Miguel Pino

Bachelor of Computer Science and Engineering

UAV Cloud Platform for Precision Farming

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser:	Luís Miguel Campos, R&D Director, PDMFC
Co-adviser:	João Costa Seco, Associate Professor, Universidade Nova de Lisboa

Examination Committee

Chairperson:	Carlos Augusto Isaac Piló Viegas Damásio
Raporteur:	Paulo Orlando Reis Afonso Lopes
Members:	Luís Miguel Campos
	João Costa Seco

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2021

UAV Cloud Platform for Precision Farming

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

Firstly, I would like to thank my advisers for providing me with the opportunity to research and develop in the growing field of Unmanned Aerial Vehicles applications. Without them, I couldn't have the possibility to collaborate in a European project and use my skills to make advancements in this technology.

Next, I want to thank my friends and colleagues that worked directly with me on this project and that without their help, I am sure the outcome would be very different. I want to show my gratitude to Guilherme Rolo, João Falé Madeira, and Ricardo Sacoto Martins, for all the work they developed and for the constructive criticism they gave regarding this dissertation. Furthermore, I would like to thank Dário Pedro and João Carvalho for their guidance during the process, and for the opportunity to improve my thesis with the publication of an article regarding this technology.

Last but not least, I need to thank my close family and friends for their advice and patience while I worked and developed my thesis. I want to directly thank Adriana Fonseca, Inês Serra, and Marta Gomes for always reminding me that I am capable and good enough to be doing this work. Moreover, I want to show my appreciation for the time you took to make sure everything was going as supposed, and every time you took the time to make sure I was all right. Finally, I want to thank my parents. For not only making sure I always knew which course to take, but also for never letting me feel I wouldn't be supported in every way, shape, or form regardless of the path I choose.

Abstract

A new application for Unmanned Aerial Vehicles comes to light daily to solve some of modern society's problems. One of the mentioned predicaments is the possibility for optimization in agricultural processes. Due to this, a new area arose in the last years of the twentieth century, and it is in constant progression called Precision Farming. Nowadays, a division of this field growth is relative to Unmanned Aerial Vehicles applications.

Most traditional methods employed by farmers are ineffective and do not aid in the progression and solution of these issues. However, there are some fields that have the possibility to enhance many agriculture methods, such fields are Cyber-Physical Systems and Cloud Computing. Given its capabilities like aerial surveillance and mapping, Cyber-Physical Systems like Unmanned Aerial Vehicles are being used to monitor vast crops, to gather insightful data that would take a lot more time if being collected by hand. However, these systems typically lack computing power and storage capacity, meaning that much of its gathered data cannot be stored and further analyzed locally. That is the obstacle that Cloud Computing can solve. With the possibility to offload computing power by sending the collected data to a cloud, it is possible to leverage the enormous computing power and storage capabilities of remote data-centers to gather and analyze these datasets.

This dissertation proposes an architecture for this use case by leveraging the advantages of Cloud Computing to aid the obstacles of Unmanned Aerial Vehicles. Moreover, this dissertation is a collaboration with an on-going Horizon 2020 European project that deals with precision farming and agriculture enhanced by Cyber-Physical Systems.

Keywords: Cyber-Physical Systems, Unmanned aerial vehicles, Cloud Computing, Precision Farming

Resumo

A cada dia que passa, novas aplicações para Veículos aéreos não tripulados são inventadas, de forma a resolver alguns dos problemas actuais da sociedade. Um desses problemas, é a possibilidade de otimização em processos agrículas. Devido a isto, nos últimos anos do século 20 nasceu uma nova área de investigação intitulada Agricultura de alta precisão. Hoje em dia, uma secção desta área diz respeito à inovação nas aplicações com recurso a Veículos aéreos não tripulados.

A maioria dos métodos tradicionais usados por agricultores são ineficientes e não auxiliam nem a evolução nem a resolução destes problemas. Contudo, existem algumas áreas científicas que permitem a evoluçao de algumos métodos agrículas, estas áreas são os Sistemas Ciber-Físicos e a Computação na Nuvem. Dadas as suas capacidades tais como a vigilância e mapeamento aéreo, certos Sistemas Ciber-Físicos como os Veículos aéreos não tripulados estão a ser usados para monitorizar vastas culturas de forma a recolher dados que levariam muito mais tempo caso fossem recolhidos manualmente. No entanto, estes sistemas geralmente não detêm grandes capacidades de computação e armazenamento, o que significa que muitos dos dados recolhidos não podem ser armazenados e analisados localmente. É aí que a Computação na Nuvem é útil, com a possibilidade de enviar estes dados para uma nuvem, é possível aproveitar o enorme poder de computação e os recursos de armazenamento dos datacenters remotos para armazenar e analisar estes conjuntos de dados.

Esta dissertação propõe uma arquitetura para este caso de uso ao fazer uso das vantagens da Computação na Nuvem de forma a combater os obstáculos dos Veículos aéreos não tripulados. Além disso, esta dissertação é também uma colaboração com um projecto Europeu Horizonte 2020 na área da Agricultura de alta precisão com recurso a Veículos aéreos não tripulados.

Palavras-chave: Sistemas Ciber-Físicos, Veículo aéreo não tripulado, Computação na Nuvem, Agricultura de alta precisão

Contents

List of Figures

List of Tables

Glossary

Back end	Commonly referred as the data access layer of a piece of software in web development. It handles the connections of the application to a database or middleware.
Bitrate	Corresponds to the number of bits that are conveyed or processed per unit of time in telecommunications and computing.
CIR	Color-infrared (CIR) aerial photography renders the scene in colors not normally seen by the human eye, is widely used for interpretation of natural resources.
Cloud Robotics	Field of robotics that attempts to invoke cloud technologies such as cloud computing, cloud storage, and other Internet technologies centered on the benefits of converged infrastructure and shared services for robotics.
Front end	Commonly referred as the presentation layer of a piece of software in web development. Can be comprised of many user interfaces and can be implemented in multiple frameworks.
Gazebo	Is an open-source 3D robotics simulator. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments.
Heatmap	Graphical representation of data where the individual values contained in a matrix are represented as colors.
IMU	An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers.

Middleware	Is computer software that provides services to software applications beyond those available from the operating system.
NDRE	The Normalized Difference Red Edge Index (NDRE) is a metric that can be used to analyse whether images obtained from multi-spectral image sensors contain healthy vegetation or not.
NDWI	The Normalized Difference Water Index (NDWI) is used to monitor changes related to water content in water bodies. As water bodies strongly absorb light in visible to infrared electromagnetic spectrum, NDWI uses green and near infrared bands to highlight water bodies.
RGB	The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors.
TCP	Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite, that allows two computers to communicate.
Three-tier architecture	Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.
YAML	Is commonly used for configuration files and in applications where data is being stored or transmitted.

Acronyms

API	Application Programming Interface.
CPS	Cyber-Physical Systems.
CSV	Comma-separated Values.
DSS	Decision Support System.
GPS	Global Positioning System.
GUI	Graphical User Interface.
HEIFU	Hexa Exterior Intelligent Flying Unit.
HTTP	Hyper Text Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
IoT	Internet-of-Things.
JSON	JavaScript Object Notation.
NDVI	Normalized Difference Vegetation Index.
ODM	OpenDroneMaps.
OOP	Object Oriented Programming.
REST	Representational State Transfer.
ROA	Resource-Oriented Architecture.
ROS	Robot Operating System.
RTT	Round Trip Time.
SOA	Service-Oriented Architecture.

SOAP	Simple Object Access Protocol.
TSDB	Time-series Database.
UAS	Unmanned Aircraft Systems.
UAV	Unmanned Aerial Vehicle.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
VTOL	Vertical Take-off and Landing.
WSN	Wireless Sensor Network.
WSS	WebSocket Secure.
XML	Extensible Markup Language.
YOLO	You Only Look Once.

Chapter 1

Introduction

1.1 Motivation and Context

The usage of Unmanned Aerial Vehicle (UAV) has become increasingly widespread over the years, and the potential applications that are already in development are endless [1]. We can find by researching this field various applications that go from habitat mapping [2] to monitor above-ground biomass (AGB) [3], all the way to identify erosion and deposition in an agricultural landscape [4]. All these applications are possible due to the high-mobility nature and ease of development characteristic of the UAV [5]. Not only has the UAV gained popularity over the overall consumer, but it also has entered many industries, being a high demand solution for certain professional situations. However, there are still some obstacles we need to overcome regarding the capabilities of these UAVs.

In the first place, the low computational power of the on-board computers present in UAVs limits the computation of complex algorithms. A solution to this problem is to offload the computational power by connecting a UAV to the internet. By having a cloud infrastructure capable of communication with the UAV, we can send relevant data from the UAV sensors to the cloud application and run the algorithms there. The second obstacle is the low storage capacity of the UAV due to its small size. One could employ the same cloud solution for this problem since it is also possible to offload storage capacity. If we integrate a UAV in a cloud platform designed for this purpose, we could meet these two requirements [6].

It was due to these obstacles that the possibility to integrate UAV with the field of precision farming appeared, and that led to implementation of the AFarCloud project. The lack of productivity in agricultural processes led farmers to adopt a new kind of paradigm called precision farming, where high-tech solutions can automate many traditional work procedures. These solutions can be the use of a Wireless Sensor Network (WSN) with

the capability to get real-time values of sensors positioned across crop fields or even the implementation of fully automatic devices that can perform jobs such as, seeding, harvesting, sorting, packaging and livestock management [7]. However, a holistic system that could gather sensor information from a WSN and feed it to a Decision Support System (DSS) is still missing. Moreover, this system could also provide inputs for automatic devices based on gathered information instead of being manually fed by users. The main focus of the AFarCloud project is to propose a holistic system oriented to precision farming. With improvements in agricultural Cyber-Physical Systems, and the application of a network of sensors and cameras connected through the Internet-of-Things to *" (...) provide a distributed platform for autonomous farming that will allow the integration and cooperation of agriculture Cyber Physical Systems in real-time to increase efficiency, productivity, animal health, food quality and reduce farm labor costs."* [7].

In that context, this dissertation proposes a platform capable of interconnecting Cyber-Physical Systems (CPS), in this case, multiple UAVs, and WSNs through the internet with the ability to develop custom solutions over it. This document will present the core functionalities of the platform and some additional custom features.

1.2 Objectives

The purpose of this dissertation is the development of a platform that connects multiple UAVs to a web platform. The objectives are part of the European project AFarCloud, namely:

1. Registration and configuration of UAV in the platform by assigning a type, token, and organization, plus the management tools such as token revocation, peripherals administration, and sensor calibration;
2. In-flight real-time monitoring with GPS coordinates, velocity, IMU, battery, flight mode, and video stream;
3. Automated mission planning tool for a single UAV and for multiple UAV;
4. Gathering of all flight data in a time-series database for big data queries;
5. Possibility to replay a flight mission information, gps coordinates, velocities, and video stream if present, through the platform.

Another goal for this project is the need to create an easily extended platform. Given the nature of the AFarCloud project, we developed a feature regarding a field of precision farming, dedicated to the creation, processing, and analysis of multispectral maps. These maps resulted from the upload of aerial pictures collected from UAVs during a flight mission. After the upload, these images must be processed and stitched together to create different indexes that would then overlap over a satellite map.

1.3 Solution Overview

The proposed solution is a cloud platform deployed in a private environment that will connect end-users to a UAV. This platform gives the possibility of offloading most of the intensive computing tasks that may be needed to run by the UAV to accomplish a user-specified task. Furthermore, it leverages the enormous data storage capacity of cloud computing data centers by sending most of the data sensed from devices present on the UAV to the cloud.

This system is based on the typical Three-tier architecture where there will be a presentation tier (user interface or Front end), a logic tier (Back end), and a data tier (database) [8]. Furthermore, a middleware layer is also present. This layer will be responsible for the connection to a WebSocket gateway that collects and broadcasts the data from the UAV as close to real-time as possible. Moreover, a media gateway will also be a part of the middleware layer, making possible for a low latency transmission of video streams.

1.4 Document Structure

This dissertation is organized in the following way:

- **Chapter 1: Introduction** - Summarizes the motivation and context of this dissertation and presents a small overview of the proposed solution.
- **Chapter 2: Problem Analysis** - Defines the domain of the problem and its relation to PDMFC and AFarCloud. There is also documented the software development process by describing the stakeholders and their needs, plus summarizing all this information in multiple diagrams.
- **Chapter 3: State of the art** - Summarizes the research done regarding the state-of-the-art for the technologies used. Furthermore, it also specifies the related work found regarding the work described in this dissertation.
- **Chapter 4: Approach** - Elaborates on the solution overview by presenting the architecture of the project in detail.
- **Chapter 5: Validation and Critical Review** - Presents multiple case studies performed in conjunction with the team of PDMFC. Subsequently, the data regarding these tests was analyzed and discussed in this chapter.
- **Chapter 6: Conclusion** - Enumerates the conclusions drawn from this dissertation and presents the future work.

Chapter 2

Problem Analysis

This second chapter describes the problem, first by answer the question of what Precision Farming is, then by presenting not only its advantages but also its challenges. After this, there is an introduction to the AFarCloud project. Moreover, the connection between PDMFC and AFarCloud is described. Then, the impacts that this European project plans to have on the Precision Farming field are presented. The last section of this chapter corresponds to the requirements elicitation for the platform. This section includes a study regarding the stakeholders and the platform's requirements derived from the analysis of the mentioned study.

2.1 Precision Farming

Precision farming or precision agriculture is the introduction of information technology (IT) solutions to agricultural processes [9]. This introduction led to the creation of farming management systems that leverage data to make insightful and environment-conscious decisions.

The traditional method of agriculture is an open feedback loop in the sense that its outputs aren't automatically used to impact the input. For over many years, farmers have used the results of a former successful harvest to optimize parameters on future crops. Agriculture engineers used this loop process only qualitatively, which lead many times to imprecise feedback. Unfortunately, there were some external factors that were also unpredictable, like the weather and pest infestations. The goal of precision farming is to define a Decision Support System (DSS) for a whole farm [10]. A DSS is an information system that requires data as input to generate decision-making activities. Sensors all around the farm gather data from their environment and feed it to the DSS. The type of data may vary from soil humidity to air quality and temperature. Then with the help of

some already developed and tested algorithms, the DSS provides the user with guidelines. These guidelines, with the corresponding GPS coordinates, will ensure the maximum growth for the crops, given that we have confidence in the algorithms chosen for the DSS.

Another field that helped in the improvement of precision farming is the use of UAV for capturing multispectral and RGB images [11]. These images can then be stitched together by using image processing algorithms that result in the creation of map projections. These maps retain per each pixel more information than the values for red, green, and blue. They can register the value of near-infrared and red-edge spectrum values, that some algorithms can use to process and produce a Normalized Difference Vegetation Index (NDVI). A knowledgeable professional can use the data provided by this index to measure and evaluate the health of the crops and the needs for water or fertilizer inputs.

2.1.1 Advantages of Precision Farming

There are two main benefits of precision farming, the first is related to, the maximization of profits, and the second minimizing the impact on the environment:

By utilizing these insightful decisions, a farmer can optimize the amount of water, fertilizer, chemicals, and more. By maximizing these factors, a farmer will start to notice the improvement of his crop performance [12, 13]. Furthermore, the knowledge of the correct amounts of these substances will also minimize its wasteful use, translating in fewer costs for the farmer. There is also possible to use these indexes to identify pests in a more timely manner. By gathering information from multiple indexes and combining this information, it is possible to study and analyze which combination is best suited to identify pests. As of now, a colleague is using this platform to generate different indexes (RGB, NDVI, NDRE, CIR, NDWI) from aerial photographs [14]. The DSS then notifies the farmer that a crop in a given location is suffering from a disease, giving the farmer more time to take action.

The right timing and quantity of water and chemicals used on crops reduce their impact on the whole environment. First, regarding the less consumption of water, thus minimizing the unnecessary waste of water. Second, by using the bare minimum amount of fertilizers and pesticides to ensure the optimal growth of the crops while also preserving the environment.

2.1.2 Challenges of Precision Farming

Dobermann [15] exposed one of these challenges by writing, "Researchers and farmers can easily collect huge amounts of information: but assessing the quality of this information, transforming it into meaningful management decisions, and evaluating potential benefits and risks has proven to be a difficult task". This means that it is still hard to extract knowledge from these systems regardless of the amount of collected data if there isn't a competent entity capable of deducing such conclusions. In order to reach such results,

there is a need for insights from agriculture engineers, crop scientists, and more, to make sense of the gathered data.

Another challenge that precision farming faces even nowadays is the low rate of adoption. Although the adoption's rate of seed genetics and precision steering has exceeded 50% due to its viability, the same doesn't apply to precision farming [16]. This low adoption rate is due to the initial requirement of a significant investment of capital and time, although this adoption could result in cost savings through more precise management of inputs [17]. These investments encompass not only the installation and acquisition of new equipment but also the time needed to learn how to use these technologies accordingly. This overhead expense is what makes the difference between Precision Farming and the seed genetics example. Although seed genetics is also expensive to develop, the adoption process is simple in comparison with Precision Farming. Due to the overhead expense and complexity of the adoption process of Precision Farming, many farmers may choose not to implement it even though it can bring insights to minimize costs and therefore increase the profit of a harvest.

2.2 AFarCloud Project

Aggregate Farming in the Cloud (AFarCloud) is an European project that aims to provide a distributed platform for autonomous farming that will integrate with CPS in real-time to increase efficiency, productivity, and reduce labor costs. This project also aims to make farming robots accessible to more users by enabling farming vehicles to work in a cooperative mesh. "This project is co-funded by the ECSEL Joint Undertaking and by national programmes and funding authorities" [1]. Some concrete services that will be provided to the users are:

- Enable reliable, high-performance, real-time and secure data exchange for CPS.
- UAVs for accurate plant health analysis and process high-quality maps during every stage of crop growth.
- Resource Optimization, through the use of Wireless Sensor Network (WSN) and real-time monitoring of the principal parameters in crop and livestock activities, in order to minimize unintended impacts on wildlife and the environment in many agricultural production systems.

2.2.1 Organization

The AFarCloud project divided the planned work into eight distinct Work Packages(WP), each with its independent purpose. These WP's are as follows:

[1] http://www.afarcloud.eu/

- **WP1. Project Management's** first objective is to monitor all project activities and ensure that all parties follow the project plan and the quality plan established. This WP is also responsible for tracking the progress of the overall project and make sure that tasks regarding more than one WP are well coordinated.
- **WP2. System Requirements, Architecture Specification, and Implementation** are responsible for user and system requirements for the AFarCloud platform. Aside from this, WP2 is also responsible for implementing the Semantic Middleware that will encompass a secure communication system for the platform.
- **WP3. Intelligent Coordination and Decision-support Solutions for Autonomous Operations** aims at the design and implementation of a mission management tool that will be able to orchestrate (semi)-autonomous missions in the domain of precision agriculture. Moreover, this WP is also responsible for the development of a graphical user interface that will allow human-robot but also human-system interaction.
- **WP4. Environment Characterization Platform's** main task is the orchestration of multiple data input streams provided by not only the deployed distributed sensors but also by cloud and big data analytics systems.
- **WP5. Sensor and Actuator Development** aims at the design and implementation of the necessary sensors to monitor, sample, and actuate over multiple plants and animals.
- **WP6. Autonomous System Development and Legacy System Integration** focus on the development of autonomous systems such as Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicles (UGVs). Furthermore, integration technologies regarding these solutions are also to be made by partners under this WP.
- **WP7. Demonstrators Definition, Integration, Verification, and Validation** are responsible for the development and configuration of the overall infrastructure for the demonstrations and validation of the AFarCloud objectives.
- **WP8. Innovation and Business Management, Dissemination, Exploitation, and Standardization's** focus is the dissemination of news and articles related to the project, being State-of-the-Art articles, or the publication of results from AFarCloud demonstrations.

2.2.2 PDMFC Connection

Projeto Desenvolvimento Manutenção Formação e Consultoria (PDMFC), Lda is an IT company founded in 1993, that participates in national and European projects such as Portugal 2020 and Horizonte 2020. These projects typically fall in areas such as information governance, security, cognitive, cloud, digital transformation, among others. Aside

from this, PDMFC is part of PDM Group covering areas such as IT, Telecommunications, Gaming, Marketing, Electronics, Finance Investments, and Social Economy.

PDMFC is one of the 56 partners that make up the AFarCloud consortium and one of the five Portuguese parties involved. PDMFC is present in many Work Packages, but the ones that are related to this dissertation are WP3 (Intelligent Coordination and Decision-support Solutions for Autonomous Operations) and WP4 (Environment Characterization Platform). For the first one, WP3, the developed work was regarding a user interface that facilitates the interaction between human-robot. It was mainly for this finality that the Beyond Skyline platform was co-developed by 3 Portuguese partners PDMFC, APPS4MOBILITY, AND BEYOND VISION. The main focus of this dissertation is the design, development, and validation of the mentioned platform. Regarding WP4, the work developed was a mobile user interface for accessing different input data streams such as sensors data, cow's collars data, UAVs data, UGV's data, and weather data.

2.2.3 Impact on Precision Farming

As mentioned before, precision farming has many benefits, such as reduction of costs and human burden, optimization of the amount of water and chemicals used, among others. However, there are some challenges to overcome, like the lack of knowledge that professionals can derive from all the collected data, but also the limitation of time in manual work. Because of this, there is not enough time to analyze all the data, and most of it gets discarded. Aside from all the benefits that the solution from AFarCloud will bring, by being a product for Precision Farming, it also tries to mitigate this challenge with the introduction of a Decision Support System (DSS). One of the specified Work Package's focus is on the design and development of a DSS that will feed on data from input streams and also data from past decisions and missions to help support the user's next decision. The objective is to reach a point when the DSS has enough experience and data to predict what would be the best decision to make next, and suggest it to the user. In the first phase, the user will always have to make some manual operation so the system can perform the specified next action. In the second phase, the system will be autonomous enough to come up with the next move and to carry it out without any human supervision.

2.3 Requirements Elicitation

This section introduces the stakeholders identified for the AFarCloud project and which are their struggles and needs. With this information, the AFarCloud was capable of defining the functional and non-functional requirements for the platform. With this specification of various system functions, we were able to create Use Cases for each stakeholder, which summarizes the possible actions each user could make on the system.

Subsequently, to further define each step that the system needs to perform to reach an objective, the author produced some Activity Diagrams.

2.3.1 Stakeholders

The following stakeholders are entities that will personally use the platform or have a direct interest in the outcome of the former. The identification and interviews of these entities were all conducted before the development of this dissertation. The following bullet list presents a summary of each stakeholder, what are their work needs and responsibilities, and what they need to achieve with the platform.

- **Farm Owner:** Is the owner of agricultural land and is responsible for the production decisions to maximize its profits. It needs to be capable of taking insight from past collected data to make these decisions and be able to have an overall view of the farm at all times. Regarding the platform, the Farm Owner will have an **Admin** role, meaning he can see every information and take any action.
- **Farm Operations Manager:** Normally, when a production farm is sizable, and the Farm Owner can't manage all its workers, it needs a Farm Operations Manager. This entity is most common to know all the technical aspects of all agricultural processes on-going on the farm, leaving only the financial and economic responsibilities to the Farm Owner. Regarding the platform, he has the **Operations Manager** role that has all the permissions of the admin except actions concerning organizations.
- **Farm Worker:** Is typically a worker on a farm and will use the platform to submit some of its work. For example, a worker responsible for a drone mission will need to download the mission from the platform, set up the drone, perform the mission, and finally report the results. Another example is that a worker might need to check the sensors of a given area to use the data to make decisions. Regarding the platform, the Farm Worker as the lowest level of permissions and has the **User** role.
- **Farm Administrative:** Is equivalent to the Farm Worker with the added responsibility of a set of bureaucratic tasks. Regarding the platform, this user has a specific role and can manage several users. The **Administrative** must perform most of these tasks through the platform.
- **Drone Services Providers:** A farm can own a fleet of drones, or it can rent third-party drones from a Drone Services Provider. Regarding the platform, the **Drone Services Provider** is an external user that is capable of renting drones to several farms in a service manner.
- **Platform Managers:** This is the one responsible for maintaining the proper functioning and correctness of the whole platform. It will manage the different organizations present on the platform and the respective admin users. Regarding the platform, the Platform Managers will have the **Platform Administrator** role.

2.3.2 Requirements

This section will present the requirements that the AFarCloud partner produced during the interviews with the stakeholders. Moreover, this list of requirements was extended with more functionalities that were discussed with PDMFC. All the requirements will be numbered to simplify their connection with the work developed, further detailed in next chapters. Functional requirements will have the prefix (FR) and the Non-Functional Requirements will have the prefix (NFR).

2.3.2.1 Functional

From the interviews with the stakeholders, it was possible to extract some functional requirements that we can translate into platform functionalities. These requirements were grouped into four sections, each pertaining to a different topic:

- Functional requirements connected to UAVs are detailed as **Drone Functionalities** int Table 2.1.
- The requirements regarding the creation and management of missions on the platform are described as **Mission Functionalities** and are defined in Table 2.2.
- The **Maps Functionalities** encompasses all the possible actions a user can make over each map and are enumerated in Table 2.3.
- Lastly, all the functional requirements concerning the managing and administrating of the platform are under **Platform Functionalities**. These pertains all the functionalities regarding users, permissions, roles, and organizations management, and are represented on Table 2.4.

All tables follow the same structure, giving the ID and name of each functional requirement, and a small description of a user should achieve with each functionality.

