POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Politecnico di Torino

Master's Degree Thesis

October 2024

Development of a WebApp for safe mission planning for UAS in urban areas

Candidate:

ANGELO MICHELE DEL GUERCIO

Supervisors:

Dr. STEFANO PRIMATESTA

Ing. GIANLUCA RISTORTO

SUMMARY

In recent years, the use of drones, or Unmanned Aircraft Systems (UAS), has experienced unprecedented growth, with applications ranging from environmental monitoring to surveillance, extending to sectors such as logistics and transportation. However, operating these devices in urban environments presents complex challenges in terms of safety and risk management. Specifically, drones flying in populated areas can pose a danger to people on the ground in the event of malfunctions or accidents. Furthermore, the integration of these technologies into civilian contexts must comply with strict regulations to ensure both safety and privacy. The core issue lies in the fact that the impact of a drone in a populated area can have devastating consequences, making it essential to develop advanced tools that enable precise flight mission planning while minimizing risks. The need for risk assessment and careful route planning for flight operations has led to the development of this project, which aims to create an interactive platform for the safe management and planning of drone operations.

The rapid proliferation of drones has made risk management in flight operations increasingly crucial. In Europe, the SORA (Specific Operations Risk Assessment) guidelines have become the standard reference for authorities and industry operators. SORA allows for the assessment of the risk associated with a flight operation based on a series of parameters, such as population density, the characteristics of the operational area, and the required safety levels. Currently, there are few tools available to assist UAS operators in conducting risk assessments according to SORA guidelines. These tools, while attempting to simplify the assessment process, are not fully automated and often require expert intervention, leading to significant costs, especially for small and medium-sized operators. Even aviation authorities, such as EASA (European Union Aviation Safety Agency), provide guidelines and standard forms that allow operators to manually perform risk assessments. However, these free resources require in-depth knowledge of regulations, and no fully automated platforms exist that entirely replace

human intervention. Another major issue is that manual SORA procedures can be time-consuming, slowing down operational processes in a context where the number of drone operations is steadily increasing.

The primary objective of this project is to develop an interactive web application for the safe planning of flight operations in populated environments. The platform, designed for professionals such as drone operators or surveillance agencies, offers advanced mission planning and management functionalities through an intuitive map-based interface. The key features include:

- Risk map generation, based on data such as population density and the characteristics of the surrounding environment.
- Safe route planning, minimizing operational risks through the analysis of critical areas.
- Custom configuration of flight parameters, such as altitude, speed, and the drone's payload.
- Display of critical information directly on the map.

The application utilizes React.js to manage the user interface, while the backend is developed in Node.js, ensuring a smooth and high-performance user experience. Communication with a ROS server is handled via WebSocket, guaranteeing real-time data reception and processing.

The project development process was divided into several phases:

1. **User interface design**: The web app was designed with a user-friendly approach, allowing the operator to interact with the map, select areas of interest, and configure mission parameters.
2. **Implementation of risk map generation**: A key feature of the project is the ability to generate risk maps and low-risk paths. The ROS server, connected to

the web app, processes the data provided by the user and returns a map of the risk areas.

3. **Integration with the ROS server**: A crucial aspect of the project was the integration of the web app with ROS, which runs the algorithms for risk map generation and flight planning.
4. **Testing and performance optimization**: Various tests were conducted throughout development to verify the effectiveness and reliability of the application.
5. **Compliance with regulations**: The application was designed to ensure that operators can configure missions following existing regulations.

The goal of this thesis project is to provide an intuitive and fast tool for risk assessment in these operations, reducing the complexity and costs associated with existing solutions. In the future, through further development, this project could significantly contribute to the safe and compliant management of drone flights, with potential applications in multiple sectors, including surveillance, logistics, and environmental monitoring.

CONTENT

LIST OF FIGURES

CHAPTER 1

1. Introduction

The term "Unmanned Aircraft" refers to any aircraft that operates or is designed to operate autonomously or be remotely piloted without an onboard pilot. This definition includes all types of unmanned aircraft, including radio-controlled flying models, regardless of whether they are equipped with an onboard camera.

Drone regulations use the term "Unmanned Aircraft System" (UAS) to refer to a drone, its system, and all other equipment used to control and operate it, such as the control unit, any launching equipment, and more.

Figure 1. Drone delivery

1.1 Context

In recent years, drones, also known as Unmanned Aircraft Systems (UAS), have seen increasing use in various sectors, including environmental monitoring, surveillance, logistics, and precision agriculture. This widespread adoption has been made possible by technological advancements in component miniaturization, cost reduction, and improvements in autonomous flight capabilities. However, the integration of drones into urban environments presents a series of complex challenges, primarily related to safety, privacy protection, and cybersecurity.

Flight operations in cities expose drones to considerable risks, such as collisions with people or infrastructure, technical malfunctions, and interference with other aerial or ground systems. In particular, in the event of a malfunction or accident, a drone could cause significant damage to both the population and infrastructure, making risk management an absolute priority. In addition, there are privacy concerns, as drones are capable of collecting potentially sensitive video and audio data. Lastly, cybersecurity represents another critical issue: drones could be vulnerable to cyberattacks that compromise the mission or cause deliberate harm.

In this context, there is a growing need to develop advanced tools for the safe planning of drone flight operations, especially in densely populated environments. Planning must ensure not only the safety of citizens and infrastructure but also compliance with current regulations, such as those established by the Specific Operations Risk Assessment (SORA), which governs the use of drones in complex scenarios.

The research group referenced in this work has already made significant contributions to the study of these issues, with publications regarding the creation of risk maps for drone use in cities [1][2] and the planning of safe, low-risk routes [3]. Despite these advances, there remains a need to develop a practical and intuitive user interface that allows drone operators to use these advanced tools to safely manage their flight missions.

This thesis work aims to address this need by developing a web app that enables operators to plan flight missions in urban environments, providing a visualization of risk areas and suggesting safe routes.

1.2 Objectives

This thesis is part of a larger project aimed at developing innovative tools for planning safe drone routes, with a specific focus on operations in urban environments. The main objective of the project is to address the challenges related to flight operation safety, risk management, and regulatory compliance, by providing advanced technological solutions that minimize potential hazards associated with drone use in complex and densely populated contexts.

The project aims to develop a digital infrastructure capable of supporting operators in flight mission planning by offering risk assessment tools and suggesting routes that minimize the likelihood of accidents. The ability to evaluate and mitigate risks is essential to ensure that drones can operate safely, even near densely populated areas, critical infrastructure, or zones with significant operational restrictions.

Furthermore, the project aims to support compliance with international regulations and guidelines, such as the SORA (Specific Operations Risk Assessment) framework, which sets strict standards for the safety of drone operations in complex environments. Adopting these guidelines not only helps ensure safety during missions but also facilitates the authorization process of operations by the competent authorities.

The overall objective of this work is to contribute to the development of an ecosystem of digital tools that, by integrating technologies such as risk assessment and autonomous planning, can improve the safety and efficiency of drone operations in urban settings.

1.3 Contribution

The main contribution of this thesis is the development of an interactive web app for the safe planning of drone flight operations in urban environments. The specific goal of the thesis is to provide an accessible and intuitive tool that allows drone operators to plan safe flight routes, assess associated risks, and ensure compliance with current regulations. This digital tool aims to bridge the gap between sophisticated risk planning technologies developed in the academic context and the practical needs of operators in the field.

The first key contribution is the development of the user interface, which allows users to interact with maps and flight planning tools in a simple and intuitive way. The web app enables operators to define flight routes, select start and destination points, and configure operational parameters such as drone altitude and payload. The interface is designed to ensure optimal management of critical information, providing users with a clear and immediate visualization of risk areas.

The second contribution lies in the integration of the risk planning framework developed by the research group with the web app's back-end system. The ROS (Robot Operating System) server receives the data sent from the user interface, processes the information, and returns the generated risk maps. This process allows for a rapid risk assessment, enabling users to quickly adjust planned routes to minimize operational risks. Communication is handled via WebSocket, ensuring a constant data flow between the web application and the ROS server, which is essential for keeping the information updated throughout the entire planning process.

Another innovative aspect of this work is the web app's ability to implement advanced risk visualization and management functions, making analytical tools accessible to users who were previously confined to academic research or complex specialized software. The risk maps generated by the server are displayed overlaid on the planned route map, allowing operators to immediately assess the level of risk associated with each zone crossed by the drone.

Finally, the web app provides a modular and scalable platform that can be extended with new functionalities to further improve the precision and safety of flight operations. Although these functionalities are not yet implemented, the structure of the application is designed to facilitate future additions, thus contributing to a flexible planning ecosystem adaptable to users' needs and technological advancements.

Therefore, the contribution of this thesis is not limited to the development of a simple web application but extends to the creation of a comprehensive infrastructure for the safe planning of drone missions, integrating advanced risk management technologies with intuitive user interfaces. This system can support a wide range of operational scenarios, with potential applications in sectors such as surveillance, logistics, and environmental monitoring.

1.4 Outline

The thesis is structured into the following chapters, each addressing a key aspect of the development of the web app and the context in which it operates.

In Chapter 2, the state of the art regarding the currently available tools for planning drone flight missions and the technologies that assist operators in complex environments is presented. An overview of the main international regulations governing the use of drones is also provided, with particular attention to the SORA (Specific Operations Risk Assessment) guidelines, which are essential for understanding the legislative framework and the safety implications in operations within populated areas.

Chapter 3 is dedicated to the technical background of the project. Here, the theoretical foundations of the safe planning framework used for risk assessment and the generation of risk maps are examined. Additionally, a general overview of ROS (Robot Operating System), the system used to manage communication and the computation of risk maps,

will be provided, highlighting ROS's role in coordinating safe planning and integration with the web app.

In Chapter 4, the development of the interface between ROS and the web app is described in detail. This chapter explains the design and implementation process of the application, with a focus on data management, user interaction, and real-time communication with the ROS server for processing risk maps and planning safe routes.

Chapter 5 presents the results obtained, showcasing the implemented functionalities and the overall operation of the web app. The communication between the web app and the ROS server is described, illustrating a complete storyline of the flight planning phases, from defining the starting and destination points to visualizing risk maps and generating the optimized route.

Finally, in Chapter 6, the conclusions of the work performed are drawn, and potential extensions and future improvements are discussed.

CHAPTER 2

2. State of the art

This chapter delves into the regulations governing the use of drones, with particular attention to the Specific category and the SORA (Specific Operations Risk Assessment) risk assessment system, which is essential for operations in urban environments.

2.1 Regulations

The current regulations for the use of drones are primarily governed by EASA (European Union Aviation Safety Agency) and ENAC (National Civil Aviation Authority) in Italy. These regulations govern drone operations based on risk categories and the characteristics of the flight environment. EASA introduced Regulation (EU) 2019/947, which divides drone flight operations into three categories that manage operations in three different types of risk scenarios.

1. **Open**: Low-risk operations that do not require authorization but must comply with certain weight and distance limits, such as flying below 120 meters and away from people. This category is further divided into three subcategories: A1 (flying over people but not gatherings), A2 (flying close to people), and A3 (flying far from people).
2. **Specific**: This includes higher-risk operations with civil drones, where safety is ensured by the drone operator obtaining operational authorization from the national competent authority before commencing the operation. To obtain operational authorization, the drone operator is required to conduct a risk

assessment, which will determine the necessary requirements for the safe operation of the civil drone(s).

3. **Certified**: High-risk operations, similar to those in traditional aviation, require certifications for the drone, the operator, and the pilot.

The Specific category is of particular interest for operations in urban contexts as it covers missions with a moderate level of risk that cannot be classified as Open but do not require the stringent certifications of Certified operations. This is the category we take as a reference in this thesis for flight operations.

Examples of UAS operations in the "Specific" category include:

- BVLOS – Beyond Visual Line of Sight
- When using a drone with MTOM (Maximum Take-Off Mass) > 25 kg
- Flying above 120 m from the ground level
- When dropping material
- When using a drone in an urban environment with an MTOM > 4 kg or without a class identification label

Operators intending to fly in urban environments must conduct a thorough risk assessment to obtain authorization. To operate in the Specific category, operational authorization is required, which can be obtained through various methods:

1. Declaration based on a Standard Scenario (STS), applicable from January 1, 2024.
2. Operational authorization based on a PDRA (Predefined Risk Assessment), which are risk assessments predefined by EASA.
3. Authorization without a PDRA, using the SORA (Specific Operations Risk Assessment) methodology.

4. Obtaining a Light UAS Operator Certificate (LUC), which allows for operations with fewer restrictions.

Figure 2. ENAC and EASA logos

2.1.1 STS

The European Standard Scenario (STS) is a predefined operation described in Appendix 1 of Regulation (EU) 2019/947. It is one of several options for operators to initiate their operations. An operator is not required to obtain operational authorization to conduct an operation covered by an STS. If it is verified that the operation can be covered by an STS, it is sufficient to submit a declaration to the National Aviation Authority (i.e., ENAC in Italy) of the state of registration.

So far, two EU STS have been published:

- **STS 01** – VLOS in a controlled land area in a populated environment;
- **STS 02** – BVLOS with airspace observers on a controlled aircraft.

2.1.2 PDRA

The Predefined Risk Assessment (PDRA) is an operational scenario for which EASA has already conducted a risk assessment and has published as an acceptable means of compliance (AMC). The published PDRAs so far are:

- **PDRA-S01** — Agricultural work, short-range cargo transport operations
- **PDRA-S02** — Surveillance, agricultural work, short-range cargo operations
- **PDRA-G01** — Surveillance, long-range cargo operations
- **PDRA-G02** — All ranges of operations
- **PDRA-G03** — Linear inspections, agricultural work

The PDRA still requires operational authorization from the National Aviation Authority (NAA), but the process is simplified. Instead of conducting a risk assessment, the UAS operator can simply fill out the PDRA table, prepare the Operational Manual (OM), and submit the application to the registering NAA. The PDRA table is a sort of checklist on how to develop the procedures that must be included in the OM.

If the operation falls within the scope of one of the published PDRAs, the applicant can quickly develop the operator's manual and evidence of compliance using the PDRA table to demonstrate that the operation is safe. The review of documentation will also be simplified for the NAA.

Some entities caution against the use of STS and PDRA, suggesting that it would be safer to involve all stakeholders in the SORA content process and review. Operations using STS require only a declaration and not an actual authorization, so caution is needed when using these predefined schemes.

2.1.3 LUC

The Light UAS Operator Certificate (LUC) is an optional certificate for drone operators that grants specific privileges, such as the ability to conduct operations in the Specific category without needing to obtain separate authorizations from the National Aviation Authority (NAA). This allows operators to self-assess and self-authorize missions, reducing wait times for formal approvals while maintaining high safety standards.

The NAA evaluates the operator's ability to manage risk and comply with safety regulations before issuing the LUC. The level of privilege can include:

1. Conducting operations according to standard scenarios without the need to submit declarations.
2. Self-authorizing operations that fall within the Predefined Risk Assessments (PDRA).
3. Self-authorizing all operations without requiring external authorizations.

However, the LUC is not a "blank check": the operator can only conduct operations described in the approval terms established by the NAA and applicable regulations.

2.1.4 SORA

In the event that an operation is not covered by an STS or PDRA, applicants are required to conduct a risk assessment, identify mitigation measures, and meet safety objectives. To this end, the risk assessment methodology known as SORA (Specific Operations Risk Assessment) was developed by JARUS (Joint Authorities for Rulemaking on Unmanned Systems) and subsequently recognized by EASA. This methodology is of particular relevance to the work behind this thesis.

With SORA, drone operations in the Specific category are assessed, ensuring the safety of aerial operations based on a detailed risk analysis. This methodology allows

operators to evaluate the risks associated with their missions and to implement appropriate mitigation measures.

