Ingénieur ENAC 20 - Systèmes Avioniques
ENAC 20 Engineer - Avionic Systems

Théo Evan LE GALL

IENAC 20

End of Studies Project Memoir

January to July 2023

Design, development, and validation of a web-based control application for drones

2023

ACKNOWLEDGEMENTS

Je tiens tout d'abord à exprimer ma profonde gratitude envers M. VALERO Miguel, mon maitre de stage, pour sa direction, sa disponibilité, et son soutien tout au long de ce projet mais également de sa bienveillance et son intérêt pour le bien être des étudiants lors de cette année.

Je souhaite également adresser mes remerciements à YICHENG Chang, mon binôme sur ce projet. Ensemble, nous avons affronté avec détermination les différentes difficultés rencontrées, et sa collaboration a été essentielle pour le succès de notre travail.

Je tiens à exprimer ma sincère reconnaissance envers HATTENBERGER Gautier, mon tuteur de stage, pour son encadrement précieux et sa vigilance pour s'assurer que le projet progressait de manière optimale et correctement.

Enfin, je souhaite exprimer ma gratitude envers mon papa pour son aide précieuse lors de la rédaction de ce rapport.

First of all, I would like to express my deep gratitude to Mr VALERO Miguel, my internship supervisor, for his guidance, availability and support throughout this project, as well as his kindness and interest in the well-being of the students during this year.

I would also like to thank YICHENG Chang, my partner on this project. Together, we tackled the various difficulties encountered with determination, and his collaboration was essential to the success of our work.

I would also like to express my sincere gratitude to HATTENBERGER Gautier, my internship tutor, for his invaluable guidance and his vigilance in ensuring that the project progressed optimally and correctly.

Finally, I would like to express my gratitude to my father for his invaluable help in writing this report.

En primer lugar, me gustaría expresar mi profundo agradecimiento al Sr. VALERO Miguel, mi supervisor de prácticas, por su orientación, disponibilidad y apoyo a lo largo de este proyecto, así como por su amabilidad e interés por el bienestar de los estudiantes durante este año.

También me gustaría dar las gracias a YICHENG Chang, mi compañero en este proyecto. Juntos hemos afrontado con determinación las diversas dificultades encontradas, y su colaboración ha sido esencial para el éxito de nuestro trabajo.

También quiero expresar mi sincero agradecimiento a HATTENBERGER Gautier, mi tutor de prácticas, por su inestimable orientación y su vigilancia para que el proyecto progresara de forma óptima y correcta.

Por último, quisiera expresar mi gratitud a mi padre por su inestimable ayuda en la redacción de este informe.

FRENCH ABSTRACT (RÉSUMÉ)

Ce rapport présente les missions de mon stage ainsi que les différentes implémentations effectuées dans l'équipe de recherche de l'université catalane, l'UPC. Tout d'abord, le projet global est de pouvoir créer à partir d'une interface web, qui ne nécessite donc pas de téléchargement, un plan de vol sur une surface choisie, d'y prendre les photos aux points adaptés pour permettre l'assemblage des images et générer une cartographie de la surface.

Pour ce projet, nous travaillions à deux étudiants internationaux, un élève chinois et moi-même, ainsi qu'un professeur encadrant.

Nous nous sommes divisés le travail pour ce projet. J'ai travaillé sur la création de l'application web grâce au Framework Vue.js combinant HTML, CSS et JavaScript. J'étais en charge de la création du serveur d'application et de ses applications, notamment à partir des sommets de la surface à scanner et des distances entre les photos et, générer un plan de vol ainsi que des points sur lesquels nous devions prendre une photo. Pour connecter ces deux implémentations entre elles, l'application web ainsi que le serveur d'application, j'ai dû mettre en place un système de communication. Nous avons choisi d'utiliser un courtier MQTT avec les protocoles MQTT et WebSocket, cette communication m'a aussi servi à connecter l'autopilote et la caméra du drone à mon application web.

Dans ce rapport, je présente aussi quelques éléments de l'assemblage d'image même si ce n'est pas ma zone de travail dans ce projet. J'y présente ce que j'ai appris, le fonctionnement de l'assemblage d'images, quelques problèmes rencontrés et les différentes solutions que nous avons trouvées pour les parer.