ID	Name	Description
FR 1	Create Drone	The administrative must be able to add a new drone to the platform
FR 2	Edit Drone	The administrative must be able to edit drone's details
FR 3	Delete Drone	The administrative must be able to remove a drone from the platform
FR 4	View Drone	The worker must be able to view the drone's details
FR 5	List Drones	The worker must be able to list existing drone that he has access to
FR 6	Provide Drone	The drone provider must be able to add a drone to an organisation
FR 7	Withdraw Drone	The drone provider must be able to withdraw a provided drone

Table 2.1: Drone Functionalities

ID	Name	Description
FR 8	Create Mission	The administrative must be able to create a new mission
FR 9	Edit Mission	The administrative must be able to edit a mission's details
FR 10	Delete Mission	The operations manager must be able to delete a mission
FR 11	View Mission	The worker must be able to view the mission's details
FR 12	Execute Mission	The worker must be able to execute a previously created mission
FR 13	Monitor Mission	The worker must be able to monitor an on-going mission
FR 14	List Missions	The worker must be able to see the list of created missions
FR 15	Mission Analysis	The operations manager must be able to analyse data from previous executed missions

Table 2.2: Missions Functionalities

ID	Name	Description
FR 16	Create Map	The administrative must be able to create a new map
FR 17	Edit Map	The administrative must be able to edit a map's details
FR 18	Delete Map	The operations manager must be able to delete a map
FR 19	View Map	The worker must be able to view the map's details
FR 20	List Maps	The worker must be able to see the list of processed maps
FR 21	Map Analysis	The operations manager must be able to analyse data from processed maps

Table 2.3: Maps Functionalities

ID	Name	Description
FR 22	Create User	The operations manager must be able to create a new user
FR 23	Edit User	The operations manager must be able to edit a user's details
FR 24	Delete User	The operations manager must be able to delete a user
FR 25	View User	The operations manager must be able to view the user's details
FR 26	List Users	The operations manager must be able to see the list of all user's for his organization
FR 27	Create Role	The administrative must be able to create a new map
FR 28	Edit Role	The administrative must be able to edit a role's details
FR 29	Delete Role	The administrative must be able to delete a role
FR 30	View Role	The administrative must be able to delete a role
FR 31	List Roles	The administrative must be able to list all roles
FR 32	Create Permission	The operations manager must be able to create a new permission
FR 33	Edit Permission	The operations manager must be able to edit a permission's details
FR 34	Delete Permission	The operations manager must be able to delete a permission
FR 35	View Permission	The operations manager must be able to view the permission's details
FR 36	List Permissions	The operations manager must be able to list all permissions
FR 37	Create Organization	The platform administrator must be able to create a new organization
FR 38	Edit Organization	The platform administrator must be able to edit a organization's details
FR 39	Delete Organization	The platform administrator must be able to delete a organization
FR 40	View Organization	The platform administrator must be able to view the organization's details
FR 41	List Organizations	The platform administrator must be able to list all organizations

Table 2.4: Platform Functionalities

2.3.2.2 Non-Functional

In contrast to functional requirements that concern specific functions on the system, non-functional requirements specifies factors that should be part of the system. Typically these requirements are detailed in the system architecture design. For the architecture design of this platform, we arranged the non-functional requirements in four different categories: Usability, Reliability, Performance, and Supportability.

Usability concerns the ability of a system to provide an effective and efficient environment for its users to perform their tasks. These concerns might be physical, technical, or even legal since some processes might have specific regulations that the platform should have in consideration. From the interviews with the stakeholders, these are the usability requirements for the platform.

ID	Name	Description
NFR 1	Online	The platform must provide its services online
NFR 2	Notifications and real-time	The users must be able to use the services in real-time and get real-time notifications of updates
NFR 3	User Friendly	The interface for the platform must be user friendly
NFR 4	Simple and Easy-to-Use	The interface must be user driven to simplified its use
NFR 5	Easy in-platform navigation	All users must be able to effortlessly navigate the platform
NFR 6	Legal rights protection	The Platform must protect legally its users rights

Table 2.5: Usability Requirements

To ensure safety and function without failure, we also make sure to introduce some reliability requirements. **Reliability** not only gives respect to the security of the platform concerning topics like data encryption (NFR 7) and security protocols (NFR 8), but it also encompasses the reliability (NFR 11) and the ability of the platform, given some conditions, to keep functioning for some time. Table 2.6 presents the reliability requirements.

ID	Name	Description
NFR 7	Encryption Mechanisms	The platform must provide encryption mechanisms
NFR 8	Security Protocols	The platform must have data transmission security protocols
NFR 9	Security Password	The users must use a password to gain access to the platform
NFR 10	Quality	The platform must ensure quality of service
NFR 11	Reliability	All services provided by the platform must be reliable
NFR 12	Interoperability	The platform must be interoperable

Table 2.6: Reliability Requirements

Performance requirements for the platform are to ensure performance, low response time (NFR 14), high processing power (NFR 15), large storage capacity (NFR 18), among others. Given the nature of this platform and the real-time connection with UAVs, there was more emphasis on this point. Particularly on the subject of processing power and

storage capacity. UAVs produce a variety of data every second that they on, and most of that data is useful for the user to monitor. Aside from this, to be able to analyze prior flights and extrapolate the reasons why they went well or not, there is a need to store a large amount of data for each flight. Another point that stakeholders mentioned in interviews were the need for the platform to be steadily available (NFR 16). Especially the insurance that the platform will always be available when a given UAV has taken flight.

ID	Name	Description
NFR 13	Efficency	The platform must be efficient
NFR 14	Low Response Time	The platform must have low response time
NFR 15	High Processing Power	The platform must provide enough processing power so the services can be used smoothly
NFR 16	Availability	The platform must be constantly available
NFR 17	Functionality	The platform must always provide a fully functional session for a user
NFR 18	Large Storage Space	The platform must have sufficient storage space for storing information and data

Table 2.7: Performance Requirements

Regarding **supportability**, stakeholders were more concerned with the importance of maintenance of the platform (NFR 19). Moreover, they were preoccupied with upgradability (NFR 20) and the insurance that future updates, like the addition of features, would not cause the platform to mal-function. There was also great concern regarding the fault tolerance (NFR 22) and the ability of the platform to maintain proper function even when there is some type of error.

ID	Name	Description
NFR 19	Maintenance	The platform must be regularly maintained
NFR 20	Upgradability	The platform must keep functioning with future updates
NFR 21	Adaptability	The platform must be able to adapt to new user requirements
NFR 22	Fault Tolerance	The platform must be design to handle crashes

Table 2.8: Supportability Requirements

2.3.3 Use Case Diagrams

This section presents two use case diagrams, one for the Drone, Map, and Mission functionalities (Fig. 2.1), and other for the Platform functionalities (Fig. 2.2). All the functionalities presented in these use cases correspond to at least one functional requirement. These use cases demonstrate the interactions between the different roles and the system in a summarized and visual way.

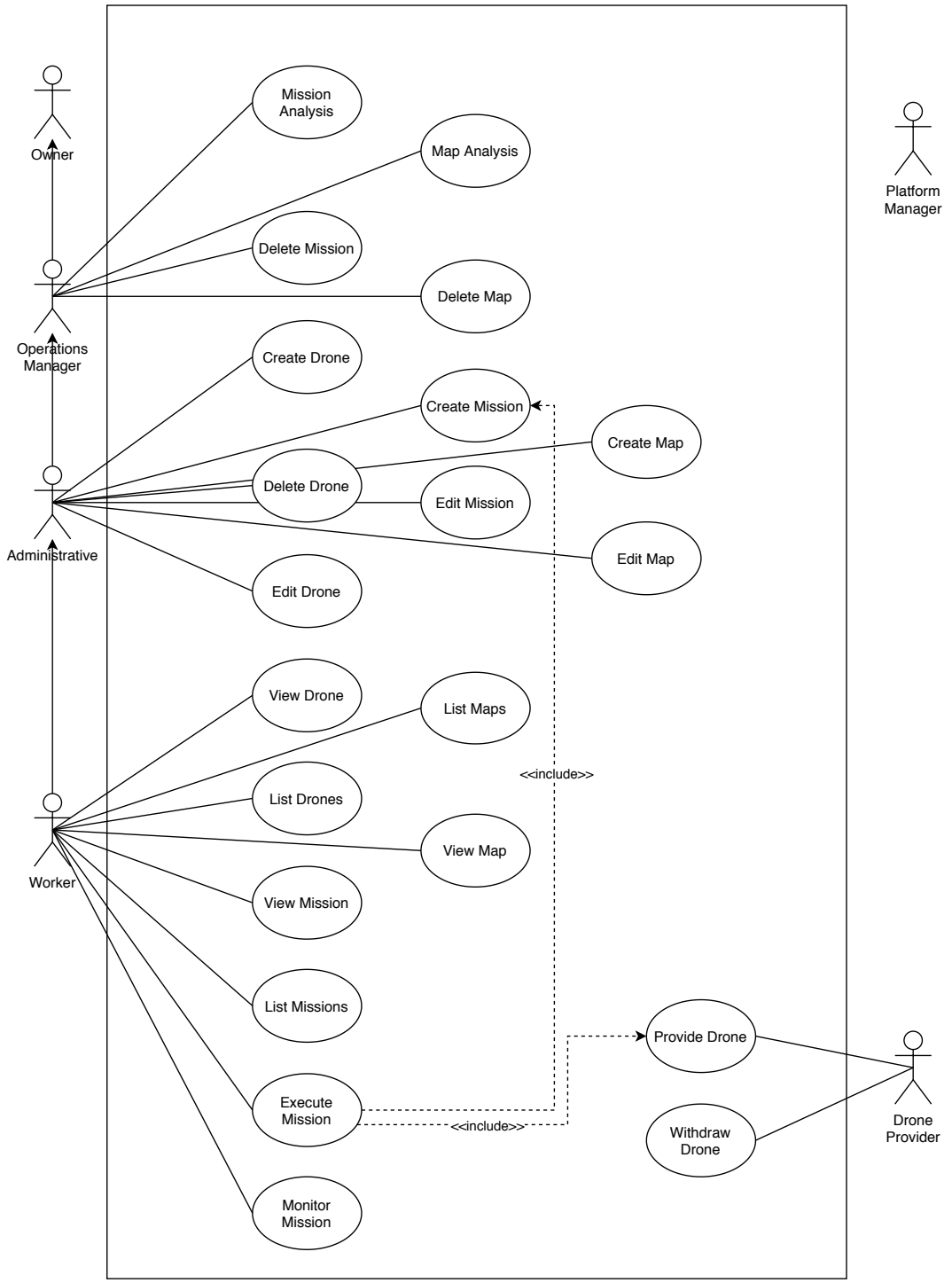

Figure 2.1: Use Case Diagram for Drone, Map, and Mission Functionalities

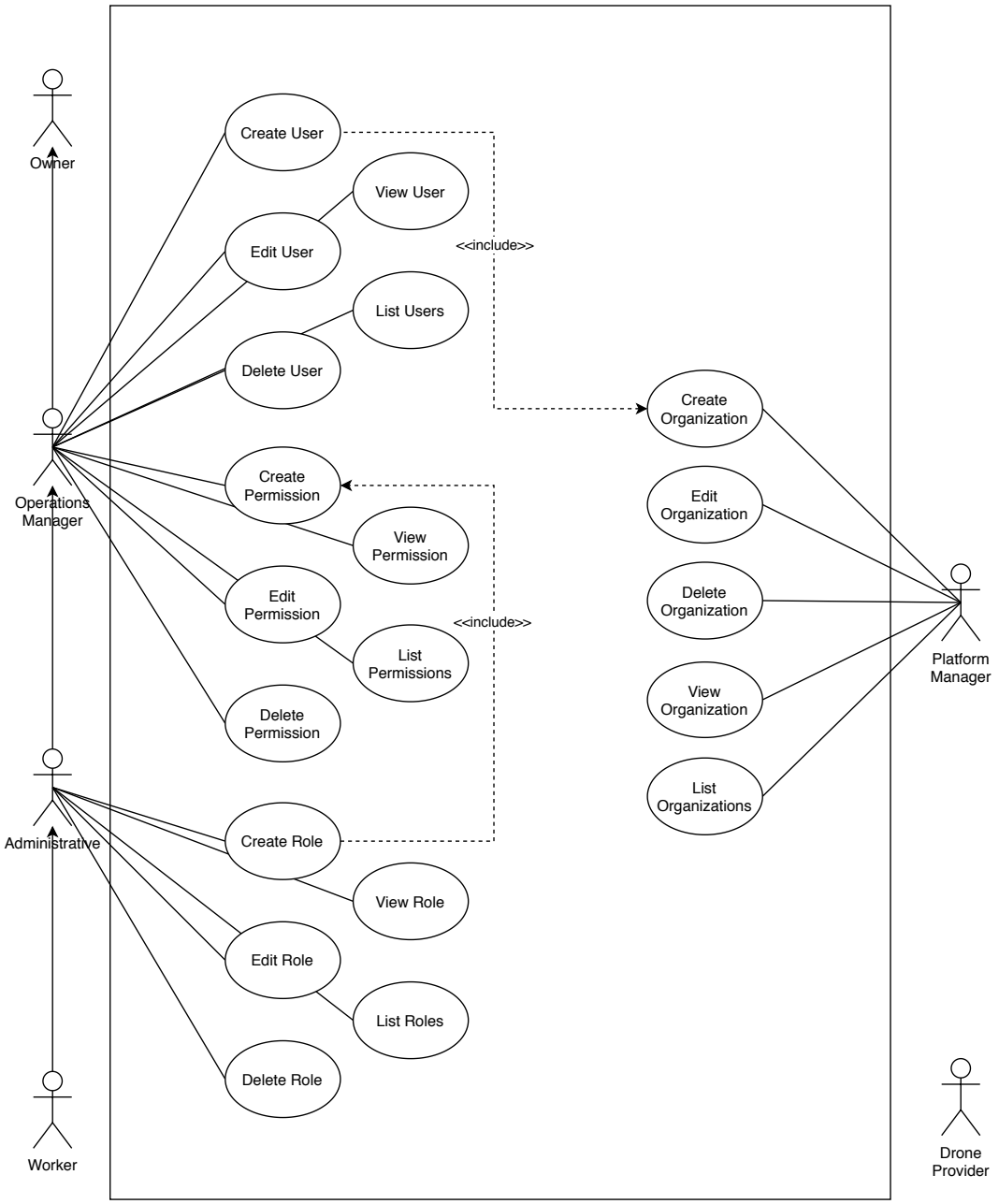

Figure 2.2: Use Case Diagram for Platform Management Functionalities

Chapter 3

State of the art

The third chapter presents the state-of-the-art of relevant technologies regarding the development of the platform. First, there is given an introduction to what is Cloud Computing, how it is organized in layers, and the different types of clouds that exist. Next, there is explained what composes a Cyber-Physical System, some of the modern ways to use it, what are its inherent challenges, and finally, what is an Unmanned Aerial Vehicle. After this, some relevant Framework Concepts are explained, such as the Robot Operating System, Node.js, and InfluxDB. Together there is also described one communication protocol called WebSocket. Finally, there is presented the research regarding the related work.

3.1 Cloud Computing

NIST[1] defined Cloud Computing in 2011 as "(...) a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)" [18].We can think of Cloud Computing as the provisioning of both the applications (service) hosted on the infrastructure on a data center but also the infrastructure itself, where there is the possibility to use computer resources in a pay-as-you-go subscription model [19]. According to multiple studies we can separate Cloud Computing in four architectural layers, Hardware/Data center, Infrastructure, Platforms, and Applications [20, 21, 22].

The hardware layer: This corresponds to the lowest level layer, where there is the possibility to manage the physical resources of the cloud, such as servers, routers, switches, among others. These are physically implemented on data centers where servers are organized in racks.

[1] National Institute of Standards and Technology

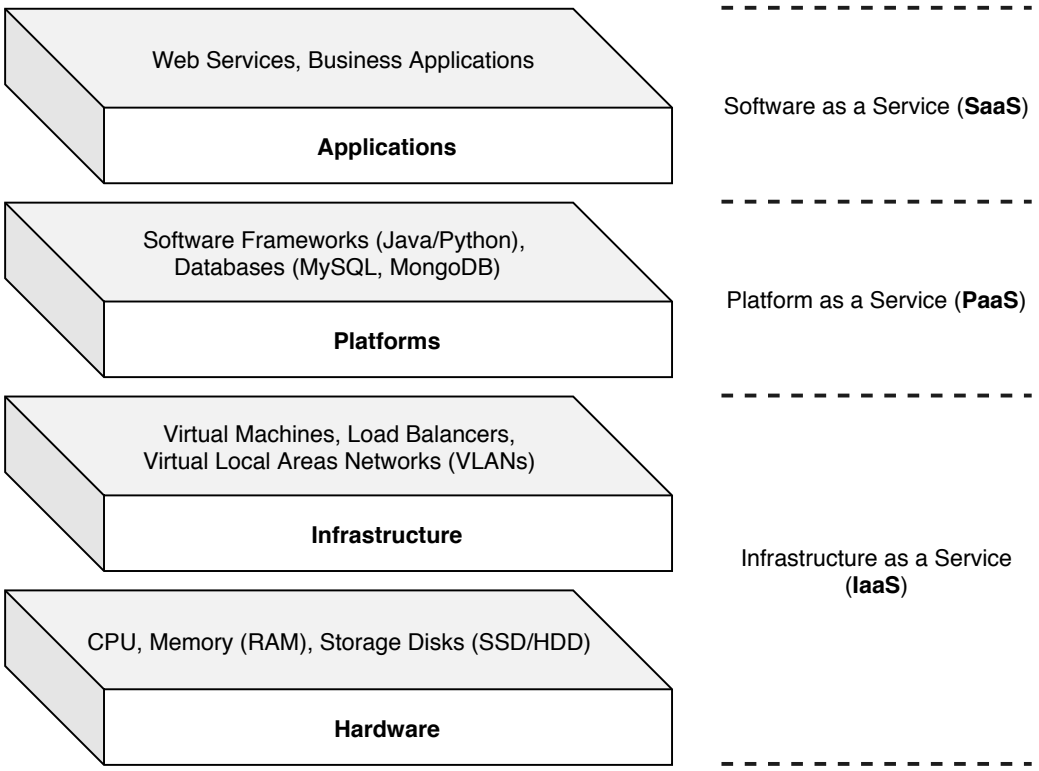

Figure 3.1: Cloud Architecture: Relation of architectural layers and business models

The infrastructure layer: This layer is also called the virtualization layer since it generates and manages a pool of computing and storage resources, it's also considered an essential component of cloud computing since it enables the possibility of dynamic resource allocation.

The platform layer: Consists of the operating systems and applications frameworks, and leverages the infrastructure layer in order to minimize the effort of deploying applications directly to virtual machines containers.

The application layer: Known as the highest level layer and consists of the actual cloud applications, it benefits of automatic-scaling to achieve better performance and availability.

In spite of this, there is also a service-driven business model that provides multiple services to the customers based on different requirements, these can be divided into *Infrastructure as a Service (IaaS)*, that manages and provides infrastructural resources on-demand, such as Virtual Machines, Private Networks, Memory (RAM), Storage Disks, among others; *Platform as a Service (PaaS)*, that offers the possibility of utilizing operation systems and software development frameworks for the implementation of IT solutions; and *Software as a Service (SaaS)*, that refers to the provisioning of on-demand applications over the Internet [23]. The relation between the architectural layers (Hardware, Infrastructure, Platforms, and Applications) and the business models (IaaS, PaaS, and SaaS) is visually represent in figure 3.1.

The last point to consider when defining Cloud Computing is the type of cloud that the application will be deployed or implemented. There are 3 main types of cloud. These are a Private Cloud, a Public Cloud and a Hybrid Cloud [24].

Public Cloud: consists of service providers that offer their resources through a network that has public access. This type of cloud has some advantages such as the possibility to start using it without any initial investment, and the possibility of shifting the responsibilities regarding security and availability to these providers. However, there are also some disadvantages like the lack of control over the data that flows through the network and some security settings.

Private Cloud: it was designed to be managed solely by a single organization, although some parts could be assigned to an external provider and it could be hosted either internally or externally. This type of cloud presents the highest degree of control over the way that data gets passed around and also regarding the performance and availability of the system as a whole.

Hybrid Cloud: It's a combination of both a public cloud and a private environment, where the last could be a private cloud or just an on-premises solution, with the goal of addressing the limitations of the types mentioned above. While part of the solution could be hosted on a typical public cloud leveraging the benefit of no initial investment, the other part can be self-deployed by the organization over a private cloud where they have a fine-grained control over all the data that gets thrown around. This is a more flexible approach where it's possible to maintain control over the security of the application but still facilitating on-demand scalability.

Given the aforementioned characteristics and the functional requirements of CPS mentioned in section 3.2, such as real-time processing and storage, many researchers have developed studies in which they use the resources of cloud computing to enhance a Cyber-Physical Systems [25]. These studies fall on a field called Cloud Robotics, a term coined by James Kuffner in 2010 [26], where the idea of offloading massive computation tasks to remote servers hosted on a cloud was born. With this solution there is the possibility to offload the computation power and storage requirements of a CPS like a UAV to the cloud, by connecting these assets to a network, leveraging the advances made in network technologies and high-speed networks. This was exactly what was done in [27] where there was require to perform real time mapping, localization, autonomous navigation and object recognition.

3.2 Cyber-Physical Systems

3.2.1 Overview

Edward A. Lee defined Cyber-Physical Systems in 2006 as "(...) integrations of computations with physical processes. Embedded computers and networks monitor and control

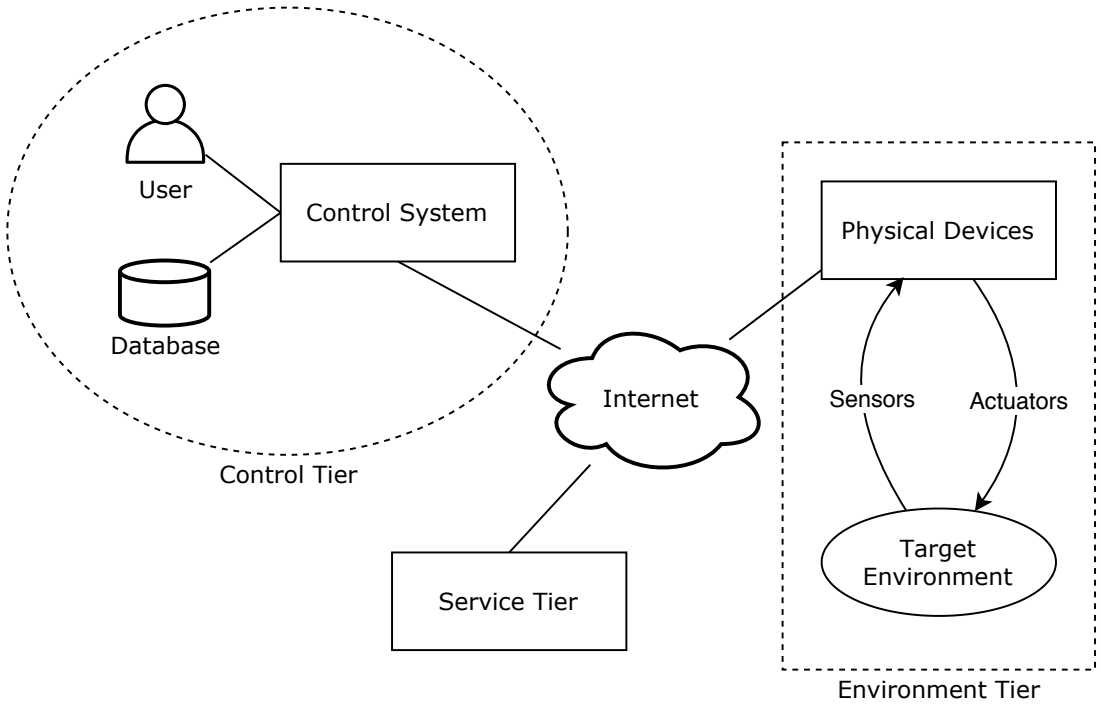

Figure 3.2: A Service-Oriented Architecture for CPS

the physical processes, usually with feedback loops where physical processes affect computations and vice versa." [28]. This tells us that, at that time, CPS were perceived as systems with a physical part that produces inputs from sensing its surroundings, and continuously send these values to a virtual part of the system so they could be analyzed and acted upon. This is the feedback loop where the values coming from the physical part directly impact the computations made on the virtual part and the resulting outputs of that computations are used as inputs for the physical part. This definition continues to evolve, like when Ji Feng He stated the importance of computation, communications, and control (3C) in regards to environmental perception [29], where he explains the mutual effects of the feedback loop on the virtual and physical processes in order to monitor or control a mechanism by leveraging real-time sensing and dynamic control in the safest, reliable and efficient way possible.

In its early stages, CPS had a two-tier architecture, one physical tier that senses the present environment and communicate that data to the computing tier that analyzes the data and then make the decision on which action the physical tier would execute. However, in 2010, Hyun Jung La proposed a three-tier architecture for CPS [30] that included a control tier, a service tier, and an environmental tier and all would be interconnected through the internet like it is shown in Figure 3.2.

The control tier functions as the "brain" of the system where data comes in from physical devices and based on the analysis of that data, a service from the service tier is activated to complete a specific task. This tier also functions as the entry-point of

any human-related input and can also include a database with the purpose to store the mentioned data. The service tier encapsulates all the services provided by the system in a typical modular web service way, where each service should be reusable and designed with adaptability [31] and commonality [32] in mind. The environment tier aggregates not only all the physical devices of the system, such as sensors or actuators, but also the target environment where these devices are located. Although these devices can run their own applications, that normally consist of transferring the information needed for the other parts of the system, they are heavily restricted by the limited resources that the underlying devices possess.

3.2.2 Applications for Cyber-Physical Systems

After the overview of what is a Cyber-Physical System, it makes sense to enumerate some of the modern applications being developed with CPS. Currently, there are multiple researchers conducting work using CPS in various fields, such as medical devices, aviation software, traffic control, and safety, advanced automotive systems, distributed robotics, biosystems, communication systems, among others [33]. From these numerous fields, we can understand that CPS is a very versatile concept that can be applied to multiple problems. Moreover, there were advancements made in other areas of research, such as Industry 4.0, where CPS is one of the components [34]. Some applications in this area can be sensor and actuator networks, intelligent robots and machines, big data, among others. In this context, CPS function more like an embedded system that can control physical processes.

There is some work being done in recent years regarding the collaboration of the fields of precision farming and CPS. Most of this work is in the area of sensing and monitoring. The next two examples demonstrate this point. The first one is a study of the development of a cyber-physical system architecture to monitor potato crops [35]. The purpose of this system is the monitoring of vegetation conditions. They used sensors at ground level to measure multiple soil properties such as nitrogen levels, pH, potassium, phosphorus, among others. The second one presents other architecture for the same circumstances, where it describes a cloud-enabled CPS platform [36]. The authors divided the platform into two sections, the edge and the cloud layer. In the edge, they deploy sensors and gateways, while in the cloud layer they deploy computational services. The purpose of this platform is to help agriculturists to monitor the ever-changing environmental conditions of their crops, as well as their historical data.

3.2.3 Challenges

This first challenge that most researchers working in this field commonly have to deal with is the limited processing and storing capabilities of most physical devices [37] such as wireless sensors, mobile robots, and so on. Although there is the possibility to enhance these devices with more processing power and storage capacity this would translate in

the increase not only in size but also in the cost of these physical devices. This increase in size and cost would negatively impact most use cases of CPSs, since many of them rely on the small size and cost of these devices. The second challenge naturally present on these systems is the amount of data that physical devices are capable of producing over time. This not only brings us to the first challenge where there is a need for great computing resources to analyze and extract insightful information from these data but also delves into the problem of big data and the need for abundant storage resources. It was due to the challenges mentioned above that many researchers suggest the use of cloud computing, also referred by Cloud Robotics [26], as a viable solution for these [38], [39].

3.2.4 Unmanned Aerial Vehicles

To understand the option for choosing a multi-rotor UAV in the development of this platform first, we need to know that there are four different types of UAV. Each one has different characteristics and is suitable for different use cases. An Unmanned Aerial Vehicle (UAV) can be defined as an aircraft that carries no human pilot or passengers, typically it is part of an Unmanned Aircraft Systems (UAS) which is composed of a UAV, a ground-based control and the communication between the two. The flight of a UAV can be controlled in two ways, guided by a pilot on the ground-based control that actively sends commands to the UAV, or it can fly autonomously on pre-programmed flight plans [40].

