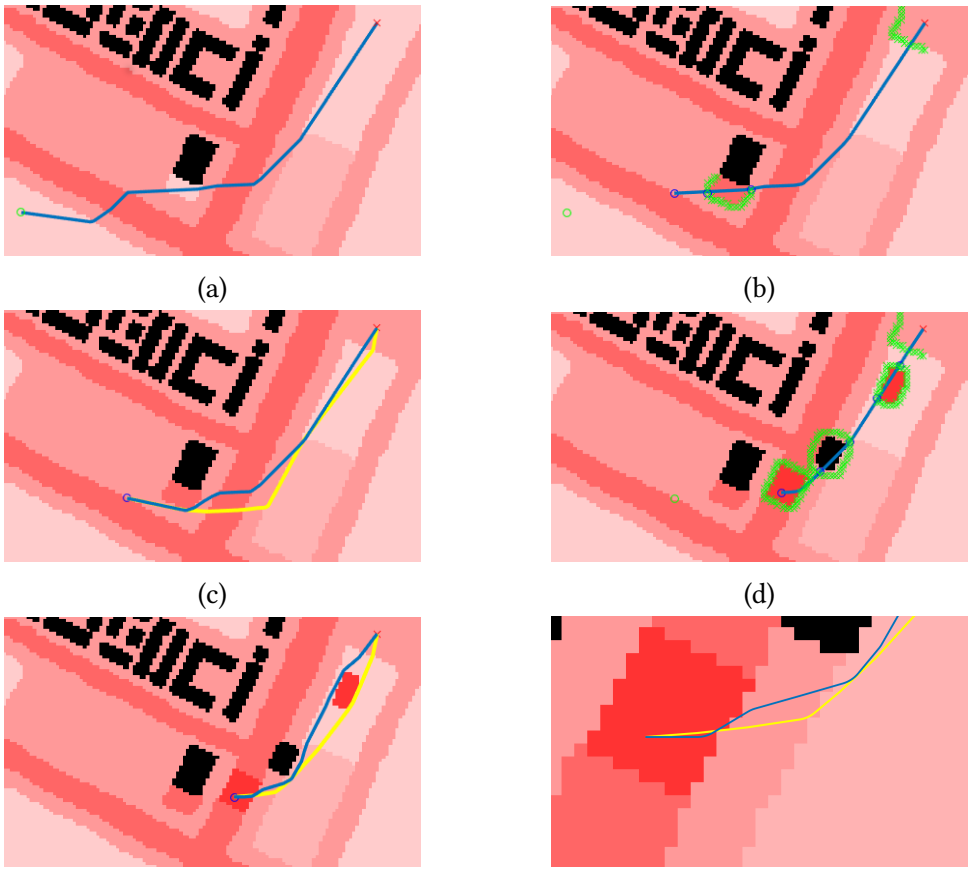
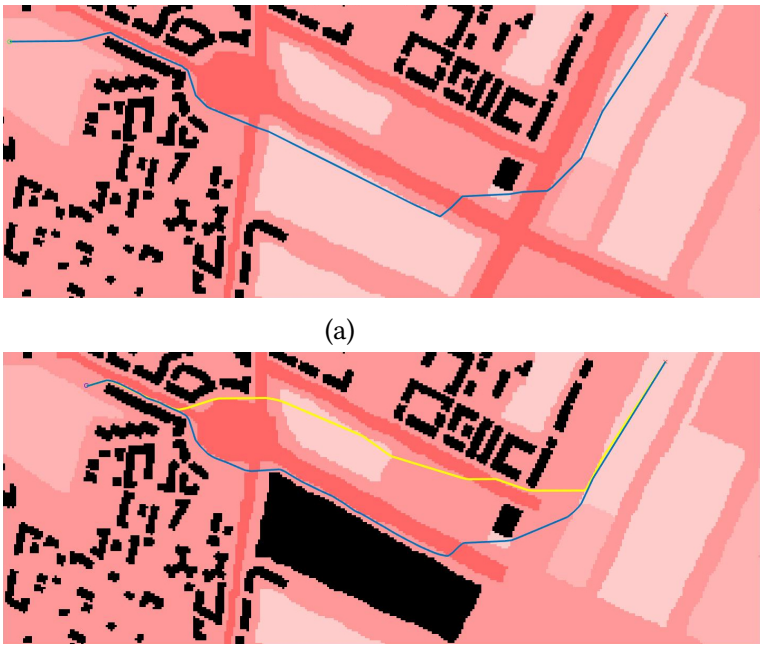


Figure 4.10: Example of the proposed risk-aware path planning approach. In (a), riskA* computes the offline path (in blue). In (b), the risk-based map changes and the Borderland algorithm checks the path exploring cells around the updated area. In (c), the path repaired by the Borderland (in blue) and the path computed from scratch with the riskA* algorithm (in yellow) are compared. Similar behavior in (d) end (e), whereby the risk-based map is updated and the Borderland algorithm is able to adapt the path. In (f) a detail of the path computed, where the path is smoothed with Dubins curves.

Table 4.5: Risk-aware path planning in a high dimension map. The percentage values compare the Borderland with the PO riskA* algorithm.

Map ID		**solve time [s]**	**path length [m]**	**cost**	**average risk-cost**
1	PO riskA*	10.7896	1813.9145	503.5395	0.2760
2	previous path		1644.6530	Invalid	Invalid
	PO riskA*	6.6231	1512.1280	447.9680	0.2966
	Borderland	0.2931 (-95.57%)	1638.6085 (+8.36%)	522.9410 (+16.74%)	0.3174 (+7.01%)



(a)

(b)

Figure 4.11: Simple scenario with a high dimensional map. In (a), the path computed by riskA*. In (b), the path computed with Borderland (in blue) and with riskA* (in yellow) are reported.

First, the riskRRT^X algorithm is described in detail. Hence, simulation results corroborate the presented risk-aware path planning approach.

4.4.1 RiskRRT^X algorithm

RiskRRT^X is a path planning and re-planning algorithm based on the original RRT^X [150].

RRT^X is a sample-based re-planning algorithm suitable to be performed when a priori offline computation is unavailable [150]. In fact, the algorithm proposed in [150] computes an initial path as fast as possible. Hence, the algorithm continues to improve and refine the path that converges to the optimal solution, similarly to anytime RRT* [99]. Moreover, the path planning algorithm adapts the search tree online according to changes in the dynamic environment, i.e. when obstacles are detected.

The main novelty of RRT^X is the re-planning phase, performed when the search space changes. Generally, dynamic RRT-based algorithms cut off and erase branches that are not valid [62], with the so-called pruning phase. Thus, they sample new states in order to compute the updated solution. This happens because standard RRT-based approaches construct an incremental tree to explore the search space [113], where each node v has a unique parent node and some children nodes (Figure 4.12a). After the update of the search space, if a node is not valid, parent and children nodes have no

alternative connections and, as a consequence, it is impossible to immediately repair the graph.

The logic used by RRT^X is simple and illustrated in Figure 4.12b. Similarly to other RRT-based algorithms, a node v has a unique parent node and some children nodes. Moreover, a set of neighbors $N(v)$ is defined, i.e. all nodes of the exploration tree near the node v, but not connected to v. If an edge or node connected with v is not valid after the update of the search space, v analyses all neighbor nodes to repair and update the graph. Moreover, each change in the exploration tree is propagated to all other children nodes, in order to rewire branches according to the updated information. This so-called *rewiring cascade* operation maintains an optimal exploration tree, but it increases the run-time complexity of the algorithm. For this reason, the *rewiring cascade* is performed only when the graph is not ϵ-consistent, i.e. if the update procedure improves the current motion cost with an improvement of ϵ. On the contrary, if the current tree is ϵ-consistent, the updated information is not propagated to children nodes.

Similarly to other dynamic sample-based algorithms, riskRRT^X generates a tree rooted at x_{goal}. In fact, since the vehicle is moving, it is easy to update a graph that is defined starting from a fixed node x_{goal}. Considering x_{bot} as the node relative to the vehicle position, the solution path is the branch connecting x_{goal} and x_{bot}. In fact, even if the node x_{bot} changes, the motion costs defined in the tree don't change, because they are computed starting from x_{goal}.

The riskRRT^X algorithm presented in this section uses the same logic of RRT^X, by using neighbor nodes to update the exploration tree. Anyway, it differs in some concepts.

The riskRRT^X optimizes and updates the exploration tree using general costs. In particular, costs change in the dynamic search space. As a consequence, after the update of the search space, the exploration tree may not be optimal anymore and the graph must be updated. On the contrary, in [150], the environment changes because unpredictable obstacles and RRT^X explores the search space optimizing the path length.

Original RRT^X is an anytime re-planning algorithm, i.e. it continuously samples new states and adds them to the exploration tree. Anyway, if the re-planning is performed for many loops, the exploration tree includes a lot of nodes and, as a consequence, the update and the rewiring cascade procedures may take a lot of time. In fact, the complexity of the algorithm increases with the number of nodes in the graph.
On the contrary, using the logic of the risk-aware path planning approach defined in Section 4.2, riskRRT^X computes the optimal exploration tree in the offline phase. Hence, in the online phase, it repairs and updates the tree without sampling new nodes and without increasing the update time. In fact, the number of nodes evaluated by the algorithm does not change. In particular, unlike traditional sample-based algorithm, the invalid nodes sampled in the obstacle region X_{obs} are not discarded, but they are stored in the invalid set $\boldsymbol{I}$. Set $\boldsymbol{I}$ is evaluated when the search space changes, in order to detect which nodes become valid. On the opposite, if a node becomes invalid in the updated search space, it is added to the invalid set $\boldsymbol{I}$. Moreover, using this technique, nodes are

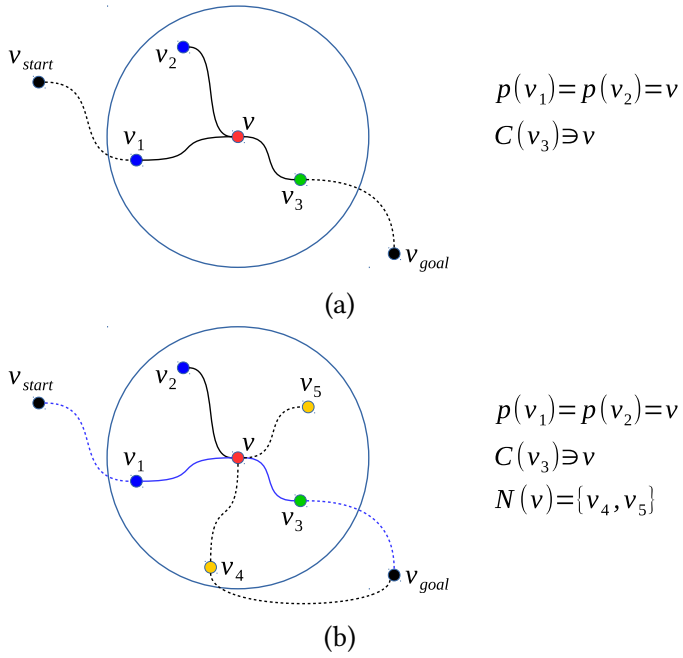


Figure 4.12: In (a), the typical RRT-based tree, where a generic node v is the parent of nodes v_1 and v_2 and it is the child of v_3. In (b), the generic node v with RRT^X, where neighbor nodes v_4 and v_5 are included in the structure. The notation $p(v_i)$ refer to the parent of the node v_i, while $C(v_i)$ the children set of the node v_i. $N(v)$ is the neighbor set of the node v.

uniformly distributed in the search space.

By definition, with RRT-based algorithms, the motion cost in the exploration tree is a "penalty" cost provided by motion from a node x_i to a node x_{i+1}. Optimal RRT-based algorithms use the motion cost to construct an optimal exploration tree. In fact, when a new node is sampled, it is added to the node that provides a lower motion cost.

As discussed in Section 4.2, the proposed riskRRT^X algorithm aims to compute a minimum risk path, considering the risk of the unmanned aircraft to the population on the ground. The risk is quantified by the risk-based map introduced in Chapter 3. Since the risk values are expressed in casualties *per* flight hours (h^{-1}), we use the concept of *time reliance*, i.e. the probability of killing a person is proportional to how long the person is exposed to the risk. Hence, in the proposed risk-aware path planning, the motion cost is computed considering the risk in respect of the flight time

$$C_m(x_i) = C_m(x_{i-1}) + \int_{t_{i-1}}^{t_i} r(x)dt \tag{4.16}$$

with $C_m(x_{i-1})$ is the motion cost at the node x_{i-1}, while the motion cost from x_{i-1} to x_i is the integral of the risk function $r(x)$ on the flight time.

Practically, the risk is defined by the risk-based map **R** that determines the search space. Due to the discrete space, the integral is computed with an approximate and incremental method, similarly to the method used in Equation (4.7) with the riskA* algorithm. Hence, the integral is defined with a trapezoidal area between two adjacent nodes

$$\begin{aligned} C_{\mathrm{m}}(x_{\mathrm{i}}) &= C_{\mathrm{m}}(x_{\mathrm{i}-1}) + c(x_{\mathrm{i}-1}, x_{\mathrm{i}}) \\ &= C_{\mathrm{m}}(x_{\mathrm{i}-1}) + \frac{r(x_{\mathrm{i}-1}) + r(x_{\mathrm{i}})}{2} \mathrm{t}(x_{\mathrm{i}-1}, x_{\mathrm{i}}) \end{aligned} \tag{4.17}$$

with $\mathrm{t}(x_{\mathrm{i}-1}, x_{\mathrm{i}})$ is the flight time expressed in hour to cover the motion from node $x_{\mathrm{i}-1}$ to x_{i}. Figure 4.13 illustrates this concept.

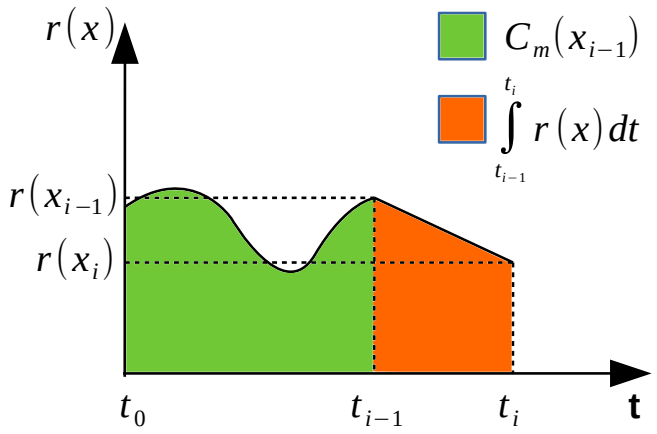


Figure 4.13: Graphical representation of the motion cost used in the riskRRT^X algorithm. The motion cost of the node x_{i} is computed using the motion cost at the parent node $x_{\mathrm{i}-1}$ and the trapezoidal area between the node $x_{\mathrm{i}-1}$ and x_{i}.

Pseudocode

The offline phase of the riskRRT^X algorithm is reported in Algorithm 4. The input of the algorithm are the start x_{start} and goal x_{goal} nodes and the risk-based map **R**. The risk-based map determines the search space X, as well as the obstacle region X_{obs} and the remaining free region X_{free}.

First, the graph G is initialized as an empty graph. G is the graph that describes the exploration tree generated by the algorithm. It consists of vertices and edges. Thus, the goal node x_{goal} is added to the graph as the initial node (line 3).

Then, the algorithm executes the main iterative routine (lines from 4 to 30) that continues until a certain number of nodes n are sampled. First, a random node x_{rand} is sampled (line 5). If it is invalid, i.e. it is in the obstacle region X_{obs} of the search space X, the node is added to the invalid set I and the procedure skips to the next loop (lines from 6 to 8). Otherwise, if the node is valid, it searches for the nearest node x_{nearest} in

Algorithm 4 Offline riskRRT^X algorithm

procedure OFFLINERISKRRTX($x_{\text{start}}, x_{\text{goal}}, \mathbf{R}$)
  Initialize G
  $G \leftarrow x_{\text{goal}}$
  for $i = 1, ..., n$ **do**
    $x_{\text{rand}} \leftarrow$ Sample
    if !Check(x_{rand}) **then**
      $I \leftarrow x_{\text{rand}}$
      continue
    end if
    $x_{\text{nearest}} \leftarrow$ Nearest(G, x_{rand})
    if dist($x_{\text{nearest}}, x_{\text{rand}}$) $> \delta$ **then**
      $x_{\text{new}} \leftarrow$ Saturate($x_{\text{rand}}, x_{\text{nearest}}, \delta$)
    else
      $x_{\text{new}} \leftarrow x_{\text{rand}}$
    end if
    if Check(x_{new}) **then**
      findParent(x_{new})
      getNeighbors($x_{\text{new}}, r_{\text{ball}}$)
      if $x_{\text{new}} \in G$ **then**
        rewireNeighbors(x_{new})
        reduceInconsistency()
      else
        $I \leftarrow x_{\text{new}}$
        $C_{\text{m}}(x_{\text{new}}) \leftarrow infiniteCost$
      end if
    else
      $I \leftarrow x_{\text{new}}$
      $C_{\text{m}}(x_{\text{new}}) \leftarrow infiniteCost$
    end if
  end for
  return G
end procedure

the graph G (line 10). Hence, the new node x_{new} is defined. If the distance from x_{rand} and $x_{nearest}$ is greater than the maximum planner range δ, x_{new} is saturated at the maximum distance δ (line 12), otherwise it corresponds to x_{rand} (line 14).

If the new node x_{new} is still valid, the algorithm searches for a parent node (line 17) and for neighbors nodes within a distance of r_{ball}. Thus, the rewireNeighbors() function verifies if neighbor nodes have a lower motion cost assuming x_{new} as parent (line 20). The reduceInconsistency() function propagates the updated information to all children and neighbors nodes if the sub-branch of the graph in not ϵ-consistency (line 21).

If the state x_{new} is not valid or it is impossible to determine a parent node, x_{new} is added to the invalid set I (lines from 22 to 29).

At the end of the iterative procedure, the algorithm returns an optimal graph G (line 31). Hence, the optimal path is the branch in G from x_{goal} to x_{start} with the minimum motion cost.

For completeness, some sub-routines are reported in Algorithms 5, 6 and 7.

The findParent(x) routine searches for a node within a distance r_{ball} from x that provides the lowest motion cost. Hence, the best node is defined as parent node and x is added to the graph G.

The rewireNeighbors(x) routine verifies if x can be the parent of neighbor nodes. In fact, if a neighbor node may have a lower motion cost passing through x, it updates the parent node. Hence, if the sub-branch after x is not ϵ-consistency, the information is propagated. The verifyQueue(x) function adds x to a priority queue Q, i.e. a set of nodes that need to be visited because are not ϵ-consistent. The rewireNeighbors() function updates the neighbor set, erasing neighbor nodes at a distance greater than r_{ball} (cullNeighbors()).

The reduceInconsistency() routine visits all nodes in the set Q. Q is a priority queue, i.e. nodes with lower costs are served before than others. For each node, the algorithm verifies if a new parent improves the motion cost and propagates the information.

Algorithm 5 The findParent routine

```
1:  procedure FINDPARENT(x)
2:      X_near ← Near(G, x, r_ball)
3:      c_min ← InfiniteCost
4:      for all x_near ∈ X_near do
5:          if CheckMotion(x_near, x) and C_m(x_near) + c(x_near, x) < c_min then
6:              x_min ← x_near
7:              c_min ← C_m(x_near) + c(x_near, x)
8:          end if
9:      end for
10:     x_min ← parent(x)
11:     G ← G ∪ x
12: end procedure
```

Algorithm 6 The rewireNeighbors routine

```
1:  procedure REWIRENEIGHBORS(x)
2:      if C_m^old(x) − C_m(x) > ε then
3:          cullNeighbors(x, r_ball)
4:          for all x_near ∈ N(x) do
5:              if C_m(x_near) > c(x_near, x) + C_m(x) then
6:                  C_m(x_near) ← c(x_near, x) + C_m(x)
7:                  x ← parent(x_near)
8:                  if C_m^old(x_near) − C_m(x_near) > ε then
9:                      verifyQueue(x_near)
10:                 end if
11:             end if
12:         end for
13:     end if
14: end procedure
```

Algorithm 7 The reduceInconsistency routine

```
1: procedure REDUCEINCONSISTENCY
2:     while size(Q) > 0 do
3:         x ← pop(Q)
4:         updateParent(x)
5:         rewireNeighbors(x)
6:     end while
7: end procedure
```

The Algorithm 4 computes the optimal graph (or exploration tree) offline.

When the search space changes because of the dynamic risk-based map, the graph may be not optimal or, in the worst case, may be invalid. Hence, an online phase is executed to update the graph. The online phase of the riskRRT^X algorithm is reported in Algorithm 8.

The inputs of the online phase of riskRRT^X are the current position of the aircraft x_{bot}, the graph G to be updated and the updated risk-based map $\mathbf{R}$.

First, the algorithm updates all costs in the graph G according to the risk-based map $\mathbf{R}$ (line 2). The updateCosts() function updates only costs in the graph, but does not modify connections between nodes.

Algorithm 8 Online riskRRT^X algorithm

```
1:  procedure ONLINERISKRRTX(x_bot, G, R)
2:      G ← updateCosts(G, R)
3:      for all x_i ∈ I do
4:          if Check(x_i) then
5:              I \ x_i
6:              getNeighbors(x_i)
7:              findParent(x_i)
8:              findChildren(x_i)
9:          end if
10:     end for
11:     if x_bot ∉ G then
12:         getNeighbors(x_bot)
13:         findParent(x_bot)
14:     end if
15:     for all x_i ∈ G visited with BFS approach do
16:         updateParent(x_i)
17:         updateChildren(x_i)
18:     end for
19:     I ← update()
20:     return G
21: end procedure
```

Then, all nodes in the invalid set I are visited (lines from 3 to 10). If a node is now valid in the updated search space, it is added to the graph G, searching for a parent, children and neighbors nodes. For this reason, it is important that invalid nodes detected by the updateCosts() are not immediately inserted in I. In fact, the invalid set is updated at the end of the algorithm (line 19).

Similarly, a node corresponding to the current position of the aircraft is inserted in the graph (lines from 11 to 14). Sometimes, this node is already included in the graph,

because sampled nodes cover the search space uniformly and, a state in a region occupied by x_{bot} may exist.

Hence, the algorithm visits all nodes in the graph $\boldsymbol{G}$ using a Breadth First Search (BFS) approach (lines from 15 to 18). For each node, the algorithm updates parent and children nodes evaluating neighbor nodes. Note that in this phase, any information is not propagated to other nodes. In fact, all nodes are already visited by the algorithm and the propagation procedure is redundant.

Finally, the online riskRRT^X returns the updated graph $\boldsymbol{G}$. The optimal path is the optimal branch from x_{goal} to x_{bot}.

The study of the time complexity of a sample-based algorithm is a challenge, because of the heuristic logic of the algorithm. The offline phase of the riskRRT^X has the same run-time complexity as RRT and RRT*. To add a new node to the graph requires an amortized run-time of $\Theta(\log n)$. Hence, considering a graph of size n, the expected time required to build a graph is $\Theta(n \log n)$ [150]. An important characteristic of the RRT^X algorithm is the low information transfer time to inform the graph, essential to face a dynamic environment. In fact, in order to maintain an iteration time of $\mathcal{O}(\log(n))$, each node maintains a set of $\mathcal{O}(\log(n))$ expected neighbors [150].

The online phase is different compared with the offline phase, because the complexity depends on how much the map changes. According to Algorithm 8, the graph is initially updated by visiting the graph with the BFS method, taking $\mathcal{O}(n)$ time. Hence, the graph is visited again by updating the graph edges. Since each node maintains $\mathcal{O}(\log(n))$ neighbors, in the worst scenario, where all nodes should be updated, the algorithm requires $\mathcal{O}(n \log(n))$ time to propagate the updated risk costs [150]. The insertion of a new node in the graph in the online phase (e.g. a node of the Invalid set $\boldsymbol{I}$) requires the same time, $\mathcal{O}(n \log(n))$ [150].

4.4.2 Simulation results

The presented risk-aware path planning strategy with the riskRRT^X algorithm is implemented as an executable process in the ROS (Robot Operating System) framework. In particular, it is implemented in C++ as a solver in the Open Motion Planning Library (OMPL) [193]. OMPL is an open source library specialized in sampling-based motion planning and it consists of many state-of-the-art algorithms.

The algorithm is tested considering three different scenarios: *(i)* a risk-based map is updated by changing risk values, *(ii)* a risk-based map is updated by adding new obstacles and changing risk values, and *(iii)* the risk-based map is updated by removing obstacles and changing risk values.

The risk-based map has the same characteristics of the map introduced in Chapter 3. Anyway, in these tests, the map is represented using different coded colors, i.e. the risk is illustrated with a greyscale, where white areas are with no risk, black areas are no-flight zones and scale of grey areas have a middle risk, in which darker areas have a higher risk value. Figure 4.14 illustrates a risk-based map used in the first scenario. The

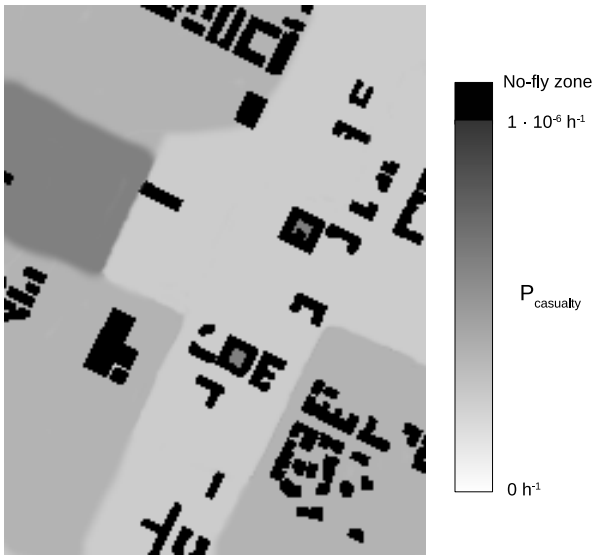


Figure 4.14: Example of risk-based map used in simulations.

risk-based map has dimension of 176×229 pixels and with a resolution of 5m/pixels.

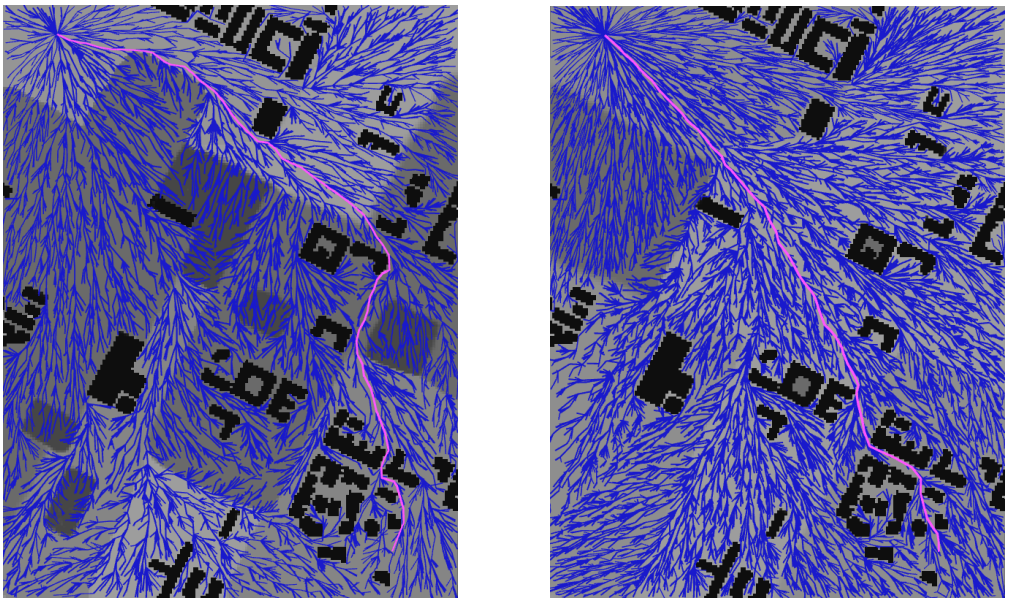
The riskRRT^X algorithm is compared with RRT* [97] and with a dynamic RRT*, based on the dynamic approach proposed in [62]. RRT* is an offline path planning algorithm and computes from scratch a new path every time the map is updated. Dynamic RRT* is an extension of RRT* in dynamic environments.

Numerical results of the three scenarios using RRT*, dynamic RRT* and riskRRT^X are reported in Table 4.6. Results are obtained by performing 10 different tests, considering the same start and target positions in the map. In fact, solutions computed by sample-based algorithms are always different, due to their randomized sampling approach. In Table 4.6 the average values are reported.

The offline path planning is performed with all three algorithms: RRT*, dynamic RRT* and riskRRT^X. In order to compare the execution time and the solution cost, all algorithms search for the optimal path by constructing an exploration tree with 10000 nodes and with a maximum planner range (δ) of 30 m. According to Table 4.6, solution costs of the offline phase are comparable. In fact, both dynamic RRT* and riskRRT^X construct the exploration tree using the same logic of RRT*. As a consequence, considering the same number of nodes in the graph, solutions are similar. On the contrary, the computational time changes. Since both RRT* and dynamic RRT* use the same offline phase, the computational time is similar. Instead, riskRRT^X is more than 6 times slower. In fact, for each sampled node, riskRRT^X determines and evaluates neighbors nodes. Hence, riskRRT^X has more complex rewiring procedures.

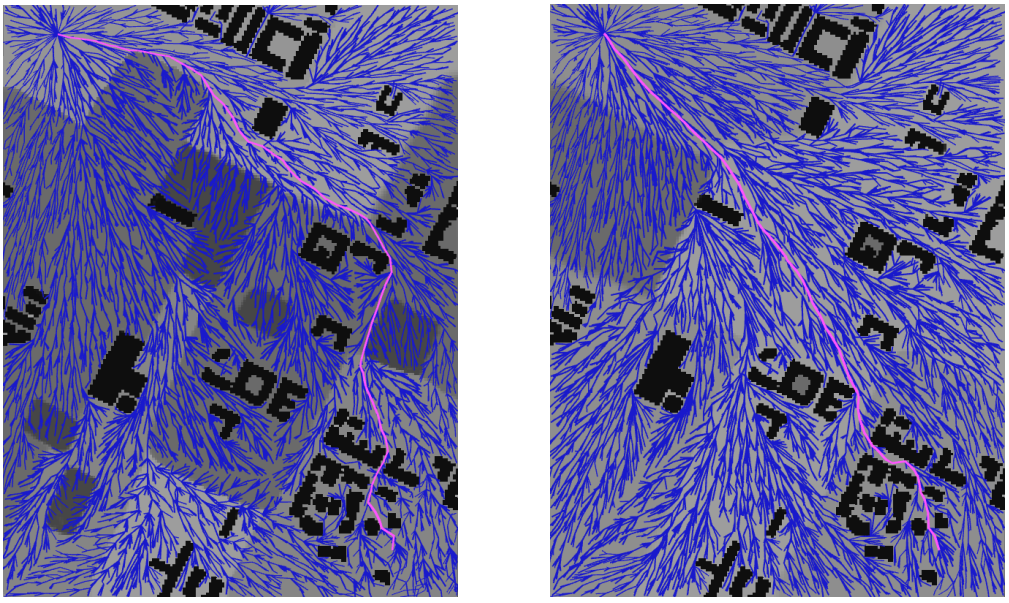
On the opposite, performances in the online phase change with the scenario considered.

In the first scenario, dynamic RRT* computes an optimal solution faster than RRT*,



(a) (b)

Figure 4.15: In (a), the offline computation with the dynamic RRT* algorithm considering the first scenario. In (b), the online one. Note that the exploration tree has more nodes in the online phase.