Vous aurez aussi un aperçu des différents tests effectués, que ça soit en simulation, ou en vol dans une volière extérieure de l'université.

Mots clés : Drones, Application Web, Serveur d'application, Assemblage d'images (Stitching), Plans de vols, Développement, Python, Courtier MQTT / WebSocket

ABSTRACT

This report presents the tasks of my internship as well as the various implementations carried out in the research team at the Catalan university, the UPC. First of all, the overall project is to be able to create a flight plan over a chosen surface using a web interface, which does not therefore require downloading, and to take photos at suitable points to enable the images to be stitched together and generate a map of the surface.

For this project, we were working as two international students, a Chinese student and myself, and a teacher supervisor.

We divided up the work for this project. I worked on creating the web application using the Vue.js framework combining HTML, CSS, and JavaScript. I was in charge of creating the application server and its applications, in particular using the vertices of the surface to be scanned and the distances between the photos, and generating a flight plan as well as the points at which we were supposed to take a photo. To connect these two implementations, the web application, and the application server, I had to set up a communication system. We chose to use an MQTT broker with the MQTT and WebSocket protocols. I also used this communication to connect the autopilot and the drone's camera to my web application.

In this report, I also present some elements of image stitching, even though this is not my area of work in this project. I present what I've learnt, how image stitching works, some of the problems we've encountered and the various solutions we've found to overcome them.

You'll also get an overview of the various tests carried out, both in simulation and in flight in an outdoor aviary at the university.

Keywords: Drones, Web application, Application server, Image stitching, Flight plans, Development, Python, MQTT / WebSocket broker

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF ANNEXES

INTRODUCTION

Drones have rapidly become part of our daily lives since they first appeared on the scene. Autonomous aircraft have demonstrated their versatility and adaptability by operating in a variety of fields and performing a variety of tasks. Drones have become increasingly important in recent years and have become an essential tool for many professional tasks. They are now used in many fields, in addition to the military and leisure sectors, such as agriculture, industrial inspection, environmental monitoring, mapping, logistics and many others.

However, although the use of drones has expanded significantly, it does present certain obstacles. These include the complexity of the applications needed to control them effectively. It is not uncommon for operators to have to download and manage multiple applications to perform a particular task, which can cause problems and discomfort. This issue needs to be addressed and resolved in order to simplify and improve the experience of using drones.

This is the wish of the team I work with. They want to simplify the use of drones to make them accessible to a wider public and eventually make their use fun in other projects. To achieve this, they are working on an ecosystem dedicated to drones called the Drone Engineering Ecosystem. This ecosystem is a software tool comprising a set of applications, functions and modules enabling a drone to be controlled in different ways and for different operations.

The project I'm working on aims to create a web application, which requires no downloads to operate, to control drones and assign them missions of varying complexity. This web application has to be as simple as possible and bring together as much information and monitoring as necessary for the flight. This application must be able to integrate with the ecosystem mentioned before without restricting the other applications usable within it.

The first mission, linked to this web-application, is to stitch together images taken by a drone. The aim is to create a flight plan adapted to the desired parameters, send the commands to the drone's autopilot and to the camera. The drone has to carry out the flight plan and take the photos at the appropriate GPS coordinates and then enable the stitching on the ground.

We are two students working together on this project. I'm in charge of the web application and flight plan, while Chang, the second student, is working on image stitching. I'll explain our roles in this project in more detail later in this report.

READING GUIDE

Creating an application from start requires a comprehensive understanding of each of its component parts. My understanding of the whole system has been defined over several months and I am aware that understanding in one reading, even a thorough reading, of a report like this can be tricky.

However, I will try to vary the levels of reading in my writing, to allow everyone to understand the project and its details at their own level. I will also try to simplify some of my comments to make them more accessible and to detail others to provide details that may be of interest to other readers.

To make it easier for everyone to read, I advise you to read the introduction and the contextualisation before embarking on the main body of the report in order to fully understand the expectations of this project, as well as the choices made throughout it.

In addition, you will find at the end of this report a glossary and a lexicon to help you with any abbreviations and definitions that you might miss.