The SORA process consists of several key phases, each designed to identify and reduce risks to both people on the ground and other aircraft. Below are the main phases:

1. **Definition of the Operation**: In this phase, the operator describes the planned mission, geographical areas, duration, and type of drone used. The operational concept (ConOps) is also defined, providing an overview of the operation.
2. **Preliminary Ground Risk Class (GRC)**: The risk to people on the ground is assessed and classified based on various factors such as population density, type of operation (VLOS or BVLOS), and drone size. The preliminary GRC is an estimate of risk based on the initial analysis of the operational area.
3. **Mitigation of Ground Risk**: After identifying ground risk, the operator can implement measures to reduce it, such as limiting the flight area or adopting safety technologies (e.g., parachutes, safe landing systems).
4. **Preliminary Air Risk Class (ARC)**: Here, the risk of collision with other aircraft is assessed. The risk is determined by the flight environment, expected altitudes, and proximity to other aircraft.
5. **Mitigation of Air Risk**: The operator adopts measures to reduce the risk of collision, such as using detection and avoidance technologies or adopting specific flight corridors.

By combining the values of residual air and ground risk, the intrinsic risk values of the entire operation, known as SAIL (Specific Assurance Integrity Level), are defined. A high-value SAIL represents an operation with a high potential risk. Depending on the SAIL index obtained, the operation will be considered more or less risky:

- Low Risk (SAIL I and II)
- Medium Risk (SAIL III and IV)

- High Risk (SAIL V and VI)

For SAIL I and II, compliance can be demonstrated with a declaration. In these low-risk zones, all operations falling under standard scenarios (STS) or PDRA can be found.

If the SORA result (after the application of mitigations) yields a SAIL III or SAIL IV, an Operational Authorization will be required. At this point, the competent authorities will indicate to the operator the possibility of needing to carry out a design verification of the entire aircraft or one of its subsystems, especially if this helps to mitigate risk and reduce the number of SAIL by 1 or 2 units.

If the result is SAIL V or SAIL VI after the application of mitigations, the operator must undergo an aircraft certification process according to the airworthiness standards described in the PART21 document, contained in Implementation Rule 1702/2003, which establishes the requirements and procedures for the certification of aircraft, products, parts, and related organizations.

Once the SAIL is determined, the applicant must review the 24 operational safety objectives (OSO) and demonstrate compliance with a level of robustness that increases with the SAIL (e.g., operations with a higher SAIL will need to demonstrate a more stringent standard).

The final step is to assess the risk level of the area adjacent to the operational area and to meet the requirements to protect that area and contain the drone within the operational area in case of unwanted flight.

After completing all the steps of SORA, the operator must submit the application form for operational authorization, the risk assessment, compliance evidence, and finally the operator's manual to the NAA. The NAA, after verifying the information provided, will issue the operational authorization.

Risk analysis is essential for operations in the Specific category due to the higher level of risk compared to the Open category. The SORA method was designed as a guide to

identify the right mitigations and reduce risk to an acceptable level. However, its application can be complex for many UAS operators, as it requires a deep understanding of drone operations and manual data collection that can take days. This is problematic for operators needing rapid approvals.

Moreover, with the increase in drone operations, national authorities will have to manage a growing number of requests, which could slow down the obtaining of approvals. While standard scenarios and predefined risk assessments (PDRA) can help, they will not completely solve the problem, especially for operations outside the EU, where SORA may be used without the applicability of standard scenarios.

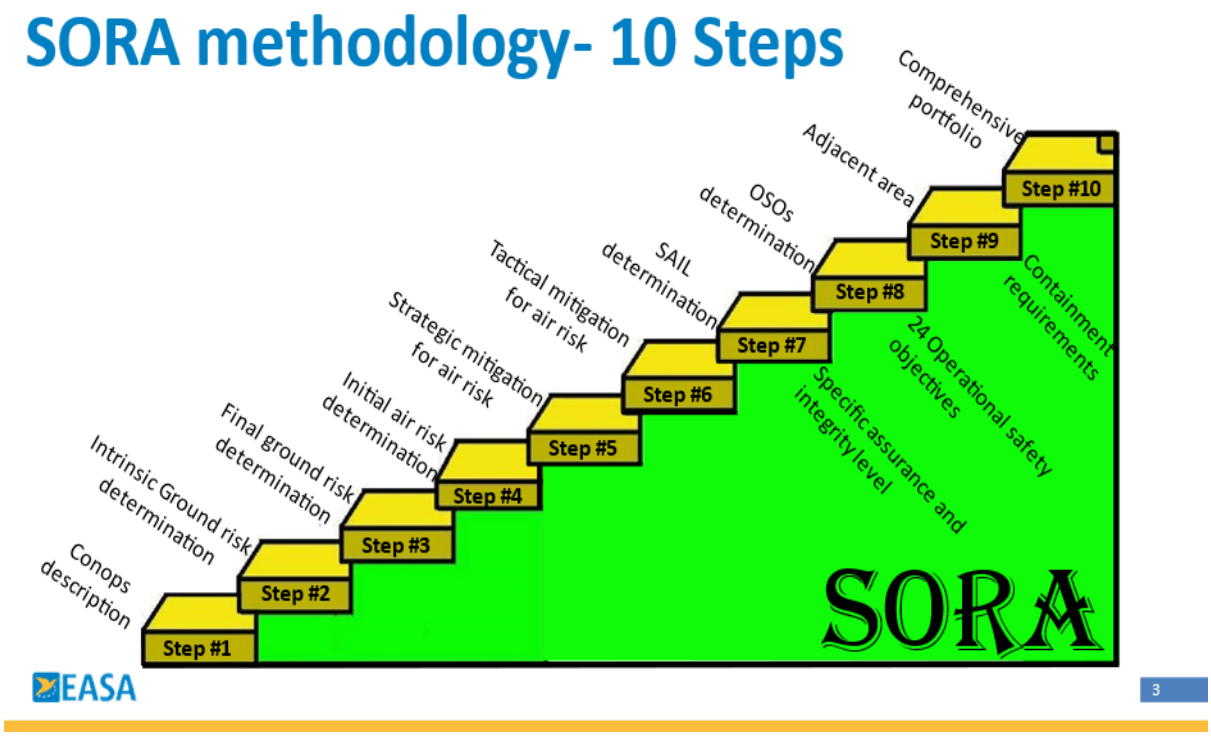

Figure 3. SORA steps

2.2 Available tools

Currently, the tools available to assist operators in obtaining a SORA assessment are limited and often require a high level of technical expertise, in addition to significant costs. One of the few existing platforms is Online SORA Samwise, which allows users

to perform a risk assessment based on SORA guidelines, but its use can be complex. This type of tool is not fully automated; the operator must still possess a thorough understanding of the current regulations and the processes necessary to manage safety and the risks associated with drone operations.

Tools like this aim to streamline the process, but they often require the involvement of an expert to assist UAS operators in entering data and verifying safety measures. These consulting services, while helpful, can entail high costs that may be prohibitive for small- to medium-sized operator groups. Many companies offering such tools provide support packages that include personalized assistance but at considerable prices.

In addition to paid platforms, some free tools provided by authorities like EASA or national aviation authorities can be used to conduct manual SORA assessments. However, these tools require the operator to independently gather all necessary information and demonstrate compliance with the required standards. For instance, EASA publishes guidelines and standard forms that operators can follow to perform their own risk assessments, but there are still no fully automated and free platforms that can replace human input and technical skills.

The main challenge lies in the fact that many drone operations need to be approved quickly to meet commercial demands, but the lengthy timelines required to manually complete SORA procedures can slow down operational processes, especially as the number of drone operations continues to rise.

In conclusion, while there are tools that aim to facilitate the risk assessment process, most still require significant expertise and associated costs, highlighting the need for new, more accessible, and automated solutions for a broader range of UAS operators. This thesis work fits into this context, aiming to provide a simple and fast tool for rapidly obtaining a risk assessment.

CHAPTER 3

3. Background

This chapter examines the technical and conceptual context underlying the development of the safe drone navigation project. Specifically, two key elements will be explored: the safe navigation framework and the technological infrastructure that enables its implementation. A central aspect is the use of the Robot Operating System (ROS), an advanced middleware platform that facilitates communication between different robotic components and allows for the implementation of complex planning and control algorithms. Finally, the main software tools used to build the application will be described.

3.1 ROS

The Robot Operating System (ROS) is an open-source middleware platform designed to facilitate the development of complex robotic applications. Although its name suggests it might be an operating system, ROS is actually a framework that provides tools and libraries for inter-process communication. Its primary goal is to standardize robotic software development, allowing developers to focus more on the specific functionalities rather than the technical details of communication and interoperability between different components.

ROS supports a distributed architecture in which various nodes, each responsible for a specific task (e.g., navigation, perception, or motion control), can communicate with one another through a publish/subscribe mechanism. Communication can take place

through different channels, such as services and topics, though in the context of this project, services are primarily used.

In ROS, services enable synchronous communication between a client node and a server node. A service allows a node to send a request and wait for a response from another node, which is useful in operations requiring immediate processing or direct feedback, such as path planning. This is exactly what happens in the project: the web app sends service requests to the ROS server to calculate paths and create risk maps based on the drone's data and surrounding environment. ROS services ensure secure and coordinated operations, enhancing system integration and the overall effectiveness of the drone's autonomous navigation system.

Additionally, ROS provides high-level services such as map management, localization, and path planning, all of which are essential for developing autonomous systems. Its modularity allows developers to extend functionality with additional packages, many of which are created and shared by the open-source community.

In the context of this flight operation planning project for drones, ROS plays a critical role in managing operations and integrating various subsystems. Specifically, it is used to manage the communication between the web app and the server, which performs advanced calculations like risk map creation and path planning. ROS's ability to handle sensors, actuators, and data processing allows for risk-aware planning, meaning flight planning that takes into account risks associated with the area being flown over, including factors like population density and no-fly zones. Thus, ROS provides a robust and flexible infrastructure that supports the safe management of autonomous flight operations, making it possible to implement advanced navigation systems.

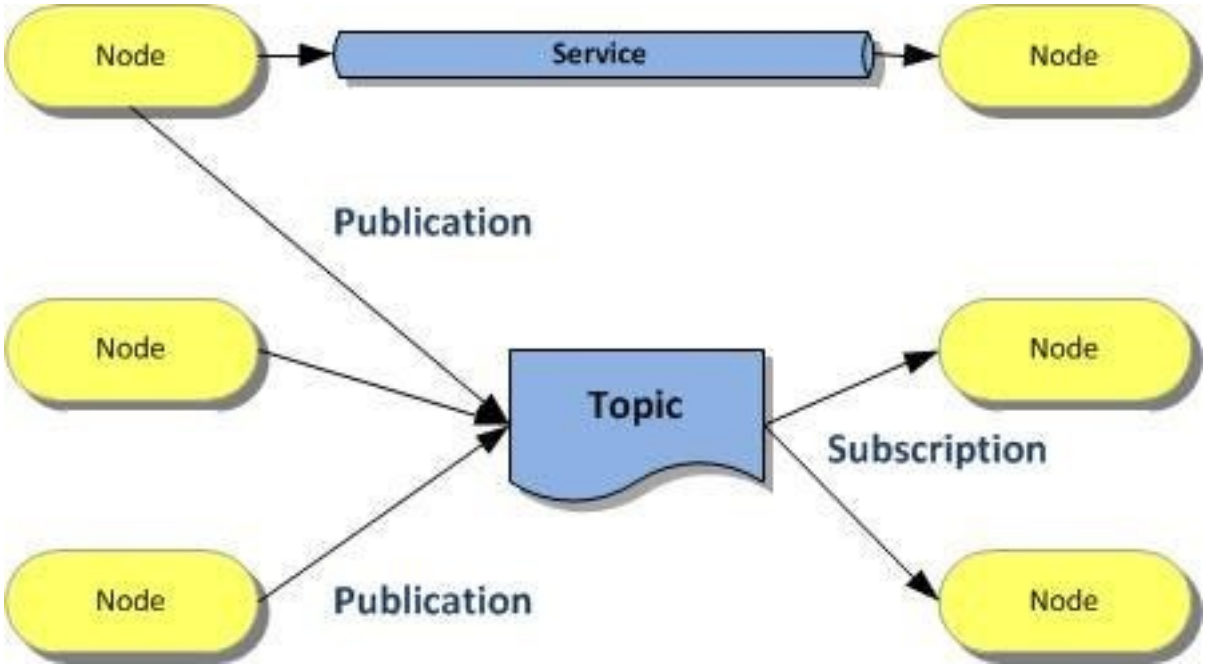

Figure 4. ROS system diagram

3.2 Framework

The safe navigation framework discussed in this work is designed to ensure that drones can operate efficiently and safely in complex and populated environments, such as urban areas. Safety is a key aspect of drone operations, as these devices must avoid obstacles, comply with safety regulations, and minimize risks to people and infrastructure. To this end, the framework integrates two crucial elements: the creation of risk maps and risk-aware flight planning.

3.2.1 Risk-aware planning

Risk-aware planning is an advanced approach to route management for drones and other autonomous vehicles that takes into account the potential risk associated with each point along the path. Unlike traditional planning, which primarily optimizes aspects such as distance or flight time, risk-aware planning incorporates risk variables to ensure the safety of both the drone and the surrounding people and infrastructure.

This type of planning uses risk maps that divide the flight area into cells, each of which is assigned a risk value based on factors such as population density, the presence of obstacles, no-fly zones, and environmental conditions. The system processes the route by selecting the least risky areas to minimize the potential for accidents or damage.

The result is a flight path that not only optimizes time or energy consumption but also minimizes exposure to hazardous areas, making this approach essential for operations in urban environments or where stringent safety restrictions apply. Risk-aware planning is thus crucial to ensuring regulatory compliance and mitigating risks in complex operational scenarios.

3.2.2 Risk map generation

The safe navigation framework employs an advanced approach to risk map generation, which is crucial for planning safe flights in complex environments. These maps are cell-based, where each cell represents a specific geographic area associated with a risk value. Each square cell is equidistant, and the map covers a predefined wide area. The generation of risk maps takes into account various factors: from environmental characteristics to drone parameters, such as flight direction and altitude, along with external factors like wind.

The risk in each cell is calculated as the probability of causing harm to people or property, considering scenarios of accidental drone descent. The final risk map is a two-dimensional matrix of cells that geographically represent the risk to the population below. Each cell is identified using a local NED (North-East-Down) coordinate system, allowing for precise association of each area with its corresponding risk level.

The map creation process is based on a multilayer model that integrates different informational layers, each containing homogeneous data such as population density, obstacles at varying heights, the level of protection offered by the terrain, and no-fly zones. Currently, the population density layer is based on realistic but not real data due

to technical and privacy reasons. In the future, real-time data reception will be integrated to provide a continuously updated risk assessment. These layers are then combined to create a global map that considers both the risk associated with potential drone descent events and the presence of flight-restricted areas and obstacles.

Essentially, the framework first produces event maps for each descent scenario, and then combines these maps with data on obstacles and no-fly zones to generate the final risk map. This multilayer approach allows for detailed risk analysis associated with each map cell, enhancing the safety and efficiency of drone flight planning.