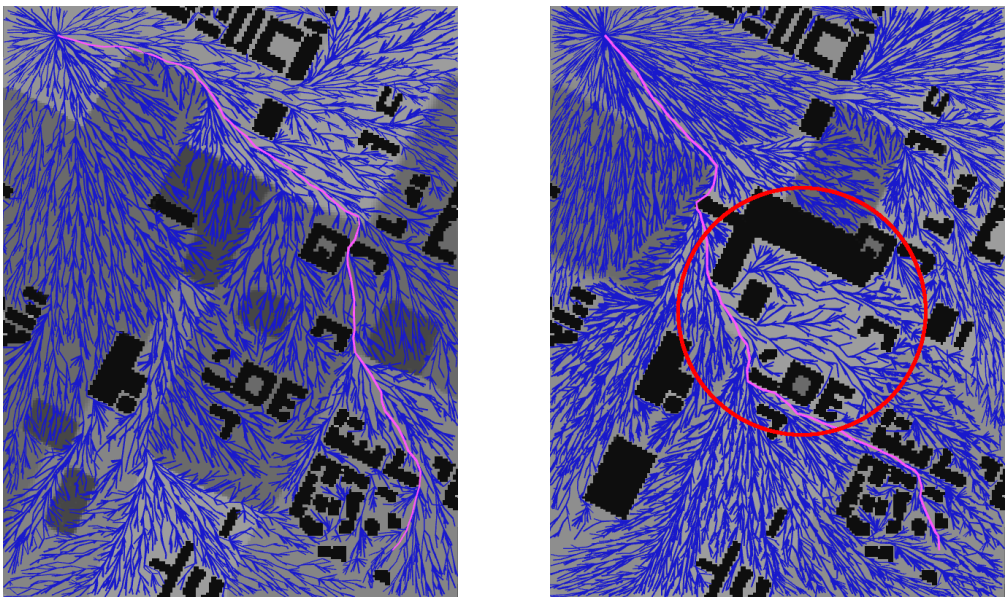


(a) (b)

Figure 4.16: In (a), the offline computation with the riskRRT^X algorithm considering the first scenario. In (b), the online one.

but it needs to sample new nodes. In fact, dynamic RRT* updates the motion costs in the exploration tree without performing any rewiring procedure. Hence, the tree is modified and rewired only when new states are added to the graph. In this test, we stop the dynamic RRT* when the solution cost is comparable with the cost obtained with RRT*. On the contrary, riskRRT^X adapts and updates the current graph according to the updated search space, without sampling additional nodes. The computational time is lower than other approaches with a similar solution cost. Figures 4.15 and 4.16 illustrates the offline and online computation with both dynamic RRT* and riskRRT^X in the first scenario.

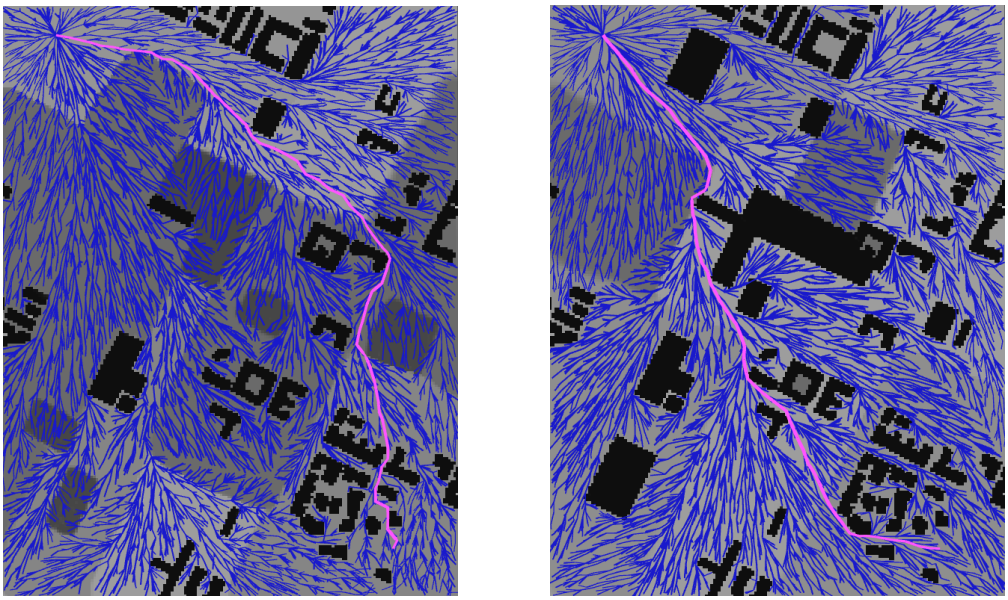
In the second scenario, some obstacles are added to the map. Dynamic RRT* erases some branches and computes the near-optimal path by sampling new nodes. In Figure 4.17 is reported the test with the second scenario with dynamic RRT*. On the Figure, an area is highlighted in which some branches are deleted. Hence, the area is populated with new nodes, but the resulting exploration tree is not uniformly distributed in the map.
On the contrary, riskRRT^X adapts the already existing graph. Due to new obstacles, some invalid nodes are removed from the tree (but included in the invalid set $\boldsymbol{I}$). The exploration tree is adapted according to the updated search space obtaining a near-optimal solution with lower computational time. This test is reported in Figure 4.18.



(a) (b)

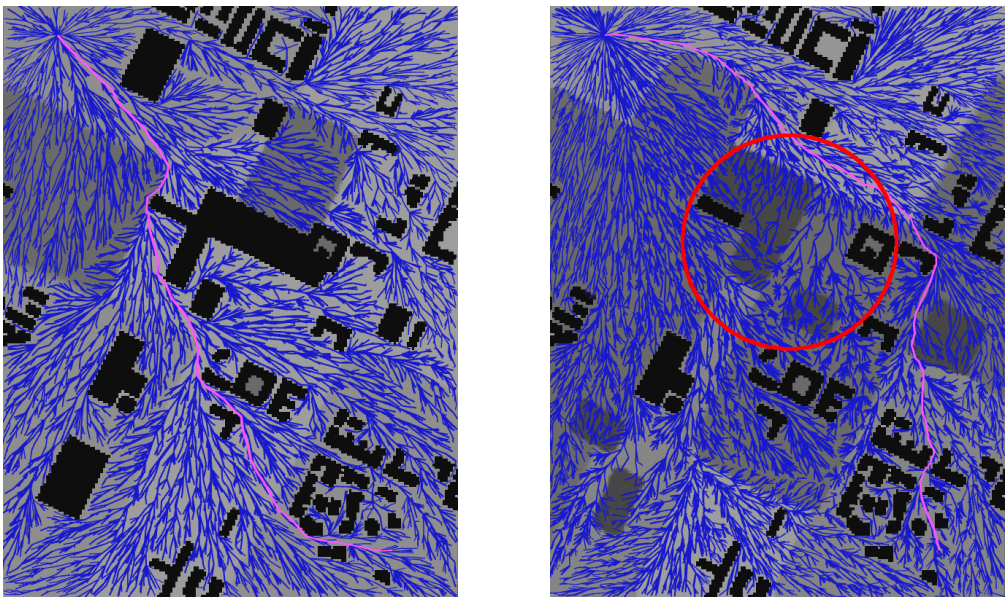
Figure 4.17: In (a), the offline computation with the dynamic RRT* algorithm considering the second scenario. In (b), the online one. The red circle highlights the area where some branches are pruned and new branches are created.

Finally, the third scenario removes some obstacles. Dynamic RRT* samples new



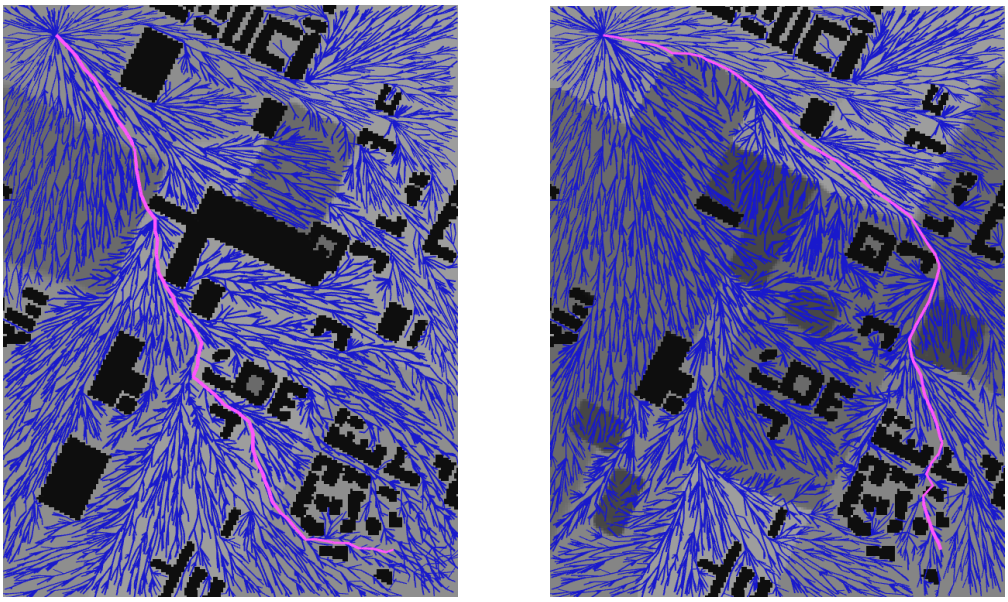
(a) (b)

Figure 4.18: In (a), the offline computation with the riskRRT^X algorithm considering the second scenario. In (b), the online one.



(a) (b)

Figure 4.19: In (a), the offline computation with the dynamic RRT* algorithm considering the third scenario. In (b), the online one. The red circle highlights the area where an obstacle is removed. Hence, the algorithm samples new nodes.



(a) (b)

Figure 4.20: In (a), the offline computation with the riskRRT^X algorithm considering the third scenario. In (b), the online one. The exploration tree is uniformly distributed also where obstacles are removed, because the algorithm evaluates the invalid set $\boldsymbol{I}$.

nodes to explore updated areas. Anyway, the resulting tree is not distributed uniformly in the map, as illustrated in Figure 4.19.
On the opposite, riskRRT^X adds some nodes by evaluating the invalid set $\boldsymbol{I}$. The resulting graph is uniformly distributed in the map, computing a path with lower computational time. This test is reported in Figure 4.20.

These tests demonstrate how riskRRT^X is suitable for re-planning operations in dynamic environments. Compared to original RRT* and dynamic RRT*, the riskRRT^X algorithm is able to update the exploration tree obtaining a path with comparable solution cost computed in less time. Moreover, the exploration tree of riskRRT^X is always uniformly distributed on the map, instead of dynamic RRT*. The use of an invalid set allows invalid sampled nodes to be reused in the re-planning phase.

The proposed riskRRT^X algorithm is suitable to perform both offline and online phases of the risk-aware path planning proposed in this Chapter. The drawback of riskRRT^X is the higher computational time in the offline phase. Anyway, according to the risk-aware path planning strategy, the offline phase is not time constrained because it is performed before the mission starts and, generally, the aircraft is still on the ground.

Table 4.6: Simulation results with RRT*, dynamic RRT* and riskRRT^X algorithms.

Scenario	Algorithm	Phase	Nodes	Computational time (s)	Solution motion cost (h^{-1})
1	RRT*	offline	10000	0.741	$1.233 \cdot 10^{-8}$
		offline	10000	0.738	$0.841 \cdot 10^{-8}$
	dynamic RRT*	offline	10000	0.731	$1.245 \cdot 10^{-8}$
		online	14102	0.649	$0.840 \cdot 10^{-8}$
	riskRRT^X	offline	10000	4.863	$1.255 \cdot 10^{-8}$
		online	10000	0.482	$0.844 \cdot 10^{-8}$
2	RRT*	offline	10000	0.739	$1.237 \cdot 10^{-8}$
		offline	10000	0.810	$0.926 \cdot 10^{-8}$
	dynamic RRT*	offline	10000	0.738	$1.231 \cdot 10^{-8}$
		online	13352	0.701	$0.925 \cdot 10^{-8}$
	riskRRT^X	offline	10000	5.027	$1.258 \cdot 10^{-8}$
		online	8327	0.449	$0.932 \cdot 10^{-8}$
3	RRT*	offline	10000	0.760	$0.913 \cdot 10^{-8}$
		offline	10000	0.742	$1.248 \cdot 10^{-8}$
	dynamic RRT*	offline	10000	0.757	$0.917 \cdot 10^{-8}$
		online	14486	0.716	$1.252 \cdot 10^{-8}$
	riskRRT^X	offline	10000	5.565	$0.944 \cdot 10^{-8}$
		online	10930	0.617	$1.245 \cdot 10^{-8}$

4.5 Discussion

In this Chapter, we present a risk-aware path planning strategy for UASs. It consists of two phases: offline and online path planning.

Offline, a globally optimal path is computed considering a static risk-based map. Since the offline phase is performed before the mission starts, the offline path planning is not time-constrained.

On the contrary, the online path planning updates the path considering the dynamic risk-based map. Unlike the offline phase, the online path planning is time-constrained, because the aircraft is executing the mission and the path needs to be updated as soon as possible.

In order to solve the risk-aware path planning problem, two different strategies are proposed. The first one is based on riskA* and Borderland algorithms. RiskA* searches for the optimal solution minimizing the risk to the population on the ground. Hence, Borderland adapts and repairs the offline path.

The second strategy relies on riskRRT^X that performs both offline and online phases. In fact, offline, it generates a near-optimal exploration tree in the risk-based map. Hence, online, it updates the tree according to the dynamic risk-based map.

According to preliminary results reported in this Chapter, both approaches are suitable to perform a risk-aware path planning for unmanned aircraft. The riskA* algorithm computes an optimal path, while, by definition riskRRT^X computes a near-optimal solution. Anyway, if the search space is properly explored, riskRRT^X provides a solution very closed to the optimal one. Moreover, riskRRT^X is suitable to explore high dimensional maps. On the opposite, the computational time of riskA* increases exponentially with the dimension of the map.

Regarding the online phase, Borderland uses a *check and repair* approach, by identifying the portions of the path involved by the updated search space. Borderland repairs the path rapidly, but the optimal solution is not guaranteed. On the contrary, riskRRT^X maintains a near-optimal exploration tree, providing a near-optimal path also in the replanning phase. Moreover, with riskRRT^X, the computation time required for the online phase does not depend on how many parts of the path are involved in the updated risk-based map. On the contrary, the time requested by Borderland is proportional to the number of nodes of the path to be repaired.

The proposed approaches are not directly comparable, because they are implemented on two different frameworks. RiskA* and Borderland are implemented and simulated using Matlab, while riskRRT^X is implemented in C++ as an executable process in the ROS framework.

To conclude, both approaches are efficient, but performances depend on the characteristics of the environment, such as the dimension of the risk-based map and how much the risk-based map changes with each update.

Chapter 5

Preliminary simulation of a UAS flight operation in urban area

In this Chapter, a preliminary simulation of a flight mission in an urban environment is described.

Using the risk-based map described in Chapter 3, a safe flight mission is planned with the riskRRT^{X} algorithm introduced in Chapter 4.

Simulation results corroborate our approach for safe flight operations in urban areas.

The Chapter is organized as follows. Section 5.1 describes the simulation environment, as well as its implementation. Simulation results are reported in Section 5.2, while we discuss it in Section 5.3.

5.1 Simulation environment

In this section, the simulation environment used in this Chapter is described in detail.

The simulation is performed using the Robot Operating System (ROS) and the Gazebo Simulator. ROS is an open-source meta-operating system for robots introduced in [167]. In particular, ROS is a general purpose robotics library for robot applications. ROS is very popular in robotics and, practically, is a standard for robot programming.

Gazebo Simulator is an open-source multi-robot simulator compatible with ROS. Gazebo is introduced in [105] and it is able to simulate robots, sensors and three-dimensional rigid body dynamics.

In particular, the simulation is based on the project proposed in the PX4 flight stack [164]. PX4 [163] is an open-source flight control software for drones and other unmanned vehicles. At present, PX4 is used on many unmanned aircraft and provides an efficient flight stack.

PX4 can be used with the Gazebo Simulator using the SITL (Software In The Loop)

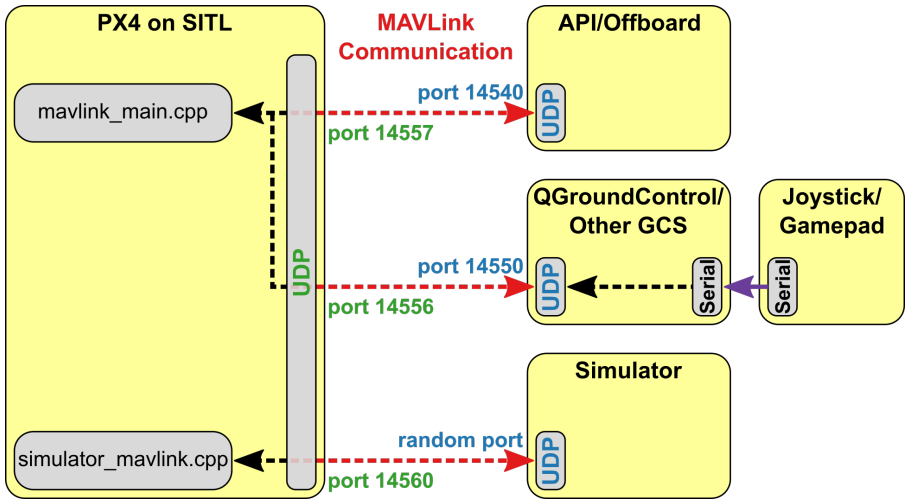


Figure 5.1: The architecture of the simulation based on SITL and the PX4 autopilot. From [164].

framework. SITL allows PX4 to be executed without using any hardware. According to the architecture explained in Figure 5.1, the PX4 is executed on SITL and communicates with the simulator using the MAVLink protocol via UDP connection. Practically, instead of exchange data with the real drone platform, the PX4 Autopilot controls the unmanned aircraft simulated in the Gazebo Simulator, which executes control commands and provides sensor data from simulated sensors. Similarly to the real scenario, a Ground Control Station (GCS) is used to interact with the drone.

Instead of control the unmanned aircraft using the GCS, we control the vehicle offboard using ROS and *mavros*. In particular, *mavros* is a ROS node that enables the communication between ROS and the PX4 autopilot via MAVLink. Mavros translates information from the autopilot (e.g. telemetry, sensor data) as ROS messages. On the contrary, it translates commands from the ROS environment in MAVLink messages to interact with the PX4 autopilot.

Figure 5.1 reports the simulation architecture, where the PX4 is executed on SITL and exchanges data with a simulator (Gazebo), GCSs and an offboard control (ROS with mavros).

In the simulation, we include the three dimensional model of the city center of Turin (Italy), in order to simulate a flight operation in an urban environment. Figure 5.2 shows the 3DR Iris+ aircraft in the simulated environment in Gazebo.

5.2 Simulation results

The simulation is performed considering a portion of the city center of Turin and using the simulation environment described in the previous section.

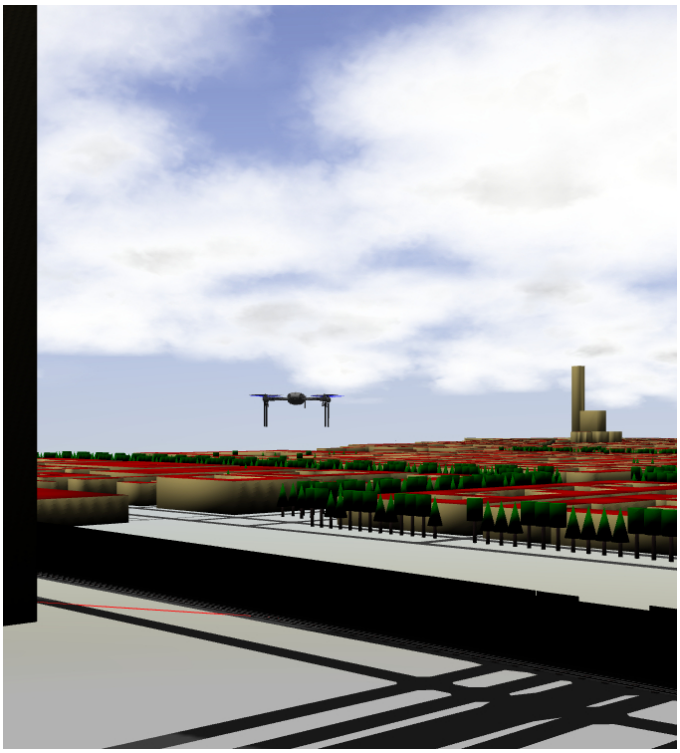


Figure 5.2: The Iris+ aircraft in the simulated environment with the three-dimensional model of the city of Turin (Italy).

First, the risk-based map is generated considering an urban area and following the procedure introduced in Chapter 3. In this simulation, we use the ADPM Evo aircraft, a small UAS suitable to fly in urban areas. In fact, according to results reported in Chapter 3, only small UASs are suitable to fly in areas with high population density, satisfying the ELOS threshold of $1 \cdot 10^{-6} h^{-1}$.

The risk-based map is computed using the same parameters used to compute the risk-based map of the Evo aircraft in Section 3.4 with a flight altitude of 50 m. Figure 5.3 reports the area used to execute the simulation, while the resulting risk-based map is illustrated in Figure 5.4a.

Given the actual position of the UAS and the target position defined in the risk-based map, the minimum risk path is computed using the riskRRT^{X} algorithm introduced in Chapter 4. The resulting path is reported in Figure 5.4a, the path length is about 2017 m with an average risk of $9.93 \cdot 10^{-7} h^{-1}$, lower than the ELOS threshold. The evolution of the risk along the path is reported in Figure 5.5a. The risk-aware path planning returns a path as a sequence of positions in the risk-based map. In order to execute the path, it is converted in a list of waypoints of GPS coordinates.

Using the minimum risk path, the UAS executes the flight mission. Autonomously, the UAS takes off at the flight altitude of 50 m and follows each waypoint sequentially.

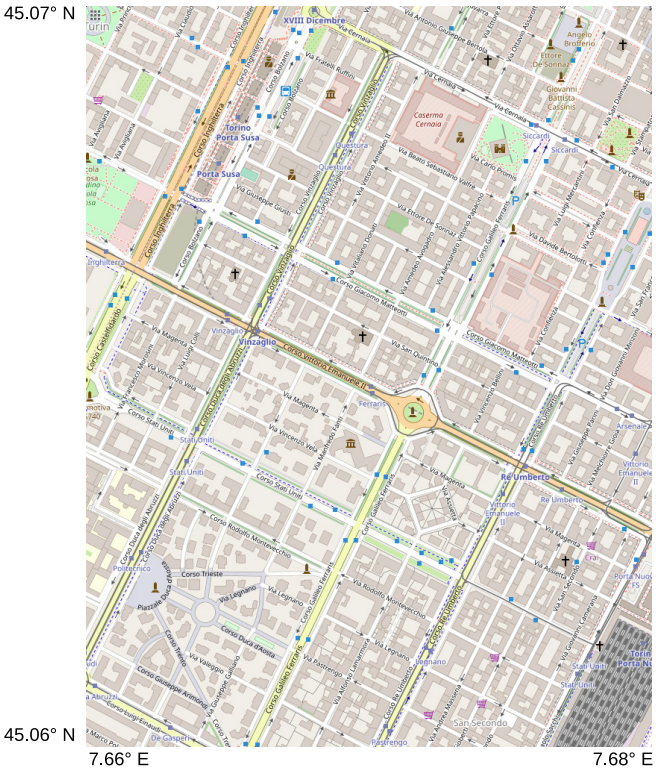


Figure 5.3: The portion of the city center of Turin (Italy) used to perform the simulation.

During the execution of the path, we update the risk-based map by updating the population density layer. Considering the updated risk-based map, the path computed with the previous risk-based map is not optimal anymore, because it crosses areas with high risk. In fact, as illustrated in Figure 5.5b, the portion of path not already executed has an average risk of $1.64 \cdot 10^{-6} h^{-1}$ and a path length of 1495 m.

Hence, using riskRRT$^{\text{X}}$, the optimal path is computed by updating the exploration tree as described in Section 4.4. The path is updated in 0.614 s, the average risk is reduced to $9.91 \cdot 10^{-7} \text{h}^{-1}$ and the path length is 1401 m. Figure 5.4b shows the updated path, while the evolution of the risk is reported in Figure 5.5b, compared with the risk of the path computed with the previous risk-based map.

The flight mission is updated and the UAS executes the updated path. Figure 5.6 shows the execution of the path on the Ground Control Station.

When the UAS reaches the target position, it lands vertically.

5.3 Discussion

In this Chapter, we discuss simulation results demonstrating how our approach can be used to plan and execute a safe flight mission.

In fact, the risk-based map quantifies the risk of flying over urban areas, determining no-fly zones and high-risk areas.

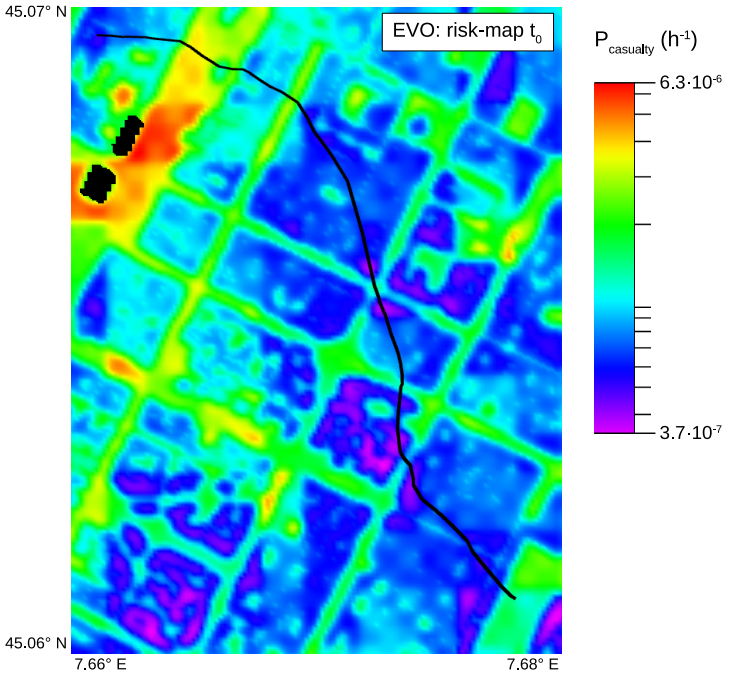
As already discussed in previous Chapters, the risk-based map can be used to plan a safe flight mission by using a risk-aware path planning. In this simulation, the riskRRT^X algorithm is used to compute an optimal path, minimizing the risk to the population and flight time.

Since the proposed risk-aware path planning consists of two phases, the path is updated during the flight operation according to dynamic changes in the risk-based map.

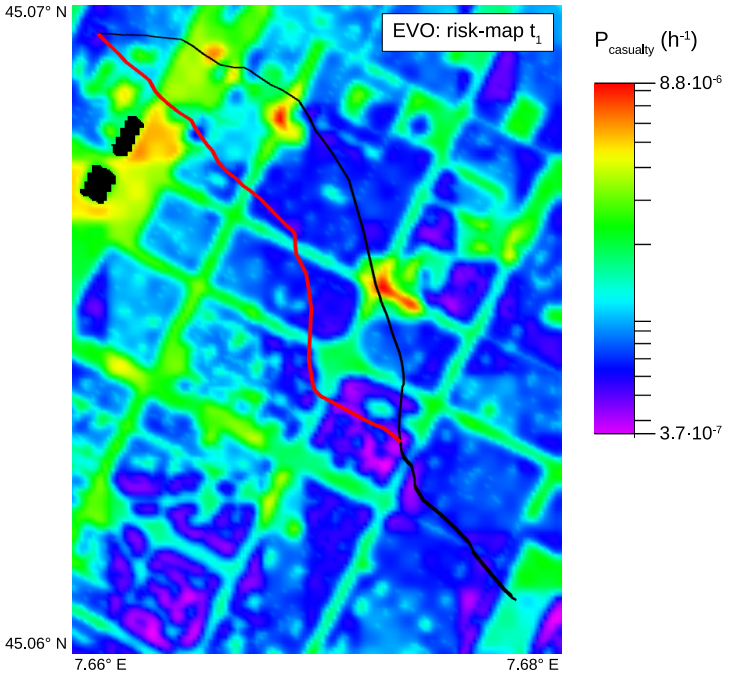
In the particular scenario of the simulation, after the update of the risk-based map, the path is no longer optimal. The average risk is greater than the ELOS threshold because the path crosses high-risk areas. The riskRRT^X is able to rapidly update the path obtaining a sub-optimal path with an acceptable average risk. In fact, as discussed in Chapter 4, riskRRT^X converges to the optimal solution with infinite samples and, as a consequence, with a finite number of sampled state, it computes a near-optimal path. Anyway, with an adequate number of states in the exploration tree, the solution is very close to the optimal one.

The path is executed using a simulated unmanned aircraft with the PX4 autopilot. The algorithm is able to update the flight mission and, as a consequence, the UAS follows the minimum risk path satisfying the ELOS threshold imposed by the National Aviation Authorities.

Thanks to the simulation environment, the same algorithm can be used with a real drone, since the PX4 algorithm is the same one executed on real UASs. Future works will include experimental tests with UASs in real urban areas.

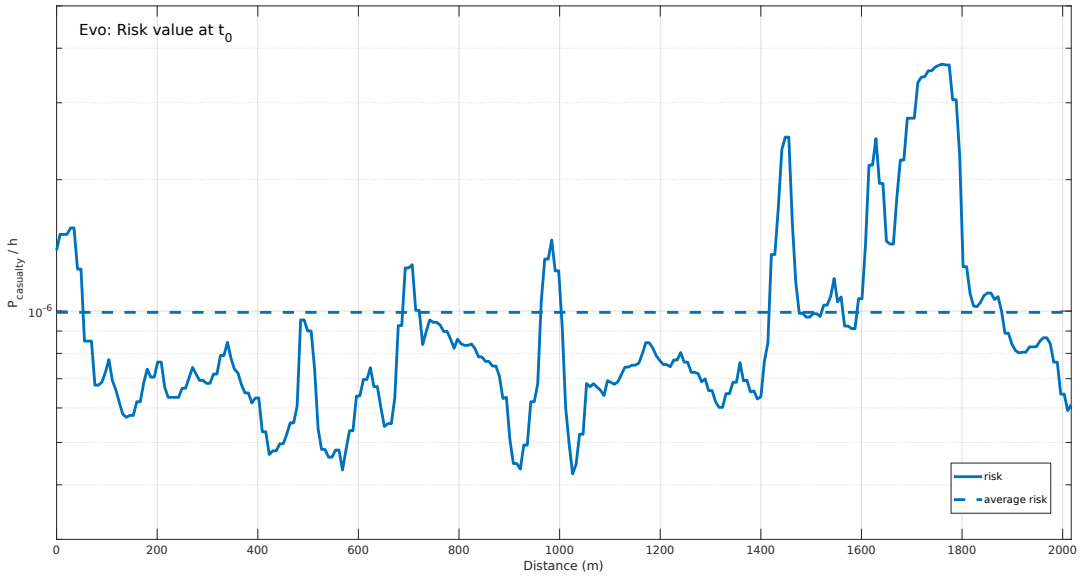


(a)

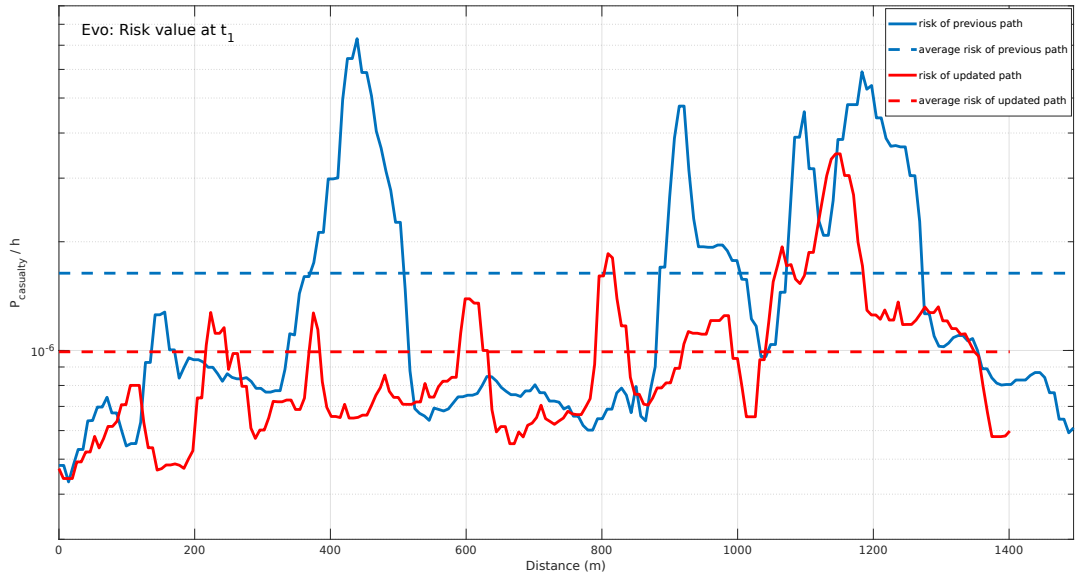


(b)

Figure 5.4: In (a), the risk-based map at time t_0 with the minimum risk path computed offline. In (b), the updated risk-based map at time t_1. On the map both offline (in black) and online (in red) paths are reported.