1. CONTEXTUALISATION

In this part, I'm going to give you the context of this project and this placement. I'll start by presenting my working environment, the school, the research group and our team on this project. I'll then go on to give you details of the expected objectives and the elements we had to take into account during our implementations. Then I'll briefly explain why I chose to do this internship, and why with this team. I'll present the work I've done during this internship and on this project, while remaining concise as this work will be detailed later. Finally, I will present the organisation of this report and why I decided to do it in this way.

1.1. Working environment

I started this final year project in the ICARUS (Intelligent Communications and Avionics for Robust Unmanned Aerial Systems). This team is attached to the research centre of the UPC (Universitat Politècnica de Catalunya) and more specifically the EETAC (Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels).

As you can see in Figure 1, the UPC is a large group of schools with many schools in different Catalan cities.

Figure 1: UPC campuses

The EETAC, whose logo is shown in Figure 2, is one of the schools on the Castelldefels campus, a city located in the south of Barcelona.

Figure 2: Logo of the EETAC institution, a UPC school on the Castelldefels campus

This team is made up of researchers from the Technical University of Catalonia - Barcelona Tech (which is another name for the UPC) and is a member of the Aeronautics and Space Research Centre (Centre de Recèrca per l'Aeronàutica i l'Espai, CRAE).

This team is mainly made up of the Department of Computer Architecture and the School of Telecommunications and Aerospace at Castelldefels.

These (scientific) faculties are permanent, and the researchers work full-time, but doctoral and master's students can join the team on a full-time or part-time basis to take part in the various project.

ICARUS began its activities in 2005, and the group quickly became a benchmark in Catalonia, being recognised by the Catalan government in 2009.

The ICARUS group is multidisciplinary, and its members include computer scientists, aeronautical engineers, and telecommunications engineers, all of whom have research experience in their respective fields.

I've joined the UAV department, and we're working together to take this project forward. We have at our disposal an open-air aviary called the Drone Lab, to carry out our test flights and test our new functionalities.

For this project, we're a team of 3. Miguel supervises and coordinates us, he also helps us to find solutions when we get stuck on certain issues, helps to find the equipment, and liaises with the other entities, for example to book slots in the drone lab. And I'm working jointly with a Chinese student, Chang, so we've divided up the overall project between us and are pooling our progress on a regular basis.

1.2. The needs and objectives

An ecosystem already exists, which I will present in the following section (2.1.), but what was missing was a web application that would allow communication with this ecosystem and the drones via the Internet. The aim of this web application would be to make drone use easier by removing the need to download drone control applications, and to be able to centralise everything needed in the same web application.

This leads us to clear objectives such as: creating an intuitive graphical interface to make it easier for everyone to use, creating an application server and its functions to meet users' needs, having a reliable communication system between the drone(s) and the web application but also between the application server and the web application so that we can have confidence in this implementation. While maintaining the central focus of the project, we need to be able to integrate into the existing ecosystem so that we can use existing applications and add new ones after we leave.

The first application to be implemented would be capable of using a drone to take photos and reconstruct images of a pre-established surface. The stitching technique, which involves assembling the various photos taken by the drone to form a complete image of the surface, would be used for this application.

1.3. My motivations

I was more interested in this end-of-study project than any other for several reasons. This project allowed us to be like a project manager, we could see all the aspects and follow all the different developments at each stage of the process. We can manage and give instructions for the elements that concern us and, on the other hand, help others with their elements of the overall project.

The team is international, so learning to communicate even with language barriers is very interesting, and learning to be clear, understand and be understood was an important part of this end-of-study project, which I think will be useful to me in the future. Secondly, we can see all the actors in the application production chain. Since we're the engineers who decide what to do, we make and test, not forgetting that we also build the drone with which we fly, which gives us complete control over our operations and developments, I'd sum it up as a global vision of the product we're creating.

Finally, we are very autonomous on this project, we propose the evolutions that we want to implement, the experiments that we want to try out, and we are free to continue in this direction or to choose another.

1.4. Overview of my work

In this section, I'm going to give you an overview of what I've done on this project. I'll go into more detail about each part of my work in the development of this report.