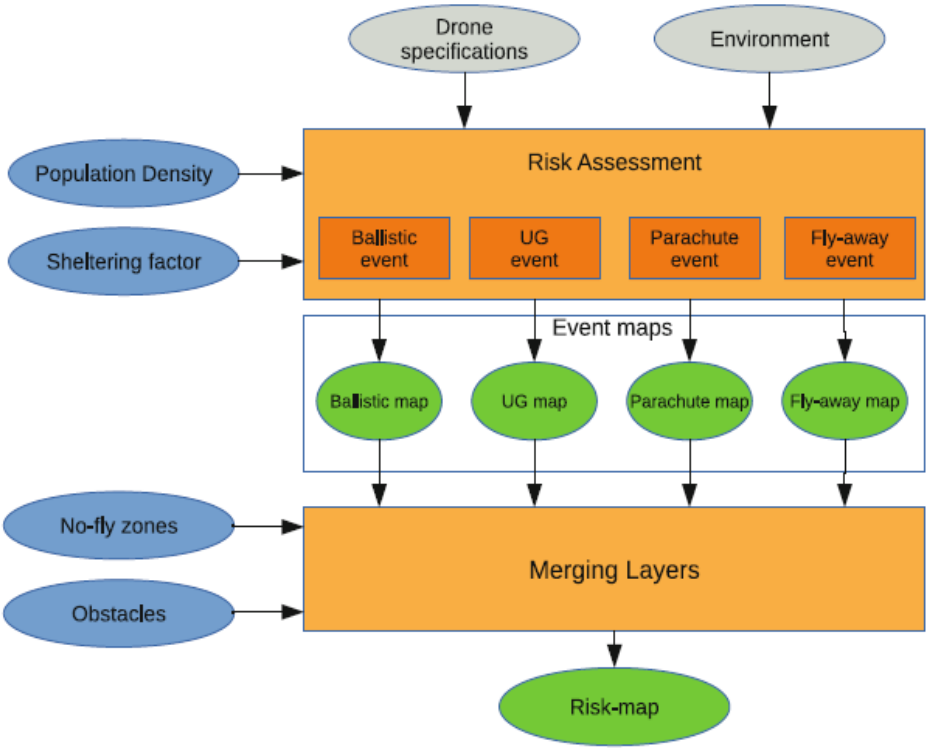

Figure 5. Main architecture of risk map generation

3.2.3 Path planning

Once the risk map is calculated, the path planning algorithm computes an optimal route that minimizes risk to people on the ground, based on the risk values provided by the generated map. The planned path consists of a series of waypoints that the drone can follow during flight. Risk is measured in terms of the probability of causing incidents per flight hour, so the planner seeks to minimize flight time in high-risk areas, thereby reducing exposure to potential hazards.

The planning method employed is based on the RRT* (Optimal Rapidly-exploring Random Tree) algorithm. RRT (Rapidly-exploring Random Tree) is a sampling-based path-planning algorithm designed to efficiently explore large, complex spaces. RRT generates a tree by progressively exploring the search space, connecting random points (nodes) with the nearest point on the existing tree and expanding into unexplored directions. This makes it well-suited for planning in complex spaces, such as those involving obstacles or environmental constraints.

RRT* is an optimized version of RRT that, in addition to connecting new nodes to the nearest point, seeks the connection with the lowest movement cost, which in the context of drone flight corresponds to risk. Moreover, the RRT* algorithm includes a "rewiring" phase that allows continuous reorganization of the tree to improve the path, gradually converging towards an optimal solution over time.

Since risk values are expressed per flight hour, the concept of Time Reliance directly relates to the risk associated with the drone’s flight. Risk depends not only on the path but also on the exposure time to potential hazards. The longer the drone flies over a populated area, the higher the likelihood of causing damage in the event of a failure or accident. Therefore, risk is modeled as proportional to flight time; the longer the time spent in a risky zone, the higher the probability of an incident occurring. This approach is inspired by the concept of reliability in complex systems, where the risk of failure increases with system usage time.

In summary, RRT* combines the efficiency of spatial exploration with path optimization in terms of risk and time, ensuring that the drone follows the safest route while minimizing exposure to high-risk zones during flight. This approach enables the planning of paths that are not only efficient in terms of flight but, more importantly, safe, reducing the likelihood of accidents in complex or densely populated environments.

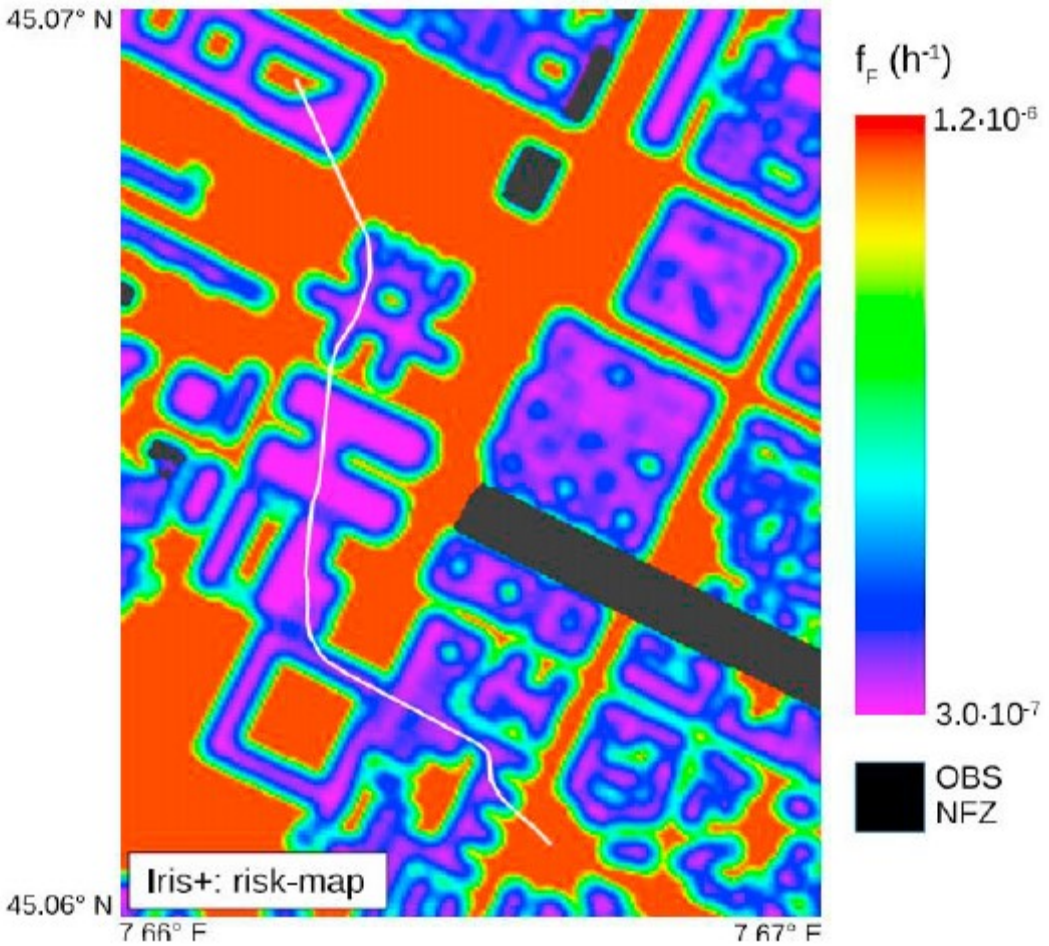

Figure 6. Example of risk map with Iris+ aircraft. The white line is the minimum risk path computed with the risk-aware path planner.

3.3 Software Tools

In this chapter, the software tools used for the development of the webapp will be described, including the development environment, programming languages, and frameworks that were essential in completing the project.

3.3.1 Editor: Visual Studio Code

The development environment used was Visual Studio Code, a highly versatile editor that supports a wide range of programming languages. With the integration of extensions like Git for version control, VSCode was essential in maintaining an efficient and organized workflow. Its ability to manage multiple environments and its ease of use make it an ideal tool for full-stack development, from front-end code to back-end logic.

3.3.2 Markup languages: HTML e CSS

HTML (HyperText Markup Language) was used to define the structure of the web app, creating a solid foundation for the content and interface. Each React component relies on a markup architecture defined through JSX (a syntax similar to HTML) for the dynamic creation of user interface elements. The HTML semantics provide a clear and accessible structure, contributing to a well-organized layout for users.
For style and visual design, the project utilized CSS (Cascading Style Sheets) to define the appearance and layout of the web app. CSS was employed to manage the positioning of elements, typography, and colors, with a particular focus on ensuring the app's adaptability to different screen resolutions by using responsive layouts.

3.3.3 JavaScript, React.js e Node.js

The core of the project is JavaScript, the primary language for client-side interaction logic. Thanks to its flexibility and power, JavaScript enabled the creation of a responsive user interface, management of communications with the ROS server, and implementation of dynamic functionalities for processing and visualizing flight data.

For front-end development, React.js was used, a JavaScript framework that simplifies the creation of dynamic user interfaces through state management and reusable components. Each part of the user interface, from input forms for flight data to map visualizations, was developed as an independent component, which enhanced the modularity and maintainability of the code. React's Virtual DOM management allowed for optimized interface updates, improving performance.

For the back-end, Node.js was used along with Express.js, a minimalist framework for building RESTful APIs. Node.js, being a server-side JavaScript environment, facilitated integration with ROS and managed client-server requests asynchronously. Express.js simplified the definition of routes for the APIs, enhancing communication with the ROS server to request path planning and manage the returned data.

3.3.4 ROS

The core of the integration with the operations calculation system is the ROS (Robot Operating System) middleware, used for managing robotic operations and communication between various nodes of the system. In this project, ROS topics were not used for data exchange; instead, services were employed, which provide a synchronous communication mode between client and server. The roslibjs library was fundamental for interfacing the web app with ROS via the WebSocket protocol, allowing for the direct and reliable sending of planning requests and the reception of path data.

3.3.5 ROSbridge_suite

ROSbridge_suite is an essential package in the project that enables communication between a ROS (Robot Operating System) system and non-native ROS applications, such as a web app. This package provides a standard interface that allows sending and receiving messages between the ROS server and external applications through common protocols, such as JSON over WebSocket, REST, or other standard communication formats.

In the context of our project, ROSbridge_suite allows our web app to send commands and receive data from a ROS system that handles the calculation of risk maps and paths. This is crucial as it enables our application to interact with the ROS system without requiring the direct installation of the ROS middleware within the web app, thereby increasing portability and simplifying the system architecture.

ROSbridge_suite consists of several components, the most important of which is the ROSbridge_server, which acts as a central server to manage requests and responses between ROS and external applications.

The ROSbridge_suite package includes:

- **ROSbridge_server**: The main component that listens for and manages external connections.
- **Protocols**: Supports various communication protocols such as WebSocket, HTTP/REST, TCP, etc.
- **WebSocket Client**: A client that enables communication via WebSocket with ROS, ideal for interactive web applications like the one we are developing.

The ROSbridge_server is the heart of the ROSbridge_suite package. It is a server that translates ROS messages into a web-readable format, such as JSON, and sends them via WebSocket or HTTP to connected clients. This server allows non-ROS applications to interact with the ROS system transparently. For our project, the ROSbridge_server

enables the web app to send service requests, receive feedback, and retrieve the necessary data to display risk maps and plan paths.

The operation of the ROSbridge_server is based on a request-response cycle:

1. **Request from the WebApp**: Our web app sends a request in JSON format to the ROSbridge_server. This may include commands for drone control, requests for sensor data, or other information managed by ROS.
2. **Translation to ROS Format**: The ROSbridge_server translates this request into a ROS message and sends it to the appropriate ROS node.
3. **Processing the Request**: The ROS system processes the request and generates a response, which is sent back to the ROSbridge_server.
4. **Sending the Response**: The ROSbridge_server translates the response into JSON and sends it to the web app, which displays it or performs the appropriate action.

One of the main advantages of using the ROSbridge_server is that it allows exposing ROS functionalities to a wide variety of clients without requiring those clients to be part of the ROS framework. This is crucial in our project since the web app, being based on technologies like React and Node.js, cannot natively interface with ROS. Thanks to ROSbridge, we can display data without compromising the modularity and lightweight nature of the web app.

3.3.6 Leaflet

Leaflet is one of the most widely used JavaScript libraries for creating interactive maps in web applications. Its main strength lies in its lightweight nature (less than 40KB) and extensive functionality provided by a broad ecosystem of available plugins. These plugins enhance its capabilities with additional features such as drawing tools,

management of vector and raster layers, support for geospatial data in GeoJSON format, and more.

Leaflet is compatible with a wide range of map sources, including OpenStreetMap, Mapbox, Google Maps, and many others. This makes Leaflet an ideal tool for projects requiring the integration of detailed, customizable maps with real-time or context-specific data, such as the risk maps used in our project.

The main features of Leaflet include:

- **Interactive Map Visualization**: Leaflet allows users to visualize maps that they can explore through pan and zoom tools, providing an interactive and dynamic experience.
- **Layer Management**: Leaflet enables the addition of multiple layers to the map, such as tile layers, markers, polylines, and polygons. This layering capability is crucial for our project, as it allows for the overlay of risk maps, flight paths, and other relevant information on the selected area.
- **Event Management**: Leaflet handles events such as clicks, hovers, drags, and zooms. This enables a high level of interactivity and customization of the map, allowing for specific actions when users interact with certain elements on the map, such as selecting areas or points of interest for path planning.
- **Support for Geospatial Data**: Leaflet provides support for the GeoJSON format, a standard for representing geographic data such as points, lines, and polygons, which is widely used for visualizing information like boundaries, paths, or selected areas.

One of Leaflet's major strengths is its modularity. Numerous libraries and plugins extend Leaflet's core functionalities, allowing for the creation of highly customized and specific applications. In our project, Leaflet is used alongside the React-Leaflet library, a wrapper that facilitates the integration of Leaflet into React applications. With React-Leaflet, we can leverage the reactivity and modularity of React to manage the state and

interactions of the map efficiently, improving the rendering management and dynamic updating of map elements.

For example, through React-Leaflet, it is possible to synchronize the drawing of the flight area with the selection of specific points for path planning, dynamically updating the user interface without needing to reload the page.

A key plugin for the project is Leaflet-Draw, which adds advanced drawing tools to Leaflet, such as the ability to draw polygons, rectangles, circles, and lines directly on the map. Thanks to this plugin, users can interact with the map intuitively by drawing the flight area, modifying shapes, and receiving real-time coordinates of the selected areas. This information is then used to calculate the risk map, enhancing the safety and effectiveness of flight operations.

3.3.7 Additional tools

The project was versioned using Git, with the repository hosted on GitHub. This enabled efficient code management, tracking changes to facilitate collaboration with potential future developers.

The WebSocket protocol was used to maintain a continuous bidirectional connection between the web app and the ROS server, allowing for rapid exchange of the necessary information for flight planning.

In conclusion, the combination of these tools and technologies made it possible to develop a powerful and dynamic web app capable of handling drone flight planning in complex environments safely. The integration of HTML, CSS, JavaScript, React, and Node.js created a coherent and modular workflow, while ROS provided support for mission management.

Figure 7. Tools used

CHAPTER 4

4. Development

In this chapter, we will delve into the technical and practical aspects of the implementation of communication between ROS and the web app, highlighting how the system was built and the challenges faced during the process.

4.1 Frontend

The web application was developed using React, a popular JavaScript framework for building user interfaces. The structure of the front-end is based on a component-based architecture, where each component handles a specific part of the interface and functionality. The code is modular, and each component is responsible for a defined aspect of the application's logic. Below a detailed description of the key components is discussed.

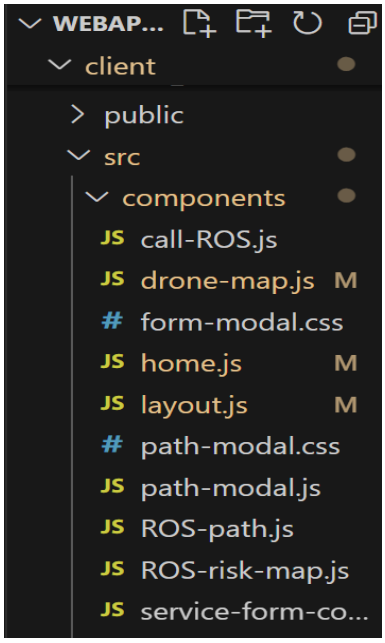

Figure 8. Structure of /src directory

4.1.1 Home.js

The Home.js component plays a crucial role in the architecture of the application. It handles several important functions, such as the selection and addition of new drones, as well as user interaction for submitting the necessary data for a flight mission.