(a)



(b)

Figure 5.5: The evolution of the risk along the path of Figure 5.4. In (a), the path at time t_0. In (b), the path at time t_1: in blue the portion of path computed with the previous risk-based map, in red the updated path.

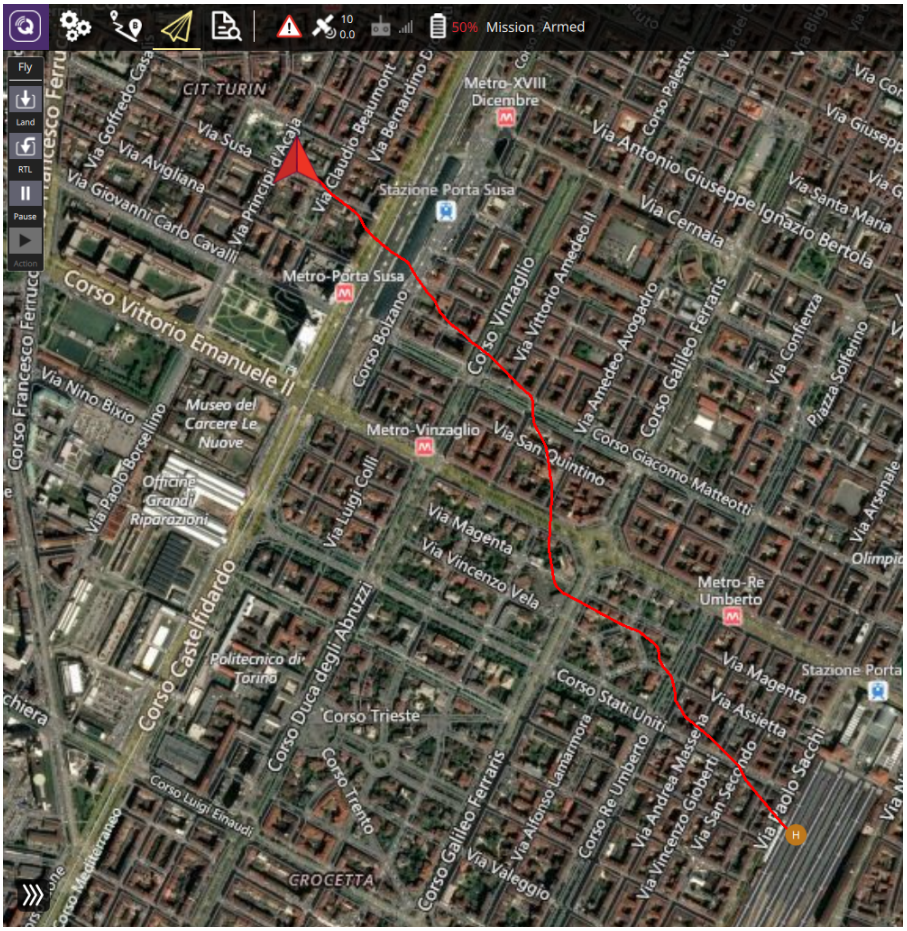


Figure 5.6: Screen of the Ground Control Station with the execution of the flight mission reported in Figure 5.4.

Part II

Autonomous navigation for Ground Robots in crowded environments

Chapter 6

Cloud-based architecture for Service Robotics Applications

In this Chapter, we introduce a Cloud-based architecture for service robotics applications. It is a general architecture used to implement some service robotics applications, such as the Virgil robot and the Courier robot, both described in Chapter 9.

This architecture is distributed between the Cloud and the mobile robot and includes all functionalities required to implement a service robotics application, such as the service management and autonomous navigation.

This Chapter is organized as follows. In Section 6.1 some background information are exposed, while in Section 6.2 the Cloud-based architecture is presented, describing each layer and module.

6.1 Background

Service Robotics is an emergent field in robotics due to the large diffusion of robotic systems in our daily life. In general, *service robots perform useful tasks for humans or equipment excluding industrial applications* [49].

Nowadays, many service robotics applications have already developed [50]. In particular, mobile robot platforms are the most popular, because they are able to execute tasks autonomously, performing autonomous navigation, monitoring and surveillance. One of the first service robotics applications is presented in [58], where a robot courier is used in a hospital. In [184], a service robot distributes fliers in a shopping mall interacting with people. In [200], a telepresence service robot is used to interact with people with special needs, while, in [111], a robot is used to provide English tutoring. An interesting project is proposed in [199], where a socially aware service robot interacts with passengers in a busy airport.

Recently, the newest mobile technologies open new opportunities to service robotics.

Using a mobile network, the robot can be connected with the Internet without any required infrastructure. As a consequence, the concept of Cloud Robotics can be used to support service robotics applications. As already explained in Chapter 1, Cloud technologies offer a lot of advantages, such as Cloud Computing, Cloud Storage, Big Data and Collective learning [102]. As a consequence, many Cloud-based Platform as a Service (PaaS) are proposed in literature [12, 29]. One of the most popular is RoboEarth [206], a Cloud Robotics framework based on Rapyuta [85].

In this Chapter, we propose a Cloud-based architecture to offer service robotics applications. The proposed framework is distributed between robot and Cloud, where the robot is a simple agent connected using a mobile network, such as 4G and, in the near future, 5G.

Thanks to Cloud Computing, many functionalities are provided by the Cloud. In fact, some autonomous navigation tasks are computed on-Cloud, as well as the management of the whole service and applications.

On the robot, only tasks that provide safety are executed. In fact, if a disconnection with the Cloud occurs, safety must be guaranteed by using on-board capabilities.

This Cloud-based architecture is already used to provide service robotics applications. In [179], a service robot is used to monitor a data center autonomously. In another project [145], a Cloud-based architecture is used to offer a museum experience to mobility-impaired people.

6.2 Cloud-based Architecture

In this section, a Cloud-based architecture to provide service robotics applications is described. The architecture is depicted in Figure 6.1. Three conceptual layers are defined:

- **Application Layer**: manages the robotics service and provides applications to end users;
- **Navigation Layer**: performs the autonomous navigation of the mobile robot, providing all the required capabilities for safe navigation;
- **Hardware Layer**: includes sensors, actuators and drivers on the mobile robot.

The three layers are distributed between the Cloud and the robot. In particular, the Application layer is on the Cloud, the Hardware layer is on the robot, while the Navigation layer is split between the robot and the Cloud, based on the service type and requirements. For instance, while the path planner performed by the *global planner* is generally executed on the Cloud, the motion controller, executed by the *local planner* can be located on the robot, because the network latency may compromise the performances. More details about each layer are described in next sections.

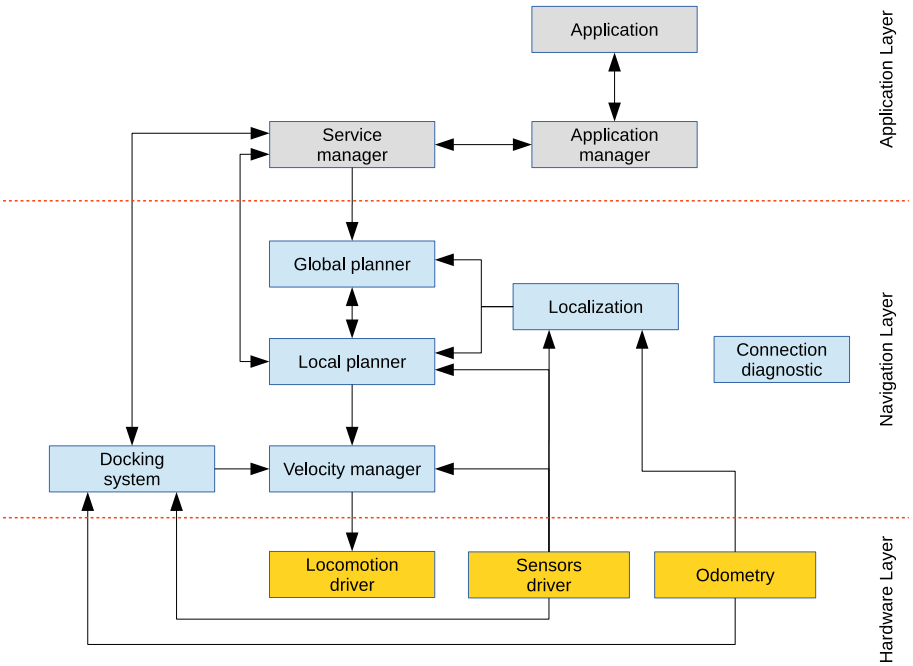


Figure 6.1: The main architecture of the proposed Cloud-based framework for service robotics applications.

6.2.1 Application layer

The application layer aims to manage the service robotics application. It is composed of three elements: the Service Manager, the Application Manager and the Application.

The Service Manager is the core of the application layer and aims to manage the correct functionality of the service. Even if in this thesis we refer to a single mobile robot, thanks to the scalability of the Cloud, the Application layer is able to work with a fleet of robots.

In order to provide the service, the Service Manager monitors the state of the robot and manages it by sending commands to the navigation layer. Simultaneously, it interacts with the Application Manager, in order to manage the Application. The Service Manager has an essential role, because it orchestrates the mobile robot and the application, in order to provide a good service, as well as to manage the interaction between users and the mobile robot.

Based on the status of the Service Manager, the Application Manager exchanges data with the Application. In fact, the Application needs to be updated based on the status of the service, interacting with users. On the opposite, all commands defined by users on the Application must be forwarded to the Service Manager.

The Application comprises all applications used by Service, such as mobile applications, web applications and any other type of applications that enable a Graphical User Interface (GUI).

Generally, the communication between the Application Manager and the Application is provided by an Internet connection on a mobile network.

The Application layer has no strictly timing requirements. Anyway, in order to provide a good service, it needs to monitor the state of the robot with a fixed frequency and to react promptly to commands from the mobile robot and the applications. In fact, a proper response time of the service in respect to commands from users is one of the most important feature to have a good user experience.

6.2.2 Navigation layer

The Navigation layer provides the autonomous navigation in order to reach the desired position, requested by the Service Manager. The architecture of the Navigation layer is based on the Navigation stack proposed by ROS [172]. The Navigation stack provides fully autonomous navigation for mobile robots, using several ROS packages. The core of the stack is the *move_base* package that implements a two-stage path planning procedure: a *global planner* and a *local planner*.

Given the current position of the robot and a map of the navigation environment, the global planner seeks for a global path to reach a target pose, avoiding obstacles marked on the map. Thus, the local planner executes the global path, providing the robot motion and avoiding unexpected obstacles with the support of on-board sensors. In particular, the local planner determines the velocity commands handed over the Velocity Manager. Generally, the global planner is not time constrained because the global path is computed before the robot starts the navigation task. Anyway, in a service robotics application, the user is waiting for the motion of the robot. As a consequence, the global planner should be executed in a reasonable time. On the contrary, the local planner requires a fixed frequency to control the mobile robot.

The robot position is provided by the localization that estimates the robot pose in a known map, based on on-board sensors. The localization block has a crucial role in the autonomous navigation, since it defines the robot's pose used by the path planning and the motion controller.

On the ROS repository, some state of the art algorithms are implemented for each block of the Navigation stack. On the contrary, the Docking system, the Velocity Manager and the Connection Diagnostic are not provided by the standard Navigation stack.

The Docking system controls the robot motion during the docking phase. Using on-board sensors, the docking system detects the docking station. Hence, it controls the robot motion to dock the mobile robot platform. The docking station recharges the battery of the robot.

The Velocity Manager filters velocity commands to the mobile robot. In our architecture, there are more than one source of velocity commands, i.e., the local planner, the docking system and, the teleoperation node, used only to manually move the robot. The Velocity Manager, based on the state of the robot, allows the proper velocity commands to be published to the locomotion driver of the mobile robot. Moreover, the Velocity

Manager provides to stop the robot for safety reasons. For instance, if an obstacle detected with on-board sensors is to close to the robot, the Velocity Manager immediately stops the robot.

The Connection Diagnostic block verifies the quality of the connection between the Cloud and the robot, provided by an Internet connection. The connection is monitored from both sides: on the Cloud and on-board the robot. The Connection Diagnostic detects disconnections and monitors the delay in communication. When disconnection occurs or there is a large delay, the Connection Diagnostic notifies the Service Manager that manages the service. For instance, the robot can be stopped, until the connection is resumed.

The Navigation layer is distributed between the Cloud and the robot. In particular, blocks that perform the control of the mobile robot and provide some safety procedures are generally located on the robot, in order to be robust to the quality of signals and disconnections.

6.2.3 Hardware Layer

The Hardware layer represents the mobile robot platform including both hardware and software, required to manage actuators and sensors. All blocks in the Hardware layer interface the hardware with the software.

The Locomotion driver sends velocity commands to wheel motors, while on-board sensors provide data to the Navigation layer.

The odometry block estimates the robot motion thanks to wheel encoders and, generally, with the support of the IMU (Inertial Motion Unit) sensor.

6.3 Discussion

In this Chapter, a Cloud-based framework for service robotics applications is presented. The aim is to define a reference framework to be used to offer service robotics applications with mobile robots.

The architecture comprises three layers distributed between the Cloud and the robot. The Application layer aims to manage the service and the application used to interact with end users. The Navigation Manager provides the autonomous navigation of the robot. In order to provide safe navigation, some modules of the Navigation layer reside on-board the robot. Moreover, the framework is able to manage disconnections. Finally, the hardware layer manages the hardware of the robotics platform.

The robot is connected with the Cloud using a mobile network. Hence, additional infrastructures are not required in the operational environment.

Thanks to Cloud technologies, most of the intelligence resides on the Cloud, while the robot becomes a simple agent. As a consequence, even if the robot has limited resources on-board, it is supported by the Cloud and its advantages.

The proposed Cloud-based architecture is already used to provide some service robotics applications. In particular, two examples are detailed in Chapter 9.

Chapter 7

Dynamic trajectory planning in crowded environments

This Chapter describes a dynamic trajectory planning for mobile robot navigation in crowded environments.

The robot navigation in crowded areas is a challenge because people are moving and continuously obstruct the robot motion. In order to solve this problem, a dynamic path planning based on the Informed Optimal Rapidly-exploring Random Tree (Informed-RRT*) is introduced, able to continuously repair and update the path, according to the dynamic environment.

The dynamic path planning approach described in this Chapter is presented for the first time in [159].

This Chapter is organized as follows. In Section 7.1 some background information and a literature review are reported. The Informed-RRT* algorithm is described in Section 7.2, while the path planning strategy proposed in this Chapter is presented in Section 7.3. Section 7.5 reports experimental results. We discuss the proposed dynamic path planning approach in Section 7.6.

7.1 Background

The presence of mobile robots in our life is growing. In particular, service robotics aims at developing robotics applications operating in human environments. Anyway, it is not easy for mobile robots to navigate in human environments, because of the presence of obstacles and people that continuously obstruct the robot motion. Moreover, when a mobile robot moves among people, safe and reliable navigation needs to be performed.

Autonomous navigation has an essential role in Service Robotics because it allows any position to be reached, in order to provide a service. For instance, in [179] the service robot is able to navigate autonomously in a Data Center. Similarly, in [145] a

museum robot explores the environment without the help of humans.

Navigation in crowded areas is a challenge because it is a dynamic environment. Generally, navigation approaches at the state-of-the-art define an initial path from the current robot pose to the target pose, considering the static environment. If an obstacle is detected and obstructs the path, the path is re-computed from scratch. Moreover, if the path planning computation requires a certain amount of time, the robot stops, waiting for the new path. In crowded environments, the path should be computed a lot of time times. In the worst case, the robot has no idea how to move in the crowd, waiting for a new valid trajectory. This is the so-called *robot freezing problem* [197].

In literature, there are several works about mobile robot navigation in crowded environments. In particular, there are two strands of thought: *(i)* prediction of people motion, and *(ii)* using a dynamic path planning.

An interesting approach for robot navigation in a dense crowd is introduced by P. Trautman. In [197], he proposes the use of Interacting Gaussian Processes (IGP) to model and predict the motion of people. In [198], he extends the method by including the cooperation model between robot and people, using the Multiple Goal Interacting Gaussian Processes (MGIGP). This approach has very promising results, but it requires the environment to be structured with a pedestrian tracking system.

In [81], crowd behavior is estimated by using an Inverse Reinforcement Learning (IRL) approach.

Anyway, these approaches require the use of a tracking system, because the use of only on-board sensors is not enough to track all people in the crowd.

Dynamic path planning is widely used to solve the path planning problem in dynamic environments. In literature there are a large variety of dynamic path planning approaches, based on graph search, sample-based method and heuristic based algorithms [127], to name a few.

Relevant works have been done with sample-based methods and, in particular, with the Rapidly-exploring Random Tree (RRT) algorithm. RRT is proposed for the first time by S. La Valle in [112]. RRT is very popular in path planning because is able to compute a solution in high dimensional environments. For this reason, there are a lot of RRT-based algorithm such as RRT* [97], RRT-Connect [107] and Transition-based RRT [91].

Thus, RRT is also used to develop some dynamic path planning algorithms. In [194] an RRT-based algorithm is used to compute a safe path in a human environment, by estimating the people motion. Similarly, in [67] the Probabilistic RRT is proposed for dynamic environments. A dynamic RRT* is proposed in [62], while in [150] a path planning and re-planning algorithm is presented, called riskRRT^X.

Anyway, sample-based dynamic path planning approaches have a drawback. Since they compute the near optimal solution, if the path planning continuously computes the path to avoid a symmetric obstacle, an *indecision behavior* may occur. In fact, if both alternative paths to avoid an obstacle have a similar distance cost, the path planning can compute different trajectories at each path computation. This causes an indecision behavior because the robot follows the path. Figure 7.1 illustrated this behavior.

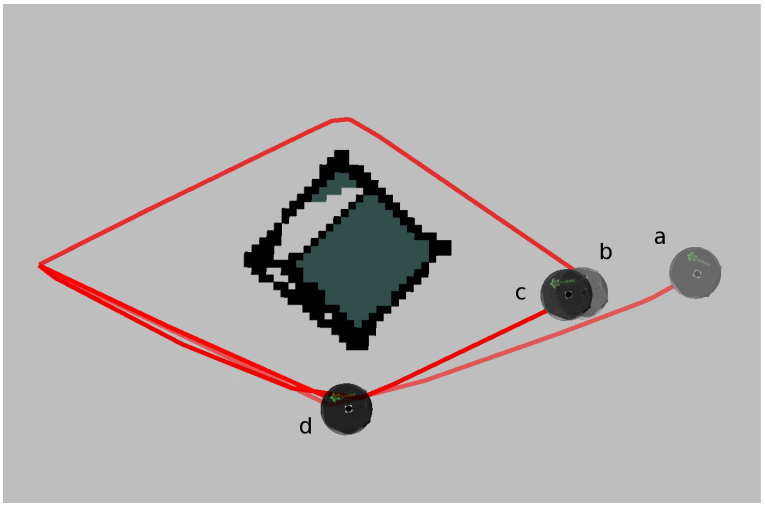


Figure 7.1: Example of indecision behavior. During the execution, the robot follows the path *a*, but the re-planning phase computes the path *b*, then the path *c*. The transition between paths *a*-*b*-*c* causes the indecision behavior. Anyway, the robot is able to overcome the obstacle.

In this paper, we propose a new framework for dynamic path planning in crowded environments, in which people are assumed as static obstacles. Initially, the path planning computes a valid path to reach the desired pose. While the robot executes the path, the path planning continuously checks, repairs and improves the path, according to changes in the dynamic environment. This re-planning method allows a valid path to be maintained until the desired target position is reached.

This approach uses a sample-based algorithm called Informed Optimal Rapidly-exploring Random Tree (Informed-RRT*) introduced in [68]. This algorithm allows a fast computation of the path to be performed even in high-dimensional space.

The proposed method aims to maintain always a valid path producing a smooth motion of the robot, solving the indecision problem and the freezing robot problem, i.e., when the robot stops waiting for a valid path to overcome the crowd. Another benefit is a reduction of CPU resources since the proposed approach adapts the path only when it is necessary.

The proposed approach is a simple method to solve the path planning problem in highly dynamic environments. Our solution does not require information from external sensors, such as cameras or tracking systems in the environment, but it relies only on on-board sensors, in order to reduce the system complexity and the application scenarios.

7.2 The Informed-RRT* algorithm

In this section, we describe the original Informed-RRT* algorithm introduced by Gammel et al. [68]. Informed-RRT* is a sample based algorithm based on the well-known Rapidly-exploring Random Tree (RRT). RRT is a popular path planning algorithm because of its speed to find solutions to single-query problems. Anyway, RRT is not optimal, because the existing tree constructed during the exploration biases future expansions. Optimal RRT (RRT*) is an improvement of RRT, able to seek for the near-optimal path. RRT* uses the rewire procedure, in which edges of the existing exploration tree can be replaced to maintain an optimal tree.

Informed-RRT* is an enhanced of RRT*. It preserves the same completeness and optimality of RRT*, improving the convergence rate and the final solution quality. In particular, Informed-RRT* improves RRT* performances when it optimizes the path length.

Assuming a standard and optimal planning problem. Let $X \subseteq \mathbb{R}^n$ be the state space of the planning problem, in which $X_{obs} \subseteq X$ are the invalid states, i.e., occupied by obstacles, and $X_{free} = X \setminus X_{obs}$ are the remaining valid states. Given $x_{start}, x_{goal} \in X_{free}$, the initial state and the final state. Let Σ be the set of all paths, where a single path $\sigma : [0,1] \to X$ is a sequence of states. The path planning algorithm seeks for an optimal path σ^* from x_{start} to x_{goal} in X_{free} minimizing a cost function $c : \Sigma \to \mathbb{R} \geqslant 0$:

$$\begin{aligned} & \sigma^* = arg\min_{\sigma \in \Sigma} c(\sigma) \\ \text{subject to} \quad & \sigma(0) = x_{start} \\ & \sigma(1) = x_{goal} \\ & \forall s \in [0,1]\,,\ \sigma(s) \in X_{free}. \end{aligned} \tag{7.1}$$

Then, assuming $f(x)$ is the cost of an optimal path. A set of states $X_f \subseteq X$ exists, able to improve the current solution path with the cost c_{best}.

$$X_f = \{x \in X \mid f(x) < c_{best}\} \tag{7.2}$$

Typically, $f(\cdot)$ is unknown, but we can estimate an admissible heuristic function $\hat{f}(\cdot)$. Hence, we define a subset of states $X_{\hat{f}}$, such that $X_{\hat{f}} \supseteq X_f$.

Informed-RRT* determines an ellipsoidal informed subset of states to improve the current solution with the current solution cost c_{best}.

$$X_{\hat{f}} = \{x \in X \mid \|x_{start} - x\|_2 + \|x_{goal} - x\|_2 \leqslant c_{best}\} \tag{7.3}$$

The resulting ellipse has x_{start} and x_{goal} as focal points. The major axis is c_{best}, while the minor axis is $\sqrt{c_{best}^2 - c_{min}^2}$, with $c_{\min}$ is the minimum possible cost. The resulting ellipse is shown in Figure 7.2.

The algorithm uses the same logic of RRT*. In the same way, it explores the search space by building an optimal incremental tree. Once the first solution is found, RRT*

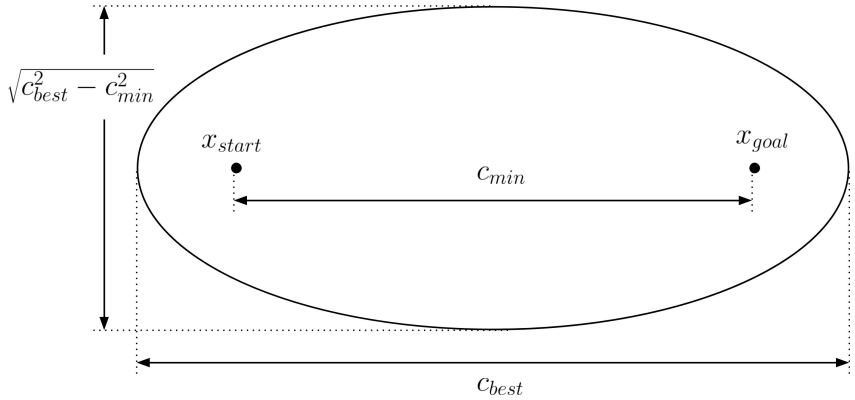


Figure 7.2: The ellipse used to describe the ellipsoidal informed subset of states. The focal points are x_{start} and x_{goal}.

continues to improve the solution using the whole search space. On the contrary, Informed-RRT* focuses the search on the part of the planning problem that can improve the solution, i.e., inside the ellipse of Equation (7.3). Hence, it is possible by sampling in the ellipsoidal informed subset of states $X_{\hat{f}}$. As a consequence, Informed-RRT* converges to an optimal solution faster than RRT*. A comparison between RRT* and Informed-RRT* is reported in Figure 7.3.

7.3 Informed-RRT*-based path planning

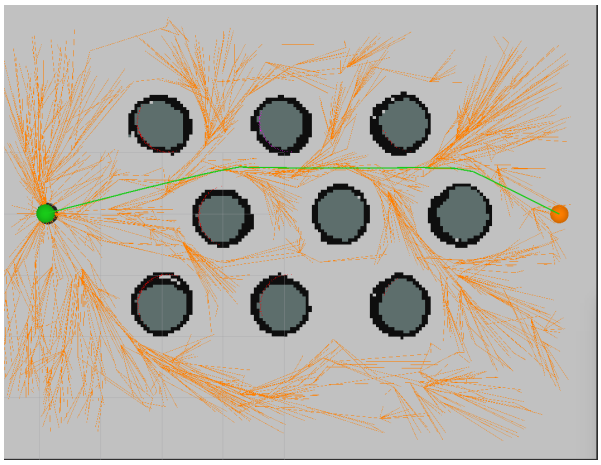
In this section, we describe the proposed dynamic path planning based on the Informed-RRT* algorithm. The architecture of the proposed approach is illustrated in Figure 7.4.

7.4 Informed-RRT* based Trajectory Planning

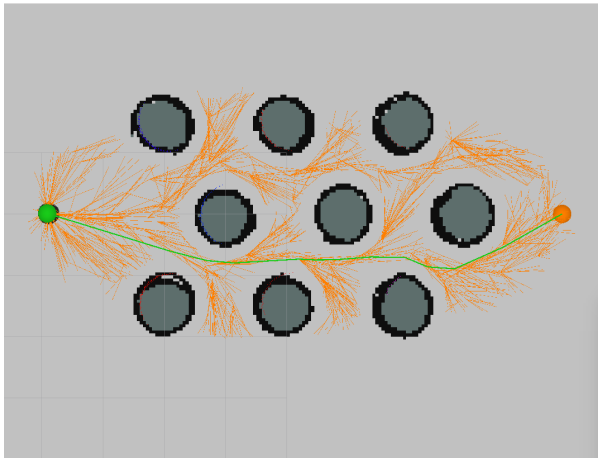
The main idea of the proposed approach is the following. Initially, using the Informed-RRT* algorithm, the initial solution path is computed. Hence, while the robot executes the current path, the path planner continues to check the validity of the path and, if necessary, it updates or recomputes the path. In fact, in dynamic environments, we are sure about the validity of the path only at the moment it is computed.

Then, considering the dynamic environment, X_{obs} and X_{free} change in time. At each iteration k, different sets $X_{obs}(k)$ and $X_{free}(k)$ can be defined, with $X_{free}(k) = X \setminus X_{obs}(k)$.

At each iteration of the algorithm, sensors measurements updates $X_{obs}(k)$ and $X_{free}(k)$. The Check routine verifies which states are still valid in the updated environment.



(a)



(b)

Figure 7.3: Example of RRT* and Informed-RRT* in a static environment and a computation time of 0.5 s. In (a), RRT* solution, while in (b), Informed-RRT*. The solution path is represented in green, the exploration tree in orange. Notice that Informed-RRT* uses the search space more efficiently, while RRT* uses the entire space.

Hence, the algorithm computes a score $\chi(\sigma)$ over the current path

$$\chi(\sigma) = \frac{\left|\left\{V_{obs}, E_{obs} \in \sigma([s_k, 1])\right\}\right|}{|\{V, E \in \sigma\}|}, \tag{7.4}$$

where V and E are vertexes and edges of the path σ. V_{obs} and E_{obs} are the invalid vertexes and edges of the path not yet executed $\sigma([s_k, 1])$, where $\sigma(s_k)$ is the state of the path corresponding to the actual robot position. The $|\cdot|$ set operator represents the cardinality of a set.

After the Check procedure, three different cases occur:

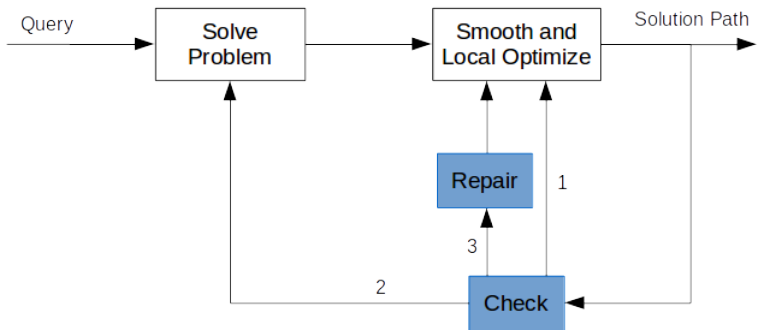


Figure 7.4: The main architecture of the proposed algorithm. After the Check routine there are three possible cases: path valid (1); path invalid (2); path invalid but repairable (3).

1. The current path is valid. There is no intersection between the current path not yet executed $\sigma([s_k, 1])$ and the set of invalid states $X_{obs}(k)$, i.e.,

$$\chi(\sigma) = 0. \tag{7.5}$$

2. The path is invalid and there are several invalid states states that intersect $X_{obs}(k)$, i.e.,

$$\chi(\sigma) \geq \chi_{max}. \tag{7.6}$$

3. The path is invalid, but there are few states that intersect $X_{obs}(k)$, that is

$$0 < \chi(\sigma) < \chi_{max}. \tag{7.7}$$

Where χ_{max} is a given threshold.

In the first case, the current path is still valid. There is no need to repair the path. Anyway, the path is always subjected to a smoothing and local optimization procedure, with the aim to improve the current path.

In the second case, the path is not valid. The search space is completely changed and there are many invalid states in the path. As a consequence, it is better to compute a new path from scratch, instead of repair the current one. Given a new start pose x_{start}, i.e. the current robot pose, the algorithm searches for a new solution path.

In the third case, the path is invalid only in a few states. In this case, it is useless to compute a new solution from scratch. Hence, we repair the invalid states.