Presently there are four main UAV categories where each UAV must fall into, these are **Multi-Rotor**, **Fixed-Wing**, **Single-Rotor**, and **Fixed-Wing Hybrid** [41].

Multi-Rotor (Fig. 3.3) are the most common type of UAVs, and also the cheapest, which are used by not only professionals but also amateurs, mainly for aerial photography and aerial video surveillance, this is due to the fact that they give the most control over position and framing. The big drawback of this type of UAVs is their big limitations when it comes to endurance and speed, not only this but they also require a lot of energy just to fight gravity and stabilize themselves, so they drain their batteries relatively fast during a flight (20 to 30 mins). These can be further classified based on the number of rotors present on the UAV, they are **Tricopter** (3 rotors), **Quadcopters** (4 rotors), **Hexacopters** (6 rotors) and **Octocopters** (8 rotors).

Fixed-Wing (Fig. 3.4) cannot lift vertically and can't hold their position while in the air because of their "wing" like normal airplanes, this type of UAV is much more energy-efficient than multi-rotor ones. Due to the lack of need in fine-grained control of the stabilization of this type of UAVs, they can also make use of a gas engine which increases their autonomy to more than 16 hours and allows them to cover much longer distances. The downsides are that they cannot hover, so there isn't the possibility to process aerial mappings, and because of their size the launching phase is a lot trickier since they need a runaway or a catapult launcher to get them into the air. The landing phase is also more complicated to process, as they will require a runway, a parachute or a net to recover them

Figure 3.3: Multi-Rotor UAV

safely. Another downside is their higher cost of manufacturing and a bigger learning curve when it comes to piloting.

Figure 3.4: Fixed-Wing UAV

Single-Rotor (Fig. 3.5) look very similar in design and structure to actual helicopters since it has one big sized rotor plus a small sized one on the tail of the UAV to control the heading. Benefits of much greater efficiency over the multi-rotor and can also be powered by a gas motor for even longer endurance. This higher efficiency is due to a rule of aerodynamics that the larger the rotor blade is and the slower it spins, the more efficient it is. In terms of complexity, operational and manufacturing costs, the single-rotor is coming in first when compared with the other. However, the big drawback is the danger that their big spinning blades pose, the long sharp blade of a single-rotor can be fatal and also cause big property losses when not taking the proper caution.

Fixed-Wing Hybrid (Fig. 3.6), also known as Vertical Take-Off and Landing (VTOL) combines the benefits of Fixed-Wing models (higher flying time) with that of rotor based models (hover). A vertical lift is used to lift the UAV up into the air from the ground. Gyros and accelerometers work in automated mode (autopilot concept) to keep the UAV stabilized in the air. Remote based (or even programmed) manual control is used to guide the UAV on the desired course [42].

The table 3.1 presents a structured summary of the main advantages and drawbacks of each UAV type, but also it introduces some of the most common and typical use cases for each one.

As is described in the previous table, the multi-rotor UAVs most common use cases are

Figure 3.5: Single-Rotor UAV

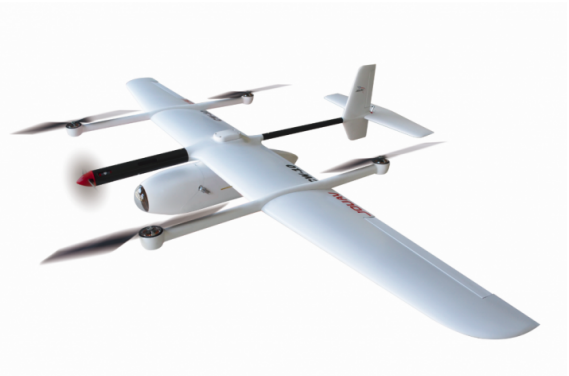

Figure 3.6: Fixed-Wing Hybrid VTOL UAV

UAV Types	Advantages	Drawbacks	Use Cases
Multi-Rotor	Easy Accessibility Cheap Manufacturing VTOL and Hover Flight	Short Flight Times Small Autonomy Small payload capacity	Aerial Photography Aerial Surveillance
Fixed-Wing	Long Endurance Fast Flight Speed Covers Higher Distances Higher Autonomy (>16 hours)	"Launching and Landing Complexity (No VTOL)" Expensive High Learning Curve	Pipeline Inspections Aerial Mapping
Single-Rotor	VTOL and Hover Flight Long Endurance (Gas Engine) More Efficient than Multi-Rotors	Expensive High Learning Curve More Dangerous	"Aerial LIDAR laser scanning"
Fixed-Wing Hybrid	VTOL and Hover Flight	"Not perfect at either Hovering or moving forward" Still in development	Drone Delivery

Table 3.1: UAV types summary

aerial photography and surveillance. That is why we chose this UAV type for our platform since those are some of the main features. After this definition of what is a UAV and in which categories they can be separated, table 3.2 introduces a more detailed comparison in regards of autonomy (the quantitative capacity for a flight of a UAV), accessibility (the easiness of maneuvering said UAV), cost (the manufacturing cost of a UAV), Hover & VTOL (the ability to hover in specified position and capability to vertically launch and

land), and flight speed (the maximum speed a UAV can reach during flight).

UAV Types	Autonomy	Accessibility	Cost	Hover & VTOL	Flight Speed
Multi-Rotor	Low	Very High	Low	Very High	Low
Fixed-Wing	Very High	Low	High	Low	Very High
Single-Rotor	High	Low	Very High	Medium	High
Fixed-Wing Hybrid	Medium	Medium	TBD	High	Medium

Table 3.2: UAV types comparison

3.2.5 HEIFU

Hexa Exterior Intelligent Flying Unit (HEIFU) is a custom made solution, aimed at the agricultural sector, developed in a collaboration between PDMFC and Beyond Vision. The HEIFU is equipped with an onboard computer running Ubuntu and Robot Operating System (ROS). This is an open platform that allows the integration of different inputs and is used to run multiple tasks, such as image processing, data relaying, remote control of a UAV (and more). HEIFU can be used with different communication systems, such as a mobile network (3G, 4G or 5G) or Wi-Fi connection. HEIFU is a Hexacopter, as it can be seen in figure 3.7, with a dimension of 1.4m in the diagonal wheelbase. The UAV weight is around 6.2Kg, including battery, and the hovering time is around 38min with a battery of 16Ah.

Figure 3.7: Hexa Exterior Intelligent Flying Unit (HEIFU)

3.3 Framework Concepts

In this section, there will be given a short description of each framework used to reach the goals mentioned in chapter 2. The first framework is ROS which was used as a Middleware to make the communication between the UAVs and the Back end of the application

possible. The next framework is NodeJS, a popular open-source asynchronous javascript runtime which enabled a cycle of request/response from the user to the server. Followed by the communication protocol WebSocket, which consists of full-duplex communications channels over a single TCP connection. Lastly, there is also a small description of what is InfluxDB, and why is it advantageous to use if in this context.

3.3.1 Robot Operating System

Joseph L. defines Robot Operating System (ROS) as a: "robot application development platform that provides various features such as message passing, distributed computing, code reusing, (...)" in [43], but to put it simply ROS is a collection of tools and libraries for robotics development that aims to simplify the task of developing complex robot software. ROS started in 2007 by Morgon Quigley with the name Switchyard, as a part of a research project being conducted in Stanford. ROS has three levels of concepts: the **Filesystem** level, the **Computation Graph** level, and the **Community** level [44], however, we are not be getting into much detail over the last one. The *ROS Filesystem level* refers to any resource that is maintained on a hard disk, such as:

- **Package:** It's considered to be the building blocks of ROS software, they may contain ROS runtime processes (nodes), a ROS dependent library, datasets, configuration files, among others. As it's said in [45]: "the philosophy is to make a piece of software that could work in other robots with only little changes to the code".
- **Message types:** It refers to the definition of the data structure for messages sent in ROS.
- **Service types:** This are the typed definition for the request/response data structures for services in ROS.

The *ROS Computation Graph level* is the abstraction of a peer-to-peer network of ROS processes that work together to accomplish the specified task for a given software and includes the following:

- **Nodes:** It corresponds to processes that perform computation tasks. These can be written in C++ (roscpp) or Python (rospy).
- **Master:** It provides name registration and lookup to nodes and services in the ROS environment, can be called a "nameservice".
- **Messages:** This is the communication unit for the ROS system, all the communication is made through messages, be it node-to-node or node-to-service and vice-versa. These data structures, comprised of typed fields, can have primitive types such as integers, floats, strings, booleans, or others, or arrays of these primitive types. These messages are sent and received through channels called topics.

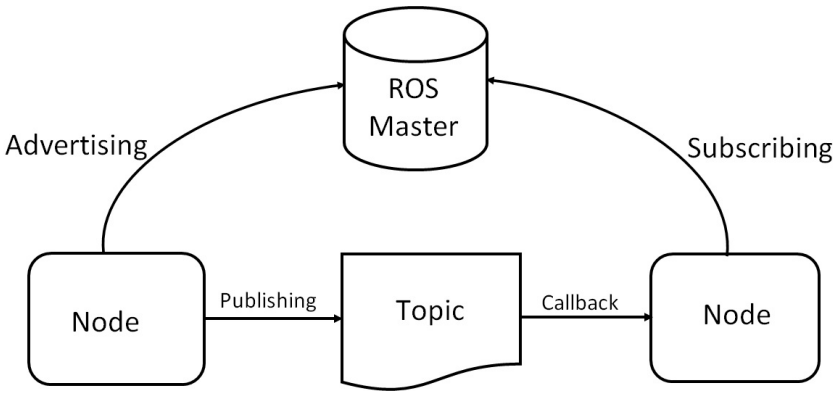

Figure 3.8: Representation of ROS communication protocol

- **Topics:** Transport system via which messages are routed to different nodes using a publish/subscribe semantics. If a node has data to send, it must subscribe to a given topic and sent the data using a defined message type. On the other side, if a node need this data, it must subscribe to the same topic and will await the data expecting it to be in the structure defined previously.

- **Services:** When there is the need for a request/reply type of interaction, in which a component may not continue to work properly while waiting for the data provided by another component and must block until it gets this data, the publish/subscribe method is not the most appropriated. This is the purpose of services, which are defined by a pair of message structures: one for the request and one for the reply [2].

A piece of ROS software starts by initiating the ROS Master node that will enable the communication between the required nodes and services. The publisher nodes start to emit data in the form of messages to a designated topic and the corresponding subscriber nodes then consume the messages gathering the data transmitted. A visual representation of this process can be seen in Figure 3.8.

3.3.2 Node.js

Due to the shift of application development from desktop programs to web (browsers) and mobile apps, JavaScript usage has increased in the last decade. It is logical that, given this fact, the appearance of new ways to create and deploy JavaScript applications comes to life. Node.js, launched in 2009, is a server-side javascript runtime environment [3], based on Google's runtime implementation denominated as V8, focusing on performance and low memory consumption [46].

[2] http://wiki.ros.org/Services
[3] https://nodejs.org/en/about/

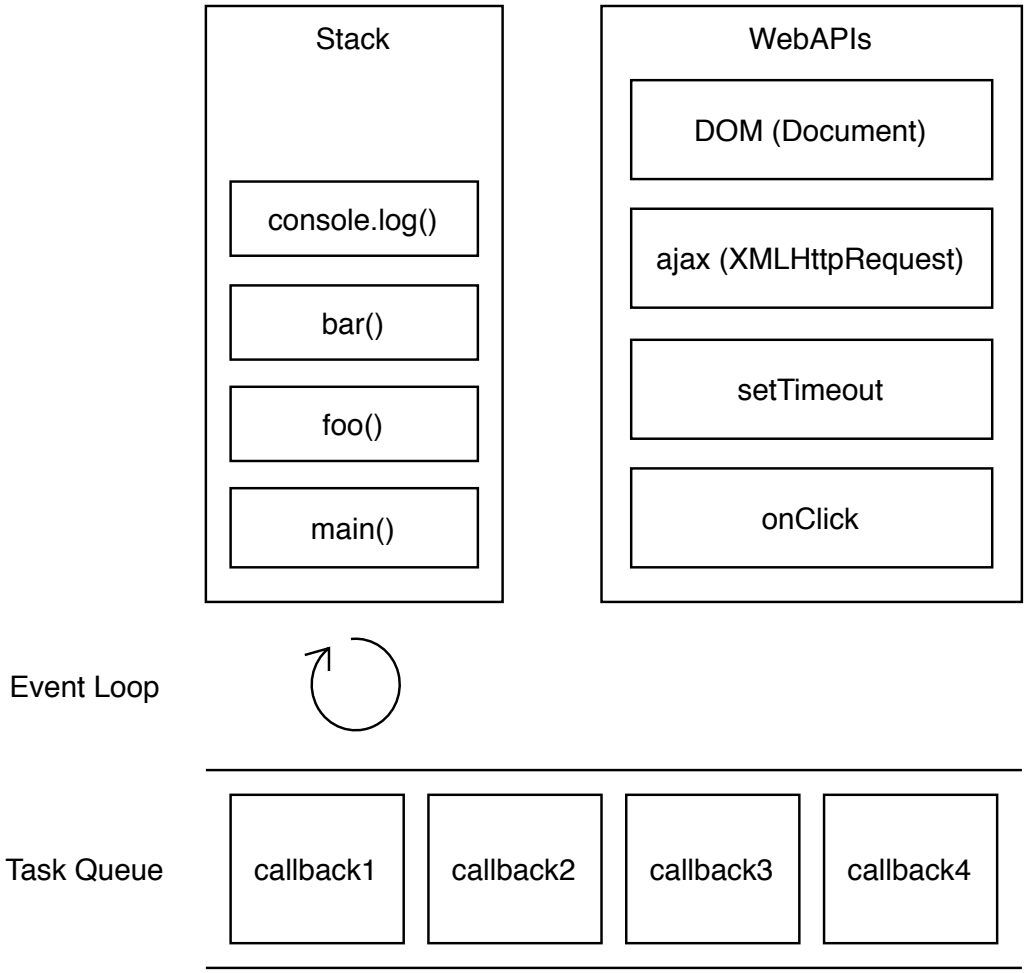

Figure 3.9: Node.js event loop diagram

The JavaScript runtime is comprised of a heap, where memory is allocated for variables and functions, and a stack, where JavaScript functions are queued. Aside from this, Node.js also includes an event loop that periodically checks the stack for queued functions and runs them. A function can do a simple computation of assigning values to variables or it can call a WebAPI like setTimeout or onClick. This WebApis typically include a callback function that should be called in the future and are respectively placed in a task queue. When the event loop sees that there isn't any function left to be run in the stack, it starts to push the callback functions in the task queue to the stack. This process can be seen graphically in figure 3.9.

With this system in place, Node.js can achieve a non-blocking single-thread that is capable of answering thousands of requests made to the same server [47] concurrently. This was the main factor that led to the adoption of a Node.js server for this platform.

3.3.3 InfluxDB

InfluxDB is a high-performance data store written specifically for time series data or, in other words, InfluxDB is a Time-series Database (TSDB). InfluxDB provides an SQL-like query language for interacting with data. It allows for high throughput ingest, compression and low latency querying, being able to handle millions of data points per second [48]. Dealing with that much data over a long period can lead to storage concerns, to counteract this InfluxDB automatically compacts data to minimize storage space. Since this is a time-driven database, every table has a time column that registers discrete timestamps and is the corresponding primary-key of each entry. The remaining attributes or columns of each table are either called *fields* or *tags*. The difference between these fields and tags is that the latter are indexed, meaning that the queries performed on tags are faster when compared to queries made to fields [49]. Another concept introduced by InfluxDB is a *measurement*, this groups fields and tags by a given name, much like a table in SQL, so it's possible to get all values for a pre-determined set.

3.4 Communication Protocols

3.4.1 WebSocket

Since the platform will handle low latency information coming from the UAVs, we needed to address this problem. A common technique used in many web-platforms is long-polling, where the client constantly checks the server for new data. Until a decade ago, this way of getting low latency data was as good as it gets. However, as more client requests reached the server, a predictable problem became more apparent. The latency between client and server increased since each connection is kept open for as long as possible. When the connection times out, the client is notified and makes a new request, repeating the process. Thus the idea of a new protocol, identified as WebSocket, was born. As I. Fette and A. Melnikov described [50], WebSocket allows a long-held, bi-directional and full-duplex TCP socket connection to establish between client and server. This procedure starts with the client sending an HTTP request to the server asking to connect to a WebSocket. This process is known as a WebSocket handshake. After the server accepts this request, it replaces the HTTP connection by a WebSocket connection. The server maintains this last connection for each client, using it to push data as close to real-time as possible to the client. From this point on, a client can publish a message to a specific channel identified by a topic, and each client that is subscribed to that topic will receive the message.

3.5 Related Work

During the initial phase of this project, research was conducted to find similar platforms that have some of the same functionalities provided by the proposed solution. There

was found only one holistic platform that had similar characteristics, the FlytOS[4], and although it presented the same idea of connecting UAVs to the cloud, with the possibility to gather data and actuate over these UAVs, one characteristic that it lacks in comparison to the one proposed in this project is the possibility of mission planning. However, it is possible to conduct mission planning using another tool called QGroundControl[5] although this is not considered to be a UAV management platform since it cannot store data sent by the UAV and has the limitation of using radio telemetry for communication. This comparison is also resumed in table 3.3.

Functionalities	Our Platform	FlytOS	QGroundControl
Online Asset Management	Yes	Yes	No
Online Visualization	Yes	Yes	No
Online Control	Yes	Yes	No
Mission Planning	Yes	No	Yes

Table 3.3: Similar platforms comparison

3.5.1 FlytOS

The decision for not using the FlytOS solution in the AFarCloud project was based on three main reasons, the need for a high-level mission planning tool, the lack of support for the custom made UAV HEIFU and absence of a communication protocol for Wireless Sensor Network.

1. The main factor that led this platform to not be considered as an option in the first place is the lack of a complex mission planning functionality that allows the creation of a UAV mission with the possibility to specify particular tasks such as vine map detection using RGB and Heatmaps.

2. Although this platform supports many DJI[6] UAVs it didn't provide full support for the UAV solution that was made for the AFarCloud project (HEIFU).

3. Another component of the AFarCloud project is the need for collecting and analyzing data gathered from a Wireless Sensor Network, in spite of being the focal point of this work, it is still a component that was taken into consideration when weighing the pros and cons.

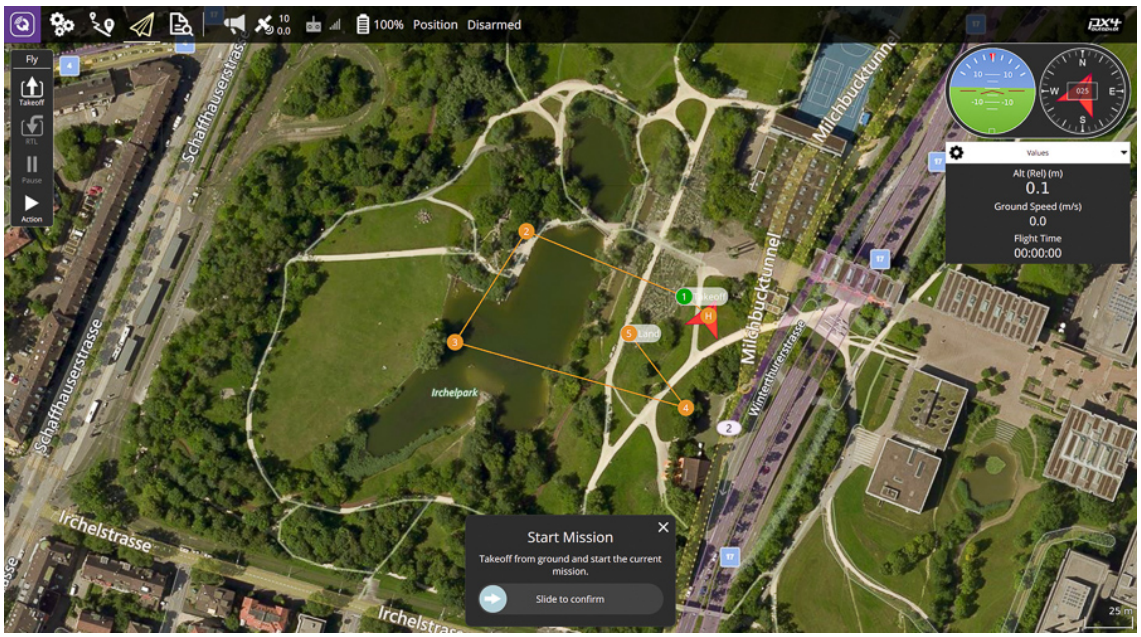

Figure 3.10: QGroundControl Interface

3.5.2 QGroundControl

Although this platform provides a tool (Fig. 3.10) capable of attaining the complexity needed for mission planning in a use case of the AFarCloud project, it doesn't go much further than that. This platform only allows us to connect a single UAV through a Pixhawk in order to receive and send data related to the real-time flight, but also there isn't a possibility to collect this data and further analyzing it in a cloud server. Another point in which this platform falls short in comparison with the others is the incapability of managing a set of connected UAVs.

[4]https://flytbase.com/flytos/
[5]http://qgroundcontrol.com/
[6]https://www.dji.com/

Chapter 4

Approach

This fourth chapter has two parts, the first one addresses the prototype that was developed by following the guidelines of the AFarCloud project, and it is composed of just user interfaces. The second part is a more detailed report on the development of the platform, where it is discussed the selected architecture for the system and its components. Furthermore, this description is divided into the Backend, the Middleware, and the Frontend, so we will be able to get into more detail about each part of the system.

4.1 First Prototype

The prototype for the platform was developed by following the guidelines of the AFarCloud project for a mobile web application. This application had the functional requirements 1 through 7 related to UAVs, and 8 through 15 regarding the missions. These were two of the main building blocks for this prototype since they were the focal point of the mobile web application. Aside from this, the AFarCloud consortium also asked for other functionalities. They were: integration with a real-time weather API, list and monitoring of placed sensors, and reports page creator. Since the purpose of this prototype was to integrate with an already implemented middleware, it was only possible to ensure the fulfillment of the usability non-functional requirements.

The first figure (Fig. 4.1) presents the main page for the vehicles. This page has a satellite world map centered on the location of a specified farm and shows all the available assets. The currently connected vehicles have a green icon, while the idle ones have a blue icon. The draggable side panel lists all the UAVs and UGVs and their corresponding names. When a user expands the side menu, more information appears (Fig. 4.2). Here the user can confer the current battery of each asset and which sensor or actuator is attached to it.

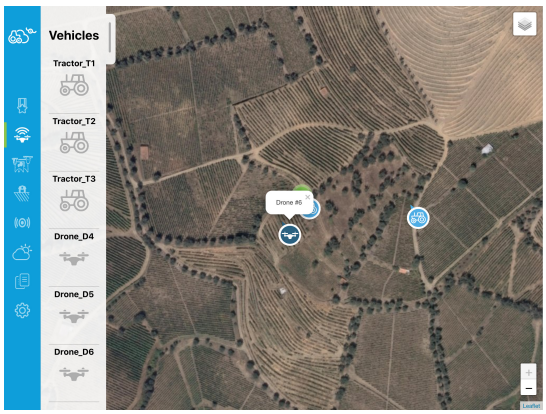

Figure 4.1: Vehicles Page

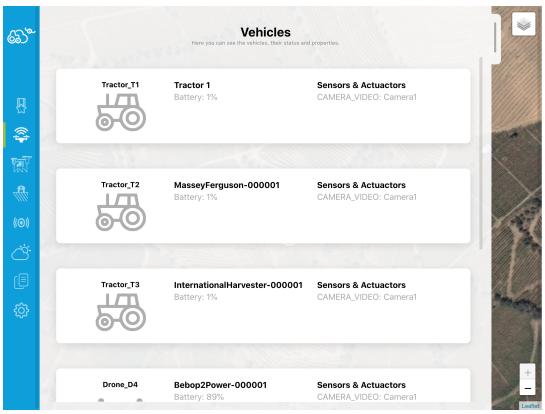

Figure 4.2: Vehicles Menu Expanded

The second set of screenshots pertains to the mission functional requirements. Here we have the same satellite map with some changes like the vehicles shown are only the ones that are assigned to a currently active mission. Moreover, we have a geographic presentation of every mission with a drawn polygon on the map (Fig. 4.3). Next to it, we have Fig. 4.4 with the draggable side panel expanded, showing relevant information for each mission. The data can be the number of tasks and what each one has to accomplish, the current status of the mission, and a start and end time for each task. The user can also check which vehicle is assigned to each mission. Furthermore, the user can stop an on-going mission.

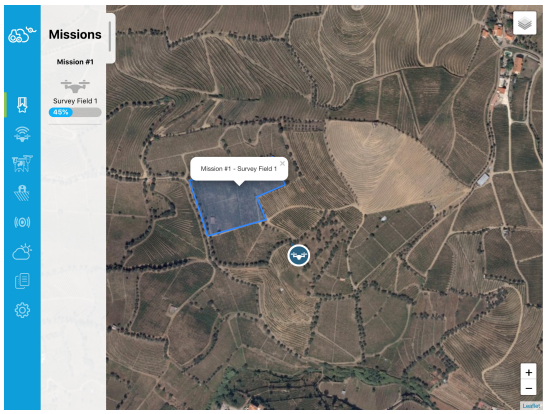

Figure 4.3: Missions Page

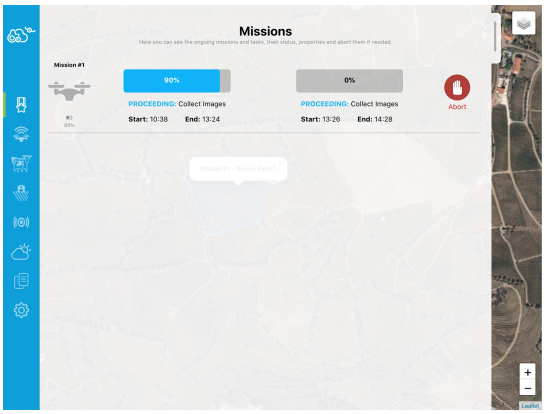

Figure 4.4: Missions Menu Expanded

The next set of screenshots concerns the placed sensors and how the user can monitor and analyze their data. On Fig. 4.5, we can see the sensors page with the satellite map and the draggable side panel. Each placed sensor is present on the map with an icon on their exact coordinates. By clicking on this icon, the user can see which sensor this icon refers to and the last registered value. By dragging the side panel (Fig. 4.6), an extensive list of all sensors appears where a user can see their information in more detail. This information includes the type of the sensor, the last recorded value, the recorded time, and value domain. Contrary to the other expanded side panels, this one has an additional

feature. The user can click on an entry of the sensors list, and the interface will show a graph. This graph displays the values registered by that sensor in a given period (Fig. 4.7). With this, the user can easily create a timeline for each area with different characteristics in mind, like soil humidity, temperature, air pressure, among others.