The component allows users to either select an existing drone from a list or add a new one through a dynamic form. The form is toggled by clicking a button, which alternates visibility between selecting from existing drones and displaying the form to add a new drone. A dropdown menu shows the list of already saved drones, allowing the user to select one and move on to the next screen. The technical data related to the selected drone, stored in the database, is fetched via API and passed to the next component.

Alternatively, if the user wants to use a drone not already in the database, they can fill out a form containing the following fields:

- **Model**: The name of the drone model
- **Max cruise speed**: The drone's maximum cruise speed
- **Max flight time**: The maximum flight time the drone can sustain
- **Mass**: The drone's weight in kilograms
- **Radius**: The drone's radius (or interaxis)
- **Max payload**: The maximum weight the drone can carry

These technical parameters are defined for each drone and can be displayed in a technical sheet. Once the new drone and its parameters are defined, the data is saved to the database through an API and passed to the next component.

Upon clicking the 'Submit' button, a function is called to distinguish between the two cases. If a drone is selected from the list, its data is retrieved and passed along. If the

form is submitted, the function checks that the form fields are filled correctly before saving the data and passing it to the next step.

```
6   export const Home = (props) => {
49    const handleSubmit = (event) => {
50      event.preventDefault();
51
52      if (droneSelected) {
53        const droneFound = droneList.filter(drone => drone.id === droneSelected);
54
55        if (droneFound) {
56          console.log("Drone found:", droneFound[0].model);
57          props.setModelName(droneFound[0].model);
58        } else {
59          console.log("No drone found with this id");
60        }
61      } else {
62
63        let flag = false;
64
65        if (model === '') { setShowErr(true); setModel(''); flag = true; }
66        if (max_cruise_speed === '') { setShowErr(true); setMCS(''); flag = true; }
67        if (max_flight_time === '' || !(parseInt(max_flight_time) > 0)) { setShowErr(true); setMFT(''); flag = true; }
68        if (mass === '') { setShowErr(true); setMass(''); flag = true; }
69        if (radius === '') { setShowErr(true); setRadius(''); flag = true; }
70        if (max_payload === '') { setShowErr(true); setMaxPayload(''); flag = true; }
71
72        if (flag) return;
73
74        setShowErr(false);
75
76        const formData = new FormData();
77        formData.append('model', model);
78        formData.append('max cruise speed', max_cruise_speed);
79        formData.append('max flight time', max_flight_time);
80        formData.append('mass', mass);
81        formData.append('radius', radius);
82        formData.append('max payload', max_payload);
83
84        API.addDrone(formData)
```

Figure 9. Section of handleSubmit function in home.js

A CSS file (App.css) manages the layout and visual elements of the page, ensuring that the Home.js component remains simple and intuitive for the user. This CSS file is responsible for defining styles such as positioning, spacing, fonts, colors, and responsiveness, ensuring that the interface is user-friendly and visually appealing.

4.1.2 Drone-map.js

In the project, one of the key components for user interaction with the simulated flight environment is the Drone-Map.js component. This component, built using the Leaflet and React-Leaflet libraries, is essential for providing an interactive map that allows users to select the area of interest for the calculation of the risk map and the planning of the drone's path. The component integrates with other modules of the web app, such as the flight mission form and the interface for selecting the start and end points for path planning.

Within the component, the user has the ability to use drawing tools to select the specific mission area. These tools are enabled through the integration of a plugin for Leaflet that allows the creation of shapes such as polygons, rectangles, and circles. For this application, only the creation of rectangles is enabled, as the software for calculating the risk map requires an area exclusively in this shape.

In this component, requestData is defined and managed through useState. This contains the data for the mission_bb, that is, the values of the flight mission's bounding box parameters, which will be sent to the ROS server for the map calculation. These include the coordinates of the northwest and southeast points of the selected area, model name, date and time, payload weight, altitude, and operating speed. The parameters defined here in JavaScript are managed on the ROS side in a .srv file written in C++. Both cases are shown below in the figure.

```
# UAV model
string uav_model
float32 payload_mass

#Mission Bounding Box
BoundingBox mission_bb

#Time: format YYYYMMDD_hhmm[ss]
string datetime

#Altitude in meters
float32 altitude_min
float32 altitude_max
uint32 altitude_resolution

#Resolution of the map
uint32 resolution

bool save_map_image
bool save_bag_file
bool use_meteo_layer
bool use_gnss_layer

---
#Bounding Box resized according to map discretization and resolution
BoundingBox discretized_bb

#identification number (coincides with the uav_model + datetime)
string id

uint32 image_height
uint32 image_width

sensor_msgs/CompressedImage risk_map_image

sensor_msgs/CompressedImage gnss_errorHor_image
sensor_msgs/CompressedImage gnss_errorUp_image
sensor_msgs/CompressedImage gnss_hpl_image
sensor_msgs/CompressedImage gnss_vpl_image

float32 wind_velocity
float32 wind direction
```

Figure 10. .srv file for risk map service request

```
const [requestData, setRequestData] = useState({
  uav_model: props.modelName,
  mission_bb: {
    upperleft: { latitude: 0.0, longitude: 0.0 },
    lowerright: { latitude: 0.0, longitude: 0.0 }
  },
  datetime: "",
  payload_mass: 0.0,
  altitude_min: 0.0,
  altitude_max: 0.0,
  altitude_resolution: 5,
  resolution: 10,
  save_map_image: true,
  save_bag_file: false,
  use_meteo_layer: false,
  use_gnss_layer: false
});
```

Figure 11. requestData fields. These values are used to compute the risk map.

Using the tools offered by Leaflet, the creation, modification, and deletion of the shape (i.e., rectangle) for area selection are managed, in addition to the map visualization. The display of the risk map image returned by the server (with ImageOverlay) and the drone's path (with Polyline) is also handled.

```
196   return (
197     <Container className="map-container" onClick={handleClick}>
198       <div id="map">
199         <MapContainer
200           center={center}
201           zoom={13}
202           ref={setMap}
203         >
204
205           <FeatureGroup>
206             <EditControl
207               position="topright"
208               onCreated={_onCreate}
209               onEdited={_onEdited}
210               onDeleted={_onDeleted}
211               draw={drawOptions}
212             />
213           </FeatureGroup>
214
215           <TileLayer
216             attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
217             url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
218           />
219
```

Figure 12. Section of drone-map.js for the visualization of map and drawing tools

```
220           {mapDone && (
221             <ImageOverlay
222               url="http://localhost:3001/images/risk_map-0.jpg"
223               bounds={bounds}
224               opacity={0.5}
225               errorOverlayUrl="https://developers.google.com/static/maps/documentation/maps-static/images/error-image
226               alt="Risk Map"
227               interactive={false}
228             />
229           )}
230
231           <ModalComponent
232             requestData={requestData}
233             setRequestData={setRequestData}
234             setMapDone={setMapDone}
235             bounds={bounds}
236           />
```

Figure 13. Section of drone-map.js for the visualization of risk map and form component

Finally, two fundamental components are called: the one containing the form to be filled in with the mission parameters (i.e. ModalComponent) and the component that allows the selection of the start and end points of the mission for route calculation (i.e. PathModal). The MarkerHandler manages the creation of markers on the map for these start and goal positions.

```
<MarkerHandler />

<PathModal
  startCreated={startCreated}
  setStartCreated={setStartCreated}
  startSetting={startSetting}
  setStartSetting={setStartSetting}
  startPos={startPos}
  goalCreated={goalCreated}
  setGoalCreated={setGoalCreated}
  goalSetting={goalSetting}
  setGoalSetting={setGoalSetting}
  goalPos={goalPos}
  mapDone={mapDone}
  setWaypoints={setWaypoints}
/>

{startCreated && (
  <Marker position={startPos} icon={defaultIcon}>
    <Popup closeButton={false}>START</Popup>
  </Marker>
)}

{goalCreated && (
  <Marker position={goalPos} icon={redIcon}>
    <Popup closeButton={false}>GOAL</Popup>
  </Marker>
)}

{waypoints.length > 0 && (
  <Polyline positions={waypoints} pathOptions={{ color: 'blue' }} />
)}
```

Figure 14. Section of drone-map.js for the visualization of computed path and path selection component

4.1.3 Risk map

The service-form-component.js file manages the form submission. It displays the fields for the flight mission parameters, which is the requestData previously discussed. The form contains, in addition to the button for submitting the data, a button that allows resetting the fields.

When the completed form is submitted, the handleSubmit() function checks that all fields have been correctly filled in. Regarding the ‘datetime’ field, a specific function was created to ensure that the string sent is in the correct format and that the date entered is not earlier than the current time.

If there are one or more incorrect fields, or if an attempt is made to submit the form without first defining an area on the map, error messages are displayed, indicating what needs to be corrected.

If everything is correct, the riskMap() function is called. This establishes a connection with the ROS server, creates a service request with requestData, and finally calls the service. If the operation is successful, the images returned by the server are saved in the /public/images folder located in the backend, via the saveImage() API. A Promise is used to wait for each image to be saved, after which the service ends. If there are no errors, the risk map image is taken from the /images folder and displayed on the map over the selected area.

```
32  const request = new ROSLIB.ServiceRequest(requestData);
33
34  service.callService(
35      request,
36      async (result) => {
37        console.log('Result of /compute_risk_map is...', result);
38
39        const images = [
40          { key: 'risk_map_image', name: 'risk_map' },
41          { key: 'gnss_vpl_image', name: 'gnss_vpl' },
42          { key: 'gnss_hpl_image', name: 'gnss_hpl' },
43          { key: 'gnss_errorUp_image', name: 'gnss_errorUp' },
44          { key: 'gnss_errorHor_image', name: 'gnss_errorHor' }
45        ];
46
47        const saveImagePromises = images.map(async (image) => {
48          const imageData = result[image.key];
49          if (imageData && imageData.data) {
50            const base64Str = imageData.data;
51            const fileName = `${image.name}-${imageData.header.seq}.${imageData.format}`;
52
53            try {
54              const saveResult = await API.saveImage(fileName, base64Str);
55              console.log(`Image ${fileName} saved successfully:`, saveResult);
56            } catch (error) {
57              console.error(`Error saving image ${fileName}:`, error);
58            }
59          } else {
60            console.warn(`No valid data for image key: ${image.key}`);
61          }
62        });
```

Figure 15. Section of the service call to ROS server for risk map request

4.1.4 Path planning

The path-modal.js file creates a component that contains two buttons: one to set the start point and the other for the goal point for calculating the path that the drone must follow. Pressing either button activates the MarkerHandler function, which, as mentioned earlier, manages the click on the map to define the location of the point.

Once the two points are defined, a button appears that, when clicked, calls the pathPlanning() function defined in the ROS-path.js file. This function, similar to the previous riskMap() function, establishes a connection with the ROS server. It then creates a request message with the values of the required parameters: an ID that identifies the risk map image, the coordinates of the start and goal positions, and the altitude. These parameters are similarly defined, just like for the risk map, in a .srv file on the ROS side.

```
##path planning service
##the variable altitude is not used now,
##the flight altitude is defined with the map_manager,
##where the user can define the flight altitude,
##thus he obtains the resulting risk_map/output map
##

string id
sensor_msgs/NavSatFix uav_position
sensor_msgs/NavSatFix[] goal_positions
uint8 altitude
---
mavros_msgs/Waypoint[] path_waypoints
float32 dist
float32 av_risk
uint8 status
```

Figure 16. .srv file for path planning service request

The service is then called, which, in case of success, will return an array of waypoints, that is a sequence of points that, when displayed on the map, will show the low-risk path calculated by the software.

```
48      // Call the service
49      console.log('Calling service... /path_planning_wp', request);
50
51      service.callService(request, (result) => {
52          console.log('Result of /path_planning_wp is...', result);
53          const waypoints = result.path_waypoints.map(waypoint => ({
54              lat: waypoint.x_lat,
55              lng: waypoint.y_long,
56          }));
57
58          // Resolve the promise with the waypoints
59          resolve(waypoints);
60          client.close();
61
62      }, (error) => {
63          console.error('Service call failed: ', error);
64          reject(error); // Reject the promise in case of service call failure
65          client.close();
66      });
```

Figure 17. Service call to ROS server for path planning request

Both the service-form-component and the path-modal components have corresponding .css files (form-modal.css and path-modal.css, respectively) that manage the layout of the two elements on the page.

4.2 Backend

The backend of this application is organized according to a structured and modular architecture, which facilitates the management of data retrieval, insertion, and modification related to drones within a database. Each component of the backend has a specific responsibility, allowing the code to remain organized, reusable, and easily maintainable. This separation ensures that each component focuses on a precise aspect of data management, improving internal cohesion and system understanding.

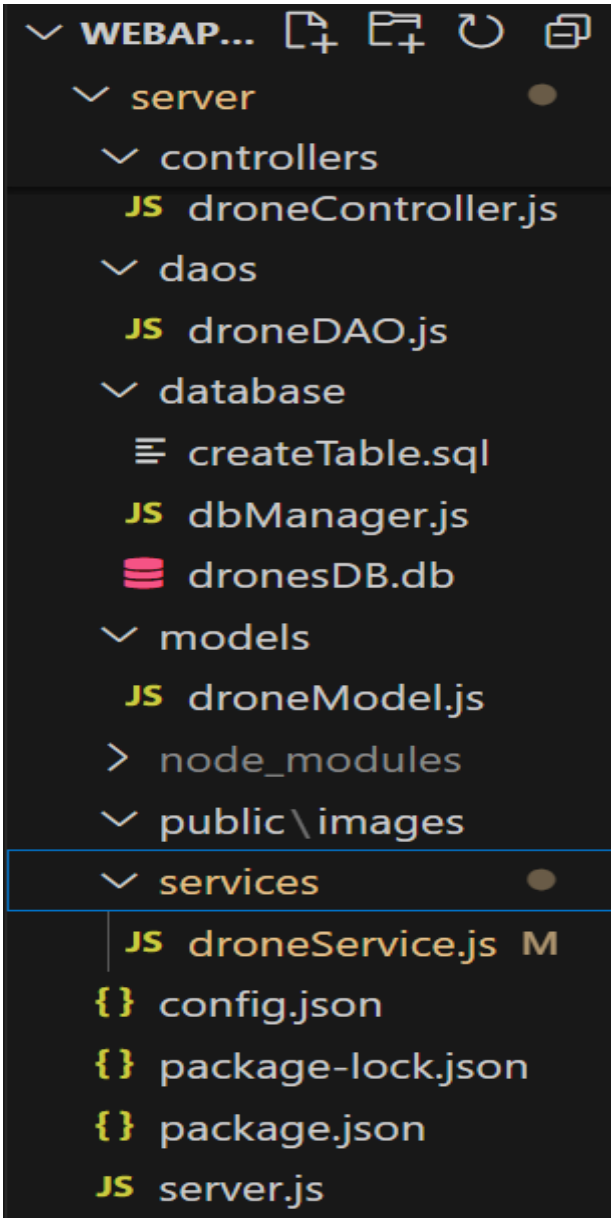

Figure 18. Backend structure

4.2.1 Drone data management

The models folder contains the definition of the data models used to represent drones within the application. Each drone is defined by a set of key attributes such as model, maximum cruise speed, maximum flight time, mass, operational radius, and maximum payload capacity. These attributes are mapped to the database tables where drone data is stored. The Drone model serves as a blueprint for saving and retrieving data from

the database, as well as ensuring that the data complies with the defined format and constraints.

The DAO (Data Access Object) layer is responsible for direct interaction with the database. In this layer, CRUD (Create, Read, Update, Delete) operations on drone data are implemented. This layer abstracts the complexity of database management and provides a clean interface for other parts of the system. The DAO encapsulates all database-related operations and provides simple methods for data insertion and retrieval.

The Service layer orchestrates the business logic of the application. It receives data from the DAO and implements the necessary logic to process it before sending it to the client or controller. This layer provides an additional level of abstraction between the DAO and the Controller, ensuring that the controller does not have to worry about business logic, but only about orchestrating requests and responses. The Service can also contain validation logic, data transformation, or interaction with other system components before passing the data to the DAO or the Controller.