As already discussed, while the algorithm verifies and updates the path, the robot follows the last solution path. This means that the robot can follow an invalid path for a brief time. According to [2], where the authors model the motion of walking people, in the worst case the mean value of walking velocity is 1.38 m/s. Hence, if we define a minimum planner frequency of 2 Hz, a robot follows an invalid path for 500 ms, while

people move 0.69 meters. With a lower planner frequency, the path can be invalid too many times, and the executed path might move towards people.

This new approach brings some advantages. First, the main trajectory remains stable, because the algorithm adapts the path with respect to the dynamic environment. On the contrary, if a new solution is computed each time, a robot can change continuously the trajectory, causing a discontinuous movement.

Anyway, preserving the old path, the optimal solution is not always guaranteed, but the resulting robot behavior is considerably improved.

7.4.1 Pseudo-code

This section explains the proposed algorithm. The main routine of the proposed dynamic path planning is reported in Algorithm 9, where the path computation and evaluation are performed. Algorithms 10 and 11 describe the repair routines, respectively, the RepairStates and RepairEdges subroutines.

Algorithm 9 Trajectory Planning Informed-RRT* based

```
1:  procedure DynamicPlanning(x_robot, x_goal)
2:      τ ← reset()
3:      σ ← reset()
4:      while GoalReached() = False do
5:          UpdateSearchSpace()
6:          χ ← Check(σ)
7:          if χ > χ_max or σ is empty then
8:              τ ← reset()
9:              τ ← InformedRRT*(x_robot, x_goal)
10:             σ ← BestTrajectory(τ)
11:         else if χ > 0 then
12:             σ ← RepairStates(σ)
13:             σ ← RepairEdges(σ)
14:         end if
15:         σ ← LocalShortcut(σ)
16:         σ ← Smoother(σ)
17:         σ ← Interpolate(σ)
18:         ExecutePath(σ)
19:     end while
20:     return
21: end procedure
```

According to Algorithm 9, the inputs of the dynamic path planning are the start state x_{robot}, defined according to the current robot pose, and the goal state x_{goal}. First of

Algorithm 10 RepairStates() routine

```
1: function RepairStates(σ)
2:     for i = 1 to i = InvalidState.size() do
3:         x_n = InvalidState[i]
4:         σ ← Disconnect(x_n)
5:         n_new = NeighborSample(x_n, r_max)
6:         if StateValid(x_new) then
7:             σ ← Reconnect(x_new)
8:         end if
9:     end for
10:    return σ
11: end function
```

Algorithm 11 RepairEdges() routine

```
1: function RepairEdges(σ)
2:     for i = 1 to i = InvalidEdge.size() do
3:         E_n = InvalidEdge[i]
4:         SearchNearState(x_{n-1}, x_{n+1})
5:         σ ← EraseEdge(E_n)
6:         n_new = NeighborSample(x_{n-1}, r_max)
7:         if StateValid(x_new) then
8:             σ ← Reconnect(x_new)
9:         end if
10:    end for
11:    return σ
12: end function
```

all, the exploration tree τ and the path σ are initialized, in order to erase old data (lines 2, 3).

Then, the main routine starts, until the robot reaches the goal state x_{goal} (from line 4 to 19). The function *GoalReached()* verifies if the robot position is in the goal region, i.e., the area that comprises the goal state x_{goal} inflated with a tolerance radius. It returns the true value only if the robot reaches the goal position.

At each iteration step, the search space must be updated (line 5). The search space is updated according to sensor measurements, by determining $X_{obs} \subseteq X$ and hence X_{free}. Initially, a static map of environments is used to define the search space. Hence, sensors detect obstacles and updates the map and, as a consequence, the search space.

The *Check()* function computes χ, i.e. the ratio of invalid states, according to Equation (7.4) (line 6). The algorithm checks if each state and edge of the path are in the free space $x_n, E_n \in X_{free}$. This procedure is carried out using the radius of the robot r_{robot} as tolerance, in order to take into account the occupation of the mobile robot platform. At this point, the function evaluates the $\chi(\sigma)$ in respect to the threshold χ_{max} (lines from 7 to 14). The value of $\chi(\sigma)$ is evaluated according to Equations (7.5), (7.6) and (7.7).

If the path is invalid, the algorithm searches for a new solution (lines from 8 to 10). Given the current robot pose x_{robot} and x_{goal}, the Informed-RRT* algorithm seeks for the optimal path. Before computing the new plan, the exploration tree is initialized, in order to erase old data. Informed-RRT* builds a new exploration tree τ, in which the solution path is the branch with the lower cost, computed with the function *BestTrajectory()*.

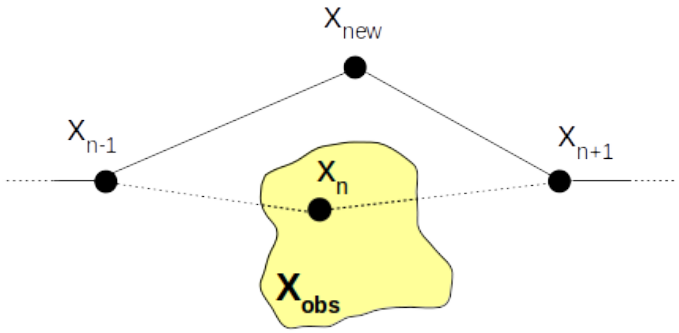
If the path is invalid but repairable, the algorithm tries to repair the invalid states and edges (lines 12, 13). As reported in Algorithms 10 and 11, there are two simple cases: an invalid state or an invalid edge between states. Figure 7.5a exemplifies the first case, where the algorithm searches for a new state by sampling around the invalid state. New sample must belong to a restricted area around the old state with a radius r_{max}. If a new state is found, the procedure erases near edges $e_n = \{x_{n-1}, x_n\},\ e_{n+1} = \{x_n, x_{n+1}\}$ connected to the invalid state x_n, and connects x_{new} with other states, then $e_n = \{x_{n-1}, x_{new}\},\ e_{n+1} = \{x_{new}, x_{n+1}\}$.

Similarly, Figure 7.5b illustrates the case with valid states, but invalid edges. Here, there is no need to replace any state. The algorithm erases the edge $e_n = \{x_{n-1}, x_n\}$, and searches for a new state x_{new}. Hence, the new state is connected to neighbors states. Then $e_n = \{x_{n-1}, x_{new}\},\ e_{n+1} = \{x_{new}, x_n\}$. With a sequence of invalid states, the procedure is the same as for a single state, i.e., more states are sampled in order to replace the invalid ones. Generally, with a *Invalid Ratio* less than χ_{max}, the repair function works successfully. In case of a failure, a new path must be computed, as in lines 8, 9, 10.

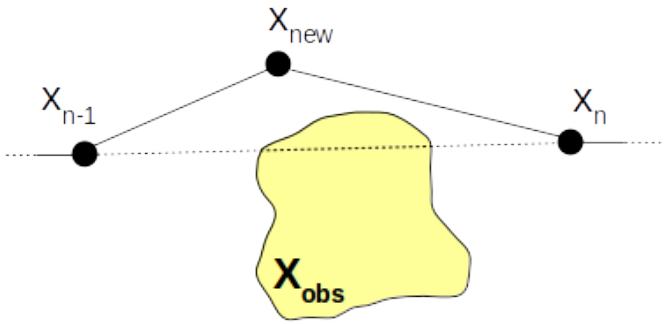
If the path is completely valid, no extra routines are executed on the path.

We remark that in the first iteration, the solution path σ is empty. Then, the path is computed with Informed-RRT*.

Indifferently to the value of χ, the path is improved using three different routines: *LocalShortcut()*, *Smoother()* and *Interpolate()* (lines 15, 16, 17).



(a)



(b)

Figure 7.5: The main simple cases of the repair procedure. In (a), there is an invalid state and it is replaced with a new one. In (b), there is an invalid edge. Hence, a new state is added to avoid the X_{obs} area.

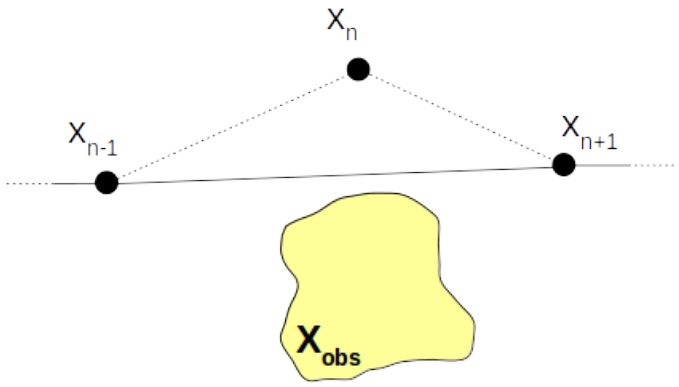


Figure 7.6: Example of the short-cut procedure. x_{n-1} and x_{n+1} can be connected directly without connecting to x_n. The function erases x_n only if the resulting cost is lower.

LocalShortcut() provides a simplified solution. As illustrated in Figure 7.6, if a line-of-sight edge can connect directly x_{n-1} and x_{n+1} with a minor cost and without passing through the state x_n, x_n can be eliminated.

The *Smoother()* function smooths the path using a Spline function.

The *Interpolate()* function adds new states in the path, in order to have states every 0.10 meters. This is useful when the robot executes the trajectory since it follows each state in sequence. Using densely placed states, the robot follows the solution path more closely.

At the end of each iteration, σ is handed over the trajectory controller that executes the solution path.

7.5 Results

The dynamic path planner algorithm presented in this Chapter is tested in an experimental test in a crowded environment.

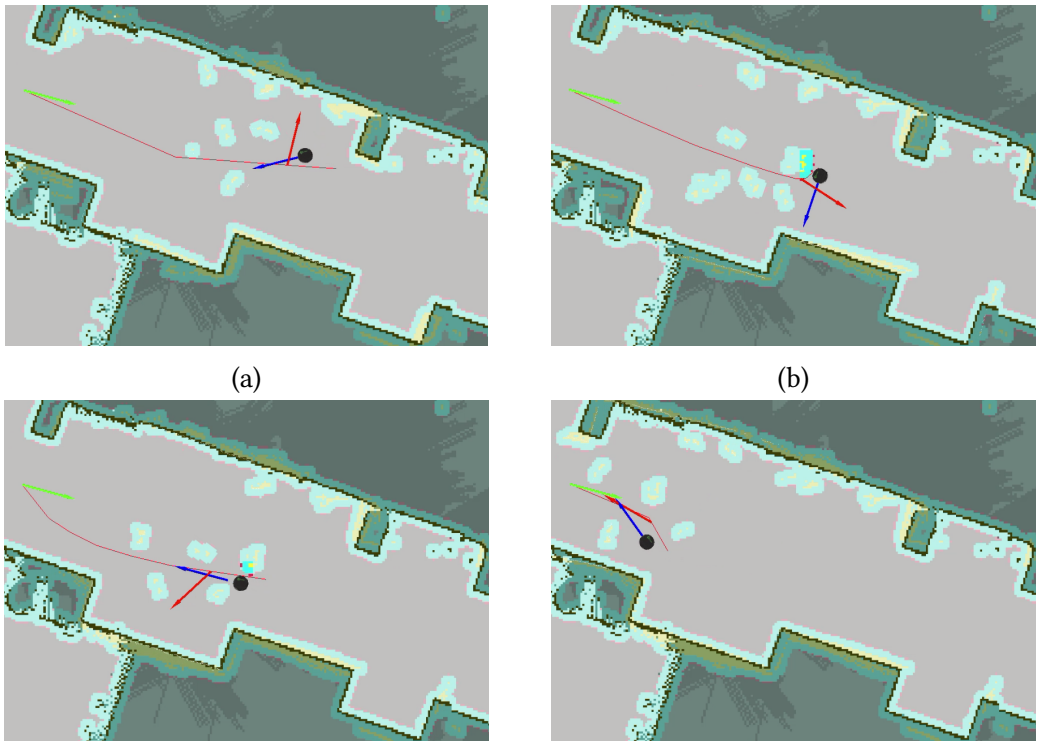
The algorithm is implemented in C++ using the Open Motion Planning Library (OMPL) [193]. OMPL is an open source library that contains implementations of many sample-based algorithms for path planning, in particular, many RRT-based planners, including the original Informed-RRT* algorithm. Moreover, the proposed solution is implemented using the Robot Operating System (ROS) [167], which enables easy deployment in simulation and on real robots.

The algorithm is tested in a real case scenario, with a Turtlebot 2 robot navigating in a dynamic and crowded environment.

Our autonomous navigation approach is implemented using the *navigation stack* of ROS, in which the autonomous navigation consists of a two-stage problem, composed by global planning and local planning. The *global planner* computes the entire path from the robot position to reach a given target point. Once the path is computed, the *local planner* follows the global path, controlling the robot. As already discussed, the path planning strategy here reported focuses on the global planner computation. In fact, the local planner is implemented using the Enhanced Vector Field Histogram (VFH+), a real-time motion planning algorithm for obstacle avoidance introduced by Borenstein et al. in [202]. This is an improved version of the original Vector Field Histogram described in [22] by the same authors.

Autonomous navigation requires a map to define the search space. Moreover, the map supports the localization task. In particular, localization is performed using the Adaptive Monte-Carlo Localization (AMCL), proposed in [65] and already implemented by ROS in the navigation stack. Monte-Carlo localization approaches recursively estimate the posterior probability of the robot's pose using particle filters (sample-based implementation of Bayesian filters).

A Turtlebot 2 robot is used. It includes a differential drive Kobuki base robot equipped with a gyro to improve odometry and a bumper to detect collisions. Moreover, on top



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

Figure 7.7: Navigation result in a real environment. The map is known and it is updated by sensors that detect people and new obstacles. Note that laser detects people in about 5 meters. The algorithm continuously checks and improves or computes the global path. It searches a gap between obstacles. The robot follows the path and finally reaches the goal point. The red line represents the current path. Green arrow the final pose, blue arrow the robot pose, while red arrow the pose gives to the local planner.

the robot, a Hokuyo Laser Range Scanner Sensor (URG-04LX-UG01) is mounted, able to detect obstacles at a maximum distance of 5.5 m.

The experiment was carried in our University corridors. Several parameters are necessary to the algorithm and they have been set finding the best configuration experimentally. The main algorithm loop routine should be able to run at a rate of at least 2 Hz. With lower frequency, the algorithm may fail to face the moving people, while with this frequency it guarantees always a valid and feasible path.

Figure 7.7 reports the experimental test in a mapped environment of about 35 m × 30 m. With the available hardware (CPU Intel CORE i7 with 2-core at 1.9 GHz) the main loop was able to run at a frequency of about 4 Hz. With higher frequency, the algorithm may return an incomplete or fragmented path, because there is less time to search for an optimal solution. This frequency has been possible due to Informed-RRT*, which optimizes its first solution in Informed space. In fact algorithms like RRT* search

and optimize the solution in all the search space.

In this test, the algorithm continuously returns the solution path or tries to improve the previous one. In this way, the robot always has a valid path to navigate in crowded environments. Following the path, the robot reaches the goal pose, avoiding collisions with people or other obstacles.

This approach requires a lower CPU usage since the path planning algorithm is called fewer times; this feature is important, especially regarding the on-board computer power consumption.

7.6 Discussion

In this Chapter, a dynamic path planner for mobile robots in crowded environments is proposed. The algorithm uses a *check and repair* approach that continuously evaluates and updates the current solution path. This routine allows a safe and valid path to be provided to move among people. Moreover, the proposed approach solves the freezing robot problem and avoids the indecision behavior, typical of re-planning approaches. Moreover, the proposed approach saves computational resources, in fact, the path is rarely computed from scratch, but it is continuously repaired according to the dynamic environment.

The main advantage of this approach is the simplicity. In fact, the presented dynamic path planning strategy does not require any external sensor and can be executed using only on-board sensors. Experimental tests conducted in a real case scenario demonstrate that the proposed approach is able to move among people avoiding collisions, reaching the target position in 100% of the tests.

This dynamic path planning strategy is implemented and used on the Virgil robot to perform a Service robotics application in a museum. More details about Virgil are reported in Chapter 9.

Chapter 8

Motion Control with Particle Filter Model Predictive Equilibrium Point Control

This Chapter presents an optimal motion controller for mobile robots, called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC). This approach implements a motion controller for autonomous navigation with obstacle avoidance and providing a safe and smooth motion.

This is based on a Model Predictive Control (MPC) approach, where the optimal trajectory is computed evaluating the behavior of the robot in the prediction horizon. Inspiring to the Equilibrium Point approach, the motion controller searches for an optimal equilibrium point near the robot that implies an optimal trajectory. Moreover, particle filters are used to consider uncertainties in the prediction, improving safety. This motion controller is presented for the first time in [155].

This Chapter is organized as follows. In Section 8.1 some background information are reported, as well as the literature review. Section 8.2 describes the kinematic equations, while in Section 8.3 the control law used in the proposed approach is explained. The proposed approach is presented in Section 8.4, while Section 8.5 describes the autonomous navigation using the PF-MPEPC method. Results are reported in Section 8.6 and discussed in Section 8.7.

8.1 Background

The large diffusion of service robotics applications in our life requires that robots are able to move autonomously in a safe and reliable way.

Autonomous navigation is an essential element of service robotics applications because the robot needs to move in autonomy in order to reach the desired position, in order to provide the service. For instance, in [145] a service robot in a museum is able

to move autonomously, in order to stream a real-time video to remote users. In [179], a service robot monitors a Data Center with autonomous navigation capabilities. Generally, the autonomous navigation consists of two phases: path planning and motion control.

The path planning searches for the globally optimal path from the robot pose to a target position, based on a reference map. Then, the motion controller aims to execute the path and, generally, to control the effective robot motion. Anyway, if the environment is unknown, i.e., the robot has not the map of the navigation area, the path planning procedure is useless. Hence, the motion controller reaches the target pose using a purely reactive method.

However, in both scenarios, the motion controller has an essential role, because it provides the robot motion.

In literature, there are many works about the motion controller for mobile robots [151]. One of the first is presented in [104] by Khatib, based on the potential field method. Other promising approaches rely on Bug algorithm [125, 144], a simple and efficient obstacle avoidance method.

A popular technique is the Vector Field Histogram algorithm [22], a fast obstacle avoidance method based on vector field. Same authors propose enhanced version of VFH, called VFH+ [202] and VFH* [201]. In [168] the Elastic Band approach is presented to provide real-time motion control. An enhanced version, called Timed Elastic Band is proposed in [175]. A reactive obstacle avoidance method, called Nearest Diagram is presented in [135], able to navigate in cluttered spaces. An enhanced version is explained in [52]. A popular and widely used approach is the Dynamic Window Approach (DWA), proposed in [66], in which optimal velocities are computed to avoid obstacles.

Sometimes, traditional reactive approaches may have unsatisfactory behavior. The robot can have oscillatory and uncomfortable motion. This happens because reactive motion controllers don't evaluate the robot behavior, but they use only the current and past state of the vehicle.

On the contrary, Model Predictive Control approaches evaluate the robot behavior, by optimizing the control in the prediction horizon. MPC-based approaches are widely used in motion control [95]. In [109] a linear MPC is used for trajectory tracking, while the same authors use a non-linear MPC in [110]. In [153] the Model Predictive Equilibrium Point approach is presented, by combining the equilibrium point method with MPC.

In this Chapter, we present a novel method of motion controller, called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC). It is an MPC based approach and includes the Equilibrium Control Approach. Like other MPC-based approaches, it minimizes a cost function in order to optimize the robot motion. In particular, the optimization procedure searches for an equilibrium point near the robot that implies the optimal trajectory. Particle filters are used in the prediction step, in order to evaluate uncertainties due to disturbances.

The proposed approach generates an optimal trajectory, avoiding obstacles with a smooth and safe motion.

In the following sections, the PF-MPEPC approach is described in detail, as well as the kinematic equations and the control law of a mobile robot with a differential drive. Finally, the approach is implemented both in simulation and on a real robot platform.

8.2 Kinematic equations

The motion controller presented in this Chapter is applied to a mobile robot with a differential drive.

It is quite easy to control a differential drive robot because it can be modeled as a simple unicycle, where the linear and angular velocities can be controlled independently. Considering a two-dimensional Cartesian space, the robot state is defined as $X_R = [x_R, y_R, \theta_R]^T$, with x and y determine the position of the robot and θ the orientation. In the same way, the goal pose can be defined as $X_G = [x_G, y_G, \theta_G]^T$.

According to Brockett's result [24], using the Cartesian representation of the unicycle, a smooth state feedback control law does not exist. On the contrary, a smoother closed-loop trajectory can be provided using polar coordinates. In [108], the author compares the performance of an MPC approach using both Cartesian and Polar coordinates.

Hence, considering the goal pose X_G, the Polar coordinates of the robot are defined as follows

$$X_R^{polar} = \begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix}, \tag{8.1}$$

Where the Cartesian and polar coordinates are related by:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}, \tag{8.2}$$

$$\alpha = -\theta + atan2(\Delta y, \Delta x), \tag{8.3}$$

$$\beta = -\theta - \alpha, \tag{8.4}$$

with ρ is the Euclidean distance between the robot and the goal pose, $\beta \in (-\pi, \pi)$ is the orientation of the goal and $\alpha \in (-\pi, \pi)$ is the orientation of the vehicle heading with respect to the line from the observer to the goal. Briefly, as illustrated in Figure 8.1, polar coordinates describe the error of the robot to the goal pose.

Moreover, considering a parking problem, i.e., when the robot needs to reach and stop in a goal pose, it is easier to control a system that needs to converge in a pose $[\rho, \alpha, \beta]^T = [0, 0, 0]^T$.

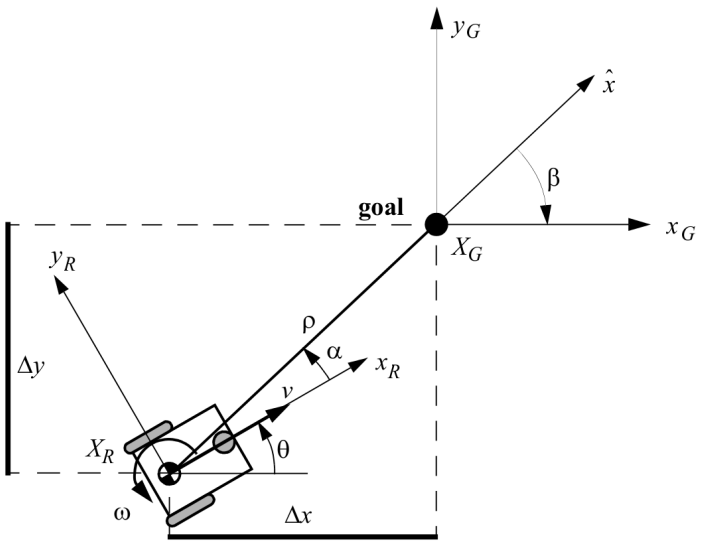


Figure 8.1: Polar Coordinates representation. Values ρ, α and β describe the error from robot to goal pose.

Then the kinematic equations are given by:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -cos\alpha & 0 \\ \frac{sin\alpha}{\rho} & -1 \\ \frac{sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{8.5}$$

8.3 Kinematic control law

In this section, the kinematic control law of a differential drive robot is defined. Similarly to [1], we study the asymptotic behavior of a dynamic system using the well-known Lyapunov stability theory. The simplest choice for Lyapunov function candidate is

$$V = V_1 + V_2 = \frac{1}{2}\lambda\rho^2 + \frac{1}{2}(\alpha^2 + h\beta^2), \quad \lambda, h > 0 \tag{8.6}$$

in which the element V_1 considers the distance error, while V_2 the alignment error. λ and h are constant parameters. Hence, considering the kinematic equations of Equation (8.5), the time derivative $\dot{V}$ is

$$\begin{aligned} \dot{V} = \dot{V}_1 + \dot{V}_2 = \lambda\rho\dot{\rho} + (\alpha\dot{\alpha} + h\beta\dot{\beta}) = \\ -\lambda\rho v\cos\alpha + \alpha\Big[-\omega + \frac{v}{\rho}\frac{\sin\alpha(\alpha - h\beta)}{\alpha}\Big] \end{aligned} \tag{8.7}$$

According to the Lyapunov stability theory, it is important to have $\dot{V}_1$ and $\dot{V}_2$ non-positive and, as a consequence, V is non increasing in time.

Similarly to [1], velocities v and ω are defined with a smooth form

$$v = \gamma \rho \cos \alpha, \quad \gamma > 0, \tag{8.8}$$

$$\omega = k\alpha + \gamma \frac{\cos \alpha \sin \alpha}{\alpha}(\alpha - h\beta), \quad k > 0, \tag{8.9}$$

with γ and k are positive constant, always greater than zero.

Notice that, according to Barbalat's Lemma, the function $\dot{V}$ converges to zero, i.e. it converges to the only possible equilibrium point at $[\rho, \alpha, \beta]^T = [0, 0, 0]^T$.

Hence, the closed loop equations are defined as follows

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\gamma \rho cos^2 \alpha \\ -k\alpha + \gamma h \beta \frac{\sin \alpha \cos \alpha}{\alpha} \\ \gamma \sin \alpha \cos \alpha \end{bmatrix} \tag{8.10}$$

8.4 Particle Filter Model Predictive Equilibrium Point Control

In this section, the proposed approach called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC) is presented. However, before explaining the PF-MPEPC, the traditional MPC is explained, as well as the Model Predictive Equilibrium Point Control that merges the MPC philosophy with the Equilibrium Point approach.

8.4.1 Traditional Model predictive Control

The Model Predictive Control approach is widely used in literature because of its performances and ability to solve a wide variety of control applications. It uses an explicit model of the plant to be controlled, in order to predict future output behavior. The aim is to optimize the behavior of the system over a future horizon, by solving an optimal control problem in real-time. Practically, at time k, an optimal control problem is solved, obtaining a sequence of optimal commands in the prediction horizon. Anyway, only the first control is actually applied to the plant, while the remaining optimal control inputs are discarded. Hence, at time $k+1$, the new optimal control problem is solved, considering new measurements for the plant.

Generally, with a MPC-based approach, a cost function J is defined

$$\begin{aligned} &J(U(k), x(k|k)) = \\ &\sum_{i=0}^{H_p-1} l(x(k+i|k), u(k+i|k)) + \Phi(x(k+H_p|k)), \end{aligned} \tag{8.11}$$

with H_p is the prediction horizon, $l(\cdot)$ is the per-stage weighting function, $\Phi(\cdot)$ is the terminal state weighting function. The notation $x(k+i|k)$ indicates the state at time $k+i$ computed with the data known at time k.

Then, the control vector $U(k)$ is computed solving the optimization problem:

$$U^*(k) = \min_{U(k)} \quad J(U(k), x(k|k)), \tag{8.12}$$

$$\text{subject to} \quad x(k+1) = f(x(k), u(k)), \tag{8.13}$$

$$U(k) \in \mathbb{U}, \tag{8.14}$$

$$x(k+i|k) \in \mathbb{X}, \tag{8.15}$$

in which (8.13) is the system dynamics, while (8.14) and (8.15) are constraints on controls and states.

8.4.2 Model Predictive Equilibrium Point Control

The use of the Equilibrium Point approach with an MPC controller is introduced in [152].

Assuming a non linear system with an equilibrium state $\tilde{x}$, we have $x(k) \to \tilde{x}$ for $k \to \infty$. Hence, the dynamic system can be defined as

$$x(k+1) = f(x(k), \tilde{x}) \tag{8.16}$$

considering the control law explained in Section 8.3, the system is able to execute a trajectory from the actual pose to the equilibrium point. Then, the control state can be determined as

$$u(k) = \pi(x(k), \tilde{x}), \tag{8.17}$$

with $\pi(\cdot)$ is the control law function, considering the actual state $x(k)$ and the equilibrium point $\tilde{x}$. This is the so-called Equilibrium Point Control (EPC) method.

According to [152], the MPC approach can be extended with the EPC method, obtaining the so-called Model Predictive Equilibrium Point Control (MPEPC). Unlike Equation (8.12), the optimization takes into account the equilibrium point $\tilde{x}$

$$\tilde{x}^*(k) = \min_{\tilde{x}} \quad J(x(k|k), \tilde{x}) \tag{8.18}$$

$$\text{subject to} \quad x(k+1) = f(x(k), \pi(x(k), \tilde{x})) \tag{8.19}$$

$$x(k+i|k) \in \mathbb{X} \tag{8.20}$$

As a consequence, the optimization searches for an optimal equilibrium point, able to minimize the cost function J.

Applied to the mobile robot navigation, $\tilde{x}$ is a target pose near the robot, expressed in polar coordinates $\tilde{x} = [\rho, \alpha, \beta]^T$. The equilibrium point is the pose that implies an optimal trajectory in the prediction horizon. Hence, using the control law of Section 8.3, the control state is defined.

8.4.3 Particle Filter Model Predictive Equilibrium Point Control

Generally, in the traditional MPC approach the measurement noise and, in general, the disturbances are not considered.

Our approach is based on the Particle Filter Model Predictive Control (PF-MPC) introduced in [190], in which particle filters are used in the prediction step to improve the stability and prevent the generation of excessive control input. In [190], two particle filters are used: *(i)* the first particle filter is a state estimator and, *(ii)* the second one is a predictor for the control input of the model. In [185], the same method is used to control a quadcopter.

The PF-MPEPC is based on the PF-MPC. In our approach, two particle filters are used in the prediction loop, as shown in the architecture of Figure 8.2. Similarly to [190], the first particle filter updates the state of the robot. Then, the second particle filter is not directly applied to the control input, but it is used to update the equilibrium point. Anyway, the equilibrium point is directly used to compute the control input of the system by using the control law designed in Section 8.3.

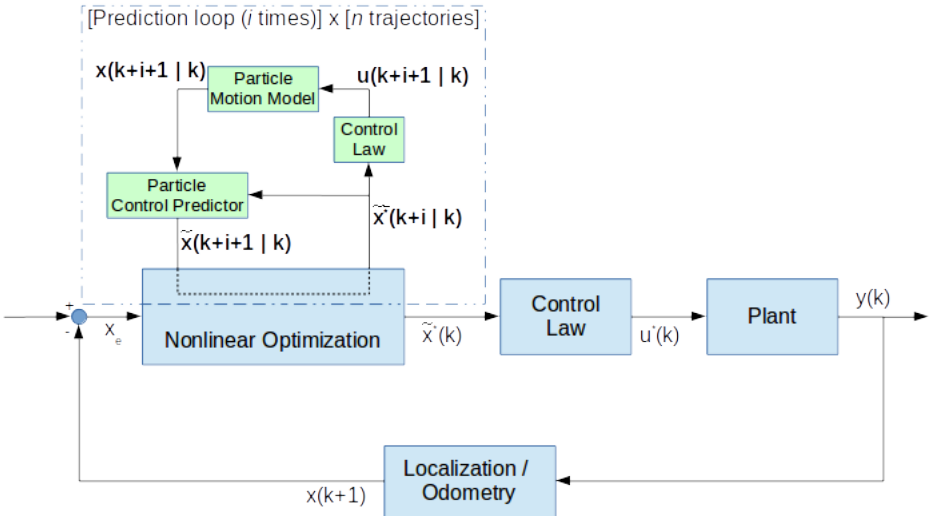


Figure 8.2: The PF-MPEPC control scheme. In green the prediction loop, while in blue the main control loop.

In order to explain the proposed approach, we assume the following nonlinear discrete-time equations

$$x_k = f(x_{k-1}, u_{k-1}, \epsilon_{k-1}) \tag{8.21}$$

$$y_k = g(x_k, \eta_k) \tag{8.22}$$

with ϵ_k and η_k are non Gaussian noise.

Initially, the first particle filter is initialized in the actual robot pose. Hence, all particles are set as $x_k^{(m)} = x_k$ with weights as $w_k^{(m)} = 1/M$, assuming M particles. In this way,

all particles are initialized in a single point. Generally, the initial pose can be defined using the robot pose estimated by the localization.