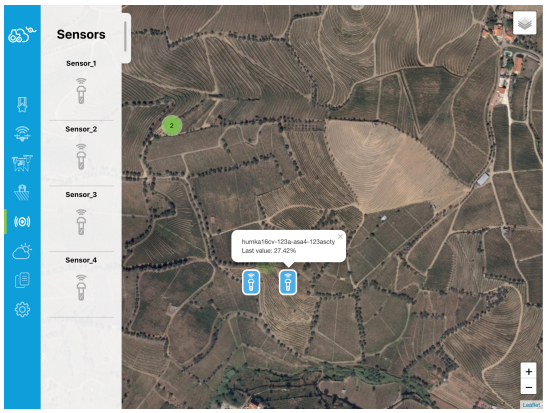

Figure 4.5: Sensors Page

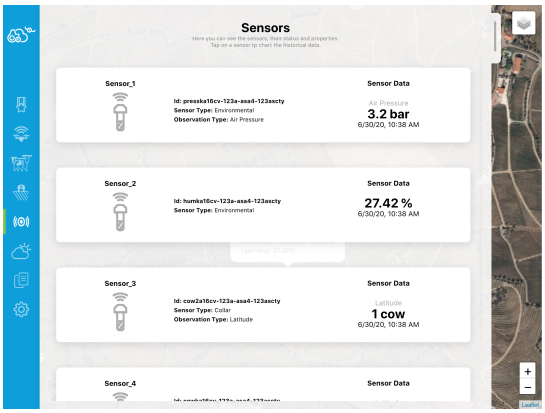

Figure 4.6: Sensors Menu Expanded

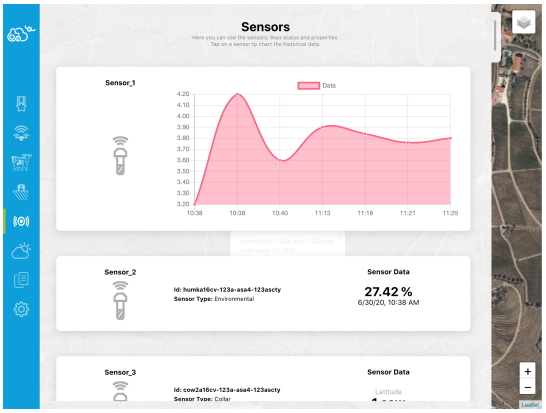

Figure 4.7: Sensors Menu Expanded Graph

The reports page has a unique functionality of generating reports for different purposes. Although this page does not utilize the visual capacities of the satellite map it was decided, in conjunction with the AFarCloud consortium, to use the same design to make it more visually consistent for the user (Fig. 4.8). The side panel lists all the possible reports the user can generate, such as the carbon footprint report and a total mix ration. The carbon footprint uses the middleware to calculate the carbon footprint of milk production, giving insight to the farmer of how much greenhouse gas the milk production is emitting. (Fig 4.9). To generate the report, the user must input some values on each specific field and choose the year for which the carbon footprint report is being generated.

The last screenshot and part of the prototype description concern the weather page. This page utilizes a real-time API that produces weather data for client applications. For this specific case, we used three different endpoints of this API, the endpoint regarding the current weather, the endpoint that gives the temperature at separate times of the day, and the seven-day forecast endpoint. In Fig. 4.10, we can distinctively see each one of

Figure 4.8: Reports Page

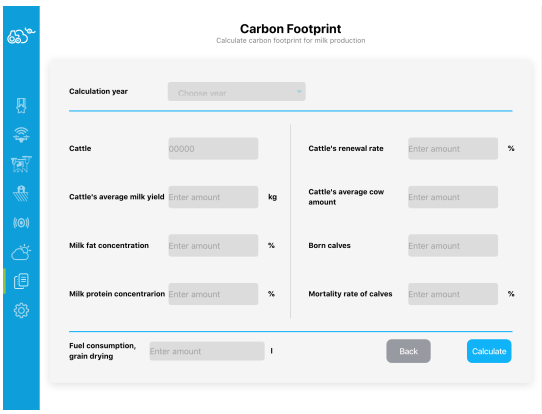

Figure 4.9: Carbon Footprint Page

these endpoints displayed. The current weather section has some additional information like humidity, wind speed, visibility, and UV index. For the hourly and seven-day forecast sections, it's only displayed the temperature value and the expected weather conditions.

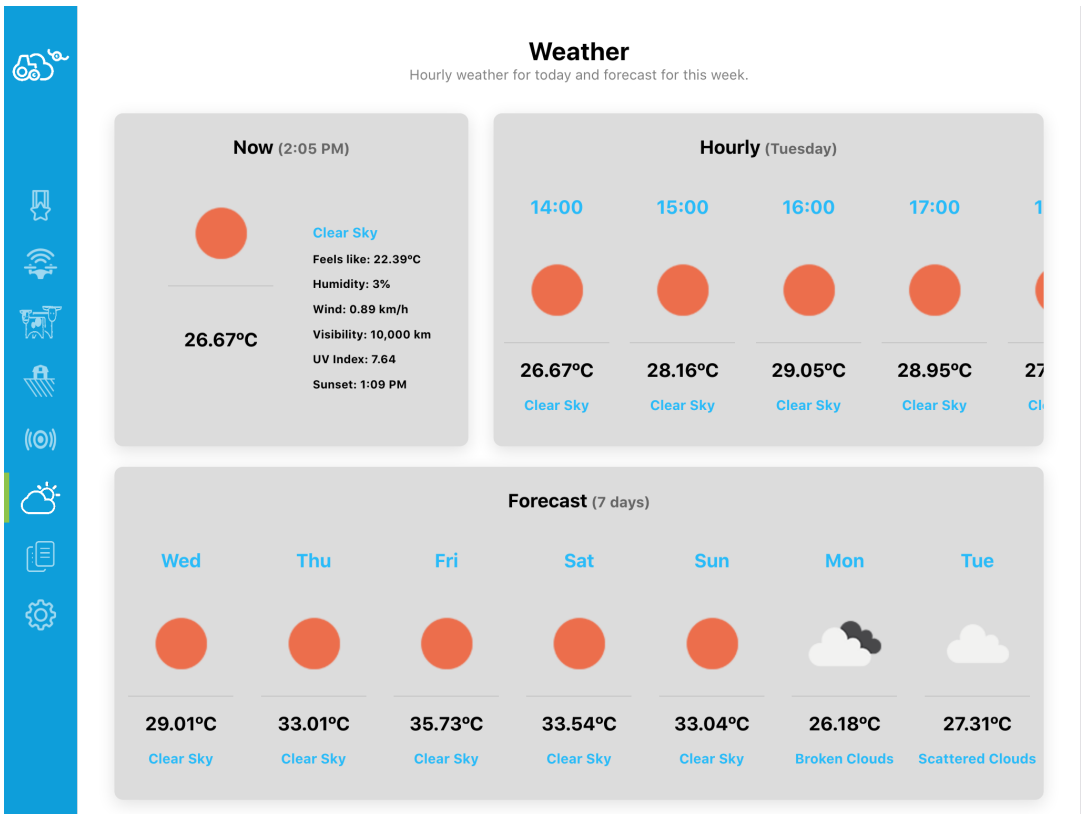

Figure 4.10: Weather Page

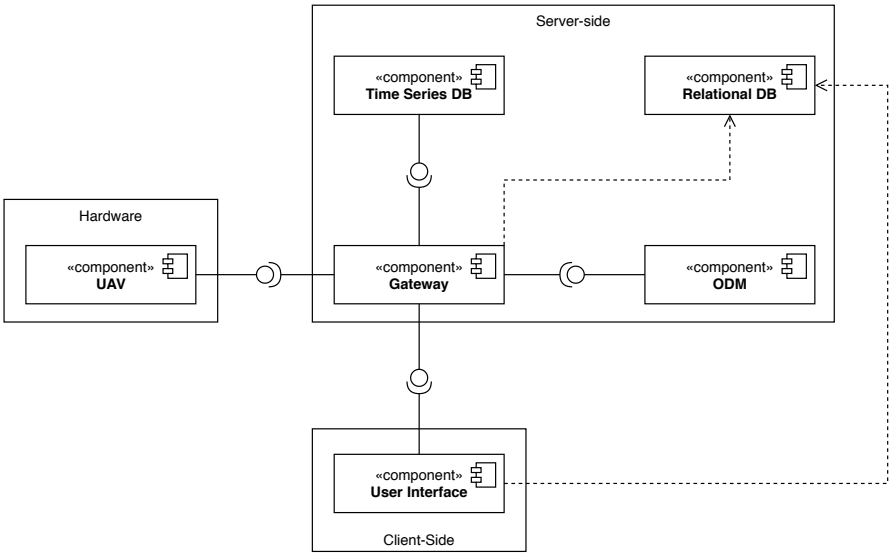

Figure 4.11: Platform Component Diagram

4.2 Architecture

In contrast with the prototype, the platform encompasses all the functional and non-functional requirements, meaning that the platform will have a more defined set of functionalities, each with some degree of complexity. Given this fact, there was a need for a more robust architecture that would be able to tackle all the requested functionalities. The following component diagram (Fig. 4.11) presents a visual representation of the defined architecture.

On the left side, we have the hardware section that is fully represented by the UAV component. Recalling that the platform focal point is its integration in a CPS, the hardware corresponds directly to the physical part on Cyber-Physical Systems. This UAV was custom-made for the platform by another division of the team. This component can communicate with the whole system by connecting to the gateway and exchange information through it.

The gateway is one of the components that make up the server-side section, and it is responsible for enabling the communication between all the parts of the system. To communicate with each other all components must exchange information through the gateway, regardless if they are inside or outside the same section. However, the method by which a component communicates can differ between Rest endpoints or Websockets, depending on the need for low latency communication. For example, the communication method needed to transmit information from the UAV to the rest of the components should use Websockets. While the data that the relational database exchanges can use a Rest endpoint.

The relational database is where the system stores all the information regarding the users and artifacts needed to manage the platform. But also, where data concerning the

UAV such as model, hashed token, and ID, are also saved. Here is where we found the first problem during the planning phase because the UAV produces a reasonable amount of data that needs to be registered. Since relational databases make use of a well-defined schema to maintain its integrity, there are some constraints there need to be addressed. The registration of large quantities of data in a relational database can be costly, and because of the schema, there are restrictions on the type of data. However, since we also needed to classify this information with a timestamp, it was an obvious choice to use a time-series database. This kind of database can handle a large number of writes to it, and it has the added benefit of each table occurring to be organized by timestamp. The focal point of the time-series database is to register various measures of the UAV, such as GPS coordinates, velocities, among others. With these metrics, it is possible for us in the future to analyze these data and check where something went wrong.

The last component present in the server-side abstraction is the OpenDroneMaps, or most known by the initials ODM. The OpenDroneMaps is an open-source command toolkit for processing UAV imagery, and it can turn 2D images captured by the UAVs into classified point clouds or 3D textured models.

Finally, for the client-side abstraction, we have the user interface component, which is composed of multiple Graphical User Interface (GUI) that are provided to the users by the server. We decided to develop these interfaces using Angular, supported by the research presented in previous chapters. Each functional requirement has its independent graphical interface, to achieve not only a separation of concerns but also more specificity in each one. Just like the other components, the user interfaces can require the needed information from the gateway. However, the functioning of the interfaces is dependent on the authentication service that commands the user to input a password that matches a saved hash on the relational database. The user can make use of the previous method, or it can also use a saved token to have access to the gateway resources.

4.3 Backend

Every application needs a logic tier that can accomplish tasks through computation operations. A developer can write these functions in numerous different programming languages, being the responsibility of the developer to choose the most adequate one for the project at hand. Nowadays, programmers can choose from multiple languages to write server-side code, such as C, Java, Javascript, Python, or PHP, to name just a few. Each one has its advantages and disadvantages, meaning that these languages can be leverage differently whatever the goal might be.

4.3.1 Server

For this specific case, there was a need for fast development, scalability, and high throughput of requests and responses. With these points in mind, the decision to use NodeJS

was a natural one. As was mentioned beforehand, the foundation of NodeJS sits on the fact that it can handle many connections concurrently while keeping one single thread. Because of this, the first limitation when using NodeJS is the execution of high computational tasks that can lead to a block on the event loop, preventing the runtime from answering anymore more requests while it hasn't finished. However, if the developers consider this restriction during the planning phase, they can mitigate it while accomplishing a channel of high throughput communication. Contrary to the way Java or C normally handles each connection by assigning it to a new thread from beginning to end, possibly maxing out the server RAM, NodeJS will use callback functions on its single thread (event loop). While in the first case, if the connection requests for database operations, the thread will be blocked until it finishes that operation. In the second case, the event loop will fire another callback function for the database operation and go immediately to sleep, waiting for a new request from the client. Furthermore, this process makes this type of server code more scalable by leaving a big chunk of RAM available for more instances of the server to be put to work.

The fact that a developer has to write NodeJs server-side code in Javascript is also somewhat a disadvantage. Javascript is an interpreted language implying that it doesn't need a compiler to translate the human-readable code into a computer-readable (machine) code. This means that interpreted languages are faster in the development phase because they can jump the "build"step, and the machine can execute the code as it is. Contrarily, compiled languages compile the source code directly into machine code that the processor can execute being faster and more efficient than interpreted languages at run time. However, this gap in run time performance has been shrinking with improvements in technology. The point here is that developers using interpreted languages like Javascript can only find errors at run time and cannot utilize tools such as types, generics, and interfaces. Given this, we decided to utilize a NodeJS framework that could mitigate these issues. For this, we choose to use NestJS. Mainly NestJS is a NodeJS backend framework that employs Typescript to introduce strong typing rules as well as some practices of Object Oriented Programming (OOP).

4.3.2 Database

The database topic has two sub-topics, one for each of the different types of databases the platform needed. Firstly, there was a need for a system to store data persistently, which ensures the consistency of configuration data regarding users, drones, and other objects. For this purpose, it was decided the design and develop of a relational database recurring to a SQL schema to enforce certain restraints and conditions. The second subtopic discussed in this section is the reason why it was necessary to introduce a time-series database in this architecture and how the integration went.

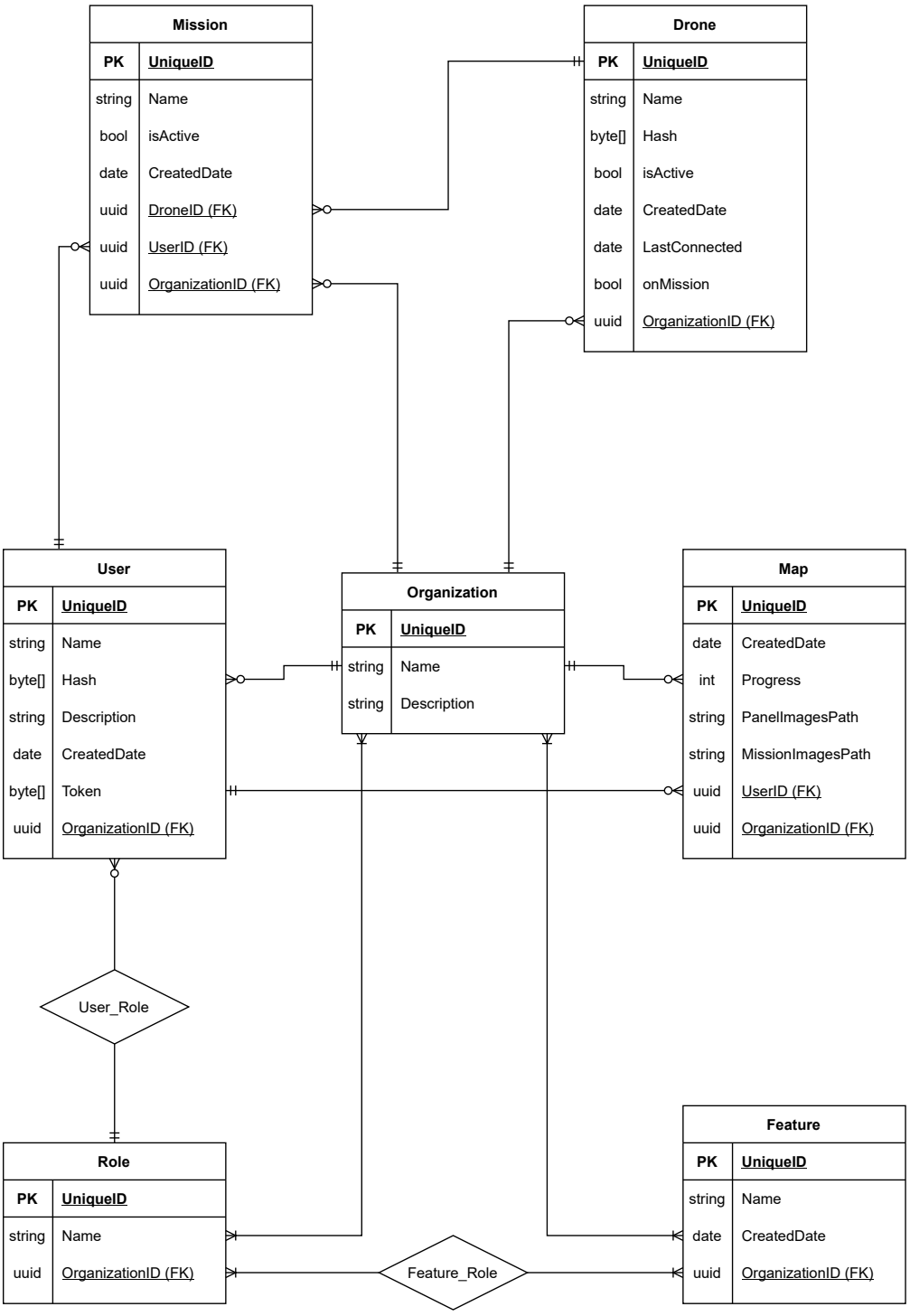

Figure 4.12: Entity-Relation Diagram for the Relational Database

4.3.2.1 Relational

The diagram in Fig. 4.12 presents the SQL schema for the relational database. By analyzing it, we can see that every entity is directly related to the **organization** entity by a foreign key. The logic behind these relationships is that every organization needs to have its roles, features, users, and assets, separated from other organizations. Given that our platform first users would-be partners and farms associated with the AFarCloud project, there was needed a way to have a clear separation of permissions and assets. When a platform manager introduces a new organization to the platform, a set of default **roles** is automatically added in parallel. This set includes the owner, the operations manager, the administrative, and worker roles. Concurrently, the platform also adds the **feature** objects related to each one of these roles to the system. These features correspond to each role's permission, and it is the logic behind the management of what each user can do in the platform. It was chosen that the representation of this tuple, **role-feature**, would be done in a relationship table with the primary keys of each table. There is anther place where this relationship table was used to express the relation between two concepts, it was used for **user-role**, to represent the relationship between each user and its corresponding role. Aside from the organization and the role, the **user** table also stores a hash of the user's password. Moreover, a token related to the last session of that user is also registered here. This token has an expiration policy attached to it, and the system will revoke it after this timestamp.

The last three entities of the entity-relation diagram correspond to the asset objects of the platform they are the Maps, Drones, and Missions.

- The **map** entity registers the date of its creation and the current processing progress. This progress can indicate if there are still missing the upload of panel or mission images. The ODM service then stitches these images together and enhances whichever features are present on them. Each map is related to a user and organization by their corresponding foreign keys.

- The author designed the **drone** entity with the requirements of the UAVs in mind. Each drone object will have a universally unique identifier (UUID), the same as all other entities, and the date when a drone provider added it to the system. Moreover, a boolean field will indicate whether the UAV is currently active or not, and another one if it is on a Mission. Also, a timestamp field registers the last time that UAV was online on the platform. Lastly, this table also stores the hash of an authentication token for each UAV; without this token, a UAV cannot do any action on the platform.

- An administrative can register a **mission** on the system by giving it a name that briefly describes the mission, and assign a UAV to it. This assignment will translate into the creation of a foreign key to the drone table. Aside from this drone table, the organization and the user table will also have foreign keys to an entry of the

mission table. The system also stores the date of creation for each mission, and a boolean value will indicate if the mission is currently on-going.

4.3.2.2 Time-series

As mentioned before in this document, the purpose of the Time-series Database (TSDB) is to store n-tuples of data in "pairs"of timestamps and values. Briefly, this means that each entry on this database will have a timestamp corresponding to the time the subject inputs one or multiple values. This method of persistently storing data is more efficient than using a relational database, for example, because it can leverage compression algorithms to reduce the size of data tables.

The first step in developing a TSDB is the design phase, like with a relational database, it is necessary to define a **schema**. These are a set of fundamental rules every subject has to comply with when adding new data entries. Specifically, for the InfluxDB, this process goes by assigning fields and tags to each table. The first concept that needs clarification is the **field** that combines a string key and a value or collection of values that can be strings, floats, integers, or booleans. Every field entry has a timestamp associated with it. However, this data structure is not indexed, meaning that queries over fields will scan all points that match a given time interval. That can lead to slow and not performant searches on the time-series database. To achieve faster queries, the developer must make use of the **tag** data structure since this one is indexed. They are quite similar to fields except in the value section, where there is only permitted the use of strings. Although these are the basic building blocks for the TSDB InfluxDB, there is one last concept that should be defined. The **measurement** serves as a collection of fields, tags, and their corresponding timestamps. This structure is mainly an organization abstraction, and it can be perceived as a table in a relational database. Every measurement must have a name comprised of a string, and the developer can then apply different retention policies to each measurement. This retention policy can specify how long the database will keep the data under a measurement, or how many copies of that data must the database keep for replication purposes.

The applications of the TSDB on the use case regarding UAVs are the following. First, there was the necessity to record the GPS coordinates of the drones while they were flying. These flights could be manually controlled or an autonomous mission. The UAV used during development produces these values with a 3 Hz frequency, meaning that we get three discrete values every second. The drone publishes these data to the respective topics, and our backend then registers these values on the InfluxDB. So for this first application, we created a measurement designated *GPS* with a field-set of latitude, longitude, and altitude. The best choice to represent these values in the field was the float representation. Then, to map these GPS coordinates with the corresponding UAV we added a tag with the string regarding the droneID. Since tags are indexed, as we mentioned before, the queries to return the GPS data from a specific drone would be very performant. The

second application concerns the velocities of the UAV during the same described flights. Here we created a measurement titled *Vel*, which is short for velocity. This table has three float fields analogous to the *GPS* table, corresponding to the drone velocity in each axis of the tri-dimensional space. These fields are the x velocity (vel_x), which corresponds to forward and backward movement, the y velocity (vel_y), which corresponds to the left and right movement, and the z velocity (vel_z), which corresponds to upward and downward movement. Similar to the *GPS* table, the *Vel* measurement also has a tag which registers the source droneID.

4.4 Middleware

Many developers use the concept of middleware across multiple fields of software development. Middleware designates a mechanism that enables the communication and control of data between two software components. This tool became common to use when there was a great need to adapt legacy systems to work with new software and frameworks. Nowadays, one of the middleware's purposes is in distributed systems, with the introduction of multiple development tools and frameworks for developing applications and the need to deploy these applications in various infrastructures. All these factors contribute to increasing the complexity of software development, and there was needed a middle layer that could facilitate the connection between these different tools, being them from the client or server-side. The first use case in our platform that needed a middleware solution was the platform's API.

4.4.1 REST API Endpoints

An Application Programming Interface (API) is a software contract that defines a communication protocol between two agents, typically a client and a server, that enforces rules between the requests and responses sent from and to each other. This example falls in the category of web development and describes a commonly used type of APIs designated Web APIs. These Web APIs utilize either Hyper Text Transfer Protocol (HTTP) or Hyper Text Transfer Protocol Secure (HTTPS), the latter being a safer extension of the former. Moreover, there is also a need to specify the acceptable structure in which the messages must be formated. The more frequent options here are the Extensible Markup Language (XML) or JavaScript Object Notation (JSON). The final point in this Application Programming Interface contract is the adoption of a communication protocol over HTTP or HTTPS. In the past, the more common choice for the communication protocol was the Simple Object Access Protocol (SOAP), when Web APIs were provided as web services, and Service-Oriented Architecture (SOA) was the prevailing practice. In recent years, this paradigm has been shifting towards the adoption of web resources instead of web services. This is mainly due to a grower demand for the transition from the regular Web 1.0 to Web 2.0. For this standard, the Representational State Transfer (REST) protocol was

born with the introduction of another software design practice called Resource-Oriented Architecture (ROA). Together with HTTP, a developer can specify which kind of method a Uniform Resource Identifier (URI) will depend on, the more familiar ones being the POST, GET, PUT, and DELETE methods.

Given the present complexity in the communication layer with CPSs, the REST protocol over HTTPS was adopted to achieve flexibility and security, respectively. Regarding the message structure, it was decided the use of JSON since both the client and the server were being developed with a javascript framework.

4.4.2 WebSockets

Another central point of this work is the transfer of data, as close to real-time as possible, produce by our CPSs, the UAVs. The computational layer of each UAV operates over Robot Operating System (ROS), meaning each one has access to the topics ROS provides. These topics cover a range of fields over which a UAV can produce data every second or with even greater frequency. These can be GPS coordinates, velocity in different directions, battery percentage, propellers speed, among others. Whenever a drone is connected, it will continuously publish values under these topics, even if these values are currently stable.

For our use case, we used the first three topics described above, and we will use the first one as an example. So for each segment of a GPS coordinate (latitude, longitude, and altitude), the UAV will have access to a correspondent ROS topic (*/location/latitude*, */location/longitude*, and */location/altitude*). At a given frequency, in the HEIFU case is 3Hz, the UAV publishes its current latitude, longitude, and altitude values to these channels. Any trustworthy agent could listen to these values by subscribing to the corresponding topics. That is the first use case for WebSockets on our platform. We configured the connection over WebSocket Secure (WSS) of a separate socket connection for each topic of the UAV. This configuration goes through the process described in chapter 3, section 3.4.1, where the client sends the server an HTTP request asking to connect to the WebSocket. The server responds with an OK response and sends this response over to the client. This process is known as the handshake protocol. After this, a long-held, bi-directional channel is established between the two agents. This socket subscribes to one of the UAV topics, and on-listening to a new message, it will propagate that value over its channel to any connected agent, in this case, that would be the backend server. The controller assigned to this task will then register the values on the time-series database and, through another socket, will disseminate the values to requiring agents. On this end, the requiring agent is the client interface on the user's browser. The diagram in Fig. 4.13 shows an example of the mechanism behind the use of WebSockets on the platform.