The Controller is the part of the system that handles HTTP requests coming from the web app. It receives client requests, sends them to the appropriate services, and returns a response. The controller usually follows the REST pattern, where each resource in the system has a defined set of operations (e.g., GET, POST, PUT, DELETE). The Controller handles receiving the HTTP request, invoking the appropriate services, and returning a response to the client in the form of JSON. This allows the web app to interface with the backend in a simple and standardized way.

The dbmanager.js file manages the opening and closing of connections to the database. It is responsible for configuring the database and managing it correctly. Often, this file contains the database credential configuration and manages the connection pool to improve performance.

4.2.2 Server.js

The server.js file serves as the entry point for starting the application's backend. It is configured to handle HTTP requests, interact with the database, and save images to the server.

Here, the Express framework is initialized, which will be responsible for managing HTTP requests. The server is set to listen on port 3001, and the HTTP header x-powered-by is disabled to enhance security by preventing the exposure of the use of Express. Middleware functions are then handled, which are executed before managing requests. Specifically:

- Morgan is used for logging HTTP requests (in development mode).
- express.json() and express.urlencoded() allow the server to parse the body of requests, including JSON and URL-encoded data, with a size limit of 50MB.
- express.static() is used to expose static files (such as images), making the images saved in a specific public directory accessible.

CORS (Cross-Origin Resource Sharing) is then configured, allowing the server to handle requests from a different domain (in this case, the frontend on localhost:3000). Options include enabling credentials and managing the response in case of success. The DBManager is instantiated afterward.

APIs related to drone management are handled by the droneController, which is responsible for CRUD operations on drone data. Requests sent to the /api path are routed to the controller for handling.

An API is also defined that allows images to be saved to the server. It receives the name and image data (in base64 format) from the front-end and saves it in the public/images folder. If an error occurs during saving, a 500 error is returned; if successful, a confirmation message is returned.

Finally, the server is started and configured to listen for requests on port 3001. A log message is displayed to indicate that the server is active and listening.

In conclusion, the backend of the drone management system is constructed following a well-defined structure that clearly divides responsibilities among model management, data access, business logic, and HTTP interface. This division helps keep the code organized and easily scalable while efficiently managing drone-related operations.

CHAPTER 5

5. WebApp overview

In this chapter, the functionalities of the application will be demonstrated through the use of screenshots and appropriate comments. An example operation will also be shown, along with the overall appearance of the app, so that one can visualize what the previously discussed code renders on the screen.

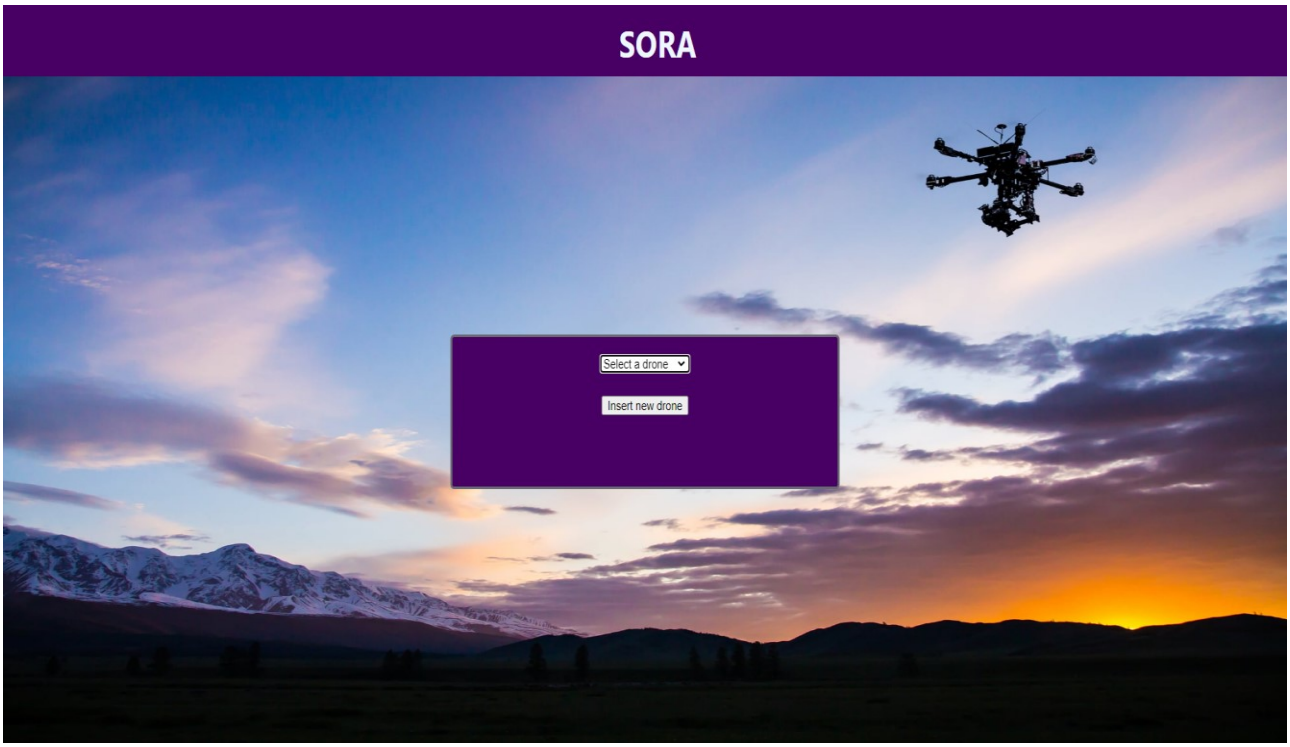

Figure 19. Home page

The Home page displays a currently very simple screen, featuring a Navbar at the top with the app's logo. The Navbar remains fixed on the screen even in subsequent pages, and clicking the logo (the name SORA is just an example) returns the user to this screen, allowing for quick access to the main page at all times.

In the central selection panel, there are two elements. The first is a drone selection menu. Clicking it will open a dropdown that shows a list of drones currently saved in the database.

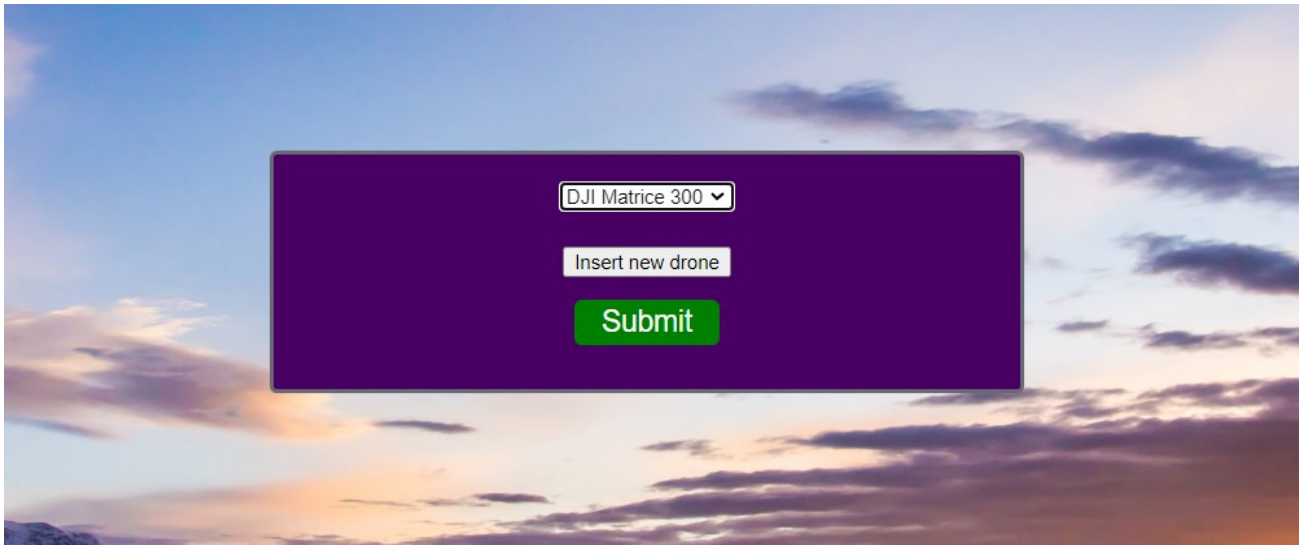

Figure 20. Drone selection in home page

Once a drone is selected, the 'Submit' button will appear, confirming the choice and navigating to the next page after retrieving the drone's data from the database.

A second option is to manually enter the details of a drone. By clicking on 'Insert new drone,' a form will open with various fields representing the technical parameters of the drone.

Figure 21. Form for new drone

Once the fields are filled out, pressing the ‘Submit’ button will navigate to the next page. The newly entered drone will be saved in the database, and its data will be used for subsequent calculations. Clicking ‘Back to selection’ will return the user to the previous panel.

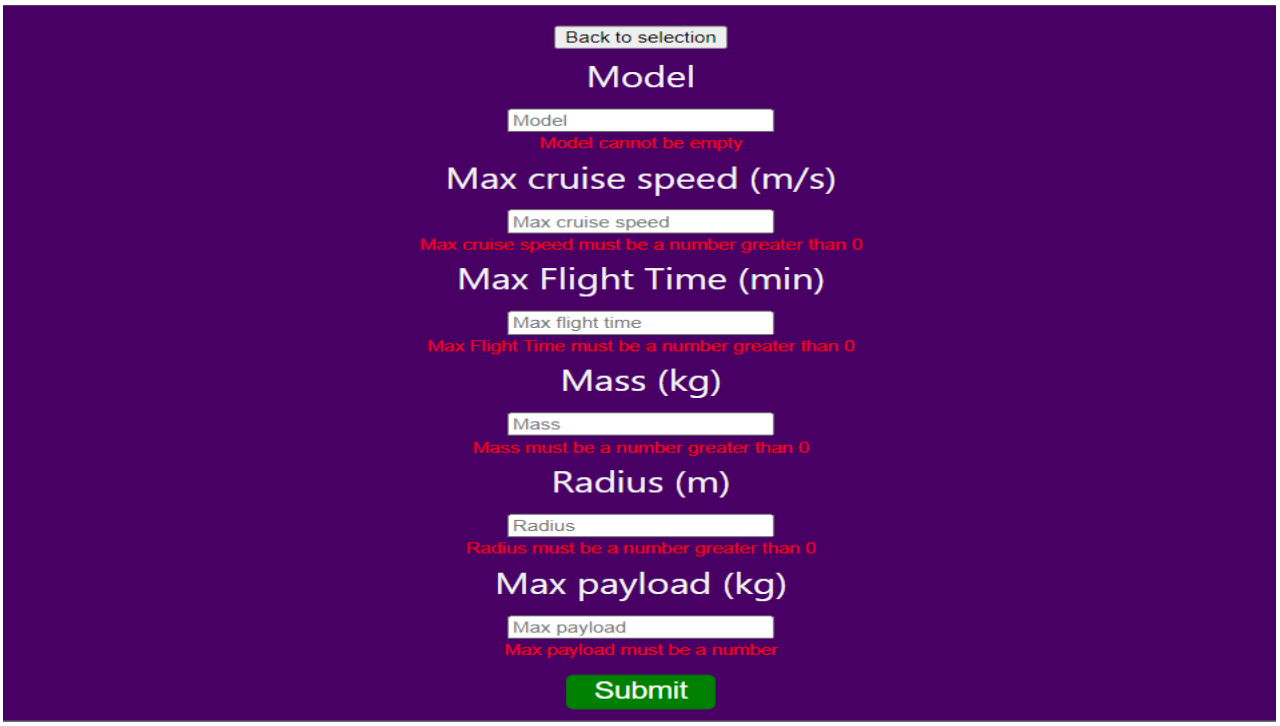

Figure 22. Form for new drone with error messages

A validation function checks that all fields are filled out correctly; otherwise, an error message is displayed under each field, as shown in the figure above.

5.1 Risk map computing

The next page represents the true core of the application. It contains the map and the components that allow users to manage the data to be sent for flight planning.

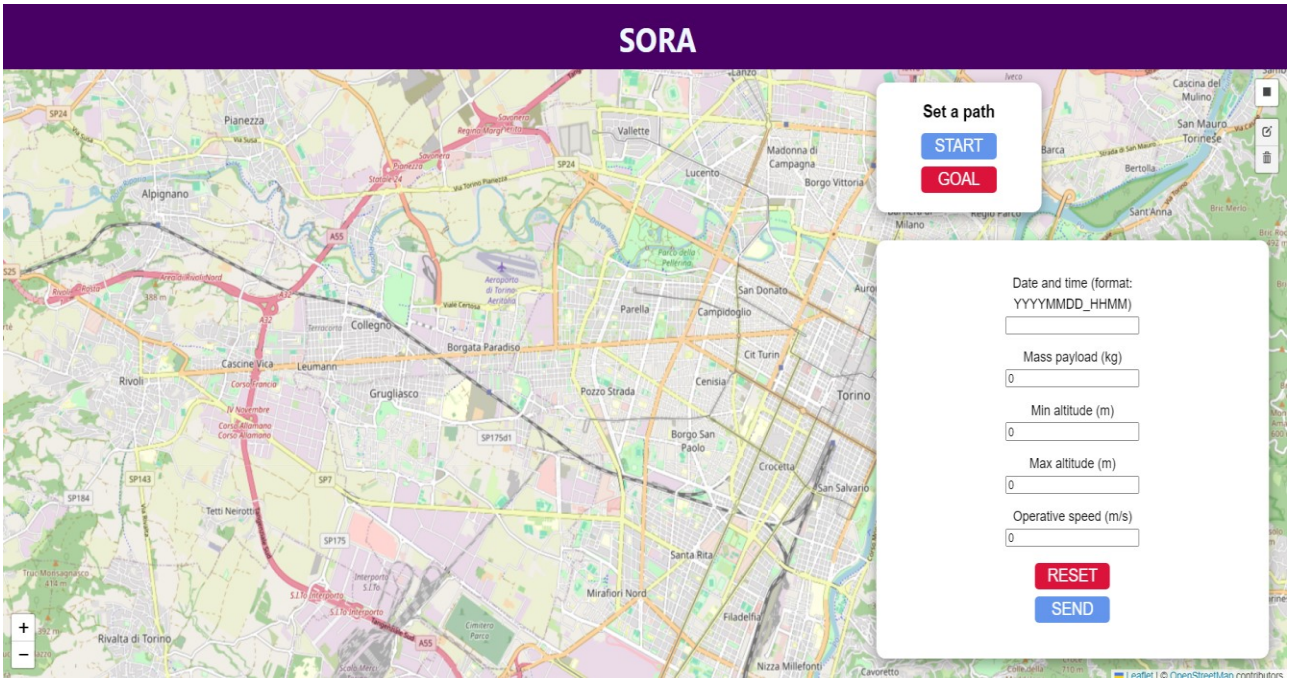

Figure 23. Main page

The page, as shown in the figure, displays the map (from OpenStreetMap) on which the user can choose the area where they want to perform the operation. In the top right corner, we find the drawing tools provided by Leaflet, which allow the user to define a rectangular area that can later be modified or deleted. Selecting the area is the first step the operator must take to proceed with the operation.