Hence, the particle filter updates the state of the robot, by using measurement data from sensors

$$x_k^{(m)} = f(x_{k-1}^{(m)}, u_{k-1}, \epsilon_{k-1}) \tag{8.23}$$

with $x_{k-1}^{(m)}$ are the particles describing the robot state at time $k-1$, while u_{k-1} is the control defined using the control law and the equilibrium point $\tilde{x}_{k-1}$ at time $k-1$

In order to implement the state estimator, we use the Probabilistic Motion Model proposed in [195] by Thrun, Burgard and Fox. Given a control u_{k-1} and a particles of the state $x_{k-1}^{(m)}$, the state estimator computes random particles $x_k^{(m)}$ distributed according to Equation (8.23), where $\epsilon = [\epsilon_{trans}, \epsilon_{rot1}, \epsilon_{rot2}]^T$ is a random noise in translation, initial rotation and final rotation. It differs from standard particle filter because it generates random poses of x_k, instead of computing the probability of a given x_k. In fact, with this sampling approach, weights don't change, but it is useful to evaluate the uncertainty of the resulting poses. Then, the output of the first particle filter, called Particle Motion Model, is the estimated state of the robot as particles $x_k^{(m)}$.

The second particle filter is the Control Predictor. Anyway, our approach does not compute directly the control of the system, but the equilibrium point. Hence, considering the updated robot pose, the Particle Control Predictor updates the equilibrium point, by computing the new polar coordinates in respect to the updated robot state and assuming disturbances. Roughly speaking, the second particle filter updates the equilibrium point as observed by the updated robot pose, considering disturbances in the observation. Hence

$$\tilde{x}_k^{(m)} = f_u(\tilde{x}_{k-1}^{(m)}, x_k^{(m)}, \upsilon) \tag{8.24}$$

with υ a random white noise with a normal distribution $\mathcal{N}(0, \Sigma)$, and $\tilde{x}_{k-1}^{(m)}$ are particles of the equilibrium point at time $k-1$. The noise υ represents the disturbances in the observation. Thus, it should be defined considering the sensor noise and the error introduced by the localization, if used.

Then, the likelihood of each particle is computed considering the target equilibrium point $\tilde{x}_k$. Then we update weights

$$w_k^{(m)} = p(\bar{\tilde{x}}_k | \tilde{x}_k^{(m)}) w_{k-1}^{(m)} \tag{8.25}$$

with $\bar{\tilde{x}}_k$ is the equilibrium point defined as setpoint computed in respect to the robot state. Thus, weights are normalized to sum to unity.

The function $p(\bar{\tilde{x}}_k | \tilde{x}_k^{(m)})$ is a conditional density function which is maximal if the estimated $\tilde{x}_k^{(m)}$ is optimal with respect to the original setpoint $\bar{\tilde{x}}_k$

$$p(\bar{\tilde{x}}_k | x_k^{(m)}) \sim \exp\left\{ - \frac{|\bar{\tilde{x}}_k - x_k^{(m)}|}{2\sigma^2} \right\} \tag{8.26}$$

Finally, if the effective number of particles is too low $n_{eff} < N_T$, we execute resampling. Where

$$n_{eff} = \frac{1}{\sum_{i=1}^{M} (w_k^{(i)})}, \tag{8.27}$$

with N_T is an appropriate threshold. This approach uses the Sequential Importance Resampling (SIR) algorithm that avoids the particle filter to degenerate.

The procedure described from Equation 8.23 to Equation 8.25 is repeated until the control horizon H_p. In (8.23), the robot pose $x_{k-1}^{(m)}$ is given by the particle motion model at previous step, and the control u_{k-1} is computed considering the particles related to the equilibrium point $\tilde{x}_{k-1}$ previously computed by the particle control predictor.

8.5 PF-MPEPC navigation

In this Section, the navigation using the PF-MPEPC approach is described, based on the architecture of our approach illustrated in Figure 8.2.

The input of the motion controller is a goal pose X_{goal}. Thus, considering the current robot pose, the error in polar coordinates X_e is computed.

The PF-MPEPC controller searches for an optimal equilibrium point in order to minimize a cost function J

$$\begin{aligned} J(U(k), \hat{x}(k|k)) = \sum_{i=0}^{H_p-1} \big[& \lambda_1 x_e(k+1|k)^2 + \\ & + \lambda_2 \omega(k+i-1|k)^2 + \\ & + \lambda_3 \Delta U(k+i-1|k)^2 + \\ & + \lambda_4 c_{obs}(\hat{x}(k+i|k)^2) \big], \end{aligned} \tag{8.28}$$

where $\hat{x}$ is the probabilistic position with a mean value μ_x and a covariance Σ_x, defined by the localization and estimated in the prediction step by particle filters. The parameters $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are constant weighting factors.

The first term penalizes the position error. The second term penalizes the angular velocity, in order to minimize excessive rotations along the trajectory. The third term penalizes the linear and angular accelerations. The fourth term penalizes the state cost in order to avoid obstacles, using the obstacle cost c_{obs}.

The obstacle cost c_{obs} is defined by a grid map, in which each element has a cost relative to the probability of collision. In particular, the cost increases with the proximity of obstacles. Minimizing this cost in Equation (8.28), the optimal trajectory avoids obstacles. This obstacle avoidance method uses the logic of the potential field approach, where repulsive forces generated by obstacles change the robot trajectory.

Moreover, control constraints on velocities and accelerations are taken into consideration

$$U_{min} \leqslant U(k) \leqslant U_{max} \tag{8.29}$$

$$\Delta U_{min} \leqslant \Delta U(k) \leqslant \Delta U_{max} \tag{8.30}$$

This is a nonlinear optimization problem. To solve it, we use a free and open-source library for nonlinear optimization called NLopt [93].

Hence, minimizing the J function, the PF-MPEPC controller finds the optimal equilibrium point $\tilde{x}^*(k)$.

The effective cost of J is computed with a prediction loop that estimates the robot behavior during the control horizon.

The prediction step uses the approach defined in Section 8.4.

At the iteration i, the prediction step considers an equilibrium point $\tilde{x}(k+i|k)$. Using the control law defined in Section 8.3, the control input $u(k+i+1|k)$ is computed. Then, the first particle filter, called Probabilistic Motion Model, updates the robot pose $x(k+i+1|k)$, while the second one updates the equilibrium point relative to the estimated pose $x(k+i+1|k)$. The iteration continues until the prediction horizon. Notice that, at the first iteration, the equilibrium point $\tilde{x}(k|k)$ is defined by the nonlinear optimization.

In order to compute the optimal equilibrium point, the nonlinear optimization evaluates n trajectories.

Given the optimal equilibrium point $\tilde{x}^*(k)$, the optimal control vector $U^*(k)$ is computed using the control law designed in Section 8.3. Following the MPC philosophy, we apply only the first control.

The control loop continues until the robot reaches the goal pose.

8.6 Results

In this section, we report the results obtained with experimental tests with a real robot.

The proposed approach is implemented in C++ using ROS (Robot Operating System) [167]. ROS enables an easy deployment in simulation and on real robots. The approach is tested using the Turtlebot 2 robot.

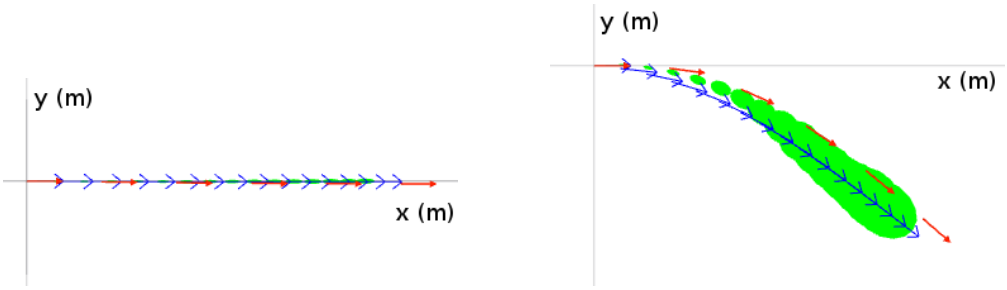
In the experimental test, the control horizon is $T = 4$ s and the controller frequency is set at 10 Hz. As a consequence, with a period of 0.1 s, the model predictive control evaluates 40 poses in the prediction horizon.

Because of the controller frequency, the optimization problem is time-constrained. In order to satisfy the controller frequency, 300 trajectories are evaluated by the optimization, guaranteeing an optimal solution. To perform the test we use a laptop with a 2-core with 1.9 GHz CPU.

The cost function minimized by the optimization is reported in Equation 8.28. Control constraints are defined as $v_{max} \leqslant 0.4$m/s and $\omega_{max} \leqslant 0.5$rad/s. After a tuning phase,

weights are set to $\lambda_1 = 2.0$, $\lambda_2 = 2.5$, $\lambda_3 = 0.9$, $\lambda_4 = 0.7$. Even if λ_4 has a low value, it has the highest priority in order to avoid collisions.

The space used by the optimization to search for the solution is bounded by $0 \leqslant \rho \leqslant 2$ (m), $-\pi/2 \leqslant \alpha \leqslant \pi/2$ (rad/s) and $-\pi/2 \leqslant \beta \leqslant \pi/2$ (rad/s).



(a) (b)

Figure 8.3: Motion prediction of simple trajectories. In red the real motion. In blue the ideal motion. In green the motion prediction using particle filters. The estimated poses are illustrated as an ellipse that describes the uncertainty. In (a), only the linear velocity is applied: ideal and real motion are similar, while motion prediction has a little uncertainty. In (b), both linear and angular velocities are applied. There is a considerable error between the ideal and real motion, then, the uncertainty is greater than the previous scenario.

Figure 8.3 shows the effect of particle filters in the prediction loop. The blue line is the ideal motion without considering disturbances. The red line is the real motion. Instead, the green line is the motion prediction using particle filters, in which each estimated pose is represented as an ellipse defined considering the covariance matrix. Because of the effect of disturbances along with the prediction, the uncertainty of poses increases with time in the prediction horizon.

Figure 8.3a reports the trajectory during a straight motion. The ideal prediction and the real motion are similar. Hence, the motion prediction with particle filters has small uncertainties. This is the simplest scenario, where only the linear velocity is applied.
On the contrary, in Figure 8.3b, the motion is given by both linear and angular velocities. The real motion differs from the ideal one. On the opposite, the prediction with particle filters evaluates disturbances estimating better the robot motion. This test demonstrates that the error is greater in rotation then in translation. Generally, it is a typical condition with wheeled robots. Moreover, a better prediction provides more safety. In fact, uncertainty is evaluated to avoid obstacles.

Figure 8.4 reports simple navigation in an unknown environment using the proposed PF-MPEPC approach. The controller continuously computes the optimal trajectory in order to reach the target position. Obstacles are detected using a laser scanner, then are marked on a dynamic map. Hence, the map is used by the optimization to compute the optimal trajectory. The resulting motion is smooth and safe.

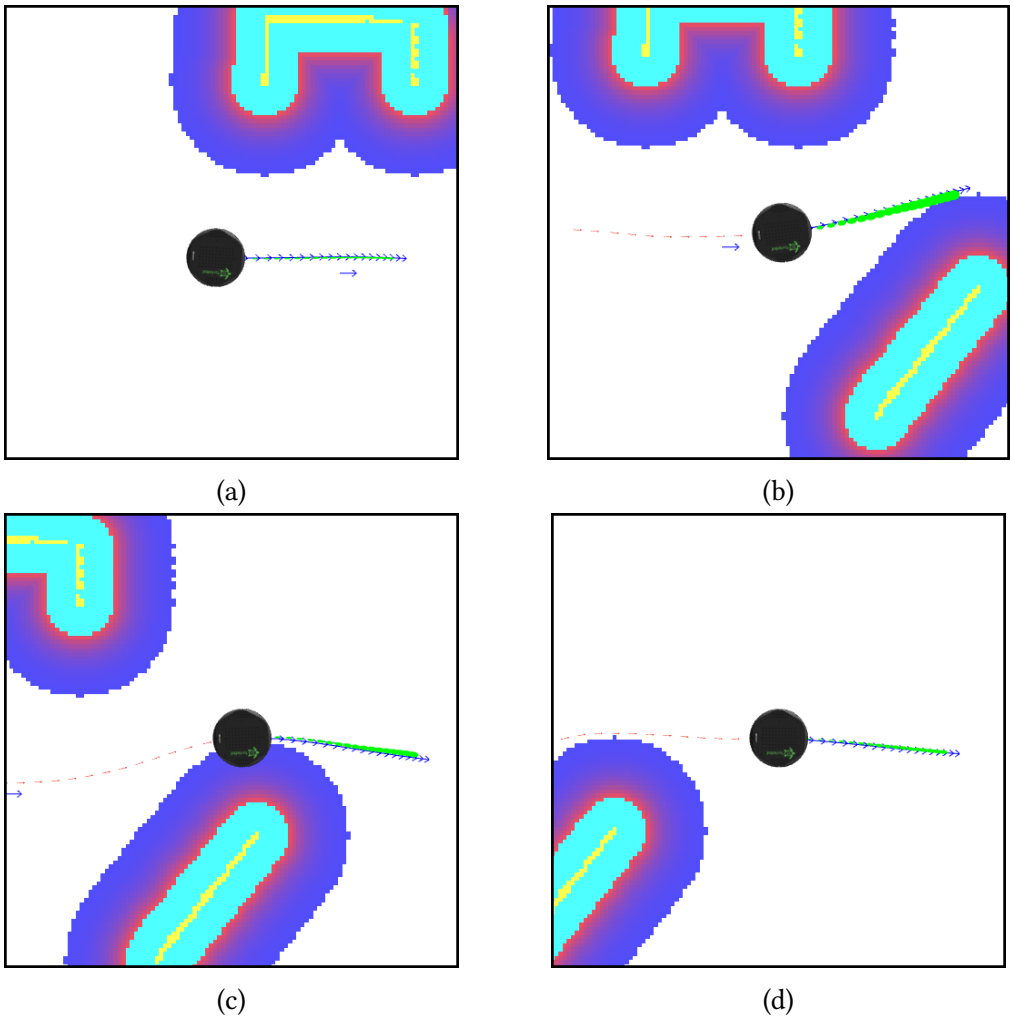


Figure 8.4: Example of the PF-MPEPC navigation in an unknown environment. In (a), the controller evaluates trajectories along horizon. In (b), robot detects unexpected obstacle and adapts its trajectory, in order to avoid the obstacle in (c). Then, the robot moves toward goal position in (d).

The dynamic map used in this work is a grid map with a resolution of 0.05 m. Each element of the map has a value associated with the probability of collision and it is represented in the range from 0 to 255. 0 refers to a safe area without obstacles and with no probability of collision (white color in the map), while 255 refers to an occupied space (yellow). Considering the robot dimension, the value of 254 refers to zones that cause collision (light blue). From 254 to 0 the probability of collision decreases. This is a common representation used in the *costmap* used and defined by ROS.

Assuming the experimental test of Figure 8.4, in Figure 8.5 the control commands applied to the robot are reported. The linear velocity is limited to 0.4 m/s while the

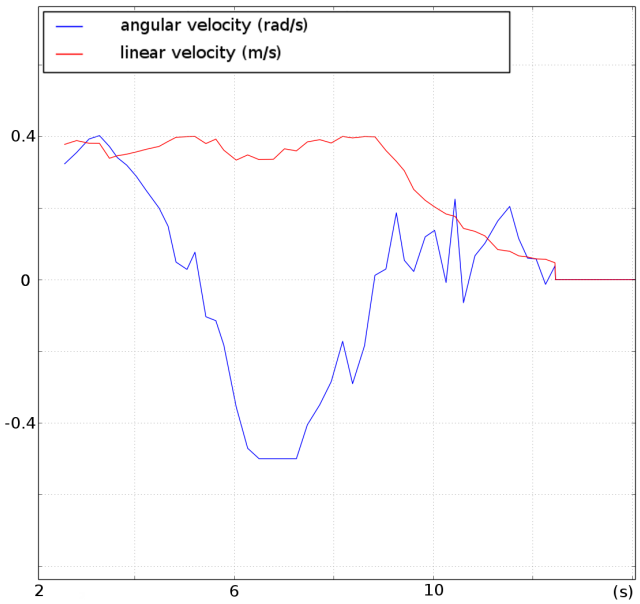


Figure 8.5: Velocity commands used in the same test of Figure 8.4. In red the linear velocity, while in blue the angular one.

angular velocity is limited to $\pm 0.5\ rad/s$. Moreover, excessive accelerations are limited, thanks to λ_3 in the cost function $\boldsymbol{J}$.

In the experimental test of Figure 8.4, the proposed PF-MPEPC is used to solve a parking problem in an unknown environment. Anyway, the same approach can be used for the trajectory tracking problem. With trajectory tracking, the implementation is similar to the parking problem. Before, the path planning computes the path from current to goal pose, composed by a sequence of poses. Then, the robot follows each pose with the PF-MPEPC controller. The PF-MPEPC is used as a motion controller for trajectory tracking in the robot Courier, a service robotics applications described in Chapter 9.

8.7 Discussion

In this Chapter, a novel method to provide optimal motion control for mobile robots is proposed, called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC).

The proposed method is based on a Model Predictive Control combined with the Equilibrium Point Control approach. The controller searches for an optimal equilibrium point near the robot, guaranteeing the safety and moving toward the goal pose. Using the MPC philosophy, the optimization computes a sequence of optimal commands and only the first one is actually applied to the robot. The proposed approach uses two particle filters in the prediction loop, in order to evaluate uncertainties due to disturbances

and measurement noise.

Unlike traditional reactive methods, our approach evaluates near future behavior. This feature provides better motion, preventing undesirable oscillation or dangerous states. In our test, the robot is able to avoid simple obstacles. The resulting motion is smooth and comfortable.

The use of the MPC approach makes it possible to apply constraints on state and input. In this work, velocities and accelerations are limited to suitable values to avoid undesirable behavior.

The proposed motion controller PF-MPEPC has some unique features. In fact, it evaluates the odometry error, sensor noise and other generic errors, such as the localization error. As a consequence, the controller computes an optimal trajectory avoiding obstacles and with a smooth motion evaluating uncertainties introduced in the system.

In this paper, we focus on differential drive robots, but this framework could be used with other robotics platforms. Use with other wheeled mobile robots implies different kinematic equations and, as a consequence, a new control law, but the method does not change.

Chapter 9

Service Robotics Applications

In this Chapter, two Cloud-based service robotics applications are described: *(i)* robot Courier, a service robot in a workspace, and *(ii)* Virgil, a robot in a museum.

The robot Courier provides a service robotics application in a workspace. The aim is to welcome a new visitor in a workspace and to escort him/her to the desired office.

Virgil is a service robot, able to provide a real-time virtual tour of a museum to remote users, by streaming a real-time video with high definition.

Both Virgil and robot Courier are tested in real scenarios, offering a service to people. Moreover, both applications rely on the Cloud-based architecture presented in Chapter 6.

This Chapter is organized as follows. In Section 9.1 some background information are reported, such as preliminary concept about the Robot Operating System (ROS) and the Cloud Robotics Platform (CRP) used in the proposed service robotics applications. The robot Courier, a service robot in a workplace is introduced in Section 9.2. Section 9.3 describes Virgil, a service robot in a museum. Hence, we discuss the proposed service robotics applications in Section 9.4.

9.1 Background

Service robotics is an emergent field in robotics. However, as discussed in Chapter 6, several service robotics applications are already developed in the last years. Moreover, thanks to the newest mobile technologies, a service robot is connected with the Internet opening new opportunities. In particular, the concept of Cloud Robotics can be used with service robotics.

Both robot Courier and Virgil robot services refer to the Cloud-based architecture presented in Chapter 6, where the Cloud has an essential role in the service robotics applications. In fact, most of the intelligent resides on-Cloud, as well as service management. On the opposite, on-board the robot, only a few tasks are executed. In particular, the two service robotics applications differ in the distribution of the software

between the robot and the Cloud because of the different on-board resources and service requirements.

Both Cloud-based services are developed using the Robot Operating System (ROS) framework and they use the Cloud Robotics Platform (CRP). In the next paragraphs, both ROS and CRP are described, in order to provide the required background to understand how both service robotics applications work.

9.1.1 Robot Operating System

The Robot Operating System (ROS) is an open-source meta-operating system for robot software development [167]. It provides hardware abstraction, low-level device control, message-passing between processes and package management. It also provides a collection of packages and tools for the development of distributed robotic applications.

At the moment, ROS is the most popular robotic framework. There are many reasons for this popularity: *(i)* the simplicity and the modularity of the framework, *(ii)* it is open-source supporting code reuse in robotics research, *(iii)* ROS code can be developed with modern programming languages, such as C++, Python and LISP, and *(iv)* it is compatible with a lot of robotics platforms.

ROS relies on the concept of the computational graph, i.e., a peer-to-peer network of ROS processes, able to provide a distributed architecture. The ROS process is called *node*, able to perform computation and to exchange data with other nodes. The *ROS Master* is the central node that provides the name registration and lookup the rest of the computation graph. Without the ROS Master, it is impossible for a node to find and communicate with other nodes. The communication between nodes is based on the TCP or UDP network protocols, using two communication models: *topics* and *services*.

Topics are based on the *publish/subscribe* logic. A node sends a message by publishing a topic, with an identification name and type. Then, thanks to the ROS Master, other nodes are able to subscribe to a topic and receive the message.

Services use the *request/reply* logic. A node offers a service with a specified identification name and type. Thanks to the ROS Master, other nodes can call the service sending a request message, then, they receive the response message from the service server node.

9.1.2 Cloud Robotics Platform

The proposed Cloud-based application is based on the Cloud Robotics Platform (CRP) developed by TIM S.p.A.. It relies on the concept of Platform-as-a-Service (PaaS) [14], i.e., a Cloud Computing service that provides a platform to costumers, in order to run and manage applications and web services.

The CRP is based on the ROS framework to be compatible with most of the robotics platforms. The CRP is presented for the first time in [173], but it is also used in [145,

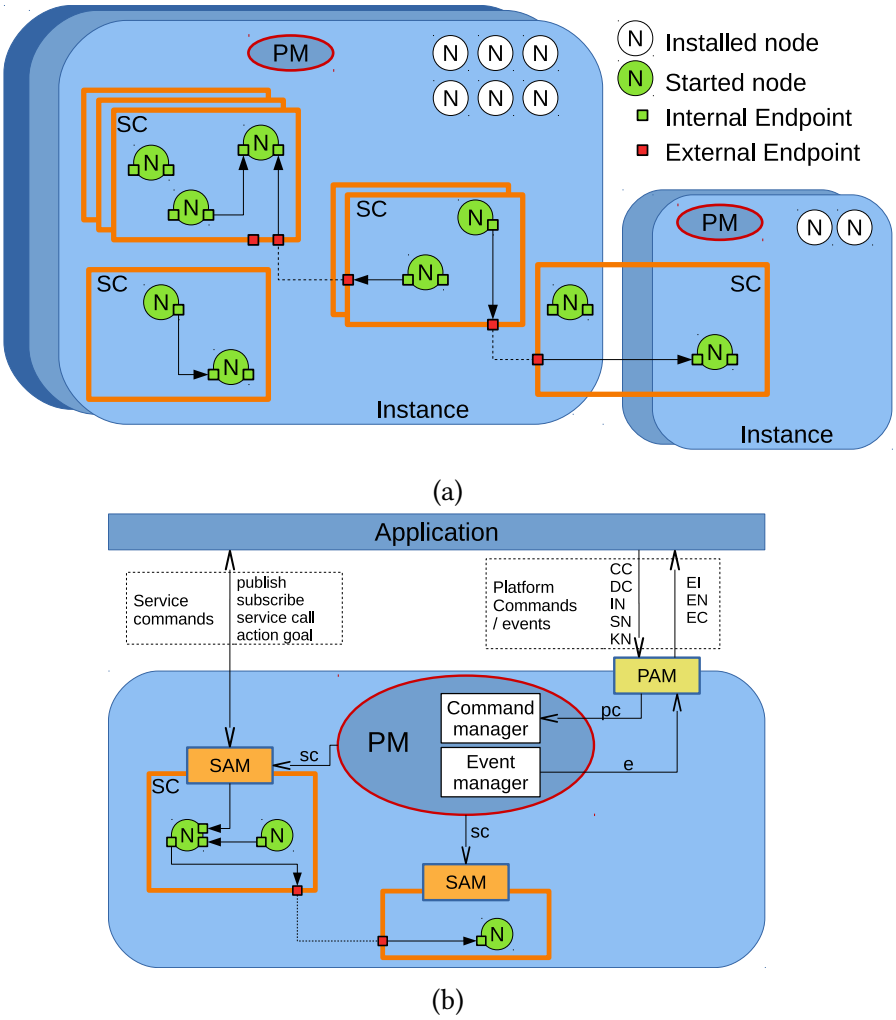


Figure 9.1: The Cloud Robotics Platform (CRP) used in this work. In (a), the main architecture. In (b), the Platform Manager (PM) and the APIs.

174, 178].

The aim of the CRP is to offer to the final user a platform to execute ROS-based robotics applications. Thanks to the CRP, most of the computational tasks can be executed on Cloud, guaranteeing a robust and reliable platform. The CRP is exposed to the user via RESTful APIs.

The CRP is composed of the following objects:

- **Node (N)** is the ROS node, i.e., the basic ROS process. In order to start a ROS node, it needs to be installed on the CRP. Otherwise, the user is able to install new nodes via API commands. The CRP supports both nodes available on the ROS repository and customized developed nodes. Each node is able to communicate to all other nodes in the CRP and to external applications via the Service API Manager.

- **Service Container (SC)** is the environment where nodes are executed. More containers may run on the same Instance and they are isolated from each other. In fact, each container is a ROS environment with an unique ROS Master that manages the ROS communication in the Service Container. However, a container is able to communicate with other containers using External Endpoints.
- **Internal Endpoint (IE)** connects nodes with others in the same Service Container or with External Endpoints.
- **External Endpoint (EE)** connects nodes with others in different Service Containers.
- **Instance (I)** is the main object where the Platform Manager resides. It comprises multiple Service Containers and nodes, but only one PM needs to be allocated in the Instance. An Instance resides on a virtual machine on the Cloud or on a robot.

These objects define the CRP architecture, depicted in Figure 9.1a. The CRP uses this architecture to develop a distributed robotic application based on the computational graph concept defined by ROS. All the Service Containers and Nodes are allocated and managed by RESTful APIs, enabled by the Platform Manager (PM). The Platform Manager architecture is illustrated in Figure 9.1b. The PM is able to manage Commands and Events. In particular, an event advertises if something happens in the CRP. For example, if an Instance allocated on a robot is disconnected to the Cloud, an Event message is provided to the PM connected with the robot. As a consequence, the PM provides to manage and, if necessary, deallocate the objects related to the disconnected robot by sending commands to SCs and to the application. Event messages refer to Instances (EI), Nodes (EN) and Containers (EC) and they are exposed using the Platform API Manager (PAM).

PAM manages also platform commands between the PM and the application. Platform commands are applied to the objects in the Instance, such as Containers with the Create (CC) and Delete (DC) commands and Nodes with the Install (IN), Start (SN) and Kill (KN) messages. The CRP comprises also more other secondary messages, that are not described in this thesis.

The CRP comprises also the Service API Manager (SAM). SAM provides external API to the user/application and manages service commands. Service commands are able to handle the message flow in Containers.

Using the above-described logic architecture, the CRP is able to provide a robust and reliable Cloud Platform for robotics applications. Moreover, CRP guarantees robustness in long-term applications: each ROS node is supervised and, if it crashes, the CRP is able to restart it.

9.2 Robot Courier, a service robot in a workspace

The Robot Courier is a service robotics application with the aim to welcome and provide assistance to visitors in a workspace. The main logic of the proposed application is depicted in Figure 9.2.

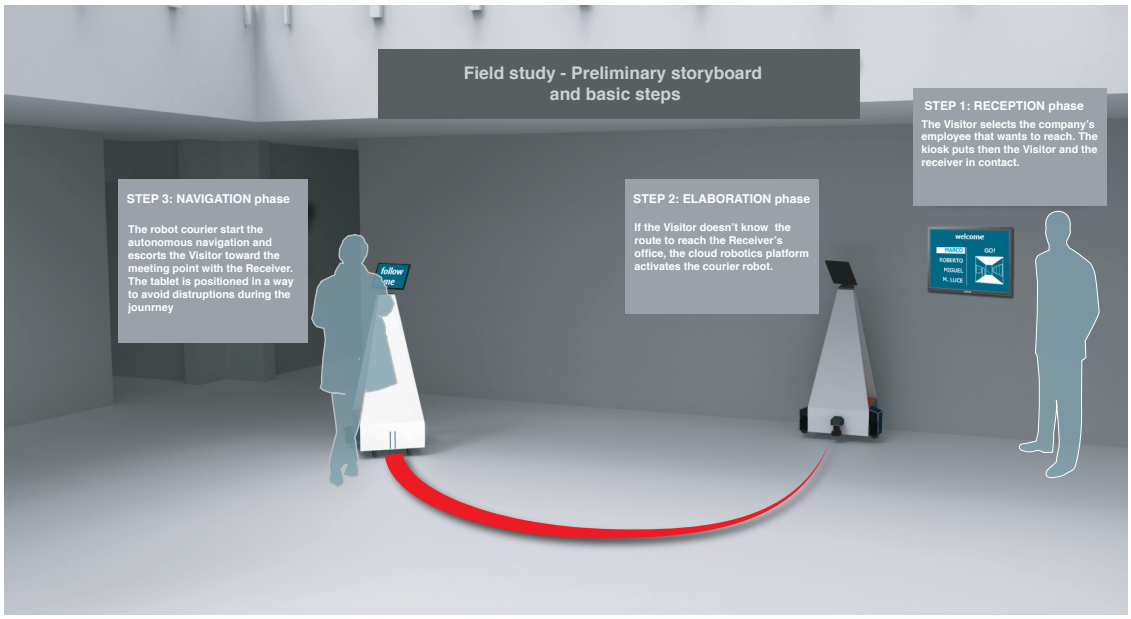


Figure 9.2: The Robot Courier service application.

The Robot Courier service comprises four main actors:

- **Visitor** is a person that visits the building.
- **Receiver** is an employee working in the building, waiting for the visitor. He is equipped with a mobile phone running the Reference App developed to provide the Robot Courier service.
- **Reception device** is used by the visitor to request the Robot Courier service. It is a tablet placed in the reception of the building on a self-service kiosk.
- **Robot Courier** is the mobile robot that escorts the visitor to the receiver. On top of the robot, a tablet is mounted with the Robot Courier App.

When the visitor arrives in the reception, he finds the Reception device. The visitor needs to compile the electronic form, defining his generalities (at least the full name) and selecting the receiver person in the building. A notification arrives on the receiver's phone, using the Receiver App, and he is able to accept or refuse the visitor. In case of not answer, after a specified time-out, the application notifies that the receiver is

not available. If accepted, the application asks to the visitor if it is able to reach the receiver's office. In the affirmative case, the visitor is able to reach the office by himself, otherwise, the robot Courier escorts him to the right office. For this purpose, the robot moves toward a *ready position* near the Reception tablet, waiting for the start command. The visitor can use the tablet mounted on the robot Courier to interact with the robot. The robot Courier is able to navigate toward the desired office autonomously, with the support of the Cloud Robotics Platform. If something happens, the visitor can stop the robot and he is able to request assistance. A notification arrives on the receiver's phone, which can help the visitor. The above mentioned storytelling is reported in Figure 9.3.