In this project, I was in charge of several different parts. I created this man-machine interface in Vue.js, a framework that makes it easy to combine codes from different languages, for example, in my case HTML, JavaScript and CSS. I also took care of the flight plan part, the idea being that from points of a polygon you could scan the whole surface to take photos spaced at a set distance defined as a function of a certain overlap, necessary to reconstruct an image from the smallest ones. Then for the drone to carry out the required flight plan I had to adapt an autopilot code in the MAVLINK language, not forgetting that this code had to work in simulation mode to test the overall operation of the application.

And finally, I took care of communications between the man-machine interface, the application server (creating flight plans in our case), and the drone's autopilot executing the flight plan created. At the end of the project, I did a lot of code simplification, debugging and added simplifying functionalities for operations.

I'd like to add that this project was first and foremost a team effort and that we exchanged opinions every week on each other's parts and attended all the tests together. Communication between the three people involved in the project, Chang, Miguel, and me, was an essential and fundamental element of the project and this strong cohesion contributed greatly to the overall success of the project.

1.5. Organisation of the report

I've chosen to organise this report by work theme, in other words, to group together elements that are similar in terms of their functions. In particular, grouping together everything to do with the web application in the same section, then grouping together the server and its applications in another section, as they were coded in the same language and are very similar in terms of operation and functionality. I've included a separate section on communication, in particular how the brokers work and what they do, because that's the link with the rest of the ecosystem, as well as with the web application and the application server. I've decided to separate the stitching, for the same reasons, because the subject is different and the tests and flights, because they concern the whole project, and I didn't want to divide the flight elements between the different parts of the project.

In each of these areas, I have chosen to adopt either a 'funnel' approach, starting with generalities before gradually refining my ideas as I go along, or to follow the chronology of my discovery and creation.

2. DRONE'S ECOSYSTEM AND HARDWARE

In this section, I'm going to give you a more detailed presentation of the drone and its components, as well as the stages involved in its construction. We worked on this drone during the project, which enabled us to test the application's various functions during simulations and flights. To give you a better understanding of the project as a whole, I'll also describe the ecosystem into which our drone and our project fit, highlighting the interactions and influences involved. Finally, I'll describe the elements of this ecosystem that I'm involved in during this project.

2.1. Hardware

In this section, I'm going to show you the various components of the drone used for this project and how they are integrated.

First, we received a kit, along with instructions on how to build it. In the kit, we had a Hexsoon box containing the structure of the drone in parts, a radio, and its related components (receiver, cables), an Orange Pixhawk autopilot, a LiDAR (laser altimeter), a Raspberry Pi kit, a GPS receiver, and brushless motors.

Firstly, as shown in the Figure 3, we assembled the frame, which contains the central body of the drone with the power distribution board, the 4 arms of the drone and the motors.

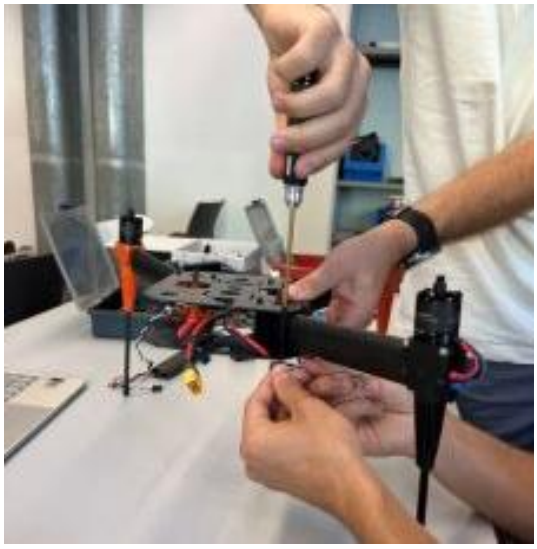

Figure 3: First assemblies of the drone

In this stage we need to pay attention to the placement of the motors, putting those turning clockwise face to face and those turning anti-clockwise in the same way, as shown in the Figure 4, this placement allows us to control the drone in flight afterwards, moreover we need to check that the direction of rotation was as expected and change the wiring accordingly.