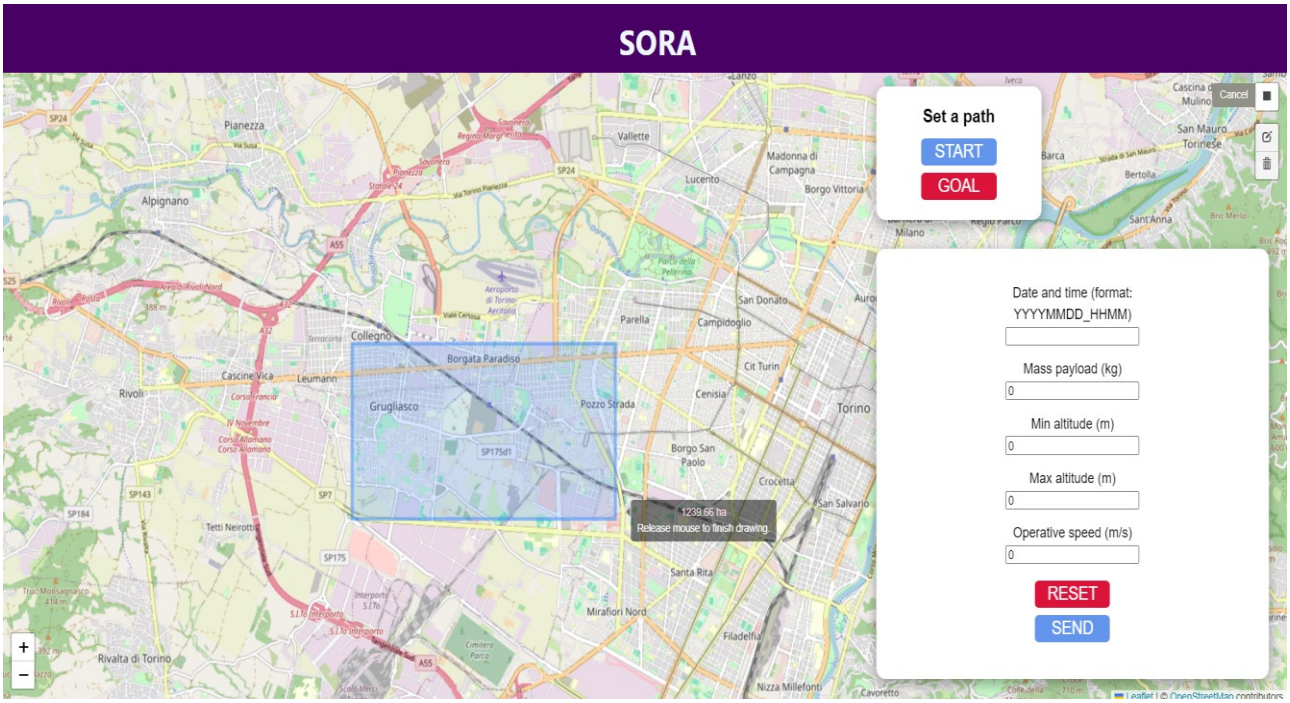

Figure 24. Area selection with drawing tools

Next, there is a form that the operator must fill out with the flight mission data. Again, a validation function checks that the entered data is correct; otherwise, an error message will be displayed, specifying which fields are incorrect.

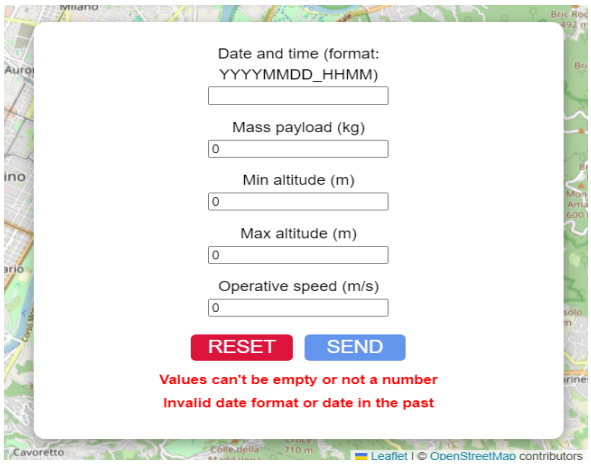

Figure 25. Values error messages

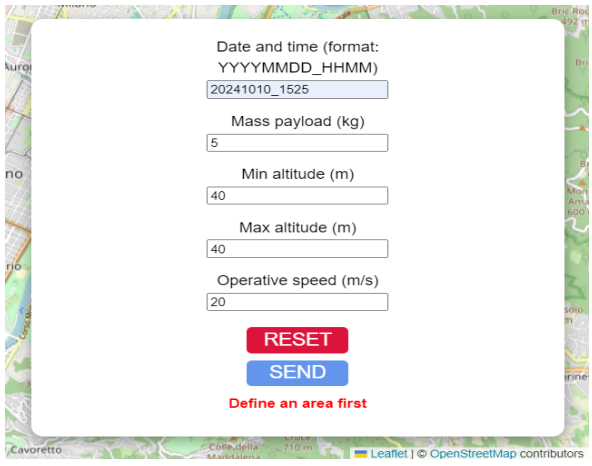

Figure 26. Error message when area is not found

Regarding the date, it must be entered in the format specified in the form and cannot be earlier than the current time. If an attempt is made to submit the data without first defining an area, an error message will appear to inform the user of this. A 'RESET' button allows all fields in the form to be cleared.

If the operator has successfully completed these two steps—defining the area and filling out the form—they can submit the data by pressing the 'SEND' button and wait for the ROS server to calculate the risk map. This will then be displayed on the map above the selected area.

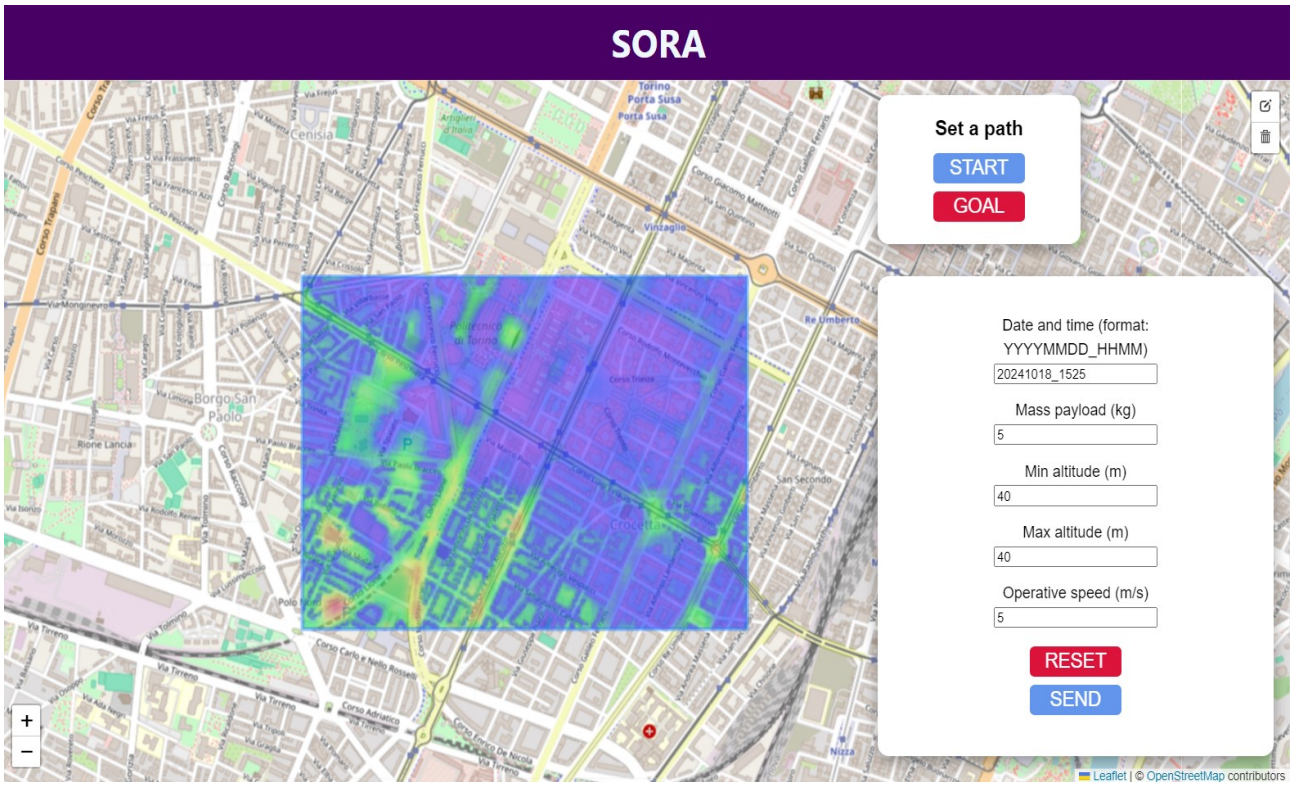

Figure 27. Risk map result

The image above shows the result of an example operation that the team performed. We can see how the returned image, made semi-transparent to allow the operator to see the map underneath, is consistent with the one processed by the ROS server, shown below. The color represents the level of risk associated with that area: red indicates a high risk, yellow/green a medium risk, and blue/purple indicates a low risk.

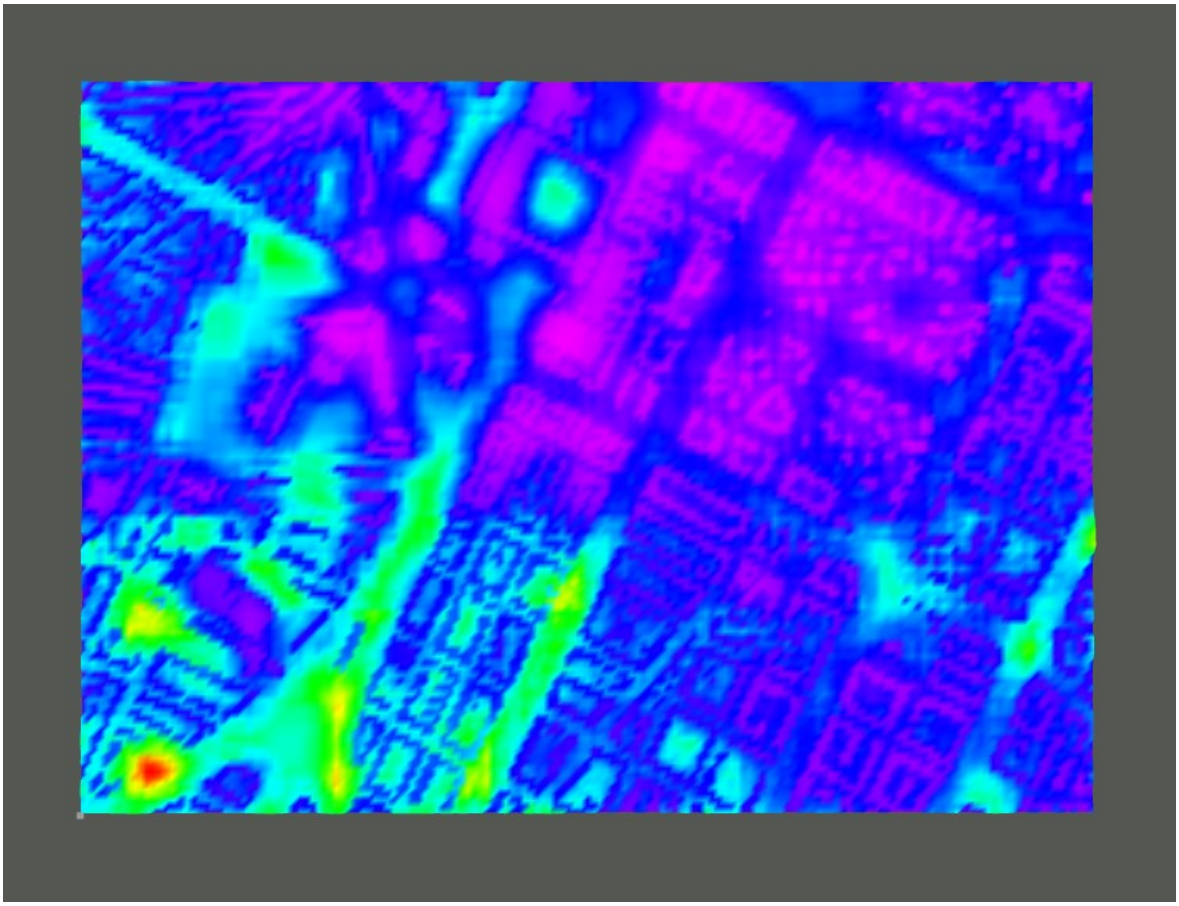

Figure 28. Risk map result ROS server side

5.2 Path computing

Once the risk map is obtained, the next step is to calculate the optimal path associated with the area in question. The "Set a path" component allows the operator to select the starting and ending points of the mission. Pressing one of the two buttons enables the selection of the corresponding point; pressing it again cancels the operation. When enabled, the pressed button will be green. If a point has already been selected, pressing the corresponding button will delete it, allowing the user to select another one.

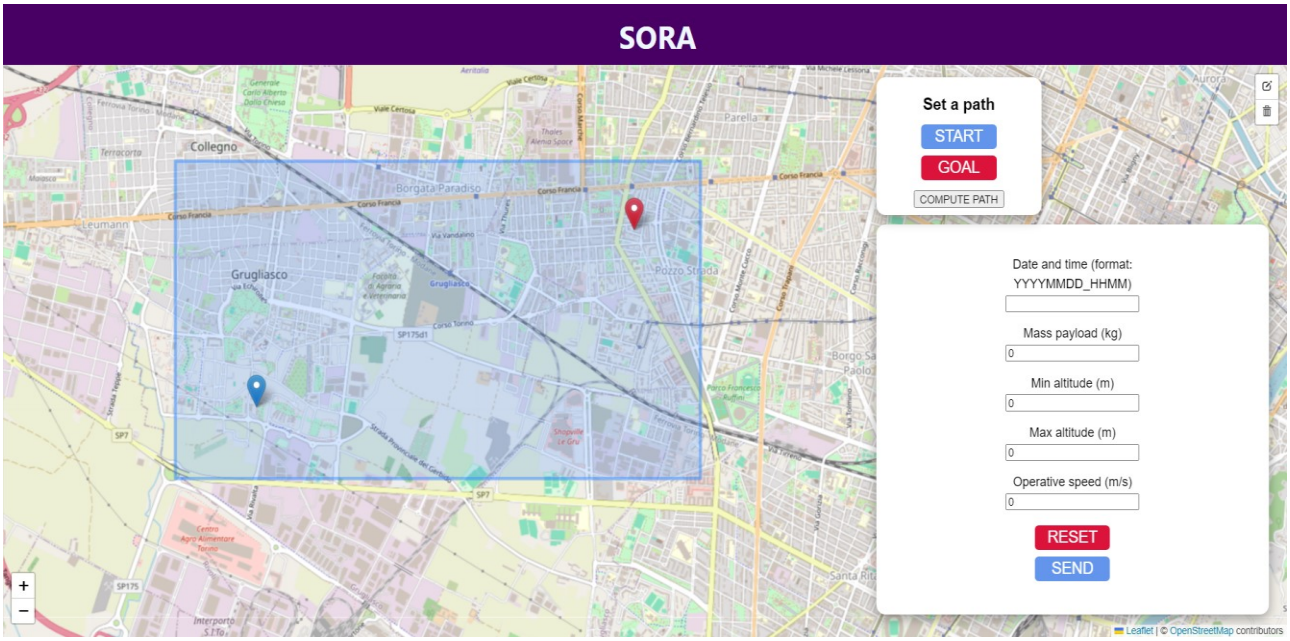

Figure 29. Example of an area with start and goal position selected

Once the starting and goal points are defined, the "COMPUTE PATH" button will appear. Pressing it will send a request to the server to calculate the path. If you attempt to do this without first calculating a risk map, an error message will be displayed reminding you to do so.