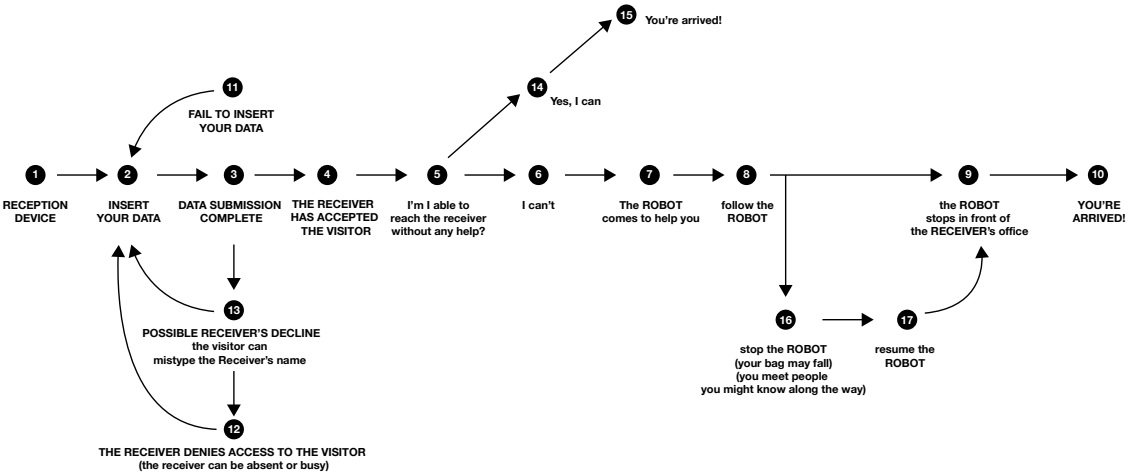


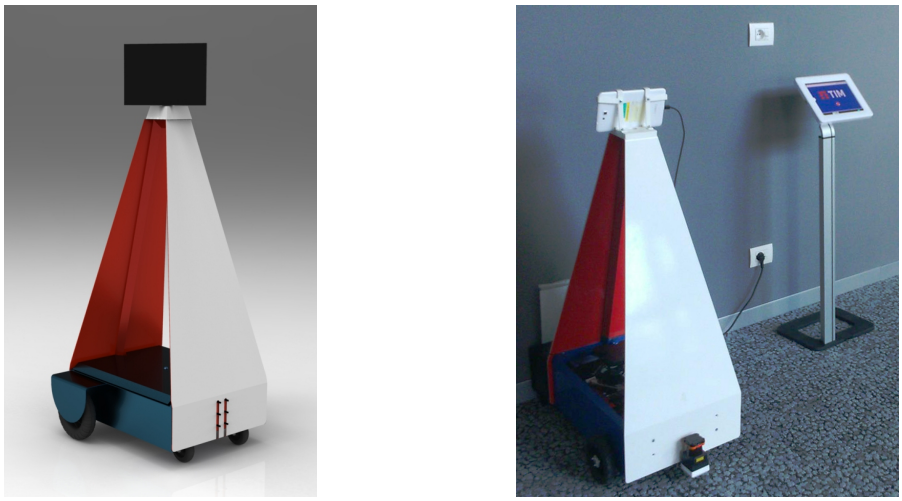
Figure 9.3: The flow chart of the Robot Courier application.

9.2.1 Devices

Robot Courier

The robot Courier is a mobile robotic platform designed for indoor navigation. It is a differential drive robot with two active wheels in the front and two passive caster wheels on the rear. It is equipped with an internal Inertial Measurement Unit (IMU), a Hokuyo UTM-30LX laser range sensor, a HD 720p USB camera module and wheel encoders. The prototype of the robot Courier is depicted in Figure 9.4a.

The robotic platform is developed by Nuzoo Robotics [147]. It is a fully ROS compatible platform developed using the R2P (Rapid Robot Prototyping) modules [21]. R2P is an open source hardware and software modular approach for robot prototyping. R2P provides hardware modules, able to perform real-time communication between hardware and software. There are some module types, such as the DC motor controller module and the IMU module used on the robot Courier. Modules communicate using a bus, while a Gateway module enables direct integration of R2P modules with ROS.



(a) (b)

Figure 9.4: The robot Courier. In (a), the render of the design project of the robot. In (b), the prototype of the robot Courier docked in the reception. Close to the robot is located the Reception device.

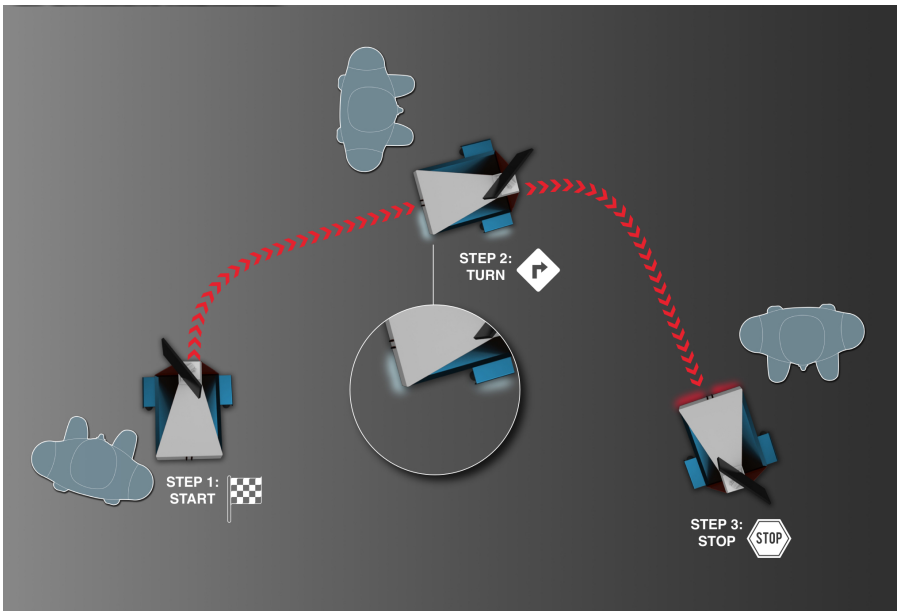


Figure 9.5: The turn and stop signals used in the Robot Courier service application.

The cover of the robot is specifically designed for the Robot Courier service. In particular, the shape of the cover and its characteristic were defined after a study, considering the usability and how the robot is perceived by people, as well as to reduce the oscillation on the robot's top part. In fact, concepts of perception and acceptance are

central in the designing of socially interactive robots [40]. For more details about the design process, please refer to [103].

The render of the project is depicted in Figure 9.4b. The robot comprises a tablet mounted on top to interact with the visitor. As depicted in Figure 9.5, the tablet is oriented in order to keep the visitor in a safe position. In fact, in case of sudden braking, if the visitor is exactly behind the robot, a collision between them can occur.

The Robot Courier App is installed on the tablet, used to communicate with the visitor. Using the application, the visitor is able to provide the start command to the robot, to stop it, to resume the navigation or to send it to the home position. The Robot Courier App is an Android application and communicates with the CRP with the Service API Manager using the LTE connection. The Graphical User Interface (GUI) of all applications used in this service is designed in order to minimize the time to train the people to approach the user's interface [103].

The robot Courier uses light signals in order to notify some motions and to provide a form of feedback to surrounding people. The use of lights can be used by the robot to interact with humans. An example can be found in [13]. Similarly to the turning signals on the car, the robot blinks the left or right turning signal when it has the intention to execute a sharp bend. If it performs a rotation in place, all turning signals blink. When the robot slows down and stops, it switches on the stop signals. If the robot is blocked because malfunction or failure in the autonomous navigation, the stop signals blink, similarly to the hazard lights on the car. Figure 9.5 illustrates an example of use of lights.

Reception device

The Reception device is placed in the reception of the building on a self-service kiosk. It is an Android tablet with a specific mobile application for the Robot Courier service, called Reception App.

The Reception App needs to be friendly and easy to use because the visitor uses it to set up the Robot Courier service. For this purpose, the Reception App is a wizard that guides the visitor to compile a registration form and select the receiver person in the building.

The Reception device used in this work is illustrated in Figure 9.4, while some screen of the Reception App are depicted in Figure 9.6.

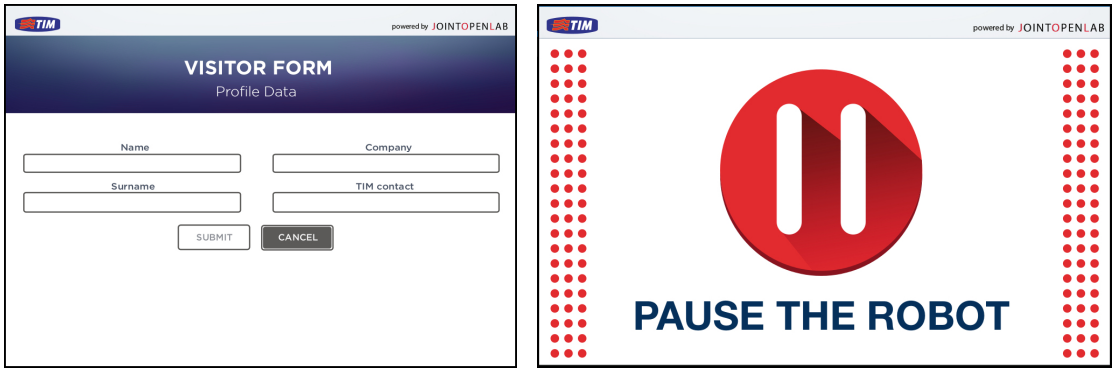
The Reception App is developed in Java, and it is able to communicate with the Cloud Robotics Platform using the Service API Manager and with a 4G Internet connection.

Receiver App

The Receiver App is installed on the mobile phone of the receiver. The receiver uses the mobile application to accept or decline the request of the visitor to reach the

receiver's office. The receiver is also alerted when the visitor stops the robot and requires help, or when the robot is blocked because of some malfunctions.

Similarly to other mobile applications used in this work, the Receiver App is developed in Java for Android and it is able to communicate with the CRP using the Service API Manager. The mobile device needs to be connected with the Internet using a 4G or Wi-Fi connection.



(a) (b)

Figure 9.6: In (a), a screen of the Reception App, while, in (b), the screen of the Robot Courier App.

9.2.2 Cloud-based service

The robot Courier service is based on the Cloud-based architecture presented in Chapter 6. Each layer and module in the architecture is customized according to the service requirement. In the next paragraphs each layer is described in detail.

Application layer

The application layer aims to manage the Robot Courier service. As described in Chapter 6, it is composed of three elements: the Service Manager, the Application Manager and the Application.

The Service Manager manages the Robot Courier Service. In particular, the mobile robot behavior is scheduled according to the state machine depicted in Figure 9.7.

Generally, the robot is in the docking, in order to maintain a full-charged battery. When the visitor requires the Robot Courier service to reach the receiver's office, the robot moves in the ready position, placed close to the Reception device. Then, the robot waits for the start command to be pushed by the visitor, who should use the on-board tablet to access the service. If the start button is not pushed in a reasonable time-out, the robot comes back to the docking. On the opposite, if the start command is pushed, the robot moves toward the target position, i.e., to the position of the selected office. The

visitor is able to stop the robot using the pause button. Then, there are three alternatives: *(i)* the resume button is pressed to continue the Robot Courier service, *(ii)* the visitor pushes the home button to come back to the reception, and *(iii)* no actions are executed in a time-out, then, the robot returns to the reception by itself.

Once the desired office position is reached, the interaction between visitor and robot ends and the robot returns to the reception. If there is a new visitor at the reception, the robot navigates directly to the ready position, otherwise, it moves toward the docking station.

When the service robot is continuously requested, the battery level of the robot discharges. In order to avoid the unexpected switch off of the robot, a minimum level of battery charge is imposed. If the level of the battery is lower than a threshold, the robot remains in docking and the Courier service returns available only when a minimum battery level is guaranteed.

If something wrong happens during the autonomous navigation of the robot Courier, the robot enters in a recovery mode, trying to restore the navigation. If it is impossible to reach the target pose because of the presence of fixed obstacles, the robot switch to the pause state, it alerts the receiver and waits for the home command or a time-out. Otherwise, if there are major issues that stuck the robot, it assumes the blocked state and the Robot Courier service needs to be restarted.

The Service Manager relies on a database with stored all the people reachable by the Robot Courier service, with associated the positions of their offices.

Based on the status of the Service Manager, the Application Manager exchanges data with the Application. The communication between the Application Manager and the Application is provided by a 4G mobile network using the Service API Manager.

The Application block comprises all the mobile applications used by the Courier service, i.e., the Receiver App, Robot Courier App and the Reception App.

Thanks to the scalability of the Cloud, the Application layer is able to work with a fleet of robots. Anyway, in this work, we assume only one robot Courier.

Navigation Layer

The Navigation layer provides autonomous navigation in order to reach the desired office position, given by the Service manager. According to the architecture of Chapter 6, the Navigation layer consists of several modules. In the following paragraphs, the implementation of each module in the Robot Courier Service is described.

Global planner The global planner computes the global path from the robot pose to the desired target pose. As inputs, it receives the target position from the Service Manager, i.e., the selected receiver's office, and the current pose of the robot from the Localization.

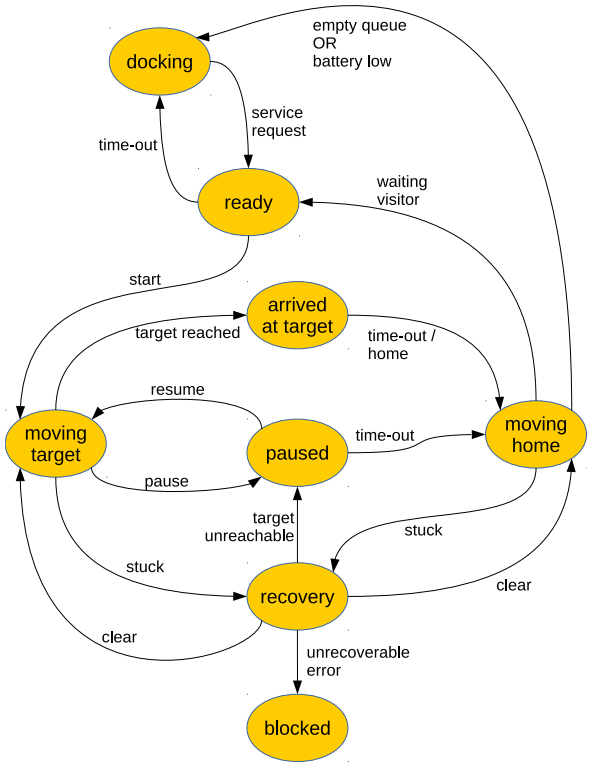


Figure 9.7: The state machine used to provide the Courier service.

In this work, the global planner is provided by the Optimal Rapidly-exploring Random Tree (RRT*) algorithm [97] that explores the search space, i.e. the map, by generating an optimal tree. The motion cost in the exploration tree is computed by trading off the path length and the path clearance to obstacles. Often, if the path planning algorithm minimizes the path length, the resulting trajectory can be dangerous because the robot could move close to obstacles and people. Generally, it is impossible to minimize the path length and maximize the clearance in the same time. For this reason, we define a simple cost function

$$\text{cost} = \delta_1 \cdot \text{length} + \delta_2 \cdot \text{clearance}, \tag{9.1}$$

where the δ_1 and δ_2 parameters balance the cost function.

The global planner computes the global path when the Service Manager defines the goal pose. Moreover, if the local planner notifies that it is impossible to follow the path because of obstacles, the global planner seeks for a new path in the updated map.

Since the network delay does not have an effect on the global planner performances, the global planner is executed on the Cloud.

Local planner The local planner executes the global path, controlling the mobile robot and avoiding obstacles using the on-board sensors.

In this work, we use the motion control method called Particle Filter Model Predictive Equilibrium Point Control (PF-MPEPC), presented in Chapter 8. It is based on a Model Predictive Control method with the integration of the Equilibrium Point Control approach. Moreover, it evaluates uncertainties due to disturbances and measurement noise by using two particle filters.

The local planner executes the global path sequentially. Practically, the target pose assigned to the local planner is a pose of the global path at a constant distance to the robot. The target of the PF-MPEPC changes at every step according to the global path, until the global goal pose is reached. If it is impossible to follow the path because of obstacles, the local planner reports it to the global planner, which seeks for a new global path.

In this work, the local planner is executed on-board, in order to be robust to network delay and disconnections with the Cloud.

Recovery behaviors The recovery behaviors are specific actions performed by the robot in order to recovery the ordinary condition of autonomous navigation. If something wrong happens, the autonomous navigation enables the recovery behaviors. Generally, it happens when the presence of obstacles around the robot obstructs the robot motion in every direction, or it is impossible to find a global path to reach the desired goal. The recovery behaviors used in the Robot Courier service are defined by ROS in the *move_base* package, where four specific actions are performed by the robot: *(i)* the robot's map is cleared outside of a user-specified region, *(ii)* the robot rotates in-place in order to update and clear the map, *(iii)* all robot's map is cleared and *(iv)* another rotation in-place is performed. These four actions are performed in sequence. If after the first action the robot remains stuck, the second action is performed, and so on. If after the last action the robot is still stuck, the robot state turns to blocked, and the receiver is alerted with a notification.

Localization The Localization estimates the robot pose in a known map, based on on-board sensors.

We use the Adaptive Monte Carlo localization (AMCL) algorithm, the default localization algorithm used by ROS in the navigation stack. It is based on the well know Monte Carlo Localization method [43], one of the most popular localization algorithms in robotics, that recursively estimates the posterior about the robot pose with particle filters, using on-board sensors. In particular, the robot Courier uses the laser scanner and the odometry data, in order to perceive the environment and the robot motion.

The Localization block resides on the robot, as well as the local planner, that needs the pose of the robot.

Velocity Manager The Velocity Manager filters the velocity commands to the mobile robot. Based on the state of the robot, the proper velocity commands are published to

the locomotion driver of the mobile robot.

Docking The Docking block provides the movement toward the docking station. When the robot Courier changes the state in docking, it moves toward the docking pose, using the combination of global and local planners. The docking pose is positioned in front of the docking station, oriented in the opposite direction of the docking station. Then, the Docking node controls the robot movement toward the docking station, using the rear camera to track the docking station.

The rear camera of the robot is used to detect, identify and track a marker located on the docking station, using the ARToolKit [84]. ARToolKit is an open-source library for Augmented Reality (AR) applications. The pose of the marker is estimated, then, the docking node provides to motion toward the docking station and docks the robot. When the robot is in charge, the docking node stops immediately the robot, which is in the docking state.

Figure 9.8 illustrates the docking station with the marker, as well as the robot with the rear camera used to track the marker.

Connection Diagnostic The Connection Diagnostic detects disconnections and monitors the delay in communication. When a disconnection occurs, the Connection Diagnostic notifies the local planner and the Service Manager. The local planner stops the robot until the connection is resumed. If the connection is not resumed in a certain amount of time, the Service Manager blocks the Service and needs to be resumed.

The network delay implies that the messages exchanged with the Cloud arrive after a delay. It does not involve autonomous navigation because the control and localization blocks are on-board. However, if the delay is too large, the quality of service decreases, because the service needs to be reactive to the visitor's commands. For instance, when the visitor stops the robot, the robot stops after a delay. For this reason, when the delay is greater than 400 ms the Robot Courier service is unavailable, similarly to disconnections.

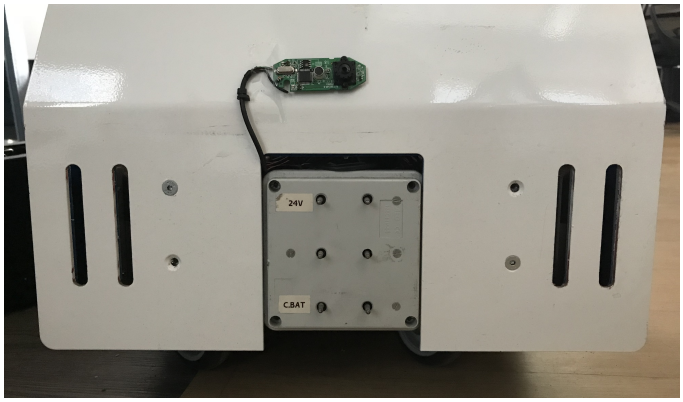
Hardware Layer

The Hardware layer represents the mobile robot platform including both hardware and software required to manage actuators and sensors. All blocks in the Hardware layer interface the hardware with the software.

The Locomotion driver sends velocity commands to the wheel motors, while the Laser scanner driver and the Camera driver provide sensor data.

The odometry block estimates the robot motion thanks to wheel encoders and the IMU (Inertial Motion Unit) sensor. The sensor fusion between them is performed using the *robot_localization* package. *Robot_localization* implements some state estimation approach, such as the unscented Kalman filter (UKF) and the extended Kalman filter (EKF) [138]. In particular, in our work, we use the EKF approach.

(a)



(b)

Figure 9.8: In (a), the docking station with the identification marker, while, in (b), the camera installed on the rear of the robot, used to track the marker.

9.2.3 Experimental results

In this section, results obtained with experimental tests in a real environment are exposed. Tests are conducted in a corporate building in Turin, Italy, suitable to test the Robot Courier service.

Mapping The mapping step is mandatory to generate the map of the navigation environment. This step is executed only once because after the mapping phase, the map is stored on the Cloud.

The mapping is performed using the *gmapping* ROS package. It is based on a state of the art grid mapping algorithm that uses Rao-Blackwellized particle filters [73].

The map is generated by teleoperating the robot in the navigation area. The map created by the mapping procedure is illustrated in Figure 9.9. The map has a resolution of 0.05 m/pixel and a dimension of about 51×26 m. Due to the non-disclosure agreement with TIM, the map depicted in Figure 9.9 has been partially edited, in order to modify

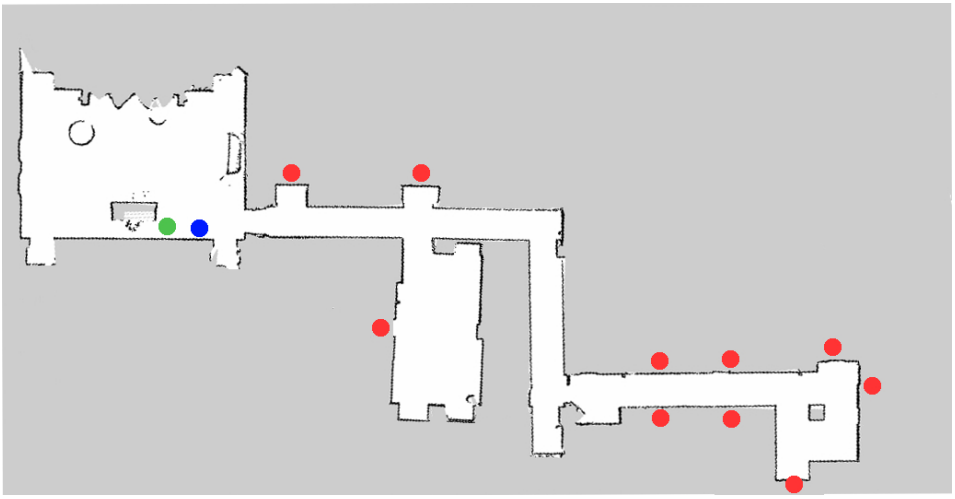


Figure 9.9: The map created with the gmapping ROS package. On the map, the positions of offices are marked with red circles, the docking position with the blue circle, while the position of the Reception device with the green circle.

the real planimetry of the building.

Localization The AMCL algorithm is tested in the map generated in the previous section. The AMCL package is quite easy to use. However, in order to have good performances, parameters need to be tuned according to the robot and environment characteristics. Figure 9.10 reports the robot Courier localized in the map.

Global Planner The global planner exposed in the previous Section is developed as a plugin in the *move_base* node in ROS. In particular, it is implemented in C++ using the Open Motion Planning Library (OMPL) [193].

Given the map, the robot position and the desired goal, the RRT* algorithm is able to find the global path by trading off the path clearance and the path length. Figure 9.10 illustrates an example of the global path.

Local Planner The motion planner based on the PF-MPEPC method is developed as a local planner plugin in the *move_base* package in ROS. The PF-MPEPC requires a tuning phase, where all the parameters are defined in order to have good performances and smooth motion. Using the same notation of Chapter 8, the parameters of PF-MPEPC are reported in Table 9.1: h and z are the constant values used to adjust the control law; λ_1, λ_2, λ_3, λ_4 are the constant weighting factors used in the cost function $\boldsymbol{J}$; T_h is the time horizon; f_c is the controller frequency and H_p is the prediction horizon.

Figure 9.11 illustrates the robot Courier, which navigates autonomously avoiding two people using the motion controller based on the PF-MPEPC approach.

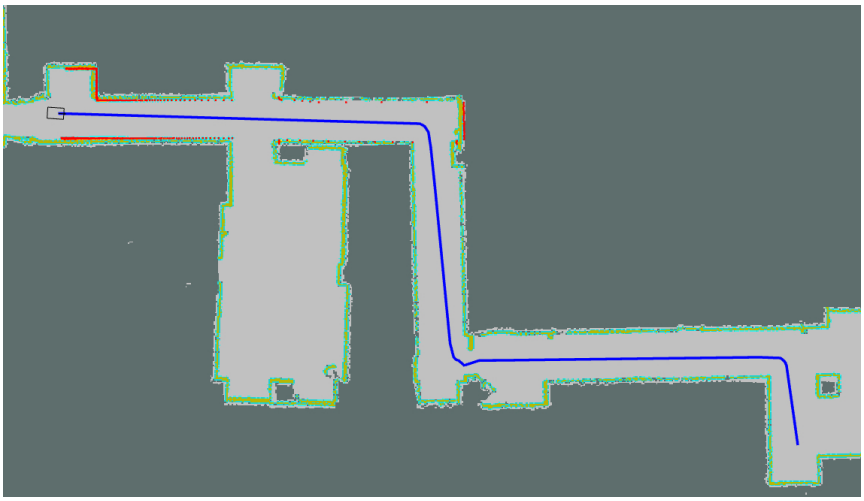


Figure 9.10: An example of the global planner used in the experiment. In blue the global path computed with RRT*, while the robot is represented with a black rectangular. The robot is localized in the map, where red lines are the laser scanner readings.

Table 9.1: The PF-MPEPC parameters

PF-MPEPC parameter	Value
γ	0.2
h	0.5
z	0.7
λ_1	3.0
λ_2	0.1
λ_3	0.5
λ_4	0.7
T_h (s)	4.0
f_c (Hz)	5.0
H_p	20

Robot Courier Service The Robot Courier service is tested in a real workspace. As illustrated in Figure 9.4, the robot, the docking station and the Reception device are placed in the reception of the building.

The Robot Courier service is tested continuously for two entire weeks with real visitors during ordinary working days, with an average of six visitors each day and a total number of 60 people escorted to specific destinations. The robot was activated in precise time spans when the reception was, on average, more congested, between 10.00 to 12.00, and between 14.00 to 16.00. The participants were selected at the entrance door, where the self-service kiosk with the Reception device is placed.

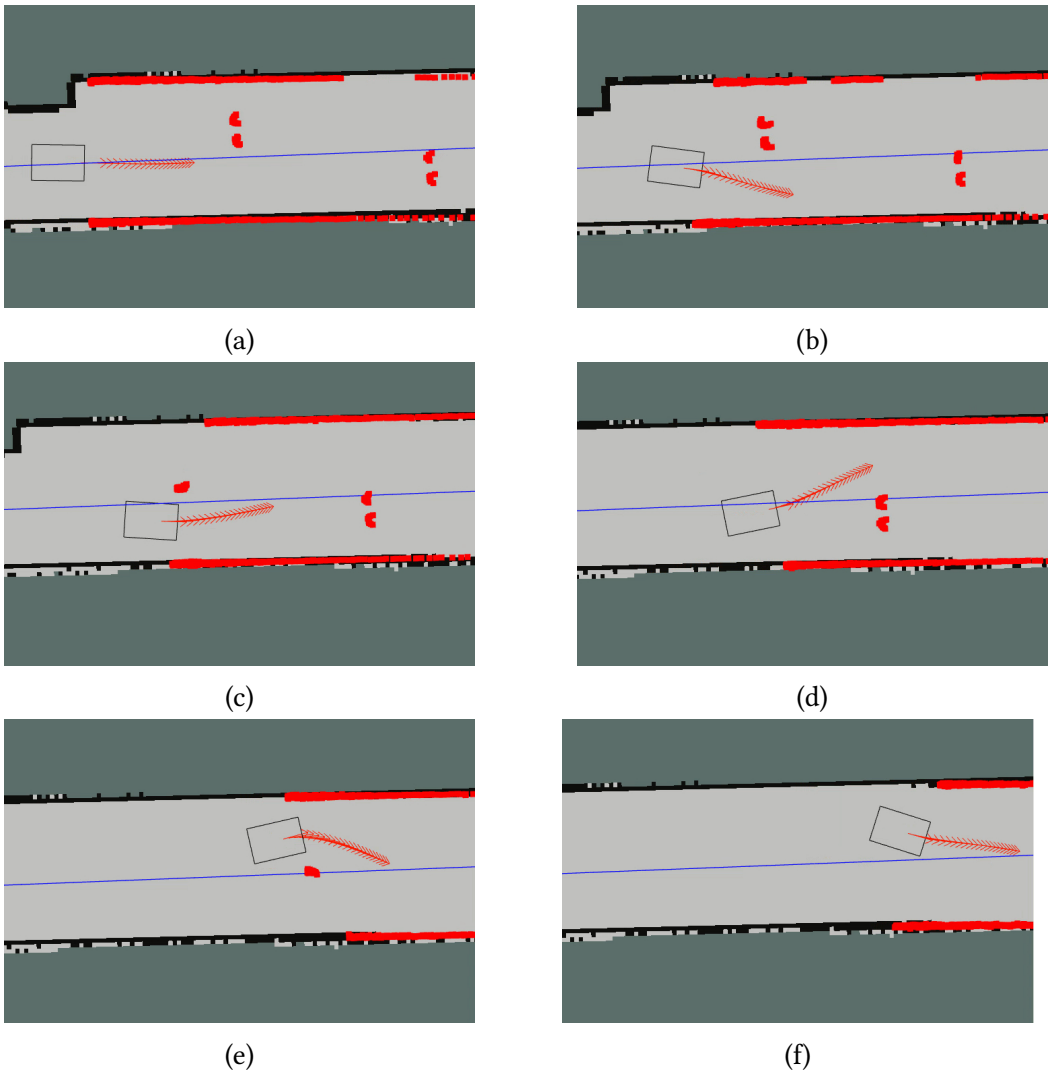


Figure 9.11: Example of autonomous navigation performed by the robot Courier using the PF-MPEPC approach. The robot follows the global path (blue line) avoiding obstacles detected with the laser scanner (red lines). The optimal trajectory computed by the MPC based motion controller is depicted with red arrows.

During the test, the service has worked without any malfunction and without requiring human assistance to unblock the robot.

Moreover, during the experimentation phase, some interesting scenarios are tested, in order to verify the robustness of the service robotic application:

1. the visitor asks for help to the robot Courier to reach the reference office, but he doesn't press the start button. According to the state machine, the robot remains in the ready state, until the time-out ends. Then, the robot returns in the docking state.

2. the visitor follows the robot and press the stop button, without resume the navigation. After the time-out, the robot returns to the docking autonomously.

3. the door between the reception and the corridor of access to offices is closed. The robot is not able to reach the reference office, because it detects an obstacle and an alternative path does not exist. As a consequence, the robot enters in the recovery mode. Once it reports the impossibility to reach the desired position, it enters in pause state and asks for help to the receiver. Then, the receiver helps the visitor, and the robot returns to the docking.

4. The robot is obstructed by fixed obstacles and moving people. The motion controller is able to avoid obstacles with a smooth and safe motion.

5. A network delay of 500 ms is manually set between the robot and the Cloud. Once the delay is detected by the Connection Diagnostic, the robot stops, until the delay returns in ordinary conditions.

These are common events tested during the test phase. The robot behavior is acceptable because the system is able to recover the navigation or detect the problem and notifies it. The robot is always able to avoid collisions with obstacles and people, performing safe navigation with smooth motion. In particular, the smooth motion is an important feature, because the visitor follows the robot. In fact, sudden acceleration and braking may have a negative impact on the visitor. Moreover, the use of light signals is very helpful for the visitor to predict the robot motion.