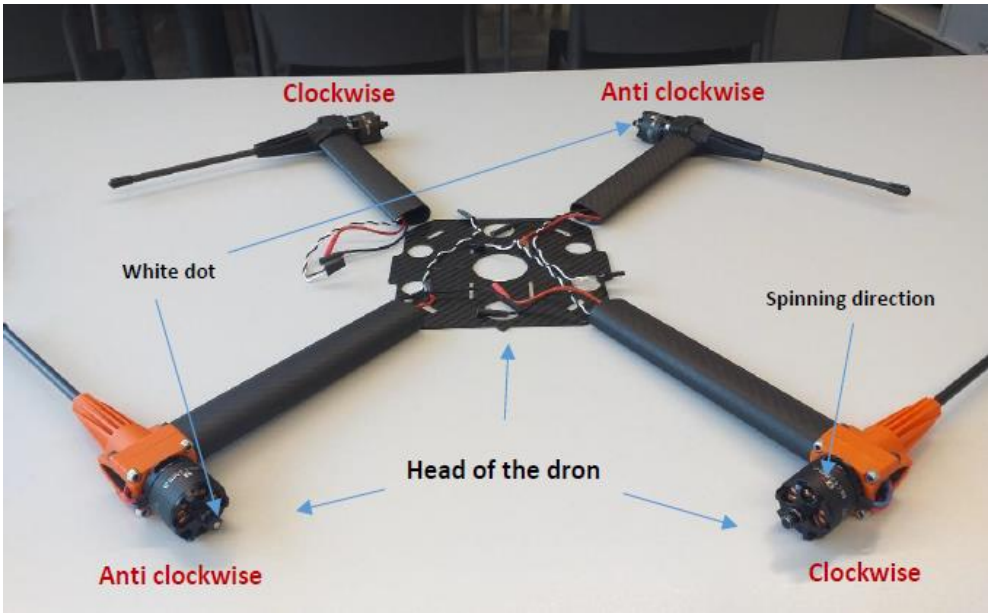

Figure 4: Positioning of motors with their respective rotation

Secondly, we calibrated and configured the radio remote control, shown in Figure 5, to establish a reliable connection with the drone. To do this, we used software to calibrate the joysticks and dials, as well as creating a flight model tailored to our needs. Once the configuration was complete, we were able to confirm the connection between the remote control and the drone.

Figure 5: Radio remote control used.

Then we set up and calibrated our autopilot, shown in the Figure 6, which is the drone's main computing unit. It translates the receiver signals into PWM signals for each engine, allowing it to perform all manoeuvres and flight operations. Different steps were followed to configure the autopilot, I will present the main ones.

Figure 6: Autopilot Pixhawk Cube Orange

Firstly, the initial configuration of connecting the buzzer, the GPS module, and the receiver to the drone. We downloaded and installed Mission Planner, our main configuration and control tool throughout the project.

Then, the management of the bus parameters is to ensure that the sensors are working properly. Like activating the GPS CAN 1 bus, configuring the GPS type, and verifying the operation of the GPS module LED lights were all included in this task.

After that, the accelerometer and compass calibration with Mission Planner. This step was crucial to ensure an accurate estimate of the drone's position and navigation.

The calibration of the radio control, which consists of recording the high, low, and middle positions of each channel. We have also changed the way switches are used to arm the drone.

And finally, the configuration of flight modes that offer specific features for each drone flight.

Furthermore, we needed an on-board computer to calculate distances and positions, and to carry out precise commands such as taking a photo at a defined position. So, we added a Raspberry Pi to our drone.

Here is a picture of the final appearance of the drone in the Figure 7.

Figure 7: Final appearance of the drone

2.2. Ecosystem

In this section, I'm going to take a closer look at the constantly evolving ecosystem created by the UPC research team. Below is a diagram of the general architecture of this ecosystem:

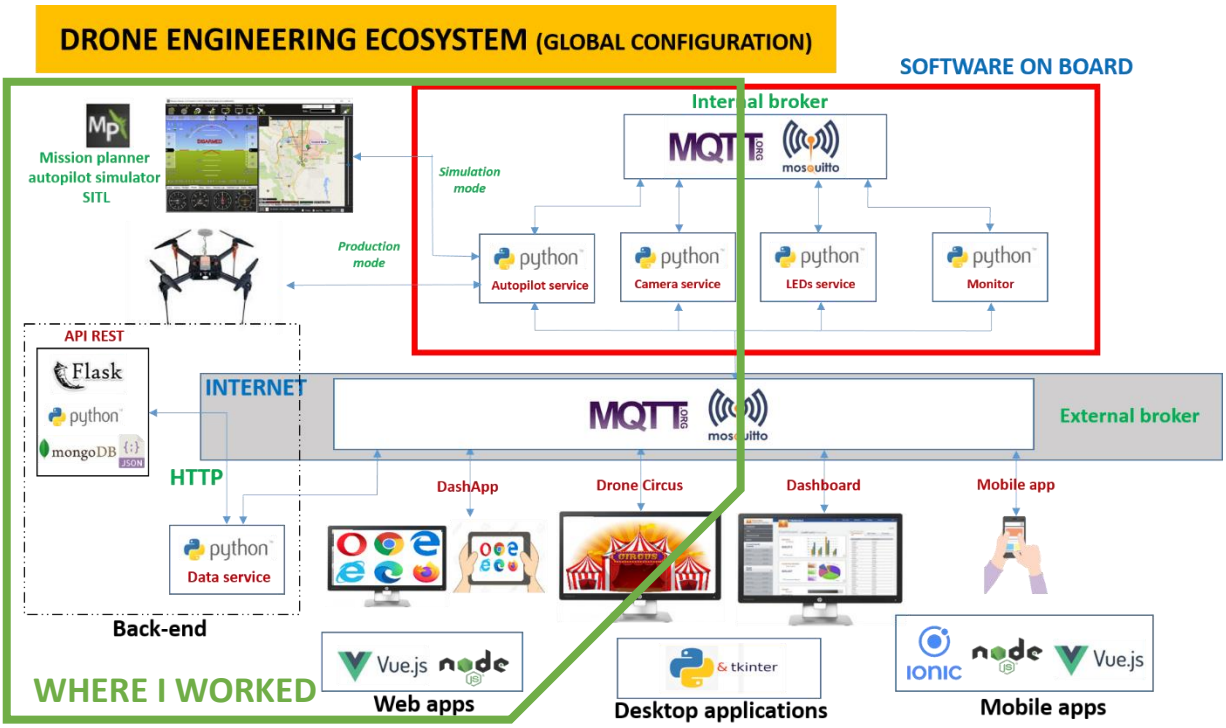

Figure 8: Drone Engineering Ecosystem's Diagram

Firstly, we can see that this ecosystem is separated into 2 distinct parts: the on-board software part, and the ground software part.

The on-board software part, framed in red in Figure 8, concerns, as its name suggests, what is on board the drone. A few exhaustive examples are presented, including the autopilot, camera, and LED services. We could add all the services used by the drone, such as the LiDAR and the GPS receiver, which are used in this project.

The second part, the ground software, covers everything else. These are all the applications, functions and systems that enable the use or operation of drones, as well as their surveillance and observation.

There is one final part to this ecosystem: brokers. There are 2 different types in this ecosystem: the internal broker and the external broker. The internal broker, individual to each drone, is used to relay information between the drone's various services, for example from the autopilot to the camera to give the order to take a photo at a particular position. The second broker, which is external to the UAV and therefore called the external broker, is used for all communications between entities in the ecosystem. It can be used to request the creation and reception of a flight plan from the application server, or to search for data in the database

without forgetting that it also concerns communications with the drone, in particular flight controls.

In this report I present the creation of the "DashApp" shown in the diagram. This creation will obviously be a ground-based system, but it will affect all the elements of this ecosystem, in particular the brokers for sending and receiving the data to be shared between the elements, the 'Back-end' which concerns all the servers and where my applications such as the creation of flight plans will be stored, and also the on-board system, since it affects the drone's autopilot and should enable the drone to be controlled and its telemetry data to be received so that it can be displayed on the screen.

2.3. What parts of the ecosystem am I involved in?

So, during this project I worked with several elements of this ecosystem, if not almost all of them. I have summarised it in a green rectangle in Figure 8.