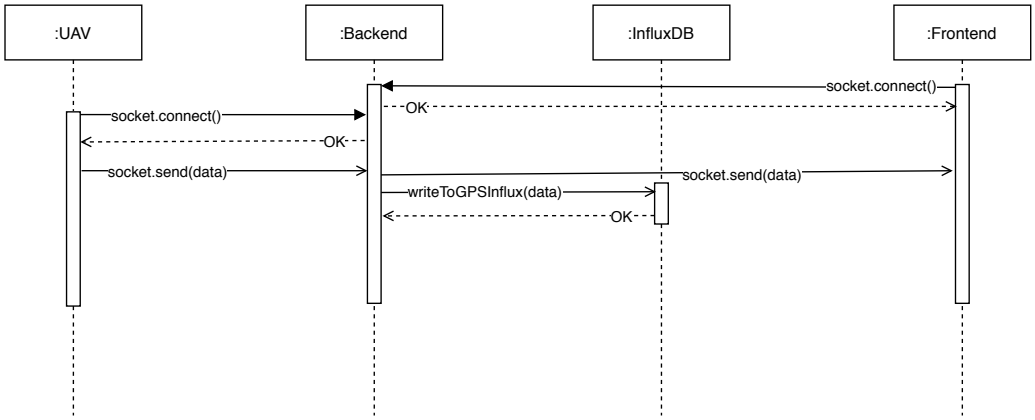

Figure 4.13: Sequence Diagram - WebSocket Data Transfer

4.5 Frontend

The final section in this approach chapter describes the user interfaces developed as the frontend of the platform. These interfaces were implemented using the Angular framework due to multiple facts.

The first practical advantage of implementing a platform with this framework is that the model-view-controller was built-in from its creation. Its component-based architecture brings not only more quality of code but also a separation of concerns. Having components, services, and models separated like classes that one must explicitly import enforces these rules and maintains the overall consistency of the project.

The second fact that lead to the choice of this framework is the entanglement of RxJS in Angular in regards to asynchronous programming [51]. RxJS is a library designed to handle asynchronous data to maintain the responsiveness of a user interface when waiting for new data to arrive. It does this by handling events independently and in parallel with the use of observables. Rather than continuously send requests to the server to check if new data is available (pooling), a component can subscribe to an observable and wait for the server to push some data. Moreover, RxJS offers a set of operators capable of manipulating the data as it arrives, meaning that when data reaches the component is ready to be presented.

Finally, another factor that solidifies our choice of this framework was its performance, derivative from Angular's hierarchical dependency injection. Dependency injection is a design pattern used in software development, first approached in Java programming language. Its main idea is that there should be a dependency injector that is responsible for connecting components to its dependent services, alternatively to the case that each component should explicitly ask for his dependencies [52].

4.5.1 User Interface

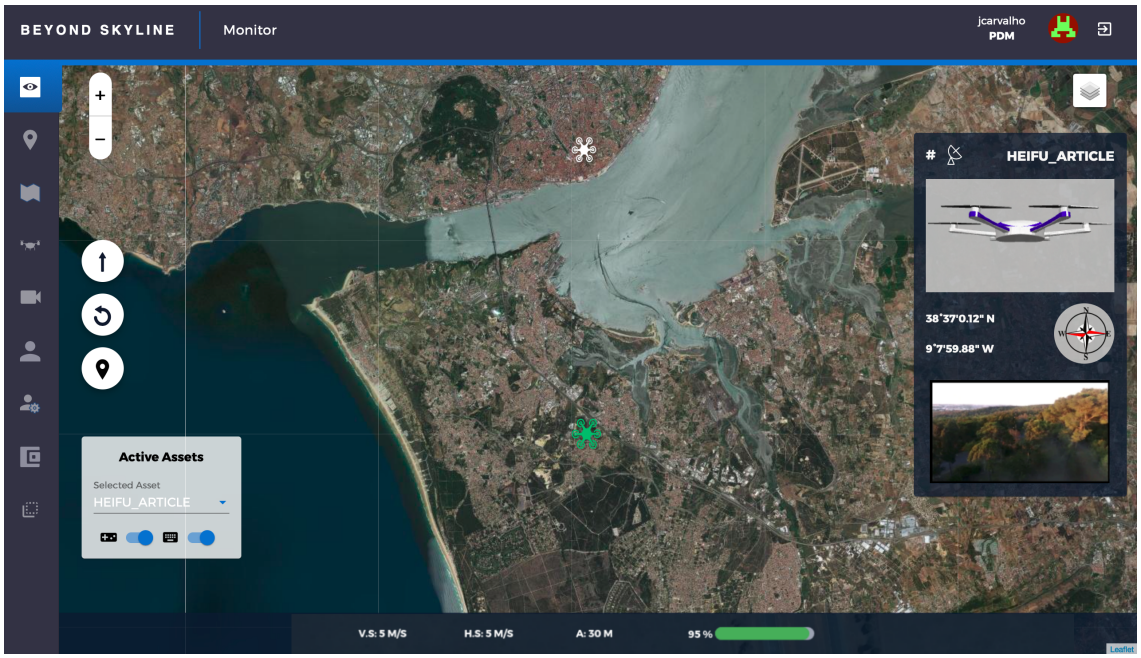

Figure 4.14: UAV Real-time Monitor Interface

Figure 4.14 presents the monitor interface for UAV connected in real-time, being possible to see the video stream of the attached camera, right above are the GPS coordinates and IMU indicator. In this section we also have a small 3D UAV that replicates the angular movement of the real UAV. This angular movement corresponds to the pitch, roll, and yaw. In the bottom bar, it is shown the vertical and horizontal velocities, the current altitude and the remaining battery. On the left side, below the zoom control, there is a section showing three buttons, the take-off, the return-to-launch (RTL), and the setpoint from top to bottom. The setpoint functionality gives the option to the user to set a point in the map with latitude, longitude, and altitude, and the selected UAV will autonomously move to that location. Under this, there is also a card that contains a dropdown of all connected UAVs. Moreover, two slide buttons show the possibility to command the selected UAV using an AC controller or the keyboard.

The screenshot in Fig. 4.15 corresponds to the mission builder user interface. Here a user can start to define an autonomous mission by first choosing the location of take-off. After this, the user can add any number of waypoints and adjust the parameters of each one. These parameters can be the waypoint altitude, the flight speed, the wait time, among others. For the interface to accept the mission as valid, it must also include a land step. The user can complete all these procedures without selecting a UAV to assign the mission to it. However, the user must first choose a UAV if it wants to specify which type of action the UAV must perform on a particular waypoint. Since the interface needs to first validate with the server, which peripherals that UAV has access to know which options it can display to the user. These actions can be, take an aerial photograph, collect

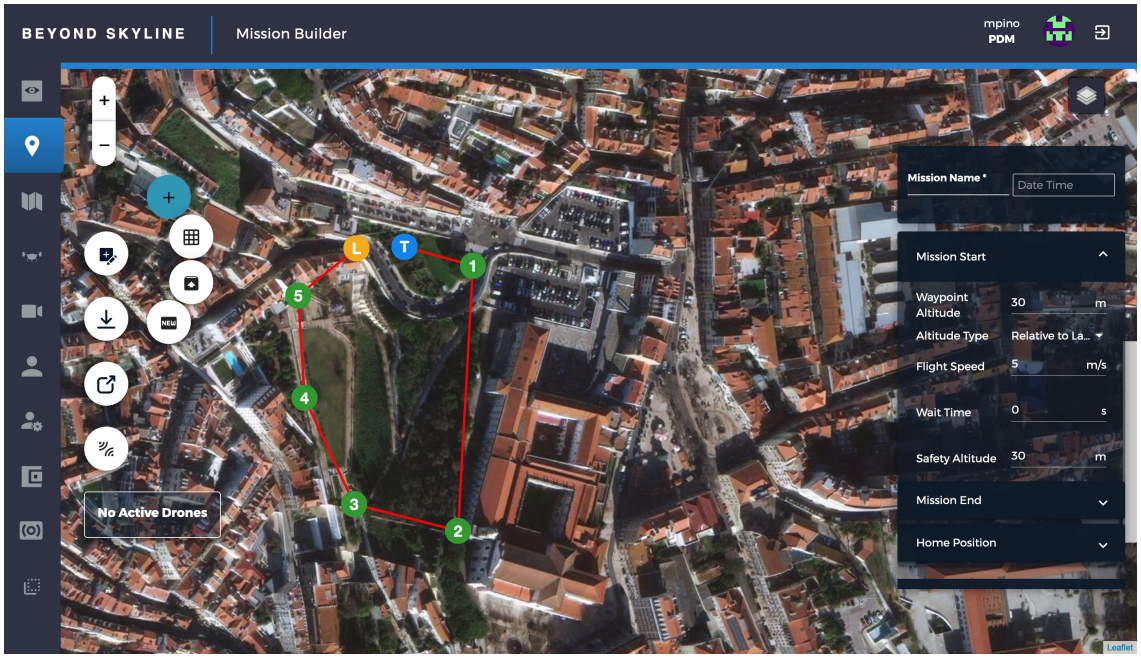

Figure 4.15: Mission Builder Interface

data from the sensor, among others.

Regarding the multispectral maps, Fig. 4.16 displays their index interface, where a user can check the list of already created maps, what is their type, when was created, and in which processing stage it is. Moreover, on the bottom left section of each map card, the user has access to a set of buttons that enables the following functionalities from left to right.

- **Visualize:** This button is only available to the user when the map has finished processing and is in the Complete stage. When a user clicks on this icon, the system redirects the user to the interface described in Fig. 4.17.
- **Download:** The option to download the finished processed map is also only available to the user when it is complete. With this, the user can download the maps resulted from the aerial photographs taken by the UAV and processed by the ODM with every distinct spectrum.
- **Edit:** Here the user has the option to edit any details regarding the resources needed to process a multispectral map. These aspects can be the upload of images and the selection of which features to process.
- **Cancel/Delete:** Finally, if the map is currently processing and by any means, the user needs to stop the processing, it can use the cancel button. However, if the map is already complete, this button switches to become a delete button.

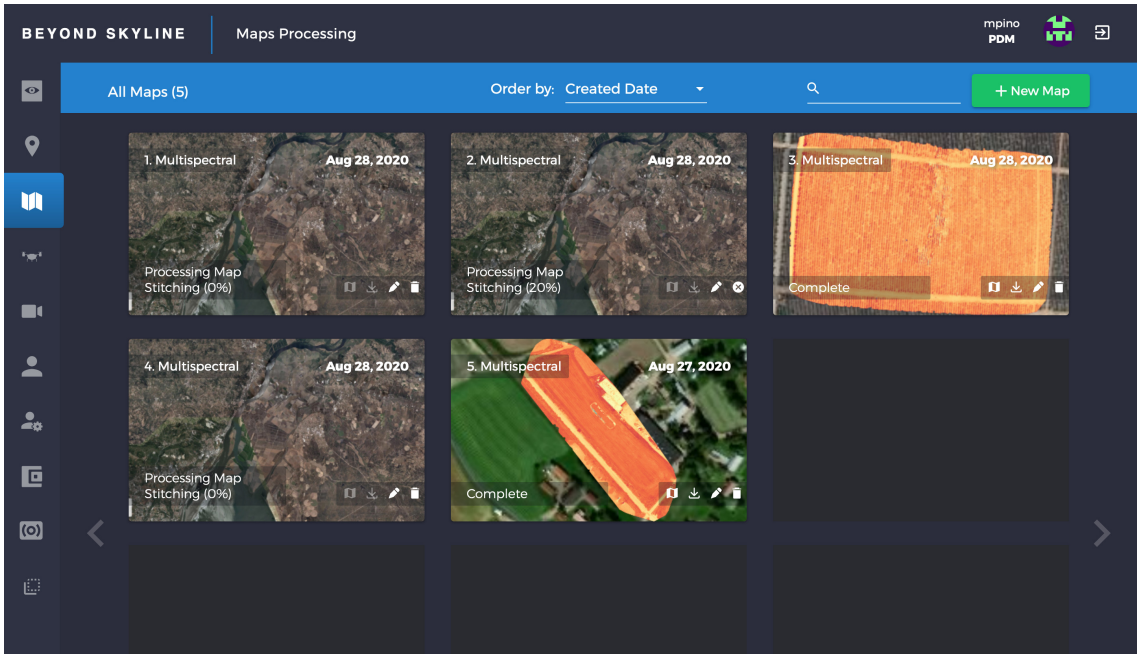

Figure 4.16: Map Indexes Interface

The 2D map indexes interface, in Figure 4.17, presents a timeline on the left side with the list of all maps created nearby, chronologically ordered. It is possible to select more than one map to compare various indexes between different maps. The right side displays a set of sliders capable of changing the opacity of each index. When selecting one map the user can toggle the 3D view and access the interface shown in Figure I.11. In the 3D interface, the user is present with a collection of points forming a point cloud that was collected with a camera placed in the UAV that surveyed the area.

Figure 4.17: 2D Map Visualization Interface

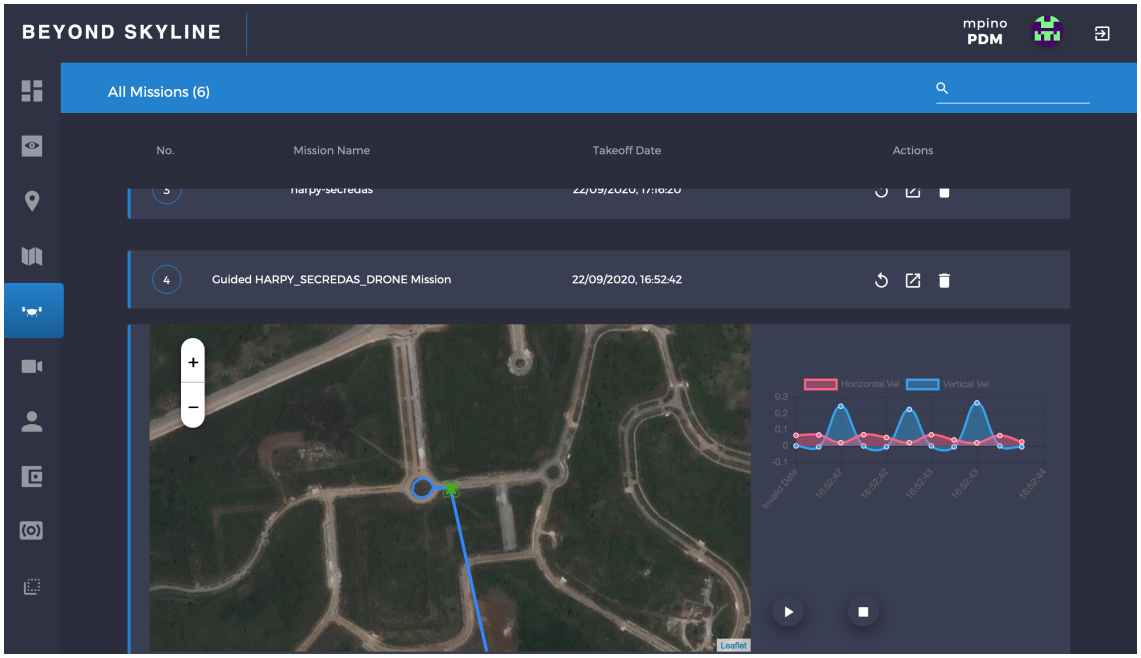

Figure 4.18: GUI for Mission Replay

Finally, Figure 4.18 presents the replay flight mission GUI, where for each UAV, a list displays all completed flights, and the user can replay each one. When the user selects a flight, the platform opens a satellite map with a marker showing the location of the drone at each timestamp. Moreover, a line is displayed representing the path the UAV took over the duration of the flight. On the right side of this map, the platform presents the play, pause, and stop buttons for each respective action. Furthermore, the team decided to add a graph to show the user the horizontal and vertical velocities of the UAV during the flight.

Chapter 5

Validation and Critical Review

The fifth chapter summarizes all the tests, results, and scientific articles that were performed regarding the platform. First, there is given an introduction to the environment used on these tests. After this, there are described the different use cases that were defined to use as validation. Furthermore, the results from these tests are also displayed and discussed in this chapter. Finally, there are descriptions of the two articles published that present results worth mentioning since they also validate a part of the platform.

5.1 Production Environment

The last task regarding the development of this platform was the implementation of a production environment. This environment is necessary to enable the planning and execution of tests and demonstrations. Although the development environment is currently working on Kubernetes, we decided to implement the production environment on a single server with the tool Docker-compose. Due to this, every component of the platform is inside a container, specifically built for each. A YAML file was produced that corresponded to the specifications for each docker and configured all the pipelines necessary to maintain the well-functioning of the environment.

The aftermost tests were conducted in conjunction with a team in the production environment. For these tests, it was selected various metrics to validate the functioning of the platform. But before these metrics are further detailed, to make sense of these metrics, there is necessary to describe the specifications of the server. The specifications are as follow:

The description of these specifications is relevant because the metrics we choose to register were: percentage of CPU usage, percentage of RAM usage, data input in GB, and data output in GB. Among other fields, these are some of the metrics possible to attain

Name	Value
Architecture	x86_64
CPU Model	AMD EPYC 7502P 32-Core Processor
Number of CPUs	64
Threads per Core	2
CPU (MHz)	1497.610
CPU max (MHz)	2500.000
CPU min (MHz)	1500.000
RAM (GB)	248

Table 5.1: Production Server Specifications

from the bash command "docker-stats". However, one peculiarity of the docker stats is that it gives the percentage of CPU usage relatively to one CPU core. Therefore, this amount can be higher than 100%, meaning it is currently using two or more CPU cores. The default behavior of containers under a docker-compose architecture is to utilize as much of the machine resources as needed. If the developer doesn't delimit restraints over these resources, this is the default behavior.

To collect and register this set of metrics, it was necessary to write a script in Python that calls "docker stats"and print the formatted output to a CSV file. Later, this output was cleaned and analyzed, producing a dataset that is graphically displayed in the next sections.

5.2 Case Studies

This section presents the decided use cases used to validate the developed platform. For each one, the server was running the testing script for the entire duration of the experiment. First, a brief introduction to each use case is presented, followed by the resulting graph and further explanation of the values.

5.2.1 Case Study 1 - 1 UAV (Simulation) with 4 Users

The first experiment consisted of connecting a simulation of one UAV to the production environment of the platform. To accomplish this, a team member must first register this UAV on the platform and get a config file. The platform generates this config file that contains a token necessary for the UAV to connect to the backend sockets. After this, the team member started the UAV simulation on Gazebo after having uploaded the config file to the UAV storage. In this phase, the UAV will already appear online on the platform, and it is possible to select on the monitor page and see it's battery, velocities, location, and camera stream. It was at this point that the script was initiated, and it began registering the metrics regarding the docker containers of each component of the platform. After this step, four users logged in to the web application and navigated to the monitor page.

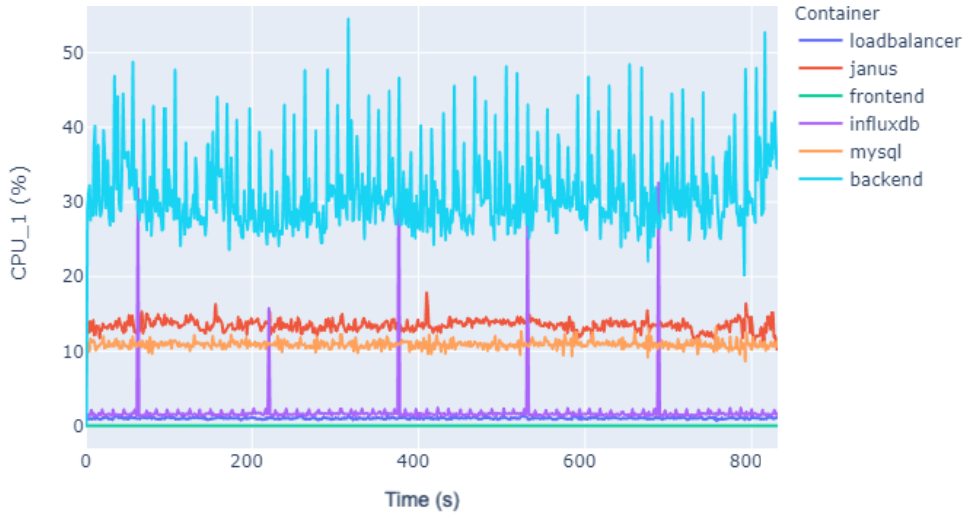

Figure 5.1: Case Study 1 - Percentage of CPU usage

Here they all selected this UAV and got access to the video stream from its camera. At this phase, it was commanded the UAV to move in a simple route while we watched the video stream, and the data showed on the monitor page. The duration of this test was sensibly more than 10 minutes, where the connection to the UAV or its video stream was not lost a single time.

The script registered the collected data to a CSV file that was later cleaned. It was necessary to use Python to accomplish this task, where most of the work was to remove unwanted metrics and adjust the units for the relevant ones. The graph in Fig. 5.1 shows the percentage of CPU usage of each component during the test. The containers shown in this graph are the loadbalancer, the frontend, the backend, the InfluxDB, and the MySQL. It is relevant to add the Webrtc Janus container to the following graphs giving that it was the component in charge of managing the video streams for the platform. We can understand from the figure that the container requesting most of the CPU was the backend, hitting a little more than 50% for a single core. As mentioned before, the percentage of CPU usage concerns one CPU core. The next two containers that used more of the CPU were the Janus and the MySQL, this is because the first was responsible for serving the video streams to the users, and the second for registering information regarding the logs of the platform.

Regarding the percentage of RAM usage, Fig. 5.2, we could conclude that none of the containers required much of the RAM to accomplish their work. The component with the highest registered value was MySQL, although this value didn't exceed 1 percent.

The next two figures represent the total amount of data, in Megabytes (MB), that were sent from and to the server during the time of the experiment. In Fig. 5.3, we can observe the container that had the most data sent to it was the Webrtc Janus, rounding the 520 MB, followed by the MySQL, and the backend. This is understandable given that the Janus works as a mid-point between the UAV and the frontend. All the data regarding the

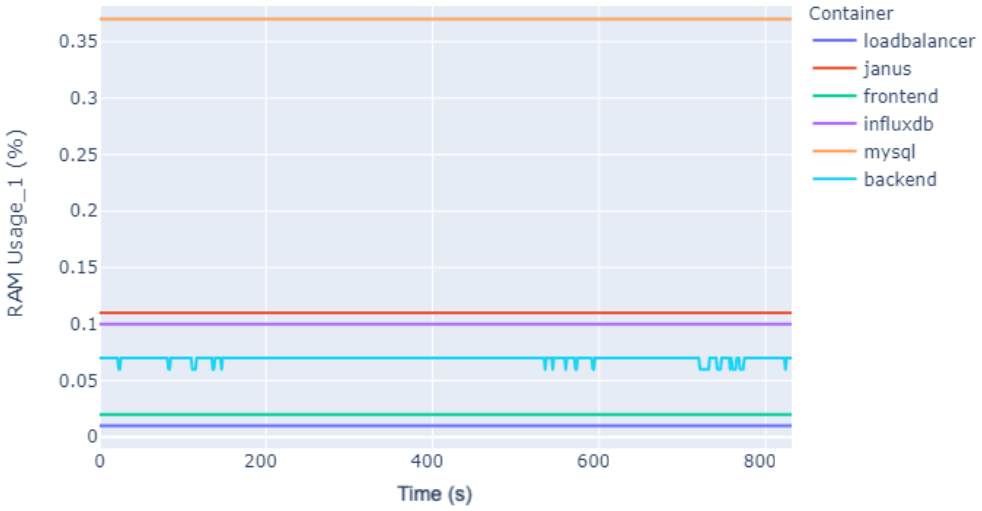

Figure 5.2: Case Study 1 - Percentage of RAM usage

video stream has to go through this Webrtc so it can send it to the users on the frontend. The reason the frontend is not present in any of the two data I/O graphs is because the amount of data input and output of the container during the test was zero.

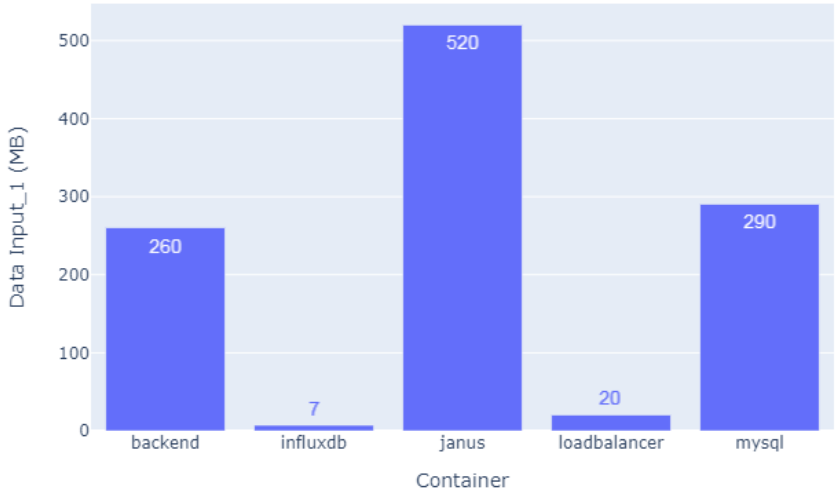

Figure 5.3: Case Study 1 - Data Input in Megabytes

Regarding the data output by each container (Fig. 5.4), we can examine that the highest value was also from the Janus. The components that precede the highest value on data output after Janus is the backend and the MySQL, respectively. Although we had four users watching the same UAV video stream, the data doesn't show a proportional increase in the amount of data. Since we had about 500MB of video stream data input, it would be expected to have four times this amount of data output (4 x 500MB = 2000MB or 2GB). This variable is due to the difference of Bitrate associated with each user. Since the network connectivity will influence the Bitrate at which a user can receive the video stream, each user will experience different Bitrates. Therefore the Janus gateway will

serve the video stream with variable quality for each user depending on their designated Bitrate.

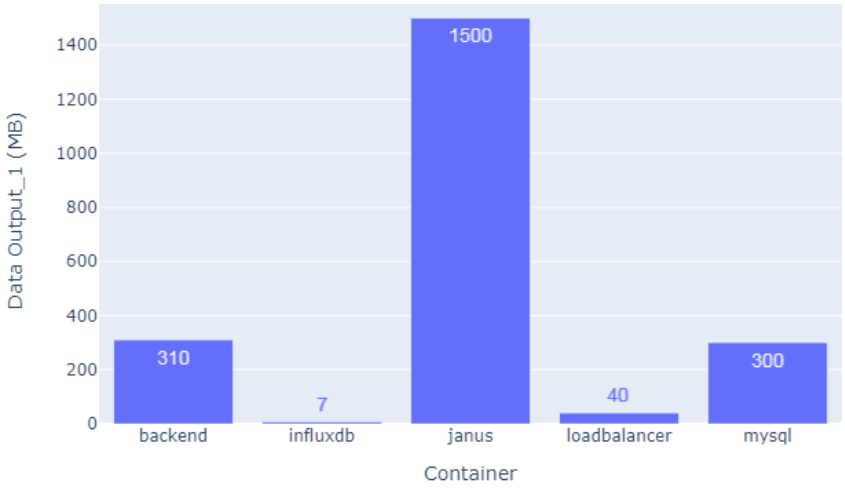

Figure 5.4: Case Study 1 - Data Output in Megabytes

5.2.2 Case Study 2 - 4 UAVs (Simulation) with 4 Users

The second experiment had a similar environment to the first one, the variable being the number of UAV simulations that were running on the server. For this test, we started four simulations each, with its video stream. The preparations steps described for the setup of the UAV were the same in this case study as it was in case study 1. After we connected all four UAVs to the production environment of the platform, four users accessed the frontend and proceeded to the monitor page. At this phase the script was initiated, and the server started registering the same metrics onto a Comma-separated Values (CSV) file. For this experiment, each user was free to change the selected UAV, therefore changing the video stream they were watching, with the constraint that they were always watching one of the streams. The duration of this test was also 10 minutes, during which there weren't any disconnection to the UAVs. The video streams were stable, and there wasn't any loss in connection.