Thanks to the simplicity of the Robot Courier service, the visitor is always capable to set up and use the robotic service. The GUI of mobile applications are intuitive and easy to use.

In particular, all the visitors report a positive evaluation of the service and they ”accept” the presence of the robot in a workspace. In fact, we have asked to people that have used the Courier robot service to complete a questionnaire about the user experience. For more details refer to [103].

Although the proposed service is applied inside an office workspace, it’s important to consider that this positive experience could be beneficial for other public contexts such as airports, hospitals, train stations.

9.3 Virgil, a robot for museum experience

Virgil is a telepresence robot designed for improving the museum experience. In particular, Virgil is developed for a specific museum: the royal residence of *Racconigi Castle*, in Italy. This castle was a holiday residence of the Savoy royal family. Anyway, due to the dimensions of the building and for safety reasons, some rooms are excluded from the visit. In fact, because of the state of conservation and fragility, more than the 60% of the royal residence remains inaccessible to visitors during the guided tour.

Virgil is used to offering a service in order to extend the museum tour through a real-time virtual tour. In particular, the virtual tour is performed by the Virgil robot able to move autonomously and providing a real-time video in high definition. Moreover, the museum guide assumes an essential role in the service robotics application, because, in addition to the standard tour, it describes and explains the virtual tour, as well as controlling the Virgil robot by selecting the desired position in the museum and manually controlling the orientation of the pan/tilt camera.

This service robotics application implements the concept of human-robot collaboration. In fact, the museum guide performs a storytelling activity with the support of the Virgil robot that has the role of a remote collaborator. It is important that the storytelling activity is performed by the museum guide because only a human is able to provide the interpretative aspect. In fact, according to [131], the interpretation is the process that provides a link between the visitor and the cultural heritage contents.

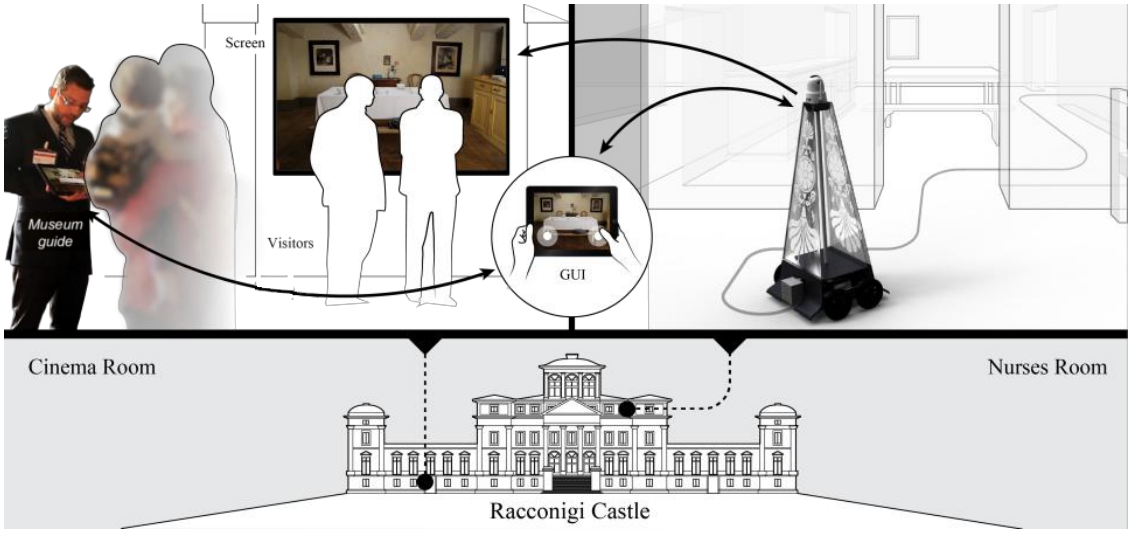


Figure 9.12: The Virgil robot service. The robot operates in the Nurses rooms by transmitting a real-time video to the cinema room, where visitors resides. The museum guide uses the guide device to interact with the service.

In summary, the Virgil service has the following actors:

- **Visitors**: are people that visit the museum;
- **Museum guide**: is the person that provides the storytelling activity, supported by the Virgil robot using the *guide device*;
- **Guide device**: is the device with the GUI (Graphical User Interface) used by the museum guide to interact with the Virgil robot;
- **Wide screen**: is the device where the real-time video of the virtual tour is shown to visitors.

At the end of the standard guided tour, visitors arrive in the cinema room of the Racconigi castle. Here, the museum guide provides a storytelling activity with the support of the Virgil robot that streams a real-time video in HD on a wide screen. The museum guise uses the *guide device*, i.e. a tablet, to interact with the Virgil robot. The guide device has a GUI with some control buttons in order to define the target position to be reached by the robot autonomously or to manually move the camera mounted on-board the robot. Figure 9.12 illustrates this scenario, where the Virgil robot operates in the *Nurses rooms*, the area where the Virgil robot is tested.

When the virtual tour starts, the robot is in the docking position. Hence, using the guide device, the museum guide selects sequentially target positions according to the storytelling. When the robot reaches the desired position, the guide is able to move the robot in the teleoperation mode, in order to explore the area locally. Otherwise, a pan/tilt camera can be moved to frame an area with more accuracy. Anyway, the teleoperation mode has some safety capabilities. In fact, the motion controlled by teleoperation is stopped when the robot is too close to obstacles detected using a laser scanner.

9.3.1 Devices

Virgil robot

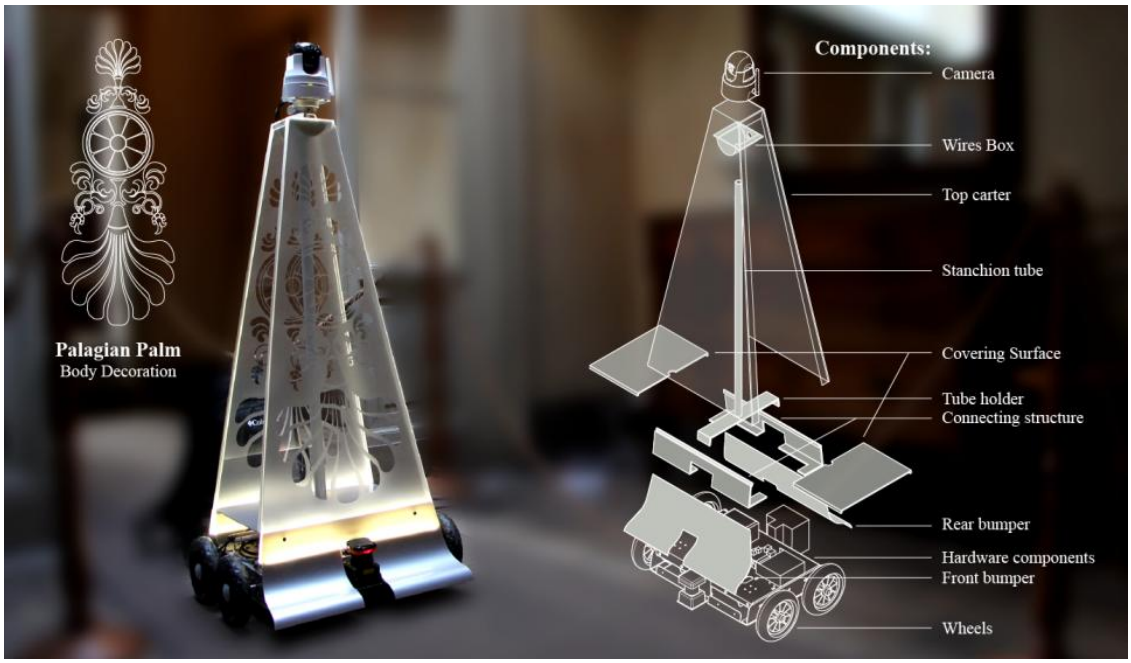


Figure 9.13: The Virgil robotics platform.

Virgil is a mobile robot with four active wheels with a differential drive. It is designed for indoor environments and it is able to perform autonomous navigation in

structured environments. It is a fully ROS compatible platform and it can be used to perform several applications.

As reported in Figure 9.13, the structure is designed in a pyramid shape and with a height of 120 cm. Two electric motors provide the motion of wheels. In particular, each motor provides traction to each side independently, with the consequent differential drive configuration.

The robot autonomy is about 8 hours thanks to a 12 V Li-Fe battery.

It has several sensors, such as wheel encoders to provide odometry data, a laser range finder Hokuyo UTM-30LX and a DCS-5222L pan/tilt camera. The hardware architecture is reported in Figure 9.14. The Electronic Control Unit (ECU) manages the robot motion, as well as transmitting serial data from motor encoders to the on-board CPU. The LTE dongle provides communication with the Cloud Robotics Platform. The on-board CPU is a mini PC NUC DN2820FY with a CPU Intel Celeron N2820 dual-core at 2.1 GHz. The on-board PC exchanges data with sensors using USB ports with the exception of the pan/tilt camera connected using an Ethernet cable.

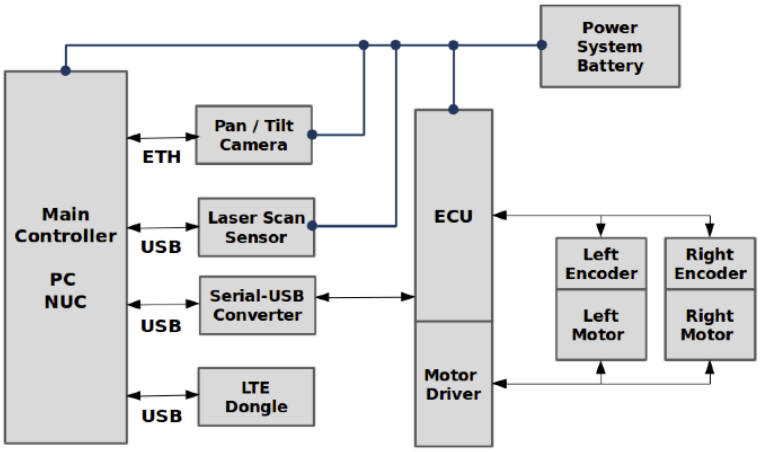


Figure 9.14: The hardware architecture of the Virgil robot.

The design of the robot is the result of an interesting study presented in [69, 126]. The robot should be unobtrusive and homogeneous with the context. The cover is made with poly-methyl-methacrylate with the shape of a truncated pyramid, reminding to the similar shape diffused in Savoy tradition. The cover has a decorative pattern that represents the *Palagiana* palm, an already existing decorative motif already applied in the castle on many elements, such as floors and furniture. Virgil and the *Palagiana* palm are shown in Figure 9.13.

Guide Device

The Guide device is used by the museum guide to provide the Virgil robot service. Practically, the Guide device is a tablet with a 4G connection, used to communicate with the CRP.

The museum guide uses the device using a Web-app with a Graphical User Interface (GUI). The prototype of the GUI is reported in Figure 9.15 and it consists of the following elements:

1. The streaming video, i.e. the real-time video stream from the pan/tilt camera on-board the Virgil robot;

2. The laser-image. It is the view of the laser range finder sensor. It is an essential element during the teleoperation of the robot because it allows obstacles to be avoided.

3. Teleoperation controller. Using these buttons the museum guide is able to move the robot in teleoperation mode;

4. Assistance panel. It provides assistance during the navigation, such as the stop button to immediately stop the motion of the robot, and the obstacle panel, that reports when obstacles are too close to the robot.

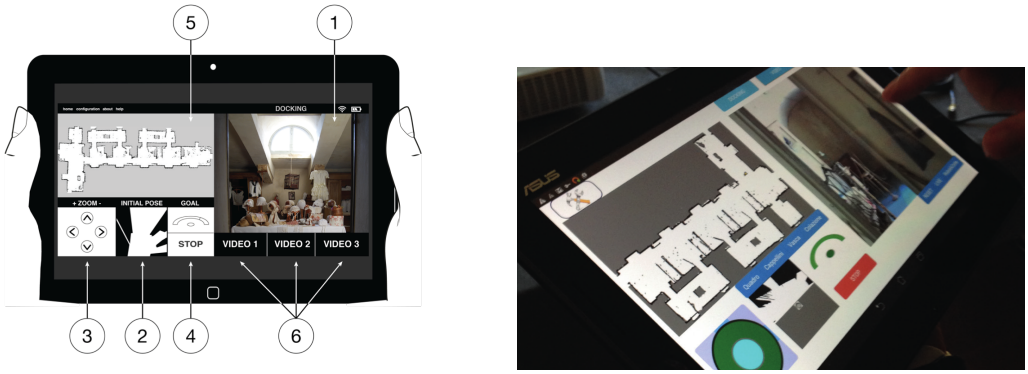
5. Map. It is the map of the explored area. Using this element the guide is able to define the desired goal to be reached, as well as determines the actual pose of the robot in the map.

6. Multimedia buttons. These buttons allow to multimedia contents to be activated and streamed on the wide screen.

In fact, during the storytelling activity, the museum guide uses the GUI to select the desired position to be reached autonomously by the robot. During the robot motion, the real-time video is always projected on a wide screen to visitors. Moreover, when the robot is not moving in autonomous mode, the museum guide is able to manually controls the robot with the teleoperation mode, by using the teleoperation controller on the GUI, as well as referring to the laser image and the assistance panel to sense the environment and avoid obstacles. Moreover, if the robot loses the localization in the map, the museum guide, using the *initial pose* button, is able to define the actual pose of the robot on the map, recovering the localization.

During the storytelling activity, the museum guide shows some multimedia contents to ensure a more exciting presentation.

The GUI is designed in order to be comfortable and easy to use. More details are explained in [69].



(a) (b)

Figure 9.15: Left panel: the prototype of the Graphical User Interface (GUI) designed for the Virgil robot service. Right panel: the real GUI used in the experimental tests.

9.3.2 Cloud-based service

The Virgil robot service is based on the Cloud-based architecture presented in Chapter 6.

The application layer manages the service, by interpreting the commands from the GUI of the *guide device* and determining the target pose to the navigation layer, as well as reporting feedback messages on the GUI.

The navigation layer provides autonomous navigation. According to the architecture of Chapter 6, autonomous navigation consists of two main elements: global planner and local planner. The global planner is implemented using the dynamic path planner introduced in Chapter 7, based on the Informed-RRT* algorithm. Thanks to this dynamic path planner the robot has always a valid and safe path in order to reach the desired target pose.

Instead, the local planner is provided by the Enhanced Vector Field Histogram (VFH+) algorithm [202]. VFH+ is an efficient motion controller algorithm able to follow the path provided by the global planner and avoiding unexpected obstacles. In particular, VFH+ provides a safe and smooth motion. Since the robot transmits a real-time video, it is essential to have a smooth motion, in order to provide a stable and comfortable video.

Both global and local planner relies on the current pose of the robot, defined by the localization. In particular, the localization is provided by the AMCL algorithm [43].

The velocity manager filters the velocity commands to the robot. In fact, in the Virgil robot service, the motion can be provided by two different modes: autonomous and teleoperation. Based on the current state of the robot, Virgil is controlled by autonomous navigation algorithms or by the teleoperation commands from the GUI. Moreover, if the robot is too close to an obstacle detected by the laser range finder, the velocity manager immediately stops the robot, avoiding the collision.

The docking system moves the robot to reach the docking station and to recharge

the battery of the robot. With this service the docking system is very simple: the robot moves autonomously toward a predefined pose in front of the docking station, hence, it turns in place of 180° and moves rearward until the robot is in charge. This docking system is more simple compared with the docking system of the robot Courier, where the rearward motion is controlled by using a camera.

The connection diagnostic verifies the connection between the robot and the Cloud using the same logic described in Section 9.2.2. Anyway, in the Virgil robot service, all the modules of the navigation layer reside in Cloud, with the exception of the velocity manager. The main reason for this choice is the good 4G signal in the navigation area. In fact, thanks to the support of TIM, a dedicated 4G antenna is placed in proximity of the Racconigi castle.

The hardware layer consists of the robotic platform, including motors and sensors drivers.

9.3.3 Experimental results

The presented Virgil robot service is tested in the Racconigi castle with real visitors. In particular, according to Figure 9.12. the robot operates in the Nurses rooms, while visitors participate at the real-time virtual tour in the cinema room, where the museum guide performs a storytelling activity.

In order to provide autonomous navigation in a structured environment, the map of the navigation environment is generated using the same approach described with the robot Courier service. Hence, the map of the Nurses rooms is reported in Figure 9.16, obtained with the Gmapping algorithm [73].

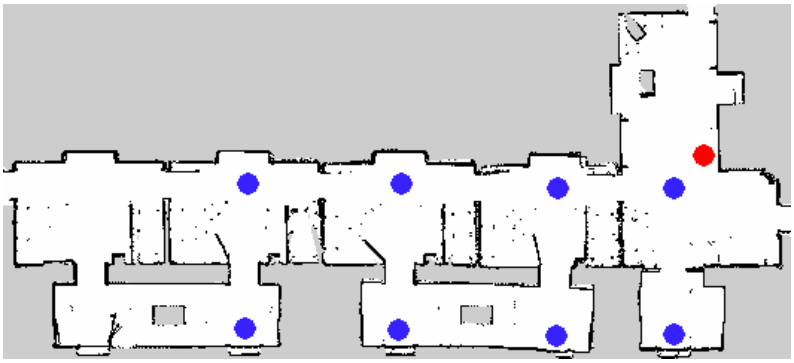


Figure 9.16: The map of the Nurses rooms at the Racconigi castle generated using the Gmapping algorithm [73]. The red circle is the docking position, while the blue circles are the predefined target positions generally used by the museum guide to provide the real-time virtual tour.

Before performing the test with real visitors, the autonomous navigation with Virgil is tested evaluating the performances of localization, global and local planning, as well

as the response time of the Cloud-based service through the Cloud Robotics Platform. In particular, because of the good 4G connection, the average network latency is about 80 ms with maximum values of 150 ms, enabling the Cloud-based control.

Using the combination of global and local planner the robot is able to move autonomously in the navigation environment, avoiding unexpected obstacles detected with the laser range finder.

Moreover, a training phase for the museum guide is performed, in order to take confidence with the Guide device and with the whole Virgil service. Anyway, the GUI is designed to be easy to use and with intuitive controls.

Finally, the Virgil robot service is tested with real visitors. At the end of the guided tour, the museum guide escorts visitors in the cinema room, specifically arranged for this service. In fact, the room is equipped with a projection system and a sound system, connected with the CRP using the Internet, while the museum guide interacts with the service and, as a consequence, with Virgil, using the Guide device, i.e. a tablet with the GUI. During the storytelling activity, the museum guide alternates the real-time streaming video with some multimedia contents, such as slideshows of historical pictures or historical videos.

The real-time virtual tour is performed 10 times distributed in several days. The robot is always able to move autonomously in the Nurses rooms in a safe and smooth way. Moreover, the museum guide interacts with the robot with no problem, setting the desired target position on the map and exploring the area by moving the robot using the teleoperation mode or moving the pan/tilt camera.

In particular, we have asked to participant people of the first two days to complete a questionnaire about the virtual visit and about the robotic experience. Generally, the experience was good: the 85% stated that the experience was entertaining and engaging; the quality of the video was good (75%), as well as the sound quality (89%); the robot was adequate and appropriate to the context, offering an interesting service (70%) and, in particular, it was very useful (85%) extending the standard visit. For more details about the design challenge, ethical analysis and results from the questionnaire refer to [126].

The Virgil robot service is also used to provide a museum experience to impaired people [145], in which an impaired person directly interacts with the service by using the GUI.

9.4 Discussion

In this Chapter, two Cloud-based service robotics applications are described. Both robot Courier and Virgil robot services are based on the Cloud-based architecture presented in Chapter 6, where three layers (application, navigation and hardware) are distributed between the robot and Cloud. The two services have many common characteristics because they rely on the Cloud Robotics Platform described in Section 9.4.

(a)

(b)

Figure 9.17: In (a), the Virgil robot in the Nurses rooms. In (b), visitors in the cinema room with the real-time video streamed by the Virgil service.

In particular, the CRP and the Cloud-based architecture allow an easy development of Cloud-based service robotics applications, managing both applications and autonomous navigation. The compatibility with ROS permits the use of a lot of state of the art algorithms already developed by the ROS community, as well as the easy implementation of new algorithms. Because of different service requirements, the autonomous navigation on the two service applications is implemented using different algorithms. Anyway, in both cases, the robot interacts with the Cloud, in order to provide the autonomous navigation task.
In particular, Virgil is controlled directly by the Cloud because of the high quality of

the 4G connection at the Racconigi castle. On the contrary, the robot Courier service is performed in a workspace where the 4G connection is not always good. Hence, the Cloud design a safe path, executed by the on-board local planner. As demonstrated, all modules of the Cloud-based architecture can be allocated as desired on-board or on-Cloud.

With the next generation of mobile network (5G), robots will be completely controlled by the Cloud. In fact, the network latency will be around $1 - 10$ ms. Anyway, on-board the robot some safety systems should be implemented, in order to tackle disconnections. For instance, with both the robot Courier and Virgil, the Connection Diagnostic and the velocity manager resides on-board. In fact, they stop the robot in case of disconnections or bad connection with the Cloud, as well as when obstacles are too close to the robot.

The presented services demonstrate how Cloud robotics can be used to support service robotics applications.

Part III

Conclusions

Chapter 10

Conclusions

In this Ph.D. dissertation, we presented two different scenarios of autonomous navigation of mobile robots in crowded environments.

First, we proposed a solution to design and execute safe flight operations in urban environments.

A ground risk-based map assesses the risk to the population on the ground when the UAS flies over inhabited areas. In fact, the main problem of flight operations in urban areas is the presence of people on the ground, exposed to the possible impact of the aircraft on the ground. The risk-based map is one of the main contribution of this thesis. It is novel tool used to quantify the risk to the population on the ground in urban areas. The risk is computed with a probabilistic risk assessment approach and by evaluating the probabilistic impact area estimated by considering four descent event types. The proposed risk-based map is a promising tool for risk-based decision making. The map quantifies the risk of a particular flight operation identifying the areas where the flight is allowed or not.

The risk-based map can be used by National Aviation Authorities to quantify the risk associated with a particular flight operation. On the opposite, operators can use the risk-based map to define a flight mission avoiding no-fly zones and high-risk areas.

In this thesis, the risk-based map is used to compute a safe path in an urban area. For this purpose, two different risk-aware path planning approach has been proposed.

Both approaches rely on the same risk-aware path planning strategy that consists of two phases. Offline, a minimum risk path is defined based on static risk information of the risk-based map. On the contrary, the online phase updates the offline path according to the dynamic risk-based map.

The first approach is based on the combination of riskA* and Borderland algorithms, able to perform the offline and online phases, respectively. RiskA* seeks for the globally optimal path minimizing the risk to the population on the ground. Hence, the Borderland algorithm updates the path using a *check and repair* method.

Another approach relies on the riskRRT^X algorithm to perform both offline and online phases. In fact, riskRRT^X is a path planning and re-planning algorithm. Offline,

it explores the search space (i.e. the risk-based map) using an incremental tree, then, online, it updates the tree according to the updated risk-based map.
Both approaches are suitable to perform the risk-aware path planning, computing and maintaining a safe path during the execution of the flight mission.

Both risk-aware path planning strategies are part of the contribution of this thesis. In fact, the algorithms are specifically designed to compute and update a safe flight mission for UASs in urban areas.

The combination of risk-based map and risk-aware path planning allows a safe flight operation to be performed. To demonstrate this, we execute a simulation, where an UAS flies over an urban area with a high population density.

The presented work about safe flight operations of UASs is part of a wide project, where a Cloud-based framework for intelligent navigation and coordination for UAS is proposed. The aim of this project is to define a reference framework to implement autonomous flight operations in urban areas with the support of the Cloud.

Another scenario tackled in this Ph.D. dissertation is the autonomous navigation of ground robots in crowded environments to provide service robotics applications.

A dynamic path planner is presented to compute and maintain a valid path through moving people. The proposed approach uses a *check and repair* logic, where the algorithm continuously checks the path and repairs only the invalid portions of the path.

Moreover, a motion controller approach is proposed, called Particle Filter Model Predictive Equilibrium Point Control (PF-MEPC). It is a MPC-based controller combined with the Equilibrium Point method. Practically, the algorithm searches for an optimal equilibrium point to be reached by the robot that optimizes the trajectory of the robot toward the goal position and avoiding obstacles. The PF-MPEPC uses two particle filters to consider disturbances and uncertainties in the prediction phase. The resulting robot behavior has a safe and smooth motion.

The dynamic path planner and the motion controller are another contribution of this thesis. Both algorithms aim to implement a safe and efficient robot navigation in a crowded environment. In particular, these algorithms are used to provide two Cloud-based Service robotics applications: the robot Courier and the Virgil robot. Both services rely on a reference Cloud-based architecture, where the robot is connected with the Cloud using a mobile network and exploiting Cloud technologies.

The robot Courier service aims to welcome visitors in the workplace and escorts them to the desired office. Instead, the Virgil robot provides a real-time virtual tour of an inaccessible area of a museum. Both service robotics applications are tested in a real scenario with real visitors, demonstrating that the proposed Cloud-based architecture is suitable to provide service robotics applications.

Future works include the development of all elements of the Cloud-based architecture for UAS described in Chapter 2, as well as experimental tests in a real urban area. Considering the second part of the thesis, future works include the improvement of current Cloud-based architecture for service robotics applications, as well as the development of new and more complex applications.

Nomenclature

Acronyms / Abbreviations

AMCL Adaptive Monte Carlo Localization

BFS Breadth First Search

BVLOS Beyond Visual Line-Of-Sight

CRP Cloud Robotics Platform

EASA European Aviation Safety Agency

ELOS Equivalent Level of Safety

ENAC Ente Nazionale per l'Aviazione Civile

ENAV Ente Nazionale per l'Assistenza di Volo

FAA Federal Aviation Administration

GCS Ground Control Station

GUI Graphical User Interface

IMEI International Mobile Equipment Identity

IMSI International Mobile Subscriber Identity

IMU Inertial Measurement Unit

IoT Internet of Things

ISO International Organization for Standardization

JARUS Joint Authorities for Rulemaking on Unmammed Systems

LOS Line-Of-Sight

MPC Model Predictive Control

MPEPC Model Predictive Equilibrium Point Control

NAA National Aviation Authority

NED North East Down

OMPL Open Motion Planning Library

OSM Open Street Map

PaaS Platform as a Service

PDF Probability Density Function

PF-MPC Particle Filter Model Predictive Control

PF-MPEPC Particle Filter Model Predictive Equilibrium Point Control

QoS Quality of Service

ROS Robot Operating System

RRT* Optimal Rapidly-exploring Random Tree

RRT Rapidly-exploring Random Tree

SIM Subscriber Identity Model

SITL Software In-The-Loop

SNR Signal to Noise Ratio

SORA Specific Operations Risk Assessment

UAS Unmanned Aerial System

UTM UAS Traffic Management

VFH+ Enhanced Vector Field Histogram

VLOS Visual Line-Of-Sight

Bibliography

[1] M. Aicardi et al. "Closed loop steering of unicycle like vehicles via Lyapunov techniques". In: *IEEE Robotics Automation Magazine* 2.1 (Mar. 1995), pp. 27–35. ISSN: 1070-9932. DOI: 10.1109/100.388294.

[2] P. Aikaterini and J. Tianjian. "Frequency and velocity of people walking". In: *The Structural Engineer* 83.3 (2005).

[3] Andrea Alessandretti and A Pedro Aguiar. "A Model Predictive Cloud-Based Control Scheme for Trajectory-Tracking: Effects of Round-Trip Time Over-Estimates". In: *2018 13th APCA International Conference on Control and Soft Computing (CONTROLO)*. IEEE. 2018, pp. 183–188.

[4] *Amazon Robotics*. https://www.amazonrobotics.com/. Online. Accessed: 2018-09-03.

[5] Ersin Ancel et al. "Real-time Risk Assessment Framework for Unmanned Aircraft System (UAS) Traffic Management (UTM)". In: *17th AIAA Aviation Technology, Integration, and Operations Conference*. 2017, p. 3273.

[6] Iouli Andreev et al. "Risks due to beyond design base accidents of nuclear power plants in Europe—the methodology of riskmap". In: *Journal of Hazardous Materials* 61.1 (1998), pp. 257–262. DOI: 10.1016/S0304-3894(98)00130-7.

[7] Ewa Andrejczuk, Juan A Rodriguez-Aguilar, and Carles Sierra. "A concise review on multiagent teams: contributions and research opportunities". In: *Multi-Agent Systems and Agreement Technologies*. Springer, 2016, pp. 31–39.

[8] Jeffrey G Andrews et al. "What will 5G be?" In: *IEEE Journal on selected areas in communications* 32.6 (2014), pp. 1065–1082.

[9] Plamen Angelov. *Sense and avoid in UAS: research and applications*. John Wiley & Sons, 2012.

[10] Ardupilot. *Open Source Autopilot*. http://ardupilot.org/. Accessed: 2019-03-08.

[11] David Arterburn et al. *FAA UAS Center of Excellence Task A4 : UAS Ground Collision Severity Evaluation*. Tech. rep. 2017, p. 148.

[12] Rajesh Arumugam et al. "DAvinCi: A cloud computing framework for service robots". In: *2010 IEEE international conference on robotics and automation*. IEEE. 2010, pp. 3084–3089.

[13] Kim Baraka and Manuela M Veloso. "Mobile Service Robot State Revealing Through Expressive Lights: Formalism, Design, and Evaluation". In: *International Journal of Social Robotics* 10.1 (2018), pp. 65–92.

[14] Daniel Beimborn, Thomas Miletzki, and Stefan Wenzel. "Platform as a service (PaaS)". In: *Business & Information Systems Engineering* 3.6 (2011), pp. 381–384.

[15] S Bertrand et al. "Ground risk assessment for long-range inspection missions of railways by UAVs". In: *ICUAS 2017, International Conference on Unmanned Aircraft Systems*. IEEE. 2017, pp. 1343–1351. DOI: 10.1109/ICUAS.2017.7991331.

[16] Sushil Pratap Bharati et al. "Real-time obstacle detection and tracking for sense-and-avoid mechanism in UAVs". In: *IEEE Transactions on Intelligent Vehicles* 3.2 (2018), pp. 185–197.

[17] Cynthia Bir and David C Viano. "Design and injury assessment criteria for blunt ballistic impacts". In: *Journal of Trauma and Acute Care Surgery* 57.6 (2004), pp. 1218–1224.

[18] H Bleuer et al. *New trends in medical and service robots*. Springer, 2017.

[19] Nicoletta Bloise et al. "A survey of Unmanned Aircraft System Technologies to enable Safe Operations in Urban Areas". In: *ICUAS 2019, International Conference on Unmanned Aircraft Systems*. IEEE. 2019.

[20] Robert Bogue. "Robots poised to revolutionise agriculture". In: *Industrial Robot: An International Journal* 43.5 (2016), pp. 450–456.

[21] Andrea Bonarini et al. "R2P: An open source hardware and software modular approach to robot prototyping". In: *Robotics and Autonomous Systems* 62.7 (2014), pp. 1073–1084.

[22] Johann Borenstein and Yoram Koren. "The vector field histogram-fast obstacle avoidance for mobile robots". In: *IEEE transactions on robotics and automation* 7.3 (1991), pp. 278–288.

[23] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[24] Roger W Brockett et al. "Asymptotic stability and feedback stabilization". In: *Differential geometric control theory* 27.1 (1983), pp. 181–191.

[25] Norlida Buniyamin et al. "A simple local path planning algorithm for autonomous mobile robots". In: *International journal of systems applications, Engineering & development* 5.2 (2011), pp. 151–159.

[26] Jonathan Cacace et al. "Aerial service vehicles for industrial inspection: task decomposition and plan execution". In: *Applied Intelligence* 42.1 (2015), pp. 49–62.