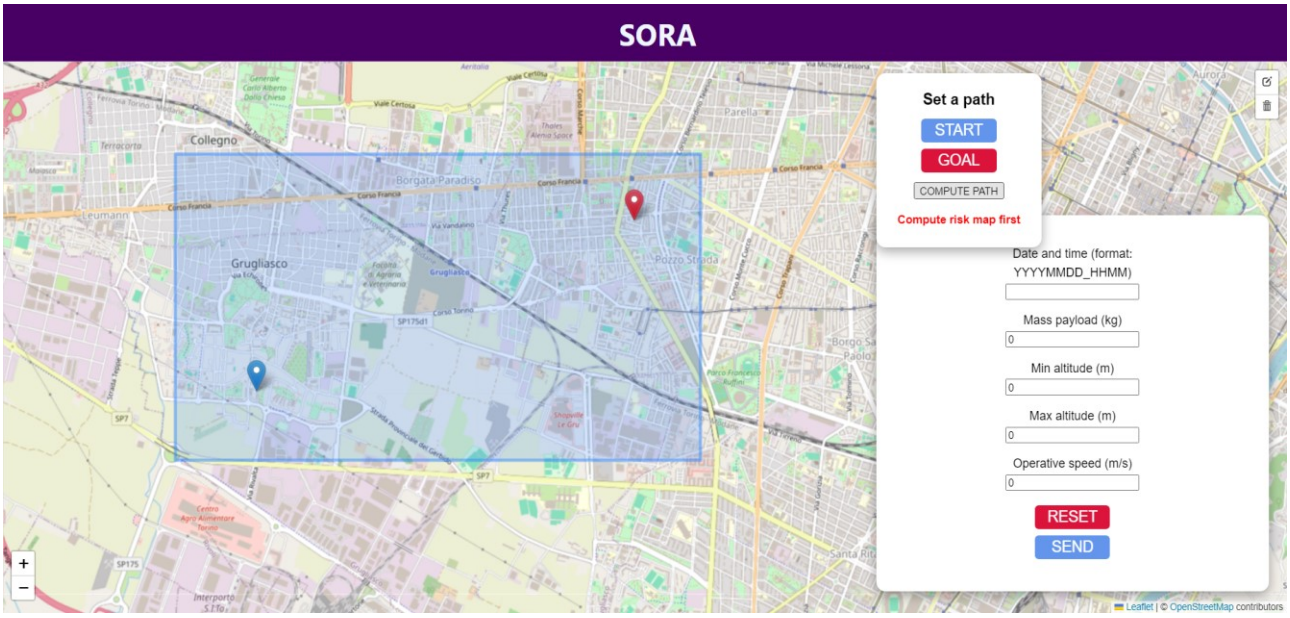

Figure 30. Path planning error message

The team, as an example, calculated a path based on the previously calculated risk map. The result is shown in the image below, which can be considered representative of what the final result of any planning may look like.

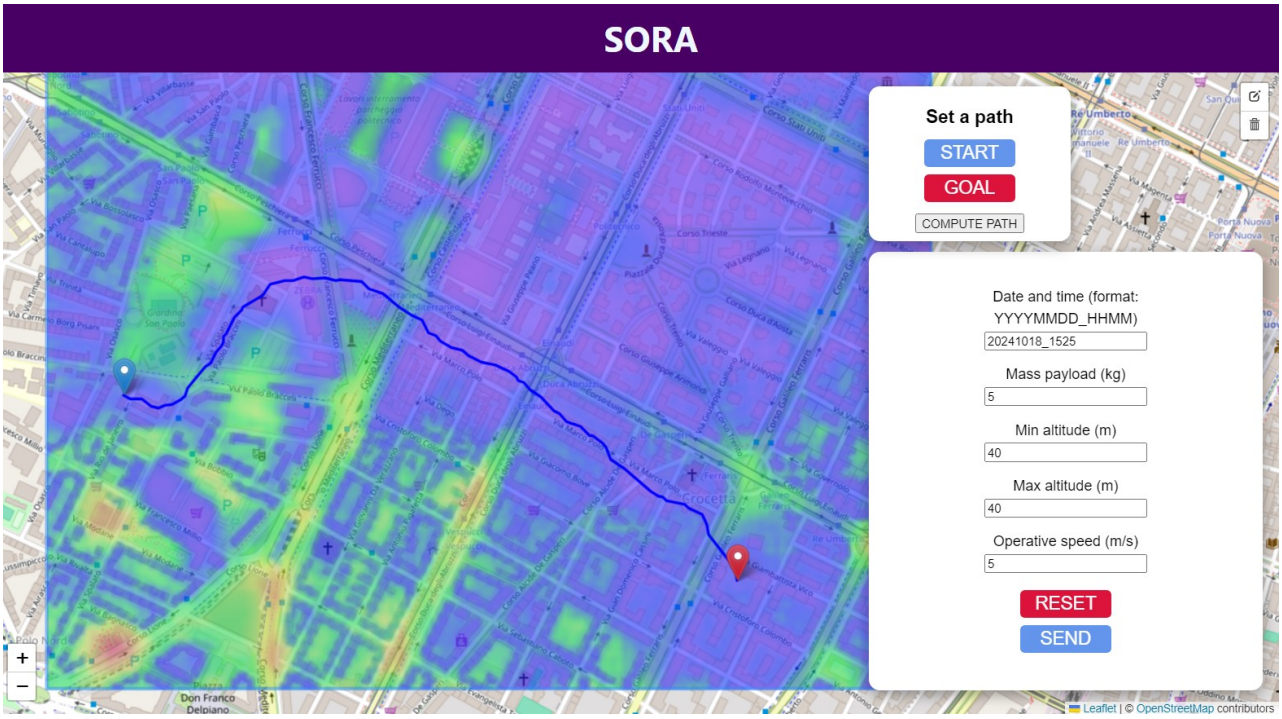

Figure 31. Path planning result

Once the path is obtained from the server, it is displayed on the risk map between the two selected positions. The image, once again, is consistent with that processed on the ROS side.

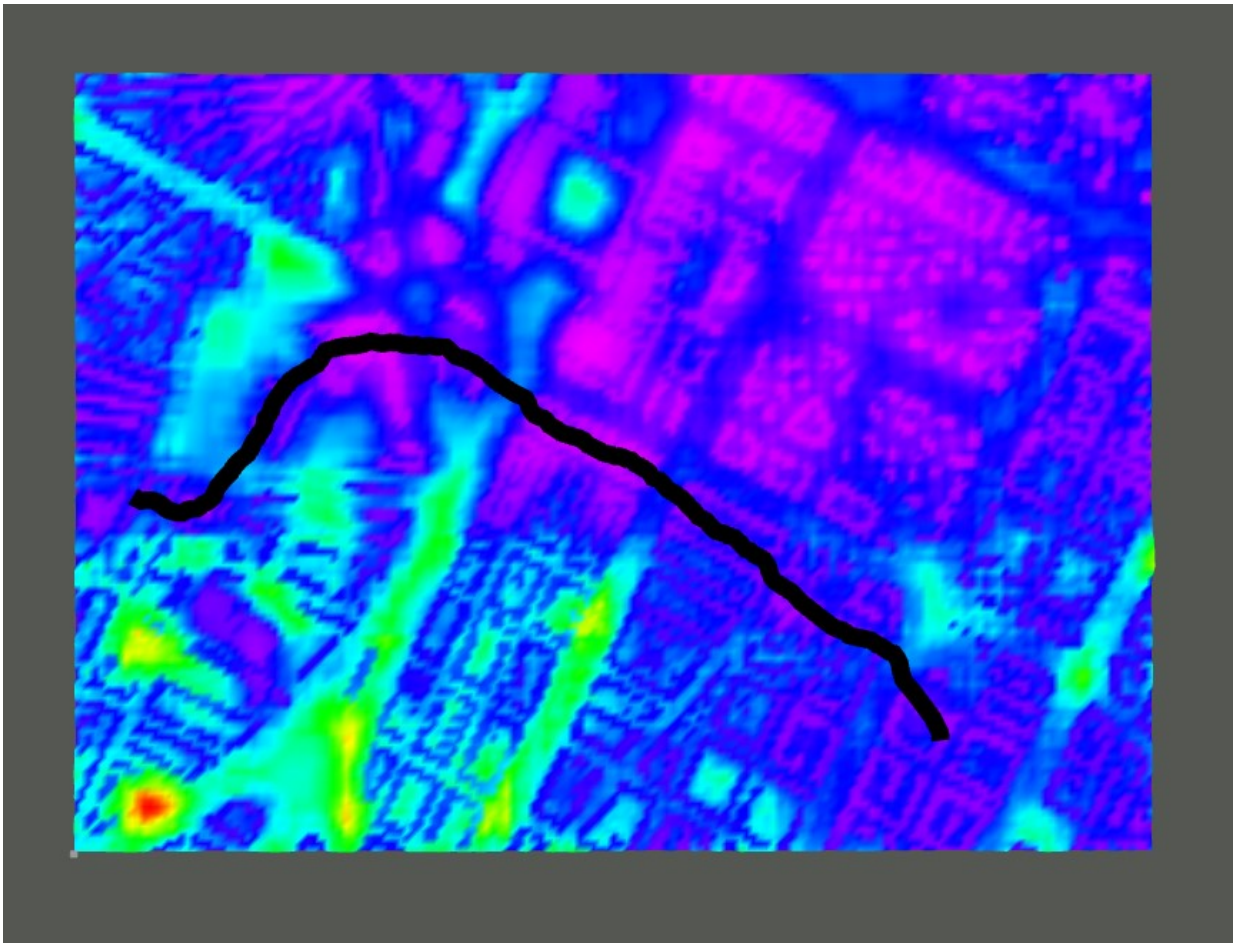

Figure 32. Path planning result ROS server side

Looking at these images, one can see how the calculation software attempts to generate a path that is as safe as possible, trying to avoid the highest-risk areas even if it means lengthening the route.

Below, as an example, are the results of other simulations. As before, each map will be followed by the corresponding result from the ROS server. Observing Figures 35 and 37, it is evident that even a slight shift in the goal point can lead to a significantly different low-risk path.