Fig. 5.5 shows the graph of CPU usage, where we can analyze that the backend was working in an interval of 50 to 80%. Followed by MySQL at 20%, and Janus between 5 and 20%. In this case study, we can examine an increase of about 20% in the working interval of the backend. Given the slightly higher workload, it has given the three additional UAVs. The graph also shows the double CPU usage of the MySQL component. With this insight, we can expect that the introduction of more UAVs will not bring an exponential increase in resource usage.

The RAM usage graph (Fig. 5.6) shows that MySQL was the most demanding, accompanied by Janus and InfluxDB. Similarly to the first experiment, none of these components used more than 1% of the RAM.

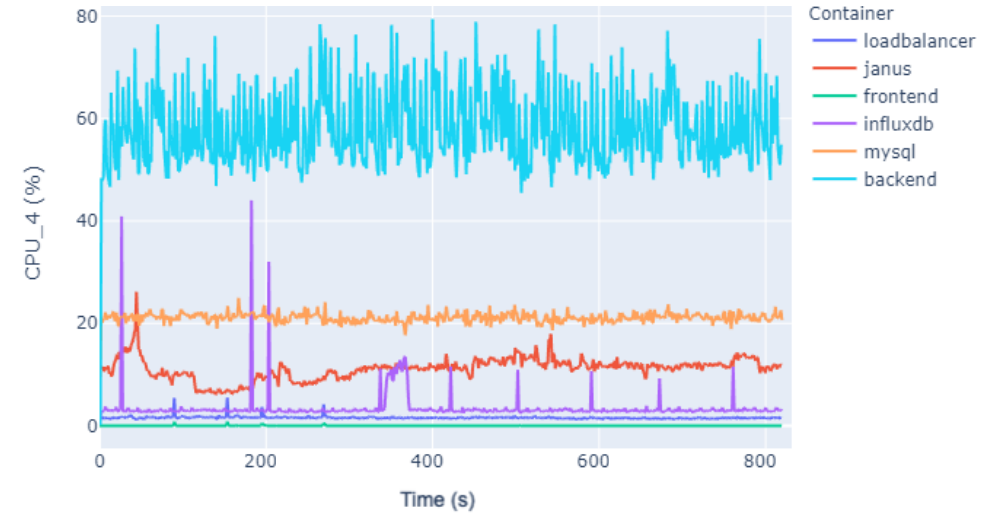

Figure 5.5: Case Study 2 - Percentage of CPU usage

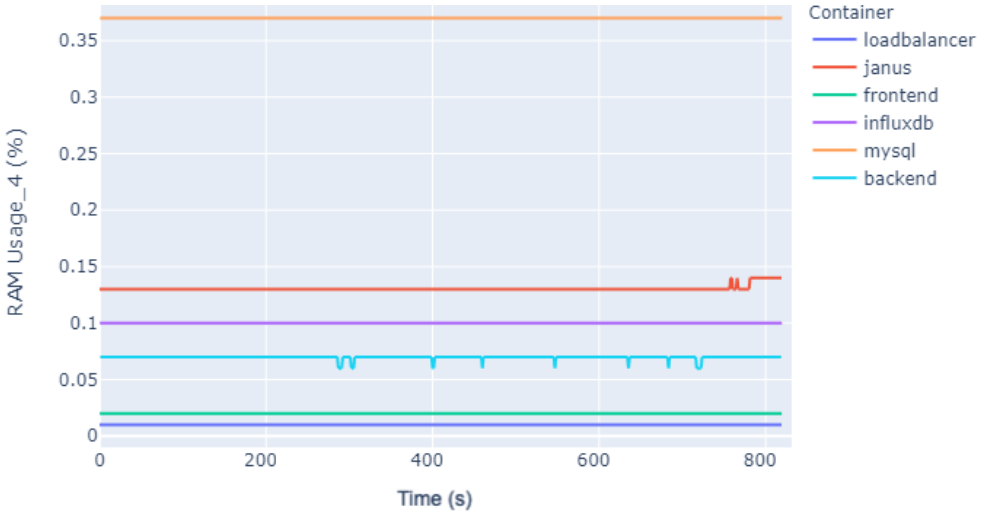

Figure 5.6: Case Study 2 - Percentage of RAM usage

Concerning the data input and output of the system, Fig. 5.7 shows us the Janus had the highest amount of data input, with total data input of 840MB. The next components with more data input are respectively MySQL with 570MB, and backend with 520MB. Here we can observe that almost all containers have slightly double the amount of data input in the second experiment, except for the loadbalancer. Once again, it is reinforced that the addition of more UAVs will not exponentially increase the resource usage of the server.

On the opposite side, the data output graph for this experiment (Fig. 5.8) shows that the container with the highest amount was again the Janus. However, this time the amount of data output from Janus (800MB) was lower than the data input (840MB). The explanation for this fact is on the reason given for the data output of the first test did not equal the "n"users times the amount of data input for the Janus container. Not only can

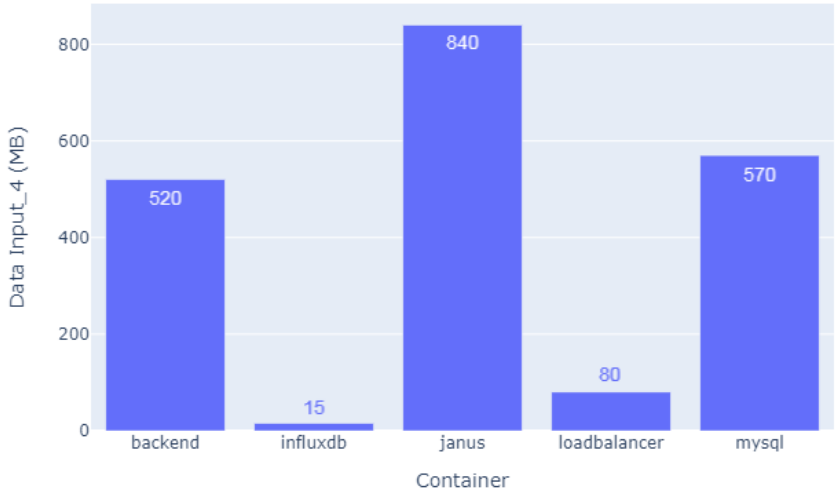

Figure 5.7: Case Study 2 - Data Input in Megabytes

each UAV have a different input Bitrate meaning each video stream will produce distinct amounts of data, but also each user can have a different Bitrate explaining the varying quantity of data each user will get. In this case, we have another factor that contributed to the fact that the data output was similar to the data input. Since each user could only see one video stream at a time, we would not have sixteen times (four video streams times four users) the amount of video stream data. As we can examine, we only had about the same amount of data input as of data output, indicating that the number of users watching will directly influence the amount of data output for the Janus component.

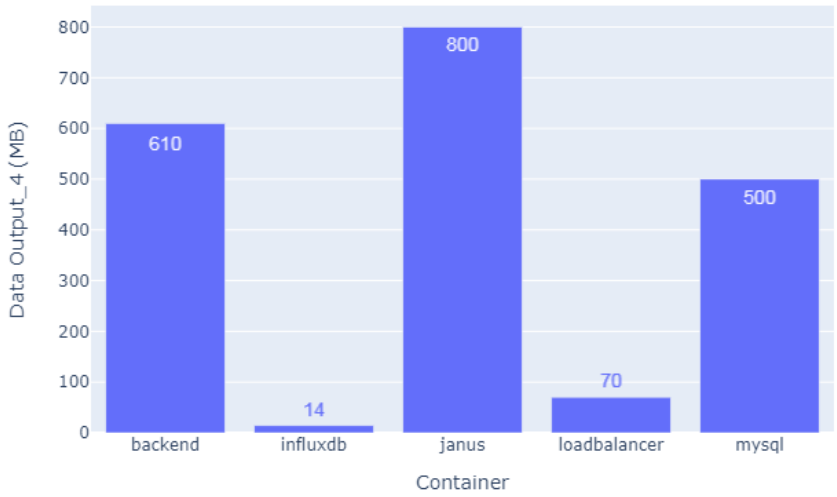

Figure 5.8: Case Study 2 - Data Output in Megabytes

To further emphasize the former statement, we can observe that the amount of data the Janus output on the first test was substantially higher than on the second test, meaning that it is not the introduction of more UAVs that will increase the overall data transmission of the platform but the addition of more users. Understandably, given that one video

stream can be seen by multiple users, but a single user cannot see various video streams at the same time.

The registration and analysis of these metrics are not only useful as a validation tool but also as an improvement mechanism. We deliberately chose to orchestrate the containers in a docker-compose file without any resource requirements and restrictions. Without the resource limits on the file, each component is working on a best-effort policy, meaning every container will scale vertically as much as it requires. This policy is not an optimal decision for a production environment since it can lead to a component to take more resources than it should. Nevertheless, we followed this best-effort paradigm to run tests and subsequently analyze the resulting data. With this knowledge, we will study and provide a minimum and maximum limit on the resources for each component for optimum performance.

5.3 Round Trip Time (RTT) Test

Aside from all the experiments conducted by the author, other team members performed some tests regarding the Round Trip Time (RTT) of a message sent through the platform [53]. These colleagues described the mentioned test in the following article. For this test, the authors measured the time for a message to arrive from the frontend to the UAV. They built an autonomous mission through the mission builder on the platform that commands the UAV to took-off and went to 120m of altitude and then descended to the ground. They repeated this mission 15 times in one day, with the particularity that the frontend was running on a machine in a different city each time. Then, they measured the RTT in distinct phases of the process and presented the results in a table.

After analyzing the resulting data, the authors found relevant results worth to discuss. First was that the RTT between the UAV and the backend remained stable, even when the frontend was in a different country. Given that the backend is currently attached to a server in Lisbon and the UAV took flight in Portugal makes sense for these values to be inside the same range. Since the variable was the location of the frontend, here is where the values for the RTT varied the most. They noted that when the frontend was in a place near the server running the backend, the RTT rounded the 13ms. But when the user was using the frontend from a distant location (e.g.) Sidney, the RTT rose to 352ms. However, the authors also elaborate that: "In aviation, the maximum RTT delay from a flying handling perspective is of approximately 300 ms". Another conclusion the authors inferred from the results was that connectivity played a big part in the performance of the platform. This conclusion came from the tests done in Portugal, where tests in locations near the server's region would perform similarly to tests done in the north of the country, farther from the server.

5.4 UAV video stream test

There was another article accepted in a conference regarding the video stream functionality of the platform [54]. In the article, the authors write about the possibility to combine computer vision algorithms to enhance the UAV capability with autonomous collision avoidance. Firstly the authors five a small overview of the platform. Afterward, they go into further detail about the inner workings of the video streams. They explain how the media gateway briefly mentioned in this dissertation utilizes a Coturn server to establish a connection to the client.

For the benchmarks, the authors used Google Chrome's tool for debugging WebRTC, which collects statistical information about ongoing streams. For this test, they used a Realsense D435i and an Insta360 Pro. They observed that the average number of frames decoded was 30, which also corresponds to the desired frame rate. Although the Insta 360 had more frame drops than the Realsense, this is understandable given that the 360 stream requires a more stable network environment.

Chapter 6

Conclusion

6.1 Summary

This dissertation started in collaboration with an European research project under the Horizon 2020 (H2020) program. Firstly by designing and implementing the detailed prototype and secondly by presenting it to the consortium. Subsequently, as a part of the research and development team at PDMFC, it was necessary to start specifying the necessities for the platform. This research led to not only to the requirements elicitation described on this document, but also to more research, now focusing on the state of the art of technology. In this research, there was given an overview of different relevant fields like Cyber-Physical Systems, Cloud Computing, and Unmanned Aerial Vehicles, followed by more specific topics regarding framework technologies such as the Robotic Operating System, NodeJS, and InfluxDB, and even the communication protocol named WebSockets. Subsequently, the research focused on encountering other types of work that had similar functionalities. This examination led to the discovery of the QGroundControl and the FlytOS. Although they were each a UAV platform, the research showed that none satisfies all the requirements.

After the requirement elicitation and research phase were concluded, the natural stage that followed was the implementation phase. For this purpose, it was produced some Unified Modeling Language (UML) diagrams regarding the architecture of the system (component diagram) and a compilation of each user use cases (use case diagram). Furthermore, there was conceived an entity-relation diagram to demonstrate the relations between each platform user. The development started with the implementation of the frontend using Angular. Then there was the need to have REST API's ready to be called by these interfaces. There is when the PDMFC team started to develop the backend using

a NodeJS framework called NestJS, which enforces typing and object-oriented programming. Concurrently, there was started to be done advancements for the transmission of low latency data by utilizing WebSocket technology. Regarding the video stream capabilities of the platform, a WebRTC gateway based on Janus was implemented, that resulted in the article stated in section 5.4. Lastly, regarding the tests and validation phase, there were performed multiple tests with the cooperation of team members that resulted in some knowledge concerning the internal functioning of the platform. The team will use most of the insights gained at this last phase as ground knowledge for future work. Another point worth mentioning is the participation in a conference where a paper summarizing the work done on this platform [55] was presented. The article starts by explaining the current trend on UAV's and their possible applications. Moreover, it is presented the proposed architecture and a summary of each component. Finally, it is also showed the results of the research concerning the related work and discusses the pros and cons of each one.

6.2 Contributions

Although there was one initial objective that was achieved, the points specified in the following list were all accomplished:

- A tool regarding the registration and management of multiple UAV;
- Flight real-time monitoring with GPS coordinates, velocity, IMU, battery, flight mode, and video stream;
- Automated mission planning tool for single UAV;
- Graphic User Interface for replaying flight missions.

The objective that was not possible to achieve in the timeline of this dissertation was the planning of autonomous missions with multiple UAVs. This was mainly due to the difficulties on performing field tests for the autonomous mission planning on a single UAV, that lead to a lack of time to implement this feature. Aside from the goals stipulated in the objectives section, we also supported the team in other developments. Such as the implementation of a management and visualization tool for multispectral maps. This task included not only the logic behind the processing of aerial photographs taken by the UAV and the integration with the platform's backend. But also the list and details interface for the resulting multispectral maps. The author displays this work in section 4.5.1 and Figure I.11 of annex I. Furthermore, the design and orchestration of the production environment were of the entire responsibility of the author, with the guidance of the infrastructure team from PDMFC.

Regarding scientific dissemination, we also wrote an article with the same title as the present dissertation, where a summary of the work here described is documented.

This paper was submitted to the 2020 12th International Symposium on Communication Systems, Networks, and Digital Signal Processing (CSNDSP) (CSNDSP2020) in July of 2020. There was another paper submitted to the same conference by some team members. That is the paper the author mentioned in section 5.4 that discusses the results regarding the low latency video stream accomplished on the platform. Finally, on the same topic, other members of the team wrote an article with the title: "FFAU—Framework for Fully Autonomous UAVs"for the Remote Sensing 12.21 (Oct. 2020). This article proposes an autonomous framework that allows the full control of UAVs as remote clients with a supervisor cloud-based platform. This article is relevant for the validation of the platform since it has a section regarding a field test performed where the authors measured the Round Trip Time (RTT).

6.3 Future Work

Although the far majority of stipulated tasks were accomplished, there are a few features the team can plan to implement to improve the platform capabilities. Moreover, some tweaks came directly from the insights gained through the testing and validation phase, resulting in a minimum and maximum requirements for each component. These will result in the improved performance of our production environment, therefore also guaranteeing a more reliable service. Moreover, the team also identified the need to add a monitoring and logging component to the production server. First, to visualize the metrics close to real-time and understand the underlying functioning of the platform during peak workloads. Second, to identify and alert the maintainers of possible problems that can result in downtime of the system.

After we ensure our production environment is more robust and reliable, we can start to focus on introducing new functionalities and use cases. These functionalities are the following:

- The integration of computer vision algorithms to enhance the UAV capability with autonomous collision avoidance, as specified in one of the articles. The team is already working on this feature, exploring the YOLO framework to understand the best way to run it on the on-board computer on the UAV.

- After the team performs more tests on the built-mission functionality and its functioning is validated, we will start to implement the capability of orchestrating swarm missions. Where a user can select a specific area on a map, and the platform autonomously assigns multiple UAVs to work cooperatively to achieve the required tasks.

- Presently, we are working on the implementation of a solar-powered charging station for UAVs. This station will double as a take-off and landing area for autonomous use cases. After the completion of this station, we will have the task to

integrate the station into the platform.

- Currently, our UAVs communicate over a 4G modem that allows them to connect with the platform. In the future, we want to integrate our UAVs with the 5G network to enhance the efficiency in the transmission of real-time data. Being this data, UAV metrics, or camera video streams.

Bibliography

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani. "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges." In: *IEEE Access* 7 (2019), pp. 48572–48634. DOI: 10.1109/ACCESS.2019.2909530.

[2] E. Alvarez-Vanhard, T. Houet, C. Mony, L. Lecoq, and T. Corpetti. "Can UAVs fill the gap between in situ surveys and satellites for habitat mapping?" In: *Remote Sensing of Environment* 243 (2020), p. 111780. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2020.111780.

[3] A. Navarro, M. Young, B. Allan, P. Carnell, P. Macreadie, and D. Ierodiaconou. "The application of Unmanned Aerial Vehicles (UAVs) to estimate above-ground biomass of mangrove ecosystems." In: *Remote Sensing of Environment* 242 (2020), p. 111747. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2020.111747.

[4] B. U. Meinen and D. T. Robinson. "Mapping erosion and deposition in an agricultural landscape: Optimization of UAV image acquisition schemes for SfM-MVS." In: *Remote Sensing of Environment* 239 (2020), p. 111666. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2020.111666.

[5] L. Gupta, R. Jain, and G. Vaszkun. "Survey of Important Issues in UAV Communication Networks." In: *IEEE Communications Surveys Tutorials* 18.2 (2016), pp. 1123–1152. DOI: 10.1109/COMST.2015.2495297.

[6] M.-G. Avram. "Advantages and challenges of adopting cloud computing from an enterprise perspective." In: *Procedia Technology* (2014). DOI: https://doi.org/10.1016/j.protcy.2013.12.525.

[7] *Aggregate FARming in the CLOUD (AFarCloud)*. 2018. (Visited on 01/2020).

[8] R. C. Shah, S. Roy, S. Jain, and W. Brunette. "Data MULEs: modeling and analysis of a three-tier architecture for sparse sensor networks." In: *Ad Hoc Networks* 1.2 (2003). Sensor Network Protocols and Applications, pp. 215 –233. ISSN: 1570-8705. DOI: https://doi.org/10.1016/S1570-8705(03)00003-9.

[9] H. Auernhammer. "Precision farming—the environmental challenge." In: *Computers and electronics in agriculture* (2001). ISSN: 0168-1699, DOI: https://doi.org/10.1016/S0168-1699(00)00153-8.

[10] A. McBratney, B. Whelan, T. Ancev, and J. Bouma. "Future directions of precision agriculture." In: *Precision agriculture* (2005). DOI: https://doi.org/10.1007/s11119-005-0681-8.

[11] S. Candiago, F. Remondino, M. De Giglio, M. Dubbini, and M. Gattelli. "Evaluating multispectral images and vegetation indices for precision farming applications from UAV images." In: *Remote sensing* (2015). DOI: 10.3390/rs70404026. URL: http://dx.doi.org/10.3390/rs70404026.

[12] M. A. Zamora-Izquierdo, J. Santa, J. A. Martínez, V. Martínez, and A. F. Skarmeta. "Smart farming IoT platform based on edge and cloud computing." In: *Biosystems Engineering* 177 (2019). Intelligent Systems for Environmental Applications, pp. 4–17. ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2018.10.014. URL: http://www.sciencedirect.com/science/article/pii/S1537511018301211.

[13] R. Finger, S. M. Swinton, N. El Benni, and A. Walter. "Precision Farming at the Nexus of Agricultural Production and the Environment." In: *Annual Review of Resource Economics* 11.1 (2019), pp. 313–335. DOI: 10.1146/annurev-resource-100518-093929. URL: https://doi.org/10.1146/annurev-resource-100518-093929.

[14] J. P. Matos-Carvalho, F. Moutinho, A. B. Salvado, T. Carrasqueira, R. Campos-Rebelo, D. Pedro, L. M. Campos, J. M. Fonseca, and A. Mora. "Static and Dynamic Algorithms for Terrain Classification in UAV Aerial Imagery." In: *Remote Sensing* (2019). DOI: 10.3390/rs11212501.

[15] A. Dobermann, S. Blackmore, S. E. Cook, V. I. Adamchuk, et al. "Precision farming: challenges and future directions." In: 2004.

[16] G. X. Gao, P. K. Enge, L. A. Davis, et al. *Global Navigation Satellite Systems: Report of a Joint Workshop of the National Academy of Engineering and the Chinese Academy of Engineering*. National Academies Press, 2012. ISBN: 978-0-309-22275-4. DOI: 10.17226/13292. URL: https://www.nap.edu/catalog/13292/global-navigation-satellite-systems-report-of-a-joint-workshop-of.

[17] D. Schimmelpfennig. *Farm profits and adoption of precision agriculture*. Tech. rep. 2016. DOI: http://dx.doi.org/10.22004/ag.econ.249773.

[18] P. Mell and T. Grance. "The NIST Definition of Cloud Computing." In: *Application Performance Management (APM) in the Digital Enterprise*. Elsevier, 2011, pp. 267–269. ISBN: 9781617617843. DOI: 10.1016/B978-0-12-804018-8.15003-X. URL: https://linkinghub.elsevier.com/retrieve/pii/B978012804018815003X.

[19] G. I. Nikolov. *Cloud Computing and Government: Background, Benefits, Risks*. Nova Science Publishers, Inc., 2011. ISBN: 1617617849.

[20] W. Venters and E. A. Whitley. "A Critical Review of Cloud Computing: Researching Desires and Realities." In: *Journal of Information Technology* 27.3 (Sept. 2012), pp. 179–197. ISSN: 0268-3962. DOI: 10.1057/jit.2012.17. URL: http://journals.sagepub.com/doi/10.1057/jit.2012.17.

[21] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin. "A view of cloud computing." In: *Communications of the ACM* 53.4 (Apr. 2010), p. 50. ISSN: 00010782. DOI: 10.1145/1721654.1721672. arXiv: 052186571997805218657199780521865715. URL: http://portal.acm.org/citation.cfm?doid=1721654.1721672.

[22] Q. Zhang, L. Cheng, and R. Boutaba. "Cloud computing: state-of-the-art and research challenges." In: *Journal of Internet Services and Applications* (2010). ISSN: 1867-4828. DOI: 10.1007/s13174-010-0007-6. URL: http://link.springer.com/10.1007/s13174-010-0007-6.

[23] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu. "Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends." In: *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 2015. DOI: 10.1109/CLOUD.2015.88. URL: http://ieeexplore.ieee.org/document/7214098/.

[24] L. Qian, Z. Luo, Y. Du, and L. Guo. "Cloud computing: An overview." In: *IEEE International Conference on Cloud Computing*. Springer. 2009, pp. 626–631. ISBN: 978-3-642-10665-1.

[25] S. Kamburugamuve, L. Christiansen, and G. Fox. "A Framework for Real Time Processing of Sensor Data in the Cloud." In: *Journal of Sensors* 2015 (2015), pp. 1–11. ISSN: 1687-725X. DOI: 10.1155/2015/468047. URL: http://www.hindawi.com/journals/js/2015/468047/.

[26] J. J. Kuffner. "Cloud-Enabled Robots." In: *IEEE-RAS International Conference on Humanoid Robots, NashVille,TN,USA*. 2010. URL: https://ci.nii.ac.jp/naid/10031099795/en/.

[27] D. Di Paola, A. Milella, G. Cicirelli, and A. Distante. "An Autonomous Mobile Robotic System for Surveillance of Indoor Environments." In: *International Journal of Advanced Robotic Systems* 7.1 (Mar. 2010), p. 8. ISSN: 1729-8814. DOI: 10.5772/7254. URL: http://journals.sagepub.com/doi/10.5772/7254.

[28] E. A. Lee. "Cyber-physical systems-are computing foundations adequate." In: *Position paper for NSF workshop on cyber-physical systems: research motivation, techniques and roadmap*. Vol. 2. Citeseer. 2006, pp. 1–9.

[29] E. A. Lee. "Cyber Physical Systems: Design Challenges." In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. 2008, pp. 363–369. DOI: 10.1109/ISORC.2008.25.

[30] H. J. La and S. D. Kim. "A service-based approach to designing cyber physical systems." In: *Proceedings - 9th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2010*. 2010. ISBN: 9780769541471. DOI: 10.1109/ICIS.2010.73.

[31] J. S. Her, H. J. La, and S. D. Kim. "A Formal Approach to Devising a Practical Method for Modeling Reusable Services." In: *2008 IEEE International Conference on e-Business Engineering*. IEEE, 2008, pp. 221–228. ISBN: 978-0-7695-3395-7. DOI: 10.1109/ICEBE.2008.20.

[32] S. H. Chang, H. J. La, and S. D. Kim. "A Comprehensive Approach to Service Adaptation." In: *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*. 2007, pp. 191–198. DOI: 10.1109/SOCA.2007.2.

[33] J. Shi, J. Wan, H. Yan, and H. Suo. "A survey of Cyber-Physical Systems." In: *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*. 2011, pp. 1–6. DOI: 10.1109/WCSP.2011.6096958.

[34] I. Dumitrache, I. S. Sacala, M. A. Moisescu, and S. I. Caramihai. "A conceptual framework for modeling and design of Cyber-Physical Systems." In: *Studies in Informatics and Control* 26.3 (2017), pp. 325–334. DOI: https://doi.org/10.24846/v26i3y201708.

[35] C.-R. Rad, O. Hancu, I.-A. Takacs, and G. Olteanu. "Smart Monitoring of Potato Crop: A Cyber-Physical System Architecture Model in the Field of Precision Agriculture." In: *Agriculture and Agricultural Science Procedia* 6 (2015). Conference Agriculture for Life, Life for Agriculture, pp. 73 –79. ISSN: 2210-7843. DOI: https://doi.org/10.1016/j.aaspro.2015.08.041. URL: http://www.sciencedirect.com/science/article/pii/S2210784315001746.

[36] K. Antonopoulos, C. Panagiotou, C. P. Antonopoulos, and N. S. Voros. "A-FARM Precision Farming CPS Platform." In: *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*. 2019, pp. 1–3. DOI: 10.1109/IISA.2019.8900717.