[27] Elisa Capello et al. "Regulation analysis and new concept for a cloud-based UAV supervision system in urban environment". In: *RED-UAS 2017, Workshop on Research, Education and Development of Unmanned Aerial Systems*. IEEE. 2017, pp. 90–95. DOI: 10.1109/RED-UAS.2017.8101649.

[28] Marco Ceccarelli. "Problems and issues for service robots in new applications". In: *International Journal of Social Robotics* 3.3 (2011), pp. 299–312.

[29] Yinong Chen, Zhihui Du, and Marcos García-Acosta. "Robot as a service in cloud computing". In: *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*. IEEE. 2010, pp. 151–158.

[30] Victor R Clare et al. *Blunt trauma data correlation*. Tech. rep. Edgewood Arsenal Aberdeen Proving Ground Md, 1975.

[31] Reece A Clothier, Brendan P Williams, and Kelly J Hayhurst. "Modelling the risks remotely piloted aircraft pose to people on the ground". In: *Safety science* 101 (2018), pp. 33–47.

[32] Reece A Clothier et al. "A casualty risk analysis for unmanned aerial system (UAS) operations over inhabited areas". In: (2007).

[33] Anders la Cour-Harbo. "Ground impact probability distribution for small unmanned aircraft in ballistic descent". In: *preprint* - (2017), pp. 1–24.

[34] Anders la Cour-Harbo. "Mass threshold for 'harmless' drones". In: *International Journal of Micro Air Vehicles* 9.2 (2017), pp. 77–92. DOI: 10.1177/1756829317691991.

[35] Anders la Cour-Harbo. "Quantifying ground impact fatality rate for small unmanned aircraft". In: *Journal of Intelligent & Robotic Systems* - (May 2018), pp. 1–18. DOI: 10.1007/s10846-018-0853-1.

[36] Anders la Cour-Harbo. "The Value of Step-by-Step Risk Assessment for Unmanned Aircraft". In: *ICUAS 2018, International Conference on Unmanned Aircraft Systems*. IEEE. 2018.

[37] D-Flight. *D-Flight: enabling autonomous flight*. https://app.d-flight.it.

[38] Erik Dahlman, Stefan Parkvall, and Johan Skold. *4G: LTE/LTE-advanced for mobile broadband*. Academic press, 2013.

[39] Konstantinos Dalamagkidis, Kimon P Valavanis, and Les A Piegl. *On integrating unmanned aircraft systems into the national airspace system: issues, challenges, operational restrictions, certification, and recommendations*. Vol. 54. Springer Netherlands, 2012.

[40] Kerstin Dautenhahn. "Socially intelligent robots: dimensions of human–robot interaction". In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 362.1480 (2007), pp. 679–704.

[41] Luca De Filippis, Giorgio Guglieri, and Fulvia Quagliotti. "A minimum risk approach for path planning of UAVs". In: *Journal of Intelligent & Robotic Systems* 61.1-4 (2011), pp. 203–219. DOI: 10.1007/s10846-010-9493-9.

[42] Luca De Filippis, Giorgio Guglieri, and Fulvia Quagliotti. "Path planning strategies for UAVS in 3D environments". In: *Journal of Intelligent & Robotic Systems* 65.1-4 (2012), pp. 247–264.

[43] Frank Dellaert et al. "Monte carlo localization for mobile robots". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[44] Pierre Deville et al. "Dynamic population mapping using mobile phone data". In: *Proceedings of the National Academy of Sciences* 111.45 (2014), pp. 15888–15893. DOI: 10.1073/pnas.1408439111.

[45] Prasanta Kumar Dey. "Managing Project Risk Using Combined Analytic Hierarchy Process and Risk Map". In: *Appl. Soft Comput.* 10.4 (2010), pp. 990–1000. DOI: 10.1016/j.asoc.2010.03.010.

[46] Carmelo Di Franco and Giorgio Buttazzo. "Energy-aware coverage path planning of UAVs". In: *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*. IEEE. 2015, pp. 111–117.

[47] Alessandro Di Nuovo et al. "The multi-modal interface of Robot-Era multi-robot services tailored for the elderly". In: *Intelligent Service Robotics* 11.1 (2018), pp. 109–126.

[48] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[49] ISO DIS. *8373: 2012. Robots and robotic devices–Vocabulary*. Tech. rep. International Standardization Organization (ISO), 2012.

[50] Kris Doelling, Jeongsik Shin, and Dan O Popa. "Service robotics for the home: a state of the art review". In: *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*. ACM. 2014, p. 35.

[51] Lester E Dubins. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". In: *American Journal of mathematics* 79.3 (1957), pp. 497–516.

[52] Joseph W Durham and Francesco Bullo. "Smooth nearness-diagram navigation". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 690–695.

[53] EASA, European Aviation Safety Agency. *Concept of Operations for Drones: A Risk Based Approach to Regulation of unmanned Aircraft*. Cologne, Germany, 2015.

[54] ENAC. *Progetti di ricerca e sviluppo SAPR - 2018*. https://www.enac.gov.it/sites/default/files/allegati/2018-Dic/R%26S_Elenco_Stakeolder_Progetti_24_dicembre_2018.pdf.

[55] European Aviation Safety Agency. *Notice of Proposed Amendment 2017-05 (A) - Introduction of a regulatory framework for the operation of drones*. Tech. rep. 2017.

[56] European Aviation Safety Agency. *Notice of Proposed Amendment 2017-05 (B) - Introduction of a regulatory framework for the operation of drones*. Tech. rep. 2017.

[57] European Aviation Safety Agency. *Opinion 2018-01 - Introduction of a regulatory framework for the operation of unmanned aircraft systems in the 'open' and 'specific' categories*. Tech. rep. 2018.

[58] John M Evans. "HelpMate: An autonomous mobile robot courier for hospitals". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*. Vol. 3. IEEE. 1994, pp. 1695–1700.

[59] FAA, Federal Aviation Administration. *System safety handbook*. Department of transportation, Washington DC, USA, 2000.

[60] FAA, Federal Aviation Administration. *UAS Integration Pilot Program*. https://www.faa.gov/uas/programs_partnerships/uas_integration_pilot_program/. Accessed: 2018-06-26.

[61] Péter Fankhauser and Marco Hutter. "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation". In: *Robot Operating System (ROS) – The Complete Reference (Volume 1)*. Ed. by Anis Koubaa. Springer, 2016. Chap. 5. DOI: 10.1007/978-3-319-26054-9_5.

[62] D. Ferguson, N. Kalra, and A. Stentz. "Replanning with RRTs". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2006, pp. 1243–1248.

[63] Seyedshams Feyzabadi and Stefano Carpin. "Risk-aware path planning using hirerachical constrained markov decision processes". In: *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2014, pp. 297–303.

[64] Jodi Forlizzi and Carl DiSalvo. "Service robots in the domestic environment: a study of the roomba vacuum in the home". In: *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM. 2006, pp. 258–265.

[65] Dieter Fox. "KLD-sampling: Adaptive particle filters". In: *Advances in neural information processing systems*. 2002, pp. 713–720.

[66] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33.

[67] Chiara Fulgenzi et al. "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 1056–1062.

[68] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2997–3004.

[69] Claudio Germak et al. "Robots and cultural heritage: New museum experiences". In: *Journal of Science and Technology of the Arts* 7.2 (2015), pp. 47–57.

[70] Mirmojtaba Gharibi, Raouf Boutaba, and Steven L Waslander. "Internet of drones". In: *IEEE Access* 4 (2016), pp. 1148–1162.

[71] Andrey Giyenko and Young Im Cho. "Intelligent UAV in smart cities using IoT". In: *2016 16th International Conference on Control, Automation and Systems (ICCAS)*. IEEE. 2016, pp. 207–210.

[72] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. "A survey of motion planning algorithms from the perspective of autonomous UAV guidance". In: *Journal of Intelligent and Robotic Systems* 57.1-4 (2010), p. 65.

[73] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. "Improved techniques for grid mapping with rao-blackwellized particle filters". In: *IEEE transactions on Robotics* 23.1 (2007), pp. 34–46.

[74] Giorgio Guglieri, Alessandro Lombardi, and Gianluca Ristorto. "Operation Oriented Path Planning Strategies for Rpas". In: *American Journal of Science and Technology* 2.6 (2015), pp. 1–8.

[75] Giorgio Guglieri, Fulvia Quagliotti, and G Ristorto. "Operational issues and assessment of risk for light UAVs". In: *Journal of Unmanned Vehicle Systems* 2.4 (2014), pp. 119–129.

[76] Giorgio Guglieri and Gianluca Ristorto. "Safety assessment for light remotely piloted aircraft systems". In: *INAIR 2016, International Conference on Air Transport*. 2016, pp. 1–7.

[77] Dingfei Guo et al. "A hybrid feature model and deep learning based fault diagnosis for unmanned aerial vehicle sensors". In: *Neurocomputing* 319 (2018), pp. 155–163.

[78] Karl D Hansen and Anders La Cour-Harbo. "Waypoint planning with Dubins curves using genetic algorithms". In: *2016 European Control Conference (ECC)*. IEEE. 2016, pp. 2240–2246.

[79] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[80] Shyamanta M Hazarika and Uday Shanker Dixit. "Robotics: History, Trends, and Future Directions". In: *Introduction to Mechanical Engineering* (2018), pp. 213–239.

[81] Peter Henry et al. "Learning to navigate through crowded environments". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 981–986.

[82] Guoqiang Hu, Wee Peng Tay, and Yonggang Wen. "Cloud robotics: architecture, challenges and applications". In: *IEEE network* 26.3 (2012), pp. 21–28.

[83] Yun Chao Hu et al. "Mobile edge computing—A key technology towards 5G". In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[84] Human Interface Technology Laboratory (HIT Lab), University of Washington. *ARToolKit*. http://www.hitl.washington.edu/artoolkit/. Online. Accessed: 2018-09-03.

[85] Dominique Hunziker et al. "Rapyuta: The roboearth cloud engine". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 438–444.

[86] Corey A Ippolito. "Dynamic Ground Risk Mitigation for Autonomous Small UAS in Urban Environments". In: *AIAA Scitech 2019 Forum*. 2019, p. 0961.

[87] Corey A Ippolito et al. "Autonomous UAS Operations in High-Density Low-Altitude Urban Environments". In: *AIAA Scitech 2019 Forum*. 2019, p. 0687.

[88] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. "Dynamic multi-heuristic A". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2376–2382.

[89] Istat. *Rapporto URBES 2015*. https://www.istat.it/storage/urbes2015/torino.pdf. Accessed: 2018-09-18. 2015.

[90] Mason Itkin, Mihui Kim, and Younghee Park. "Development of Cloud-Based UAV Monitoring and Management System". In: *Sensors* 16.11 (2016), p. 1913.

[91] Léonard Jaillet, Juan Cortés, and Thierry Siméon. "Transition-based RRT for path planning in continuous cost spaces". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 2145–2150.

[92] H Johannsen and V Schindler. *Review of the abdomen injury criteria*. Tech. rep. Institut National de Reserche sur les Transports et leur Sécurité, Tech. Rep. AP-SP51-0039-B, 2005.

[93] Steven G. Johnson. *The NLopt nonlinear-optimization package*. URL: http://ab-initio.mit.edu/nlopt.

[94] Chaogui Kang et al. "Towards estimating urban population distributions from mobile call data". In: *Journal of Urban Technology* 19.4 (2012), pp. 3–21. DOI: 10.1080/10630732.2012.715479.

[95] Kiattisin Kanjanawaniskul. "Motion control of a wheeled mobile robot using model predictive control: A survey". In: *Asia-Pacific Journal of Science and Technology* 17.5 (2012), pp. 811–837.

[96] Suat Karakaya, Hasan Ocak, and Gürkan Küçükyıldız. "A bug-based local path planning method for static and dynamic environments". In: *International Symposium on Innovative Technologies in Engineering and Science. Valencia, Spain.* 2015.

[97] Sertac Karaman and Emilio Frazzoli. "Incremental sampling-based algorithms for optimal motion planning". In: *Robotics Science and Systems VI* 104 (2010), p. 2.

[98] Sertac Karaman and Emilio Frazzoli. "Optimal kinodynamic motion planning using incremental sampling-based methods". In: *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE. 2010, pp. 7681–7687.

[99] Sertac Karaman et al. "Anytime motion planning using the RRT". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1478–1483.

[100] Lydia E Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[101] Uluhan C Kaya, Atilla Dogan, and Manfred Huber. "A Probabilistic Risk Assessment Framework for the Path Planning of Safe Task-Aware UAS Operations". In: *AIAA Scitech 2019 Forum*. 2019, p. 2079.

[102] Ben Kehoe et al. "A survey of research on cloud robotics and automation". In: *IEEE Transactions on automation science and engineering* 12.2 (2015), pp. 398–409.

[103] Sara Khan and Claudio Germak. "Reframing HRI Design Opportunities for Social Robots: Lessons Learnt from a Service Robotics Case Study Approach Using UX for HRI". In: *Future Internet* 10.10 (2018), p. 101.

[104] Oussama Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.

[105] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[106] Anis Koubâa et al. “Dronemap planner: A service-oriented cloud-based management system for the internet-of-drones”. In: *Ad Hoc Networks* 86 (2019), pp. 46–62.

[107] James J Kuffner Jr and Steven M LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *ICRA*. Vol. 2. 2000.

[108] F. Kuhne, W. F. Lages, and J. M. G. da Silva. “Point stabilization of mobile robots with nonlinear model predictive control”. In: *IEEE International Conference Mechatronics and Automation, 2005*. Vol. 3. 2005, 1163–1168 Vol. 3. DOI: 10.1109/ICMA.2005.1626717.

[109] Felipe Kuhne, Walter Fetter Lages, and J Gomes da Silva Jr. “Model predictive control of a mobile robot using linearization”. In: *Proceedings of mechatronics and robotics*. Citeseer. 2004, pp. 525–530.

[110] F Künhe, J Gomes, and W Fetter. “Mobile robot trajectory tracking using model predictive control”. In: *II IEEE latin-american robotics symposium*. Vol. 51. 2005.

[111] Oh-Hun Kwon et al. “Telepresence robot system for English tutoring”. In: *2010 IEEE Workshop on Advanced Robotics and its Social Impacts*. IEEE. 2010, pp. 152–155.

[112] S. M. LaValle. “Rapidly-Exploring Random Trees: A New Tool for Path Planning”. TR 98-11, Computer Science Dept., Iowa State University. Oct. 1998.

[113] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[114] Jangwon Lee et al. “Real-time, cloud-based object detection for unmanned aerial vehicles”. In: *2017 First IEEE International Conference on Robotic Computing (IRC)*. IEEE. 2017, pp. 36–43.

[115] Minglei Li et al. “Reconstructing building mass models from UAV images”. In: *Computers & Graphics* 54 (2016), pp. 84–93.

[116] Shancang Li, Li Da Xu, and Shanshan Zhao. “5G internet of things: A survey”. In: *Journal of Industrial Information Integration* 10 (2018), pp. 1–9.

[117] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. “ARA*: Anytime A* with provable bounds on sub-optimality”. In: *Advances in neural information processing systems*. 2004, pp. 767–774.

[118] Maxim Likhachev et al. “Anytime Dynamic A*: An Anytime, Replanning Algorithm.” In: *ICAPS*. 2005, pp. 262–271.

[119] Chin E Lin, Cheng-Ru Li, and Ya-Hsien Lai. “UAS cloud surveillance system”. In: *2012 41st International Conference on Parallel Processing Workshops*. IEEE. 2012, pp. 173–178.

[120] Xingqin Lin et al. “The sky is not the limit: LTE for unmanned aerial vehicles”. In: *IEEE Communications Magazine* 56.4 (2018), pp. 204–210.

[121] Yucong Lin and Srikanth Saripalli. "Path planning using 3D dubins curve for unmanned aerial vehicles". In: *2014 international conference on unmanned aircraft systems (ICUAS)*. IEEE. 2014, pp. 296–304.

[122] Yucong Lin and Srikanth Saripalli. "Sampling-based path planning for UAV collision avoidance". In: *IEEE Transactions on Intelligent Transportation Systems* 18.11 (2017), pp. 3179–3192.

[123] Guo-Ping Liu. "Predictive control of networked multiagent systems via cloud computing". In: *IEEE transactions on cybernetics* 47.8 (2017), pp. 1852–1859.

[124] Huimin Lu et al. "Motor anomaly detection for unmanned aerial vehicles using reinforcement learning". In: *IEEE internet of things journal* 5.4 (2018), pp. 2315–2322.

[125] Vladimir J Lumelsky and Alexander A Stepanov. "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape". In: *Algorithmica* 2.1-4 (1987), pp. 403–430.

[126] Maria Luce Lupetti, Luca Giuliano, and Claudio Germak. "Virgil robot at racconigi castle: A design challenge". In: *Proceedings of the Seventh International Workshop on Human-Computer Interaction, Tourism and Cultural Heritage, HCI-TOCH*. 2016.

[127] Thi Thoa Mac et al. "Heuristic approaches in robot path planning: A survey". In: *Robotics and Autonomous Systems* 86 (2016), pp. 13–28.

[128] Shibarchi Majumder and Mani Shankar Prasad. "Cloud based control for unmanned aerial vehicles". In: *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE. 2016, pp. 421–424.

[129] L Marconi et al. "Aerial service robotics: The AIRobots perspective". In: *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*. IEEE. 2012, pp. 64–69.

[130] Maja J Matarić and Brian Scassellati. "Socially assistive robotics". In: *Springer Handbook of Robotics*. Springer, 2016, pp. 1973–1994.

[131] Ian McDonnell. "The role of the tour guide in transferring cultural understanding". In: (2001).

[132] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 6235–6240.

[133] Richard V Melnyk et al. "A Third-Party Casualty Prediction Model for UAS Operations". In: *Georgia Institute of Technology* (2012).

[134] Hamid Menouar et al. “UAV-enabled intelligent transportation systems for the smart city: Applications and challenges”. In: *IEEE Communications Magazine* 55.3 (2017), pp. 22–28.

[135] Javier Minguez and Luis Montano. “Nearness diagram navigation (nd): A new real time collision avoidance approach”. In: *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*. Vol. 3. IEEE. 2000, pp. 2094–2100.

[136] Shashi Mittal and Kalyanmoy Deb. “Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms”. In: *2007 IEEE Congress on Evolutionary Computation*. IEEE. 2007, pp. 3195–3202.

[137] Farhan Mohammed et al. “Opportunities and challenges of using UAVs for dubai smart city”. In: *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE. 2014, pp. 1–4.

[138] T. Moore and D. Stouch. “A Generalized Extended Kalman Filter Implementation for the Robot Operating System”. In: *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[139] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. “UAV-based IoT platform: A crowd surveillance use case”. In: *IEEE Communications Magazine* 55.2 (2017), pp. 128–134.

[140] Robin Murphy. *Introduction to AI robotics*. MIT press, 2000.

[141] Alex Nash et al. “Theta^*: Any-angle path planning on grids”. In: *AAAI*. Vol. 7. 2007, pp. 1177–1183.

[142] Naviair. *Droneluftrum website*. https://www.droneluftrum.dk. Accessed: 2018-06-26.

[143] Francesco Nex and Fabio Remondino. “UAV for 3D mapping applications: a review”. In: *Applied geomatics* 6.1 (2014), pp. 1–15.

[144] James Ng and Thomas Bräunl. “Performance comparison of bug navigation algorithms”. In: *Journal of Intelligent and Robotic Systems* 50.1 (2007), pp. 73–84.

[145] Miguel Kaouk Ng et al. “A cloud robotics system for telepresence enabling mobility impaired people to enjoy the whole museum experience”. In: *2015 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE. 2015, pp. 1–6.

[146] Ioannis K Nikolos et al. “Evolutionary algorithm based offline/online path planner for UAV navigation”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 33.6 (2003), pp. 898–912.

[147] *Nuzoo Robotics*. http://www.nuzoo.it. Online. Accessed: 2018-09-03.

[148] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. Accessed: 2018-06-26. 2017.

[149] OSM2World. *Create 3D models from OpenStreetMap*. http://osm2world.org. Accessed: 2019-03-04. 2019.

[150] Michael Otte and Emilio Frazzoli. "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning". In: *The International Journal of Robotics Research* 35.7 (2016), pp. 797–822.

[151] Anish Pandey, S Pandey, and DR Parhi. "Mobile robot navigation and obstacle avoidance techniques: A review". In: *Int Rob Auto J* 2.3 (2017), p. 00022.

[152] J. J. Park and B. Kuipers. "Autonomous person pacing and following with Model Predictive Equilibrium Point Control". In: *2013 IEEE International Conference on Robotics and Automation*. May 2013, pp. 1060–1067. DOI: 10.1109/ICRA.2013.6630704.

[153] Jong Jin Park and Benjamin Kuipers. "Autonomous person pacing and following with model predictive equilibrium point control". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1060–1067.

[154] Arvind A Pereira et al. "Risk-aware path planning for autonomous underwater vehicles using predictive ocean models". In: *Journal of Field Robotics* 30.5 (2013), pp. 741–762.

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

[156] Stefano Primatesta, Giorgio Guglieri, and Alessandro Rizzo. "A risk-aware path planning method for unmanned aerial vehicles". In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2018, pp. 905–913.

[157] Stefano Primatesta, Giorgio Guglieri, and Alessandro Rizzo. "A Risk-Aware Path Planning Strategy for UAVs in Urban Environments". In: *Journal of Intelligent & Robotic Systems* (2018), pp. 1–15.

[158] Stefano Primatesta, A Rizzo, and Anders La Cour-Harbo. "Ground risk map for Unmanned Aircraft in Urban Environments". In: *Journal of Intelligent & Robotic Systems* (2019), pp. 1–21.

[159] Stefano Primatesta, Ludovico Orlando Russo, and Basilio Bona. "Dynamic trajectory planning for mobile robot navigation in crowded environments". In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2016, pp. 1–8.

[160] Stefano Primatesta et al. "A Cloud-based Framework for Intelligent Navigation and Coordination for UASs in Urban Areas". In: *ICUAS 2019, International Conference on Unmanned Aircraft Systems*. IEEE. 2019.

[161] Stefano Primatesta et al. “A cloud-based framework for risk-aware intelligent navigation in urban environments”. In: *ICUAS 2017, International Conference on Unmanned Aircraft Systems*. IEEE. 2017, pp. 447–455. DOI: 10.1109/ICUAS.2017.7991358.

[162] Stefano Primatesta et al. “An innovative algorithm to estimate risk optimum path for unmanned aerial vehicles in urban environments”. In: *Transportation research procedia* 35 (2018), pp. 44–53.

[163] PX4. *PX4 Autopilot*. https://px4.io/. Accessed: 2019-04-15.

[164] PX4. *PX4 Flight stack: development guide*. https://dev.px4.io/en/. Accessed: 2019-04-15.

[165] Sameer Qazi, Ali Shuja Siddiqui, and Asim Imdad Wagan. “UAV based real time video surveillance over 4G LTE”. In: *2015 International Conference on Open Source Systems & Technologies (ICOSST)*. IEEE. 2015, pp. 141–145.

[166] LTE Qualcomm. *Unmanned Aircraft Systems—Trial Report*. 2017.

[167] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5.

[168] Sean Quinlan and Oussama Khatib. “Elastic bands: Connecting path planning and control”. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 802–807.

[169] Range Commanders Coucil. “Standard 321-07 Common Risk Criteria Standards for National Test Ranges: Supplement””. In: *USA Dept. of Defense* (2007).

[170] S Mohammad Razavizadeh, Minki Ahn, and Inkyu Lee. “Three-dimensional beamforming: A new enabling technology for 5G wireless networks”. In: *IEEE Signal Processing Magazine* 31.6 (2014), pp. 94–101.

[171] *ROCON, Robotics in Concert*. http://www.robotconcert.org. Online. Accessed: 2018-09-03.

[172] ROS, Robot Operating System. *The 2D navigation stack*. http://wiki.ros.org/navigation. Online. Accessed: 2018-09-03.

[173] Stefano Rosa, Ludovico Orlando Russo, and Basilio Bona. “Towards a ROS-based autonomous cloud robotics platform for data center monitoring”. In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. IEEE. 2014, pp. 1–8.

[174] Stefano Rosa et al. “Leveraging the cloud for connected service robotics applications”. In: *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE. 2015, pp. 1–6.

[175] Christoph Rösmann et al. “Efficient trajectory optimization using a sparse model”. In: *2013 European Conference on Mobile Robots*. IEEE. 2013, pp. 138–143.

[176] Peter Rost et al. "Mobile network architecture evolution toward 5G". In: *IEEE Communications Magazine* 54.5 (2016), pp. 84–91.

[177] Eliot Rudnick-Cohen, Jeffrey W Herrmann, and Shapour Azarm. "Risk-based path planning optimization methods for unmanned aerial vehicles over inhabited areas". In: *Journal of Computing and Information Science in Engineering* 16.2 (2016), p. 021004.

[178] Ludovico Orlando Russo et al. "A novel cloud-based service robotics application to data center environmental monitoring". In: *Sensors* 16.8 (2016), p. 1255.

[179] Ludovico O Russo et al. "Service robotics for data centers monitoring". In: *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2016, pp. 6908–6913.

[180] Shankar Sankararaman and Christopher Teubert. "Prospective architectures for onboard vs cloud-based decision making for unmanned aerial systems". In: (2017).

[181] Miguel Sarabia et al. "Assistive Robotic Technology to Combat Social Isolation in Acute Hospital Settings". In: *International Journal of Social Robotics* 10.5 (2018), pp. 607–620.

[182] Andrey V Savkin and Hailong Huang. "The problem of minimum risk path planning for flying robots in dangerous environments". In: *2016 35th Chinese Control Conference (CCC)*. IEEE. 2016, pp. 5404–5408.

[183] Jan Schlechtendahl et al. "Extended study of network capability for cloud based control systems". In: *Robotics and Computer-Integrated Manufacturing* 43 (2017), pp. 89–95.

[184] Chao Shi et al. "A robot that distributes flyers to pedestrians in a shopping mall". In: *International Journal of Social Robotics* 10.4 (2018), pp. 421–437.

[185] Kento Shimada and Takeshi Nishida. "Particle filter-model predictive control of quadcopters". In: *Proceedings of the 2014 International Conference on Advanced Mechatronic Systems*. IEEE. 2014, pp. 421–424.

[186] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[187] Jesimar da Silva Arantes et al. "Heuristic and genetic algorithm approaches for UAV path planning under critical situation". In: *International Journal on Artificial Intelligence Tools* 26.01 (2017), p. 1760008.

[188] Meryem Simsek et al. "5G-enabled tactile internet". In: *IEEE Journal on Selected Areas in Communications* 34.3 (2016), pp. 460–473.

[189] Patricia Grace Smith. *Expected Casualty Calculations For Commercial Space Launch and Reentry Missions - Advisory Circular*. Tech. rep. 2000.

[190] Dominik Stahl and Jan Hauth. "PF-MPC: Particle filter-model predictive control". In: *Systems & Control Letters* 60.8 (2011), pp. 632–643.

[191] Anthony Stentz. *Optimal and efficient path planning for unknown and dynamic environments*. Tech. rep. Carnegie Mellon University, 1993.

[192] Claudia Stöcker et al. "Review of the current state of UAV regulations". In: *Remote sensing* 9.5 (2017), p. 459.

[193] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012), pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[194] Mikael Svenstrup, Thomas Bak, and Hans Jørgen Andersen. "Trajectory planning for robots in dynamic human environments". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 4293–4298.

[195] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[196] Carme Torras. "Service robots for citizens of the future". In: *European Review* 24.1 (2016), pp. 17–30.

[197] Peter Trautman and Andreas Krause. "Unfreezing the robot: Navigation in dense, interacting crowds". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 797–803.

[198] Pete Trautman et al. "Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation". In: *The International Journal of Robotics Research* 34.3 (2015), pp. 335–356.

[199] Rudolph Triebel et al. "Spencer: A socially aware service robot for passenger guidance and help in busy airports". In: *Field and service robotics*. Springer. 2016, pp. 607–622.

[200] K Tsui et al. "Designing telepresence robot systems for use by people with special needs". In: *Int. Symposium on Quality of Life Technologies: Intelligent Systems for Better Living*. 2011.

[201] Iwan Ulrich and Johann Borenstein. "VFH*: Local obstacle avoidance with look-ahead verification". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 3. IEEE. 2000, pp. 2505–2511.

[202] Iwan Ulrich and Johann Borenstein. "VFH+: Reliable obstacle avoidance for fast mobile robots". In: *Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No. 98CH36146)*. Vol. 2. IEEE. 1998, pp. 1572–1577.

[203] Kimon P Valavanis and George J Vachtsevanos. "Uav applications: introduction". In: *Handbook of Unmanned Aerial Vehicles* (2015), pp. 2639–2641.

[204] Bertold Van der Bergh, Alessandro Chiumento, and Sofie Pollin. "LTE in the sky: Trading off propagation benefits with interference costs for aerial nodes". In: *IEEE Communications Magazine* 54.5 (2016), pp. 44–50.

[205] Wim Van Der Hoek et al. "Towards a risk map of malaria for Sri Lanka: the importance of house location relative to vector breeding sites". In: *International Journal of Epidemiology* 32.2 (2003), pp. 280–285.

[206] Markus Waibel et al. "Roboearth-a world wide web for robots". In: *IEEE Robotics and Automation Magazine (RAM), Special Issue Towards a WWW for Robots* 18.2 (2011), pp. 69–82.

[207] Achim Washington, Reece A Clothier, and Jose Silva. "A review of unmanned aircraft system ground risk models". In: *Progress in Aerospace Sciences* 95 (2017), pp. 24–44.

[208] Achim Washington, Reece A Clothier, and Brendan P Williams. "A Bayesian approach to system safety assessment and compliance assessment for Unmanned Aircraft Systems". In: *Journal of Air Transport Management* 62 (2017), pp. 18–33.

[209] *Waymo*. https://waymo.com/. Online. Accessed: 2018-09-03.

[210] Bernhard Weiß, Michael Naderhirn, and Luigi del Re. "Global real-time path planning for uavs in uncertain environment". In: *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*. IEEE. 2006, pp. 2725–2730.

[211] Naifeng Wen et al. "Online UAV path planning in uncertain and hostile environments". In: *International Journal of Machine Learning and Cybernetics* 8.2 (2017), pp. 469–487.

[212] Graham Wild, John Murray, and Glenn Baxter. "Exploring civil drone accidents and incidents to help prevent potential air disasters". In: *Aerospace* 3.3 (2016), p. 22.

[213] Victor Wolfe et al. "Feasibility Study of Utilizing 4G LTE Signals in Combination With Unmanned Aerial Vehicles for the Purpose of Search and Rescue of Avalanche Victims (Increment 1)". In: *University of Colorado at Boulder, Research Report* (2014).

[214] Guang Yang et al. "A Telecom Perspective on the Internet of Drones: From LTE-Advanced to 5G". In: *arXiv preprint arXiv:1803.11048* (2018).

[215] Peter Yap. "Grid-based path-finding". In: *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer. 2002, pp. 44–55.

[216] Kevin Yu, Ashish Kumar Budhiraja, and Pratap Tokekar. "Algorithms for routing of unmanned aerial vehicles with mobile recharging stations". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–5.

[217] Yong Zeng, Jiangbin Lyu, and Rui Zhang. "Cellular-Connected UAV: Potential, Challenges, and Promising Technologies". In: *IEEE Wireless Communications* 26.1 (2019), pp. 120–127.

[218] Baochang Zhang et al. “Geometric reinforcement learning for path planning of UAVs”. In: *Journal of Intelligent & Robotic Systems* 77.2 (2015), pp. 391–409.

[219] Haijun Zhang et al. “Network slicing based 5G and future mobile networks: mobility, resource management, and challenges”. In: *IEEE Communications Magazine* 55.8 (2017), pp. 138–145.

[220] Fu Zhuang et al. “A cable-tunnel inspecting robot for dangerous environment”. In: *International Journal of Advanced Robotic Systems* 5.3 (2008), p. 32.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was LuaLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.