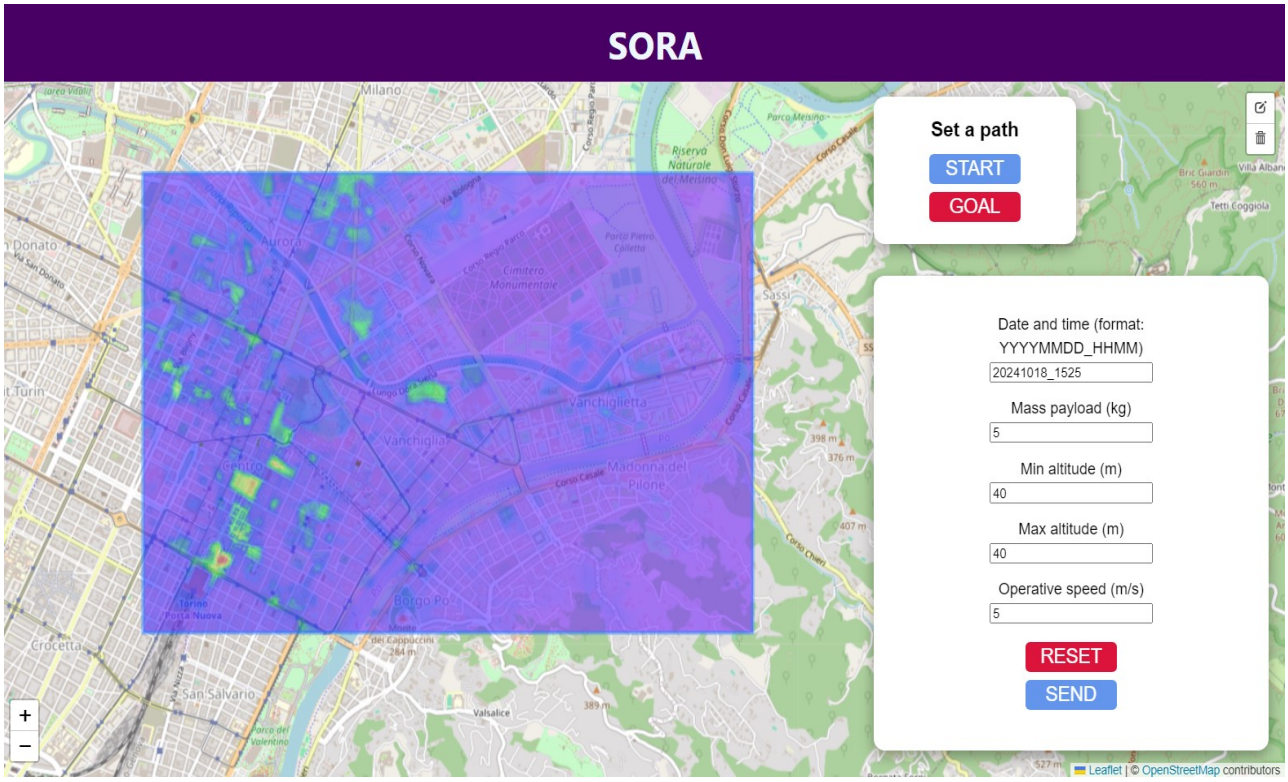

Figure 33. Risk map simulation result

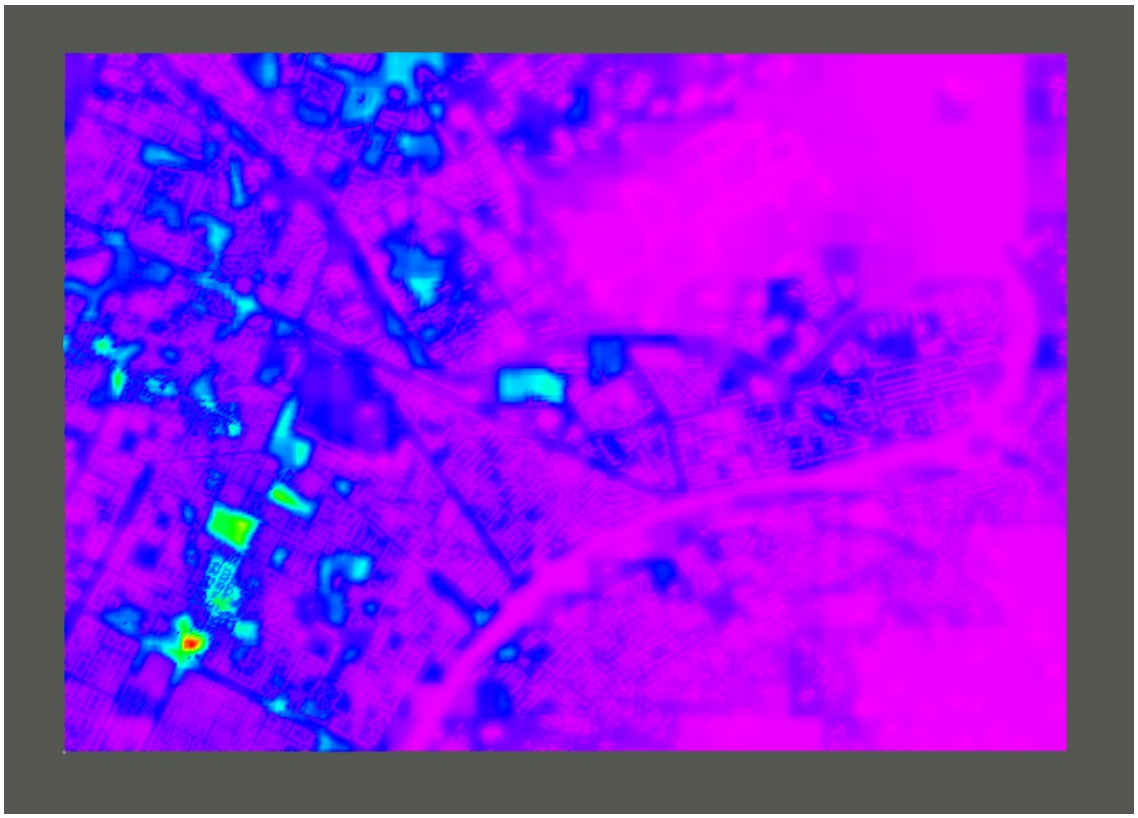

Figure 34. Risk map simulation result ROS server side

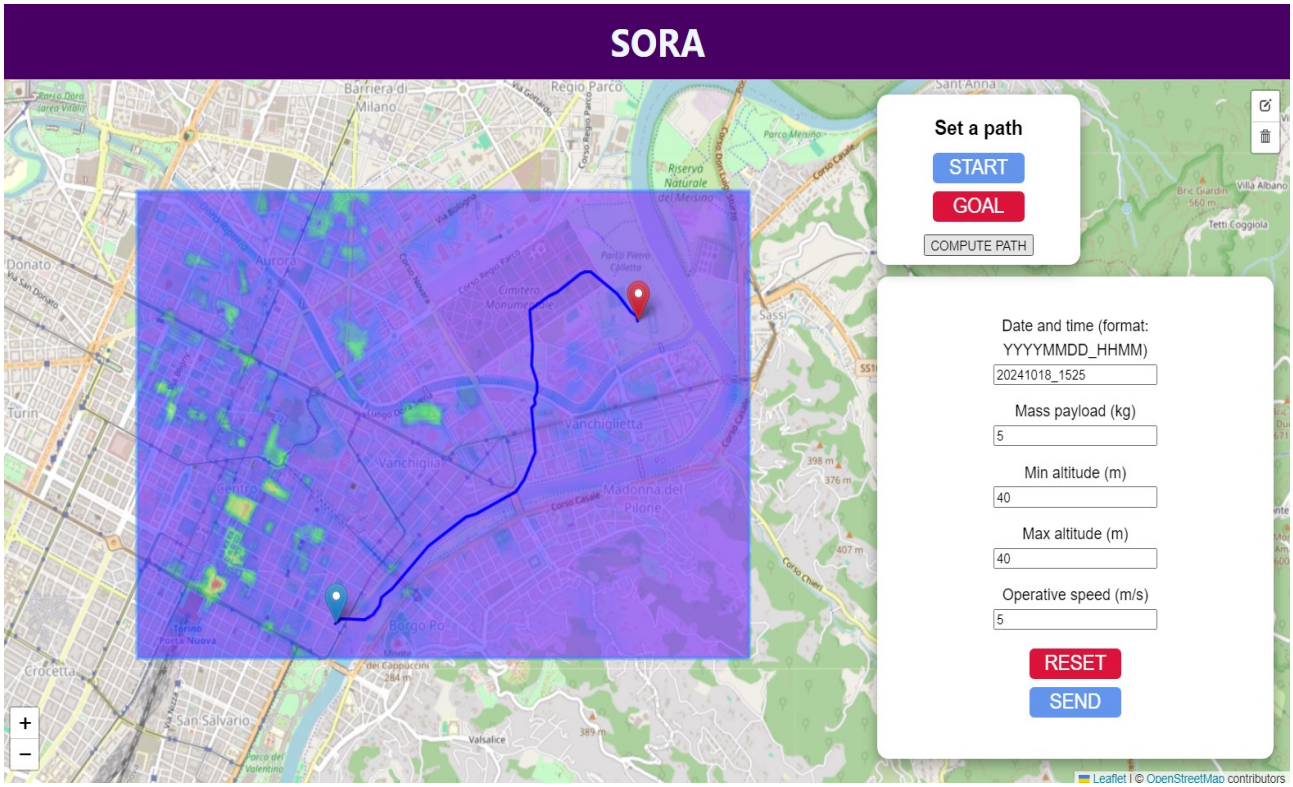

Figure 35. Path planning simulation result

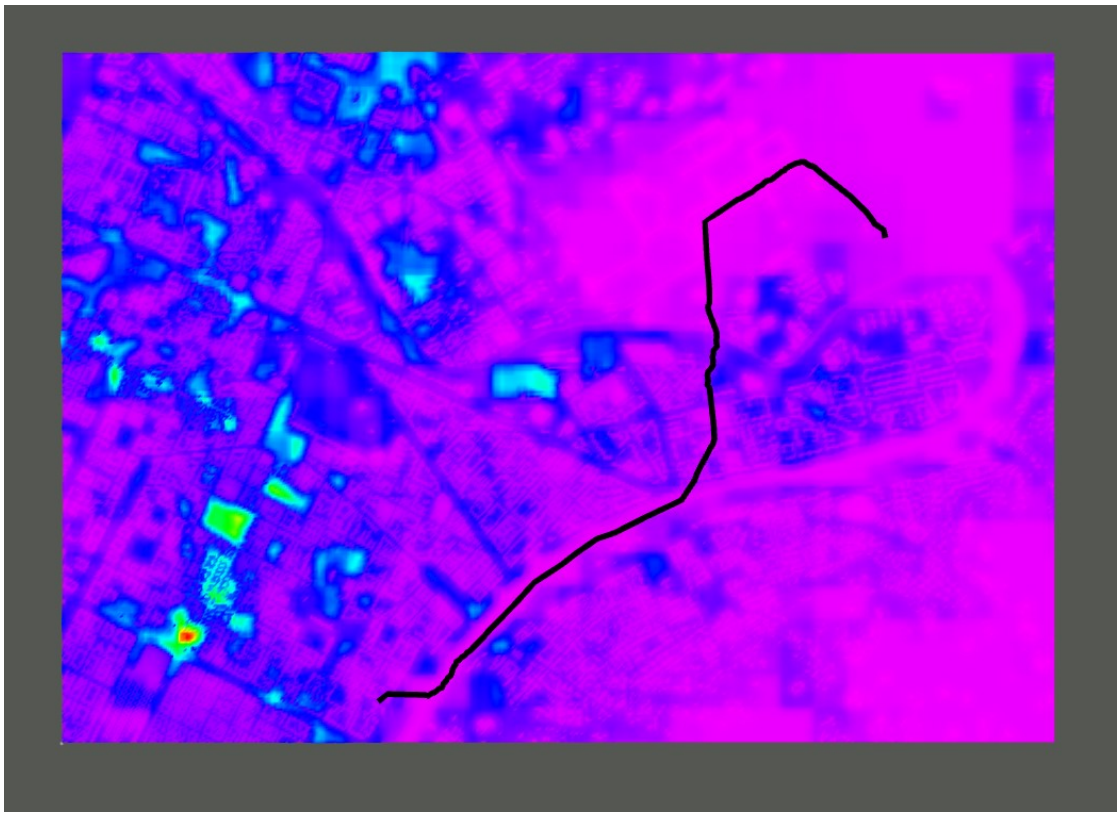

Figure 36. Path planning simulation result ROS server side

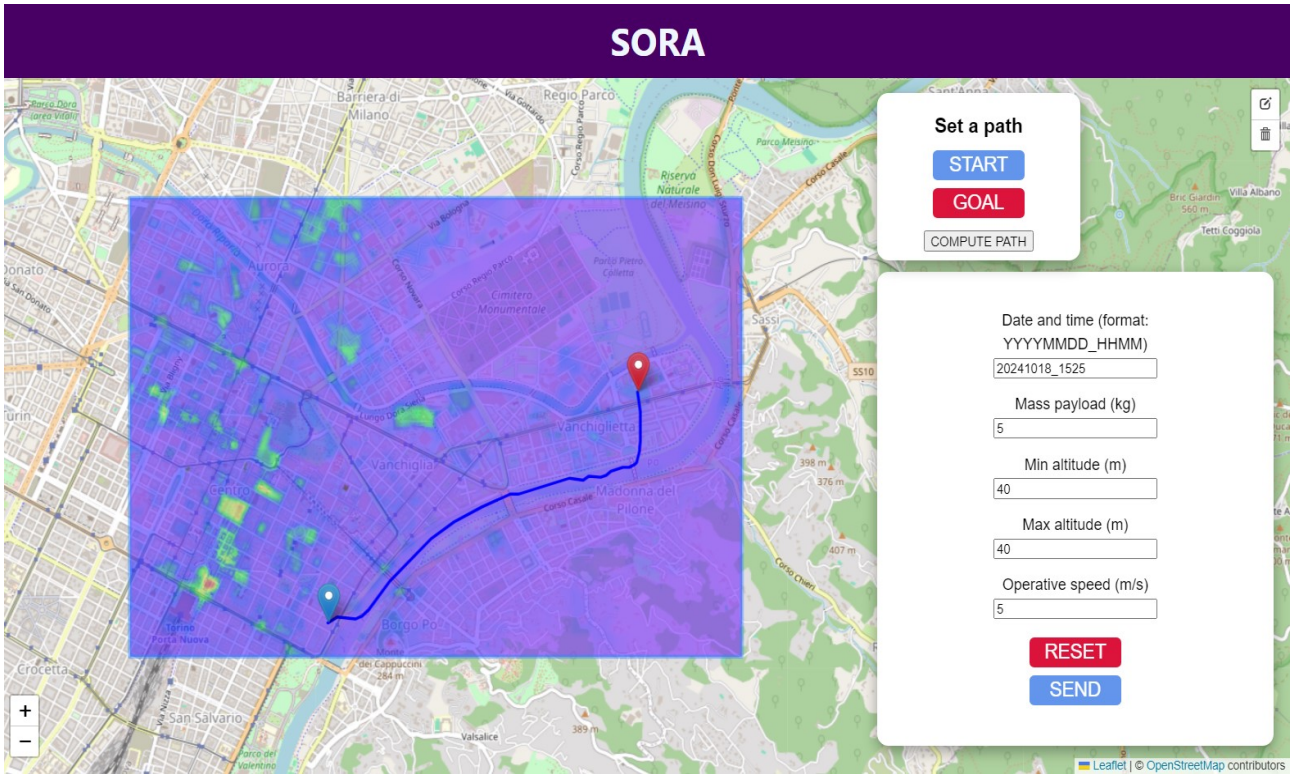

Figure 37. Another example of path planning simulation

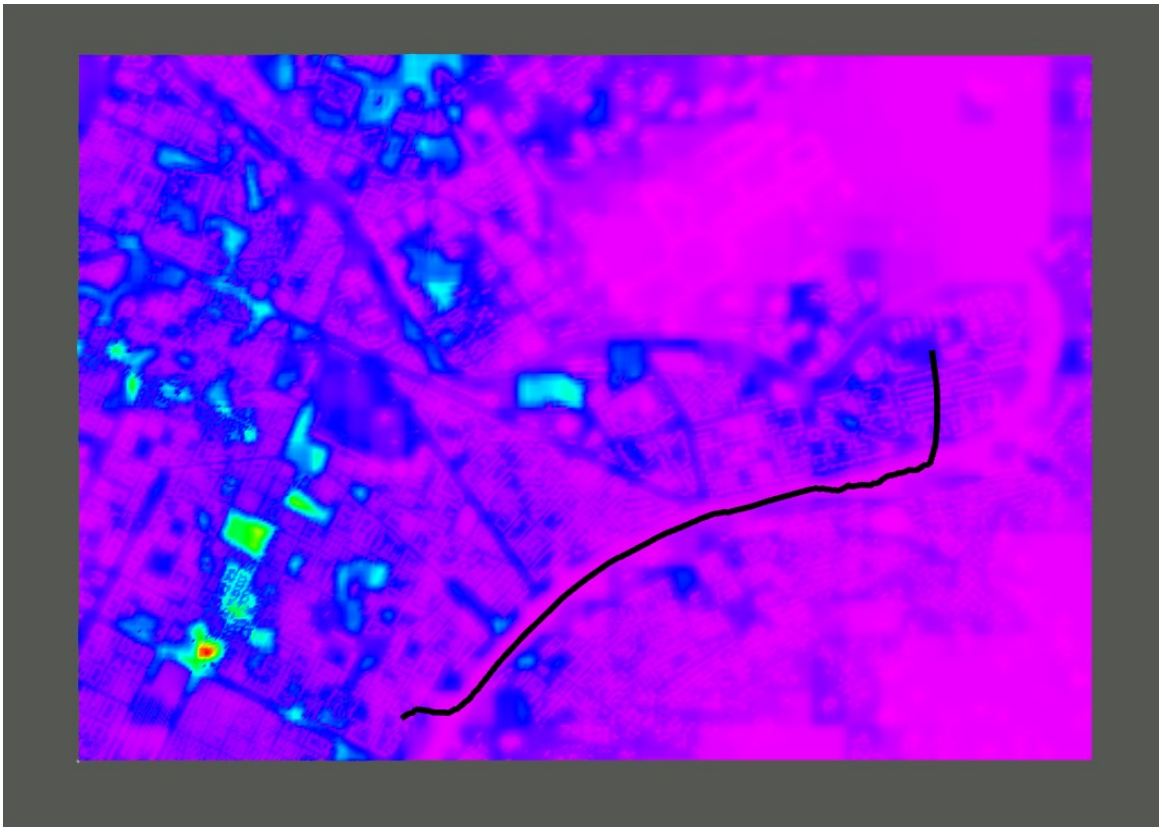

Figure 38. Result ROS server side of fig.37 simulation

CHAPTER 6

6. Conclusions and future works

The project developed for planning drone flight operations in populated areas represents a significant contribution within a continuously evolving technological context. The integration of a web application with advanced planning and risk management tools, compatible with the Specific Operations Risk Assessment (SORA) required by EASA, has demonstrated the potential to provide operators with an effective tool to ensure safety and regulatory compliance.

The user interface, based on React and libraries like Leaflet and React-Leaflet, allows for interactive maps to visualize mission area selection and risk map calculation. The interface with ROS (Robot Operating System) enables the web app to connect to a backend system that manages risk map calculations and sends commands to the drone. The backend handles requests asynchronously, allowing for smooth communication with the ROS server for processing maps and supporting images. Systems have been implemented for managing and saving data, such as images generated from risk maps, ensuring traceability of operations and the ability to analyse data in post-processing.

Future works may include the introduction of a user authentication and registration system, allowing operators to create personal accounts. This feature would enable the saving of previous flight missions and their associated risk maps in the user's area, facilitating the retrieval and analysis of past operations. Additionally, users could customize their accounts to receive suggestions or notifications about regulatory updates. Another useful enhancement could be the ability to export planning results, such as risk maps or flight trajectories, in JPG or PDF formats. This would be advantageous for creating documentation to share with other team members or regulatory bodies to demonstrate compliance with safety regulations, providing a structured archiving of mission data.

A crucial aspect for future development will also be user feedback. As the platform is used by an increasing number of operators, their contributions will be essential in identifying areas for improvement. Features such as post-mission surveys or the ability to report issues directly from the application could accelerate the implementation of new functionalities and enhancements.

The project has achieved remarkable results, creating a functional platform that could serve as a foundation for the future development of advanced planning tools for drone operations. Continuing down this path, the system could become an indispensable tool for safe and compliant operations in the drone industry.

7. References

[1] Primatesta, Stefano, Alessandro Rizzo, and Anders la Cour-Harbo. "Ground risk map for unmanned aircraft in urban environments." *Journal of Intelligent & Robotic Systems* 97 (2020): 489-509.

[2] Milano, Matteo, Stefano Primatesta, and Giorgio Guglieri. "Air risk maps for unmanned aircraft in urban environments." *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2022.

[3] Primatesta, Stefano, et al. "An innovative algorithm to estimate risk optimum path for unmanned aerial vehicles in urban environments." *Transportation research procedia* 35 (2018): 44-53.

[4] https://dronefacile.it/regolamento-droni/

[5] https://eudroneport.com/it/blog-it/metodologia-sora/

[6] https://droneitalia.online/valutazione-del-rischio-sora/

[7] https://it.wikipedia.org/wiki/Specific_Operations_Risk_Assessment

[8] https://towardsdatascience.com/what-why-and-how-of-ros-b2f5ea8be0f3

[9] https://www.smartcitiesworld.net/news/dhl-launches-its-first-regular-urban-drone-delivery-service-4189

[10] https://www.enac.gov.it/

[11] https://medium.com/level-up-web/amazingly-useful-html-css-and-javascript-tools-and-libraries-d73b10fbae29

[12] https://www.designveloper.com/blog/what-is-react/

[13] https://react-component-depot.netlify.app/

[14] https://leafletjs.com/

[15] https://react-leaflet.js.org/