[37] R. Chaâri, F. Ellouze, A. Koubâa, B. Qureshi, N. Pereira, H. Youssef, and E. Tovar. "Cyber-physical systems clouds: A survey." In: *Computer Networks* 108 (2016), pp. 260–278. ISSN: 13891286. DOI: 10.1016/j.comnet.2016.08.017.

[38] A. Koubaa, B. Qureshi, M.-F. Sriti, Y. Javed, and E. Tovar. "A service-oriented Cloud-based management system for the Internet-of-Drones." In: *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. Section 3. IEEE, Apr. 2017, pp. 329–335. ISBN: 978-1-5090-6234-8. DOI: 10.1109/ICARSC.2017.7964096. URL: http://ieeexplore.ieee.org/document/7964096/.

[39] B. Qureshi and A. Koubâa. "Five Traits of Performance Enhancement Using Cloud Robotics: A Survey." In: *Procedia Computer Science* 37 (2014), pp. 220–227. ISSN: 18770509. DOI: 10.1016/j.procs.2014.08.033. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050914009983.

[40] *The UAV*. URL: https://www.theuav.com/ (visited on 01/2020).

[41] *Types of Drones: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL*. 2016. URL: https://www.auav.com.au/articles/drone-types/ (visited on 01/2020).

[42] *Types of Drones – Explore the Different Models of UAV's*. 2017. URL: http://www.circuitstoday.com/types-of-drones (visited on 01/2020).

[43] L. Joseph. *Mastering ROS for Robotics Programming*. 2015. ISBN: 9781783551798. DOI: 10.1007/s13398-014-0173-7.2. arXiv: arXiv:1011.1669v3.

[44] A. M. Romero. *ROS/Concepts*. 2014. URL: http://wiki.ros.org/ROS/Concepts (visited on 02/2020).

[45] A. Martinez and E. Fernández. *Learning ROS for robotics programming*. Packt Publishing Ltd, 2013. ISBN: 1782161449.

[46] "Node.js: Using JavaScript to Build High-Performance Network Programs." In: *IEEE Internet Computing* 14.6 (Nov. 2010), pp. 80–83. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.145. URL: http://ieeexplore.ieee.org/document/5617064/.

[47] C. R. Pereira. *Aplicações web real-time com Node. js*. Editora Casa do Código, 2014. ISBN: 8566250141.

[48] P. Dix. *InfluxData (InfluxDB) | Time Series Database Monitoring & Analytics*. 2017.

[49] S. N. Z. Naqvi, S. Yfantidou, and E. Zimányi. "Time series databases and influxdb." In: *Studienarbeit, Université Libre de Bruxelles* (2017).

[50] I. Fette and A. Melnikov. *The websocket protocol*. 2011. (Visited on 01/2020).

[51] *Angular RxJS Integration*. https://angular.io/guide/rx-library. Accessed: 12-02-2020.

[52] D. Prasanna. *Dependency Injection: Design Patterns Using Spring and Guice*. Manning Pubs Co Series. Manning, 2009. ISBN: 9781933988559. URL: https://books.google.pt/books?id=b6060gAACAAJ.

[53] D. Pedro, J. P. Matos-Carvalho, F. Azevedo, R. Sacoto-Martins, L. Bernardo, L. Campos, J. M. Fonseca, and A. Mora. "FFAU—Framework for Fully Autonomous UAVs." In: *Remote Sensing* (2020). DOI: 10.3390/rs12213533. URL: https://doi.org/10.3390/rs12213533.

[54] R. Sacoto-Martins, J. Madeira, J. P. Matos-Carvalho, F. Azevedo, and L. M. Campos. "Multi-purpose Low Latency Streaming Using Unmanned Aerial Vehicles." In: *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. 2020, pp. 1–6. DOI: 10.1109/CSNDSP49049.2020.9249562.

[55] M. Pino, J. P. Matos-Carvalho, D. Pedro, L. M. Campos, and J. Costa Seco. "UAV Cloud Platform for Precision Farming." In: *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*. 2020, pp. 1–6. DOI: 10.1109/CSNDSP49049.2020.9249551.

Appendix

Accepted Paper - UAV Cloud Platform for Precision Farming

UAV Cloud Platform for Precision Farming

Miguel Pino[1,4], J. P. Matos-Carvalho[2,4], Dário Pedro[3,4], Luís M. Campos[3] and João Costa Seco[4,5]

[1] Apps4Mobility, 7800-295 Beja, Portugal;
[2] Beyond Vision, 3830-352 Ílhavo, Portugal;
[3] PDMFC, 1300-609 Lisboa, Portugal;
[4] FCT, NOVA University of Lisbon, 2829-516 Caparica, Portugal;
[5] NOVA Laboratory for Computer Science and Informatics, NOVA LINCS, Caparica, Portugal;

***Abstract*—The advancements made on Unmanned Aerial Vehicles (UAV) related areas such as Cyber-Physical Systems and Embedded Systems made them more suitable for some industry-specific problems. Applications in fields like Precision Agriculture and Aerial Mapping could benefit considerably with the introduction of UAV use cases. However, there are still some obstacles nowadays in UAV related topics like low processing power and storage capacity. To develop robust and complex systems integrating these UAVs first, we need to address these problems. This paper proposes a cloud platform capable of two-way communication with UAVs and further management of multiple UAVs.**

***Index Terms*—CPS, UAV, Cloud Computing, Communication Protocols**

I. Introduction

An Unmanned Aerial Vehicle (UAV) consists of a small, manually or autonomously controlled aircraft that doesn't need an onboard human pilot. One characteristic of these aircraft is its high-mobility granted by its small size. Nowadays the capabilities of these aircraft can be enhanced with cameras capable of capturing multispectral pictures from high-altitudes. Common scenarios present in fields such as aerial surveillance, industrial automation, and precision agriculture would benefit from the integration of UAVs in regards to efficiency and performance.

However, these aircraft present some obstacles. In the first place, low computational power limits the computation of complex algorithms. A solution to this problem is to offload the computational power by connecting a UAV to the internet, we could send the data necessary to run the algorithm, and then get the results back and act on it. The second obstacle is the low storage capacity of the UAV due to its small size. The latter solution could also be applied here since it is possible to offload storage capacity. If we integrate a UAV in a cloud platform designed for this propose, we could meet these two requirements.

This paper proposes a cloud platform developed for precision farming use cases. Such use cases consist of planning autonomous missions for collecting aerial mapping pictures from different spectrum's and multiple sensors. The first use case urges from the need to analyze the vineyard's life cycle through RGB mapping and heatmaps to folow its development. It is also possible to prevent vineyard pests and diseases by analyzing indicators on multispectral maps. The second use case came from the need to introduce temperature, humidity, and air quality sensors on vineyards. Farm owners use data collected from these sensors as insight for making the best decision.

The organization of this paper is as follows, after this introduction, we present the related work after this, we describe the experimental setup followed by the detailed implementation of the platform. Finally, we show the results obtained in some tests with our platform, and successively we compare and discuss these result data.

II. Related Work

The possible applications for an autonomous small aircraft able to capture aerial images are vast. Although there are many applications for UAVs use cases nowadays, a robust and reliable system to implement them is still missing [1]–[4]. Most UAV providers offer UAVs as an independent device, without any connection to a platform capable of managing them or, for example, planning an autonomous flights. Moreover, there is a lack of applications that take full advantage of the aerial photography capabilities of these UAVs.

There is an open-source tool used for manual flight control and autonomous flight planning called QGroundControl [5]. It provides configuration and support for UAVs that run PIX4 or Ardupilot. Its user interface uses a satellite map to display the current connected UAVs and enables the user to plan a mission by adding waypoints to the map. Furthermore, it also presents the number of connected satellites, the UAV battery life, and the current UAV flight mode. Aside from these functionalities, there is not much that this tool can offer since it cannot store

data persistently on a database, it can only give insights on real-time data.

There is also a platform that presents itself as an holistic cloud platform to connect UAVs, that offers a variety of features. FlytOS [6] provides plugins for autonomous planning, collision avoidance, and thermal camera integration. However, this is proprietary software that requires the user to have some knowledge to implement and start using this custom made solution. Not only this but, the authors of this article could not find evidence that indicated the possibility to analyze multiple spectrums of aerial pictures or even compare different indexes. This platform also lacks a feature that the user can use to get some insights by reviewing previous flights, with this a user can analyze aspects of the flight to find what went according to plan.

In addition to the platforms described above, there are softwares that focus on building geographic maps from UAV sampling. The studied and compared software are Pix4Dmapper, Photoscan, Aerial Photo Survey (APS) [7]. The Pix4Dmapper, developed by Pix4D, uses techniques based on aerial triangulation [8] and BBA (Block Bundle Adjustment) to calculate the positions and orientation of each image [9]. Another concurrent is APS, a software built by Menci Software, where the mosaic originates from Digital Terreain Model (DTM), also providing point clouds, orthomatics and also topographic contour lines skills [7]. Finally, there is Photoscan, a software developed by Agisoft, based on photographic triangulation, capable of generating point cloud of covered area, DTM, DSM and orthomosaic from the processing of multispectral imagery [7].

A well-known open source platform that uses methods similar to Pix4Dmapper is called WebODM [10], [11] where an interface is developed where it is possible to upload images and save them in an internal database (both the input images and the results of the maps).

Although these frameworks produce maps from images coming from an RGB and multispectral camera, they currently have no ability to integrate in realtime with drones, which in a way is a disadvantage when the goal to communicate over long distances and produce maps right after the flight. Thus, this article aims to design and implement an architecture that allows a swarm of UAVs to be in constant communications from an unlimited distance to the end user and integrate the mapping algorithms.

III. Experimental Setup

A. Proposed System Architecture

From a high-level point of view, there are three fundamental parts for the system architecture, a UAV, a server-side, and a client-side as it can be seen in Figure 1. The UAV corresponds to the physical part of our Cyber-Physical System and is our main point of actuation over the environment. A gateway makes the connection between the UAV and the server possible, by connecting the data sent from Robotic Operating System (ROS) topics to WebSockets. The UAV publishes data such as GPS coordinates and linear and angular velocities to a ROS topic, in which a WebSocket is subscribed and listens for updates to forward them to the server. The user interface then gets this data from the server via WebSocket also, to display the real-time location and speed of the UAV. Subsequently, this generated data is processed and stored in a time-series database for future queries.

Aside from this, there is a component called WebODM that is in charge of processing incoming map data. A UAV can capture aerial pictures of a landscape with different camera types, that then need to undergo a process of processing and stitching to be presented to a user. A user can then get insights from multiple spectrums pictures of a landscape and use them to support his next decisions.

Finally, the client-side includes all the user interfaces necessary for the access of the user to the main features of the platform. A list view presents all the registered drones, followed by a small description and some information such as if they are currently active and if not when was the last time they were. A monitor view displays all the currently active drones with the possibility to select any of them and start to monitor it, checking GPS coordinates and velocities. Furthermore, if the UAV has a camera attached to it, a video stream will be displayed on this page. For the maps processing, there is also a maps list that presents all available maps with the possibility to review each one individually. We can examine the different indexes with one another or even with the progress of the same index over time.

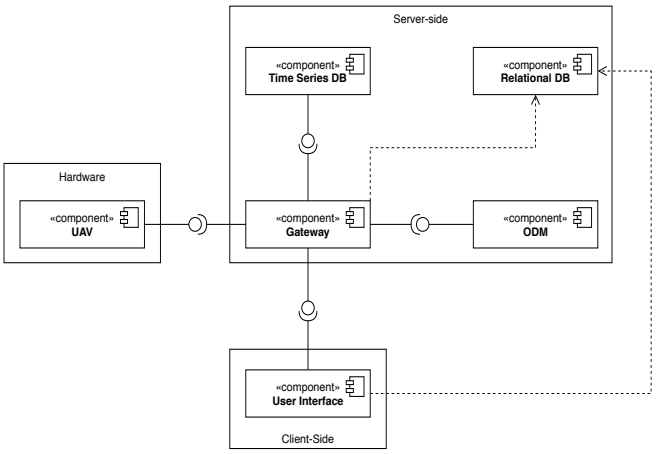

Fig. 1. Platform Component Diagram

B. UAV Design

Hexa Exterior Intelligent Flying Unit (HEIFU) is a custom made solution, aimed at the agricultural sector, developed by a collaboration between PDMFC and BEV. The HEIFU is equipped with an onboard computer running Ubuntu and ROS. This is an open platform that allows the integration of different inputs and is used to run multiple tasks, such as image processing, data relaying, remote control of a UAV (and more). HEIFU can be used with different communication systems, such as a mobile network (3G, 4G or 5G) or Wi-Fi connection. HEIFU is a Hexacopter, as it can be seen in Figure 2, with a dimension of 1.4m in the diagonal wheelbase. The UAV weight is around 6.2Kg, including battery, and the hovering time is around 38min with a battery of 16Ah.

Fig. 2. Unmanned Aerial Vehicle - HEIFU.

Some of the Hexa Exterior Intelligent Flying Unit (HEIFU) specifications are [12]: An Ardupilot (Pixhawk) hardware is used to control the low level operation. This hardware contains the IMU and GPS to know the UAV's position and orientation. Also, the Pixhawk connects to the Jetson nano embedded system via MAVlink protocol and the UAV's battery [13]. Lastly, to control the UAV's motors it is connected to a UHF receiver; An Jetson nano is used to control the high level operation. It receives data from the distance sensor (depth cameras), the RGB camera and communicates with external devices via Wifi link; A RGB camera with a gimbal stabilizer, is installed capturing onboard images at a resolution of 640x480 pixels; The camera and lens specifications are known, allowing the field of view (FOV) and the pixel size in meters to be computed [14].

C. Communication Protocol

Since the platform will handle real-time information coming from the UAV, we needed to address this problem. A common technique used in many web-platforms is long-polling, where the client constantly checks the server for new data. Until a decade ago, this way of getting real-time data was as good as it gets. However, as more client requests reached the server, a predictable problem became more apparent. The latency between client and server increased since each connection is kept open for as long as possible. When the connection times out, the client is notified and makes a new request, repeating the process. Thus the idea of a new protocol, identified as WebSocket, was born. As I. Fette and A. Melnikov described [15], WebSocket allows a long-held, bi-directional and full-duplex TCP socket connection to establish between client and server. This procedure starts with the client sending an HTTP request to the server asking to connect to a WebSocket. This process is known as a WebSocket handshake. After the server accepts this request, it replaces the HTTP connection by a WebSocket connection. The server maintains this last connection for each client, using it to push data in real-time to the client.

IV. Implementation

This section describes the implementation process of the platform starting by the backend, then the middleware, and finishing with the frontend. Typically a web platform is comprised only of a backend and a frontend, although sometimes there is a need to include some middleware logic. Moreover, we followed a standard design pattern in the industry called model-view-controller (MVC), which tries to keep the computation of each domain independent of one another. The backend involves all the server-side logic (controller), algorithms and heavy computation tasks, and database persistence (model) regarding data produced on the platform. On the other side, the frontend takes care of the presentation layer (view) since it will include all the user interfaces. Due to advancements made in recent years to frontend frameworks, there is a possibility to add some logic to these user interfaces. Finally, the middleware serves as a connector between the back and frontend, comprising all the frameworks and logic necessary.

A. Backend (NodeJs)

A server is needed to make a remote connection between two or more devices. In this case, we needed a server that can handle real-time communication. Requests to the server are processed as a loop in a "non-blocking" manner hence achieving low latency and high throughput.

The JavaScript runtime is comprised of a heap, where memory is allocated for variables and functions, and a stack, where JavaScript functions are queued. Aside from this, Node.js also includes an event loop that periodically checks the stack for queued functions and runs them. A function can do a simple computation of assigning values to variables or it can call a WebAPI like setTimeout or onClick. This WebApis typically include a callback

function that should be called in the future and are respectively placed in a task queue. When the event loop sees that there isn't any function left to be run in the stack, it starts to push the callback functions in the task queue to the stack.

With this system in place, Node.js can achieve a non-blocking single-thread that is capable of answering thousands of requests made to the same server [16] concurrently. This was the main factor that led to the adoption of a Node.js server for this platform.

B. Middleware (ROS)

Robot Operating Systems (ROS) has three levels of concepts: the Filesystem level, the Computation Graph level, and the Community level [17]. ROS also provides support for a wide range of programming languages; the possibility of visualizing data (e.g., the content of topics, nodes, packages, coordinate systems graphs and sensor data); and the possibility to write and execute code in a modular way, increasing robustness and also contributing to the standardization of this framework [18].

1) ROS Packages: First, the basic concept that the reader needs to acknowledge is the ROS package. This is where nodes' source and header files, executable scripts, launch files, among others are stored and organized.

2) ROS graph Layer: One of the packages installed from source is ros_comm contains the ROS middleware/communications' packages, known as ROS "Graph" Layer. These concepts can be found in any book, and on the ROS wide online documentation, although, some will be briefly introduced here given their relevance to the content of this dissertation.

The ROS Master is what makes it possible to different nodes to find and communicate with each other. As a distributed approach will be used in this paper, it will be initialized on one machine and nodes will be able to communicate with it from a remote machine [19].

Given that ROS is a distributed computing framework, it allows network connections and information exchange to perform different tasks. Messages allow a publish/subscribe pattern of Topics (the message named bus - its identifier), whereas Services provide a request/response behaviour. Therefore, while ROS topics are unidirectional, with ROS Services, one node is the server, from whom, a client may request a service and after completion of a procedure, send a reply [19]. This is done with Nodes, which can be seen as system's processes and may run in different machines.

Finally, Bags will be used extensively on this paper, as they allow to save the information being transmitted via different topics and the Parameter Server allows different Parameters to be accessible from a node. These can be loaded from a launch file or from inside a node and can be declared in a YAML file.

C. Frontend (Angular)

Angular is an open-source framework maintained by Google, that aims to develop client-side web applications. Not only this, but Angular also currently has long-term support for version 8 [20], which gives some assurance for platforms developed under this framework.

The first practical advantage of developing a platform with this framework is that the model-view-controller was built-in from its creation. Its component-based architecture brings not only more quality of code but also a separation of concerns. Having components, services, and models separated like classes that one must explicitly import, enforces these rules, and maintains the overall consistency of the project.

The second fact that lead to the choice of this framework is the entanglement of RxJS in Angular in regards to asynchronous programming [21]. RxJS is a library designed to handle asynchronous data to maintain the responsiveness of a user interface when waiting for new data to arrive. It does this by handling events independently and in parallel with the use of observables. Rather than continuously send requests to the server to check if new data is available (pooling), a component can subscribe to an observable and wait for the server to push some data. Moreover, RxJS offers a set of operators capable of manipulating the data as it arrives, meaning that when data reaches the component is ready to be presented.

Finally, another factor that solidifies our choice on this framework was its performance, derivative from Angular's hierarchical dependency injection. Dependency injection is a design pattern used in software development, first approached in Java programming language. Its main idea is that there should be a dependency injector that is responsible for connecting components to its dependent services, alternatively to the case that each component should explicitly ask for his dependencies [22].

The next images are screenshots from user interfaces of the platform:

Figure 3 presents the monitor page for UAV connected in real-time, being possible to see the video stream of the attached camera, right above are the GPS coordinates and IMU indicator. In this section we also have a small 3D UAV that replicates the angular movement of the real UAV. In the bottom bar, it is shown the vertical and horizontal velocities, the current altitude and the remaining battery.

The 2D map indexes page, in Figure 4, presents a timeline on the left side with the list of all maps created nearby, chronologically ordered. It is possible to select

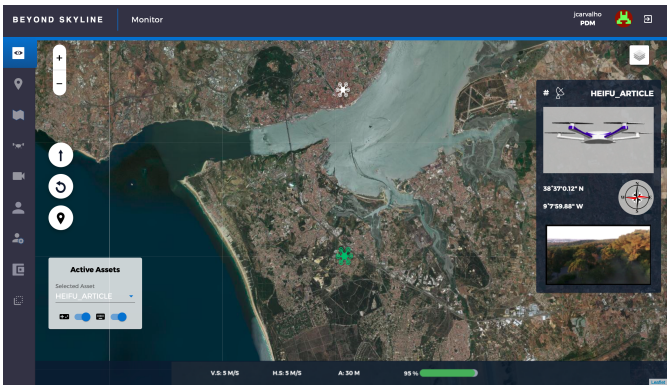

Fig. 3. UAV Real-time Monitor Page.

more than one map to compare various indexes between different maps. The right side displays a set of sliders capable of changing the opacity of each index. When selecting one map the user can toggle the 3D view and access the interface shown in Figure 5. In the 3D interface the user is present with a collection of points forming a point cloud that was collected with a camera placed in the UAV that surveyed the area.

Fig. 4. 2D Map Indexes Page.

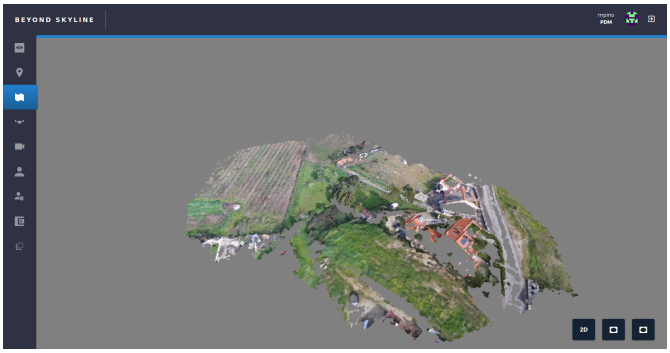

Fig. 5. 3D Map Page.

V. Experimental Results

This section aims to compare the three tools/platforms described in the related work, with the one that this paper proposes. We used six different features as indicators for this comparison.

The first feature was the possibility to setup/configure a UAV through the platform by registering a new UAV to a platform and configure certain aspects. These aspects could be the setup of a token for security reasons or the configuration of peripherals possible to integrate on the UAV. The second feature is the possibility to manage multiple UAVs, not only to view its current status and location but also with the capability to change some of its configurations. The third feature concerns the realtime monitoring of a UAV in regards to GPS coordinates, velocities, and video stream if a camera is attached. The fourth feature is the ability to send commands to a UAV directly from the platform a user could actuate on the UAV from the platform without the use of a remote control. The fifth feature corresponds to the planning of autonomous flights, the proposed platform provides a specific user interface that offers the user the capacity to determine some characteristics of the mission. The sixth and final feature concerns the creation, processing, and stitching of maps from multiple spectrums originated from aerial pictures collected by the UAV.

Functionalities	WebODM	QGroundControl	FlytOS	Proposed Platform
UAV Setup/Configuration	-	X	X	X
Multiple UAV Management	-	-	X	X
UAV Monitoring	-	-	X	X
UAV Online Navigation	-	-	-	X
Autonomous Flight	-	X	-	X
Maps Processing	X	-	-	X

TABLE I
TABLE OF SIMILAR PLATFORMS COMPARISON

These are the central points that the proposed platform offers in comparison with other pre-existence tools or platforms. The proposed platform is capable of addressing all the use cases described in table I, highlighting the third, fifth, and sixth features. These were the use cases taken into consideration in the time of its development. The possibility of not only being able to monitor the UAV movement in realtime but also to collect and process any aerial pictures taken as soon as possible was the main interest of this platform.

VI. Conclusions

The main focus of this paper was to propose a robust platform that could fit the six use cases described in

the results section while accomplishing a reliable and secure system. It was necessary to research fields such as software design patterns, realtime communication, and time-series databases to achieve this goal. Along with this research, we found some tools that could satisfy at least one of our use cases, however, none could satisfy everything. During the process of development of this platform came across some difficulties, particularly in the video stream part. The stream of video from the camera present on the UAV was a challenge in regards to managing which user could access a specific video stream.

The applications for this platform are endless, the simple fact of connecting a UAV to a cloud platform opens a group of possibilities in terms of industrial automation, aerial surveillance, civil construction, and building maintenance, among others. By adding features such as multi-spectrum camera integrations and map processing it is possible to extend the applications of this platform for other fields such as crop health surveillance, cattle management, and vine gap detection. Lastly, there are still points to improve on the platform regarding the management and configuration of UAVs, currently, we are developing a feature to calibrate internal UAV sensors directly from the platform.

ACKNOWLEDGMENT

This work is supported by the European Regional Development Fund (FEDER), through the Regional Operational Programme of Lisbon (POR LISBOA 2020) and the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework [Project 5G with Nr. 024539 (POCI-01-0247-FEDER-024539)]. This project has also received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783221. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Belgium, Czech Republic, Finland, Germany, Greece, Italy, Latvia, Norway, Poland, Portugal, Spain, Sweden. In last this work was also supported by SECREDAS project, which received funding from the Electronic Component Systems for European Leadership Joint Undertaking (ECSEL-JU) under grant agreement nr.783119 and TeamUp5Gfuse, which received funding from the MSCA-ITN European Training Networks (MSCA-ITN-ETN) under grant agreement nr.813391.

REFERENCES

[1] Evşen Yanmaz, Saeed Yahyanejad, Bernhard Rinner, Hermann Hellwagner, and Christian Bettstetter. Drone networks: Communications, coordination, and sensing. *Ad Hoc Networks*, 68:1 – 15, 2018. Advances in Wireless Communication and Networking for Cooperating Autonomous Systems.

[2] A. Ollero, S. Lacroix, L. Merino, J. Gancet, J. Wiklund, V. Remuss, I. V. Perez, L. G. Gutierrez, D. X. Viegas, M. A. G. Benitez, A. Mallet, R. Alami, R. Chatila, G. Hommel, F. J. C. Lechuga, B. C. Arrue, J. Ferruz, J. R. Martinez-De Dios, and F. Caballero. Multiple eyes in the skies: architecture and perception issues in the comets unmanned air vehicles project. *IEEE Robotics Automation Magazine*, 12(2):46–57, 2005.

[3] L. Gupta, R. Jain, and G. Vaszkun. Survey of important issues in uav communication networks. *IEEE Communications Surveys Tutorials*, 18(2):1123–1152, 2016.

[4] Emad Ebeid, Martin Skriver, Kristian Husum Terkildsen, Kjeld Jensen, and Ulrik Pagh Schultz. A survey of open-source uav flight controllers and flight simulators. *Microprocessors and Microsystems*, 61:11 – 20, 2018.

[5] T. Dardoize, N. Ciochetto, J. Hong, and H. Shin. Implementation of ground control system for autonomous multi-agents using qgroundcontrol. In *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, pages 24–30, 2019.

[6] Flytos. https://flytbase.com/flytos/. Accessed: 12-02-2020.

[7] Daniel Silva, Gerard Toonstra, Henrique Souza, and Túllio Pereira. Qualidade de ortomosaicos de imagens de vant processados com os softwares aps, pix4d e photoscan. 01 2014.

[8] J. D. Renwick, L. J. Klein, and H. F. Hamann. Drone-based reconstruction for 3d geospatial data processing. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 729–734, 2016.

[9] How to verify that there is enough overlap between the images. https://support.pix4d.com/hc/en-us/articles/203756125-How-to-verify-that-there-is-Enough-Overlap-between-the-Images. Accessed: 12-02-2020.

[10] Webodm. https://github.com/OpenDroneMap/WebODM. Accessed: 12-02-2020.

[11] H. Surmann, R. Worst, T. Buschmann, A. Leinweber, A. Schmitz, G. Senkowski, and N. Goddemeier. Integration of uavs in urban search and rescue missions. In *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 203–209, 2019.

[12] Emad Samuel Malki Ebeid, Martin Skriver, and Jie Jin. A survey on open-source flight control platforms of unmanned aerial vehicle. 08 2017.

[13] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui. Micro air vehicle link (mavlink) in a nutshell: A survey. *IEEE Access*, 7:87658–87680, 2019.

[14] J. P. Matos-Carvalho, J. M. Fonseca, and A. Mora. Uav downwash dynamic texture features for terrain classification on autonomous navigation. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1079–1083, 2018.

[15] Ian Fette and Alexey Melnikov. The websocket protocol, 2011.

[16] Caio Ribeiro Pereira. *Aplicações web real-time com Node. js*. Editora Casa do Código, 2014.

[17] Aaron Martinez Romero. Ros/concepts, 2014.

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

[19] Lentin Joseph. *Mastering ROS for Robotics Programming*. Packt Publishing, December 2015.

[20] Angular long-term support. https://angular.io/guide/releases. Accessed: 12-02-2020.

[21] Angular rxjs integration. https://angular.io/guide/rx-library. Accessed: 12-02-2020.

[22] Dhanji R Prasanna. Dependency injection. 2009.

ANNEX I

Prototype User Interfaces

Figure I.1: First Prototype - Vehicles Page

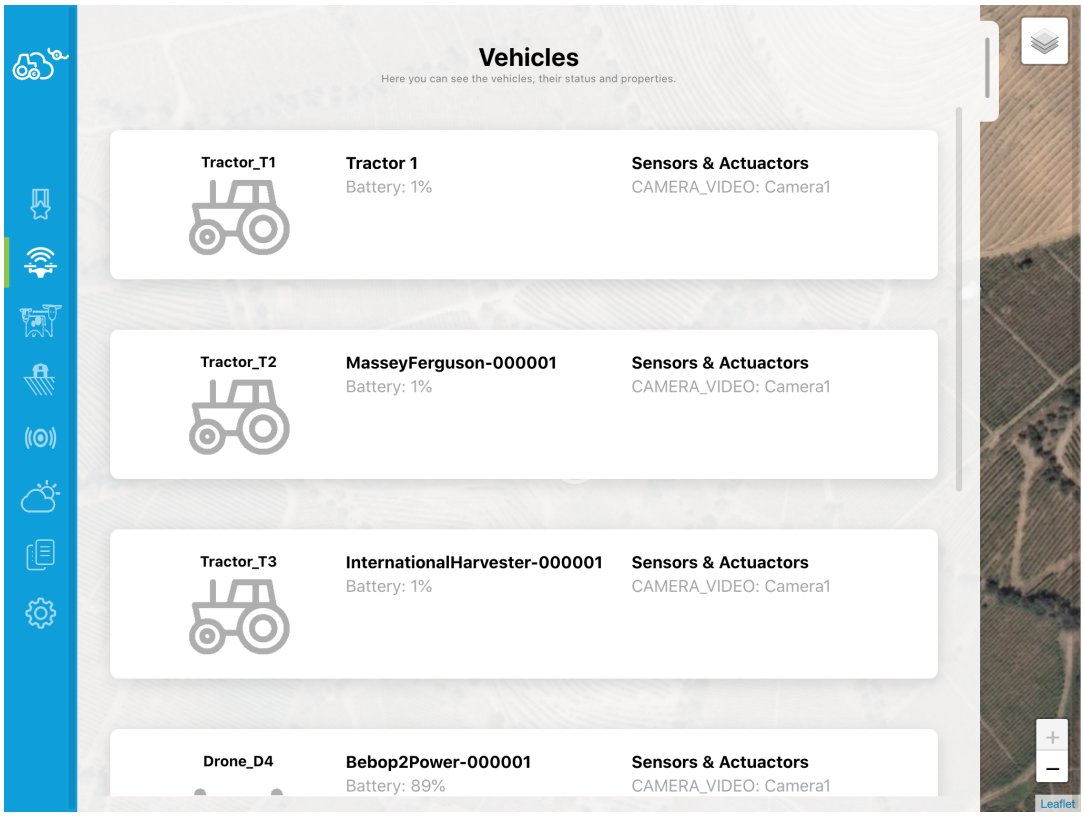

Figure I.2: First Prototype - Vehicles Menu Expanded

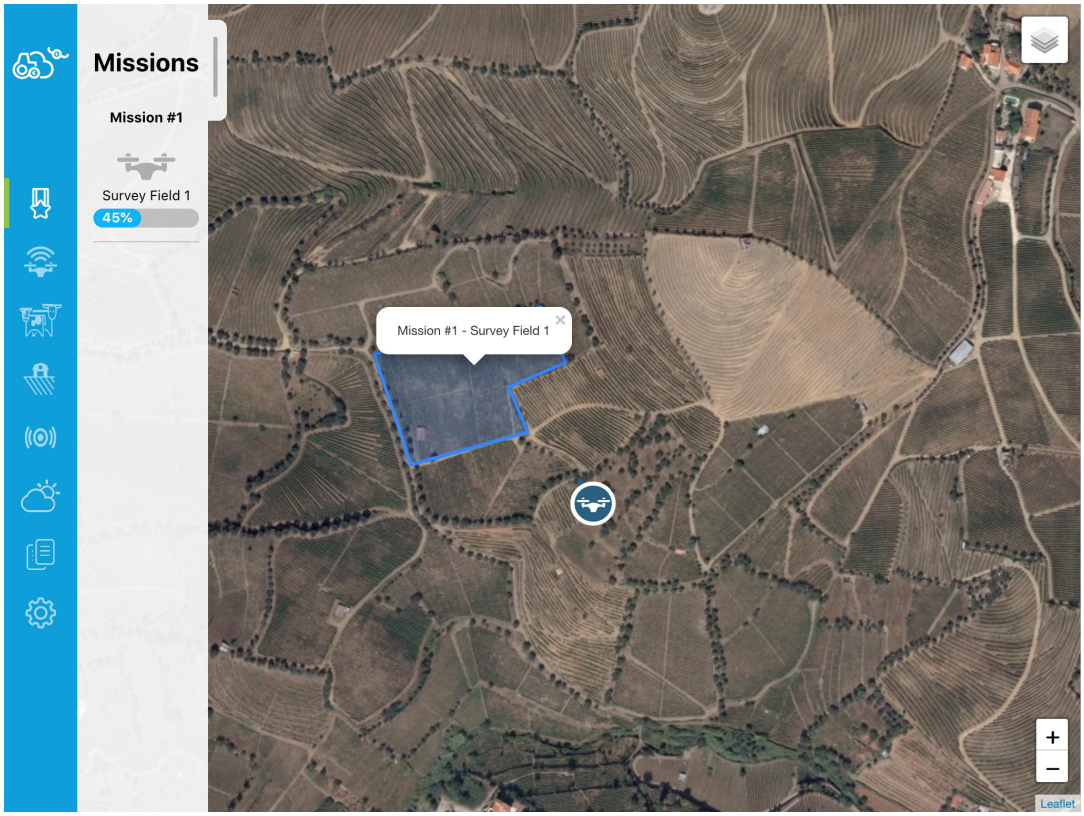

Figure I.3: First Prototype - Missions Page

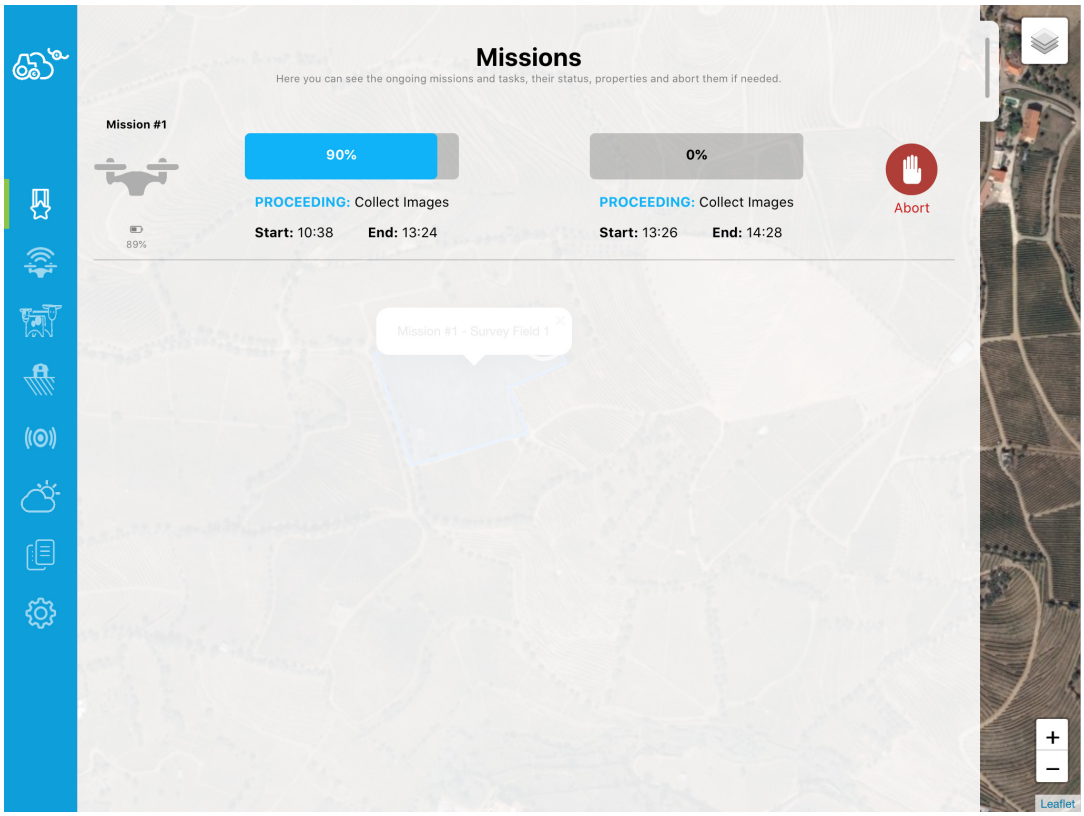

Figure I.4: First Prototype - Missions Menu Expanded

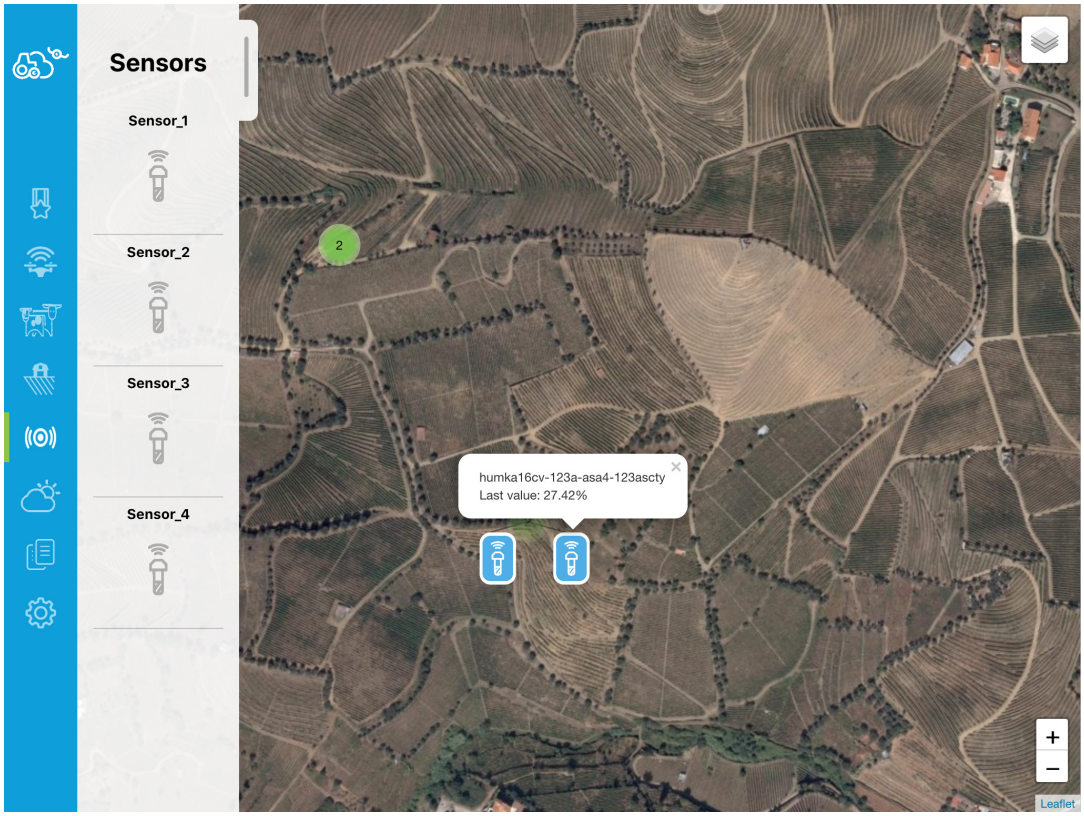

Figure I.5: First Prototype - Sensors Page

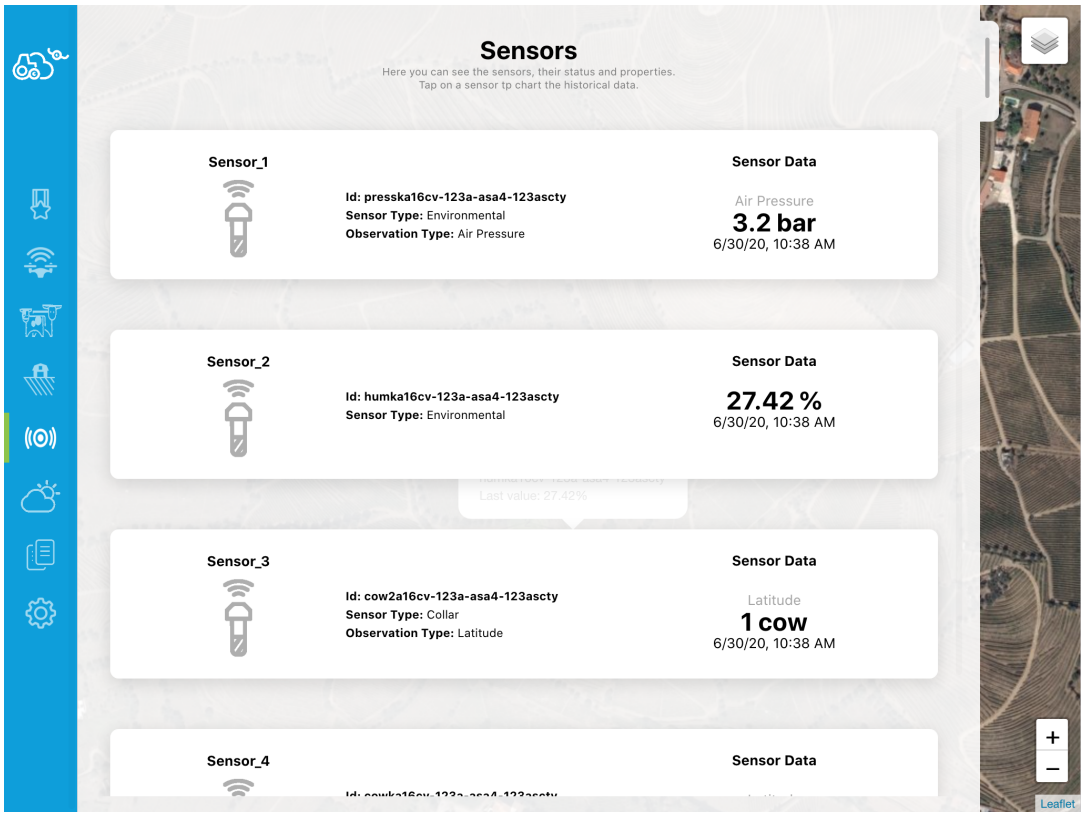

Figure I.6: First Prototype - Sensors Menu Expanded

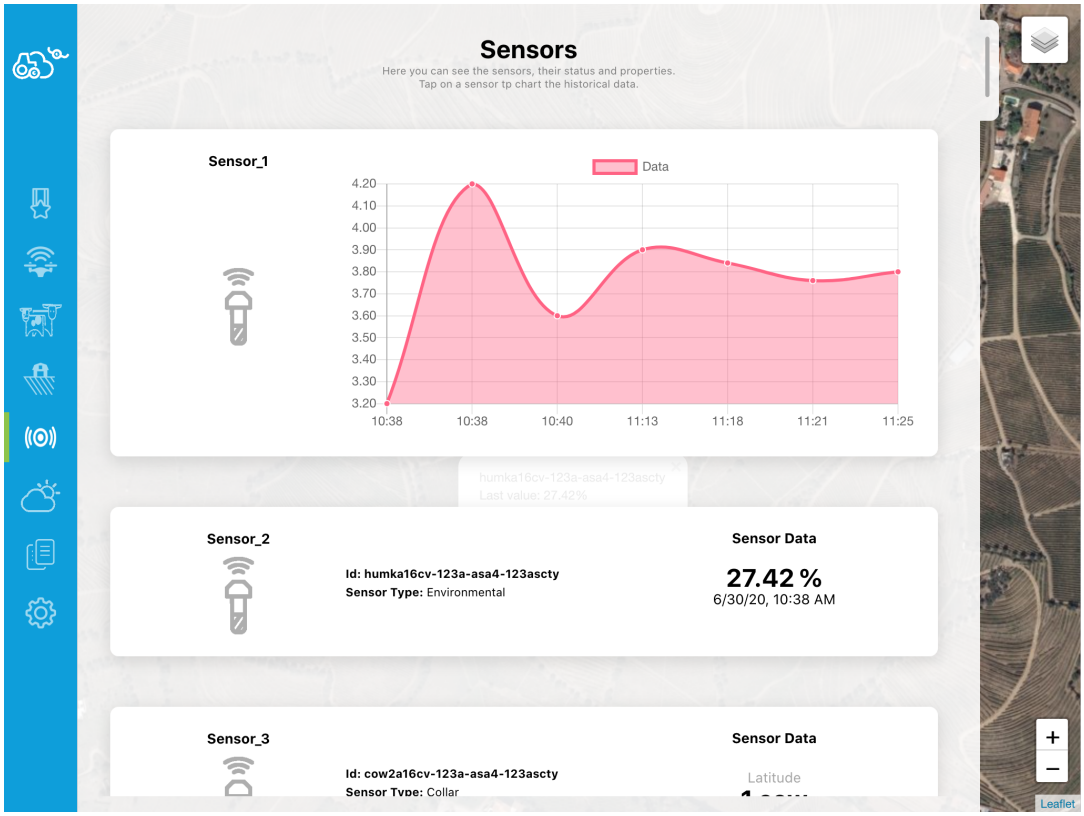

Figure I.7: First Prototype - Sensors Menu Expanded Graph

Figure I.8: First Prototype - Reports Page

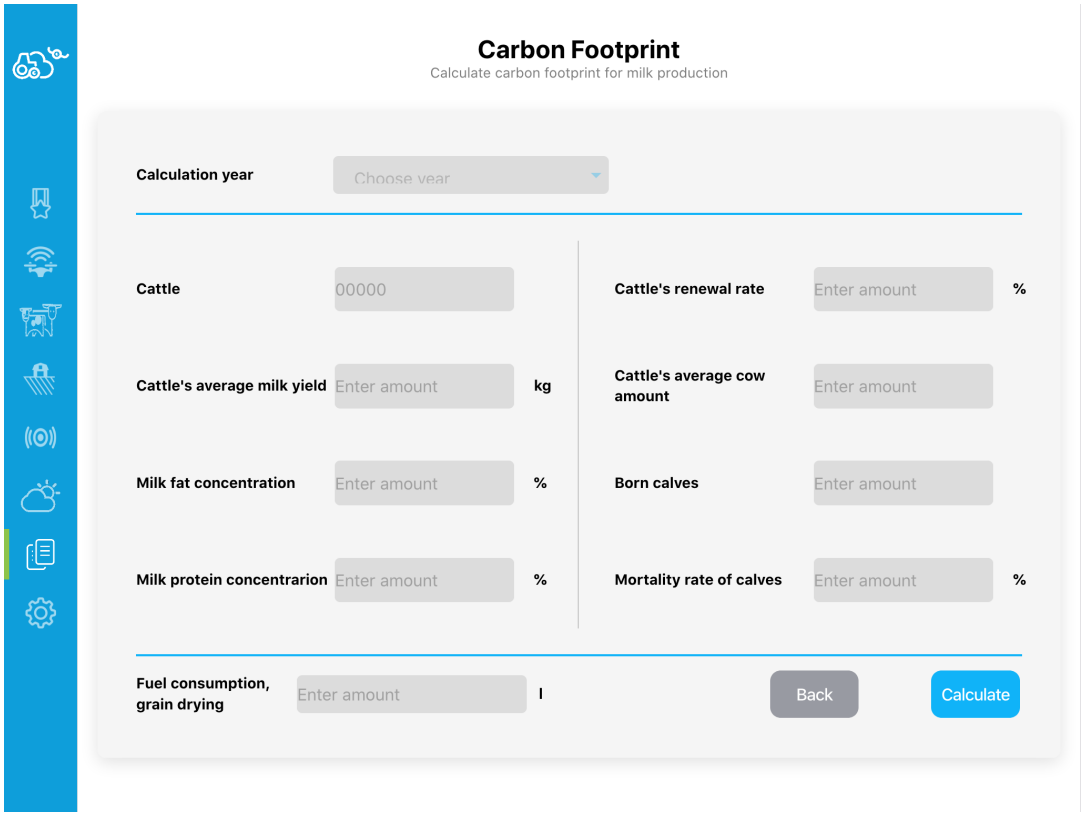

Figure I.9: First Prototype - Carbon Footprint Page

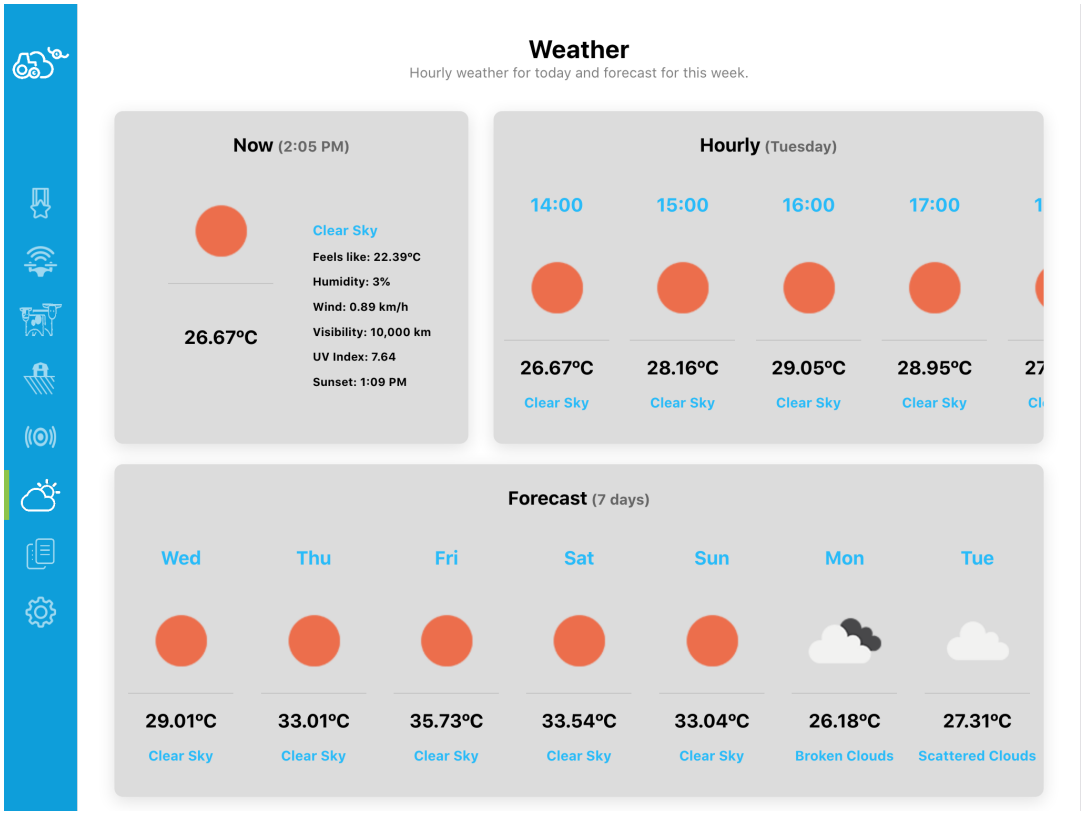

Figure I.10: First Prototype - Weather Page

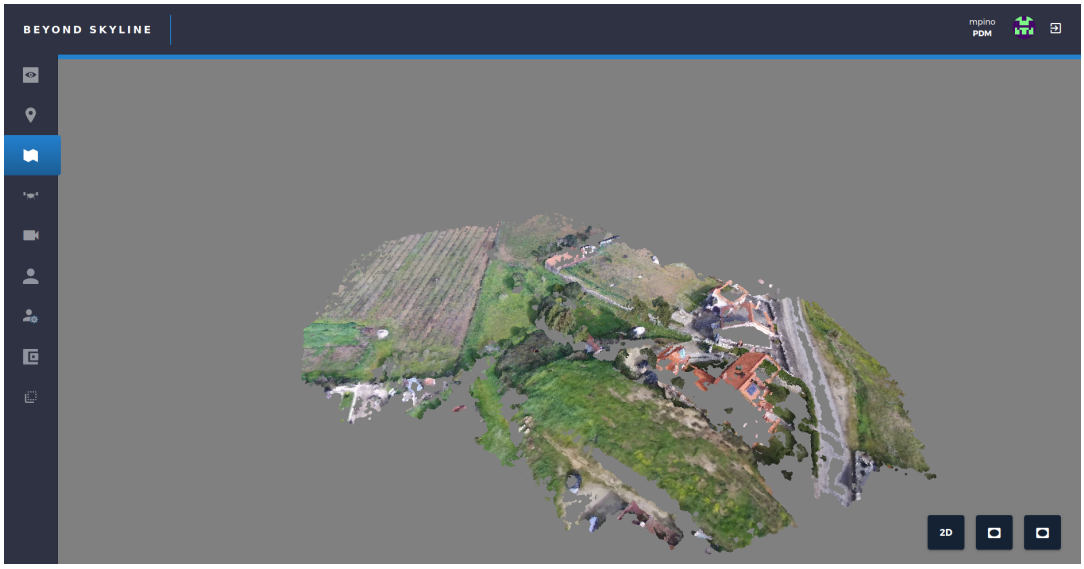

Figure I.11: 3D Map Visualization Interface