Firstly, I created the whole 'DashApp' part, which I called the web application in my project. Then I worked on the 'BackEnd' part, creating a server capable of responding to requests from the web application. All the server applications will also be stored there. All my communications are via the external broker, so I also worked with this element. And finally, of course, I worked with the drone for its construction and for the various experiments. More specifically, I worked with the autopilot and the camera service.

3. ORGANISATION

I'm now going to give you a detailed description of our organisation throughout this project. I'll start by presenting our methodology. It's inspired by Agile methodologies, which are widely proven and used in modern project management. Then I'll go on to show you how we've divided up the tasks within our team.

3.1. Agile methodology

We chose to use a working method similar to agile methods for our project. These methods are ideal for projects that require great flexibility and rapid implementation. Collaboration and transparency are essential components of these techniques. We are both customers and developers for our project, and over time we define the product precisely. At each stage of the project, we carry out regular tests to check the quality of each 'sprint' and correct any deviations. Although this takes a little longer, we also have the option of changing our minds and modifying elements if this improves the overall quality of the project. If we think we've reached an impasse, we can also go back. So, because they offer great flexibility and an iterative approach, agile methods are ideal for managing complex projects with uncertainties.

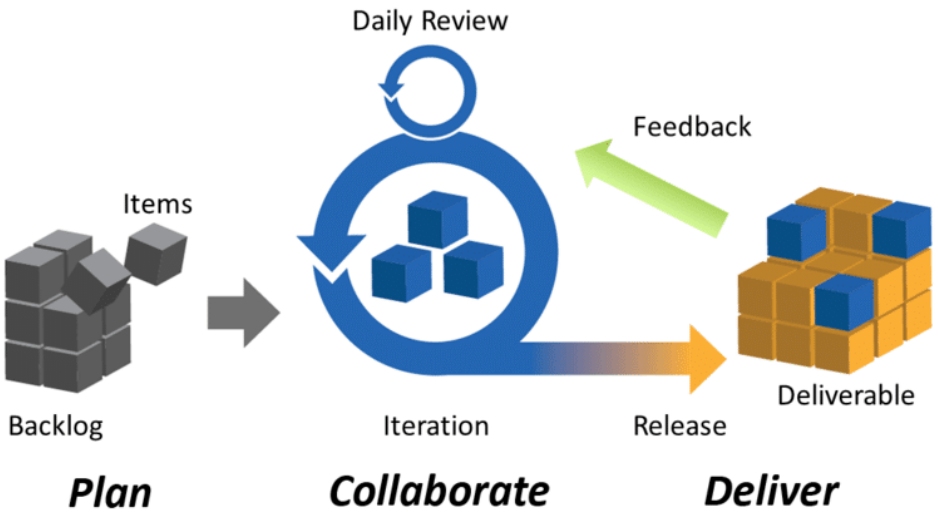

Figure 9: Diagram of the Agile methodology (taken from langerman.co.za)

3.2. Division of tasks

In this section, I'm going to tell you how we organised our work during this project and above all how we divided up the tasks.

As a reminder, the overall project consisted of creating a photo mapping of a defined area on the ground using a web application (i.e., not downloaded but online).

For this project, our team consisted of Miguel Valero, the project coordinator and organiser, Chang Yicheng, a master's companion, and me.

The three of us decided to divide up the tasks as follows:

Chang worked in the broadest sense on the stitching of images, which involves managing the photo-taking, what parameters are used to ensure the visibility and quality of the photos so that they can be stitched together. He will also manage the creation of the algorithms needed for stitching.

For my part, I worked on all the flight plan management and web application and services that I will present to you in the rest of this report.

I'll also be doing a section on image stitching, as I've had to understand how the images fit together, to reflect on the problems we've had in order to find appropriate and workable solutions, and also because our work is very closely linked.

4. WEB APPLICATION

A web application is a software application that can be accessed via a web browser. Unlike traditional applications, which require installation on a specific device, web applications are hosted on servers and can be used from any device with an Internet connection and a compatible browser.

Web applications are developed using web technologies such as HTML, CSS, and JavaScript.

HTML defines the hierarchy of elements such as headings, paragraphs, images, etc. It is used to create the overall structure of the website.

CSS is used to control colours, fonts, margins, sizes, animations, and other visual properties to give the website a consistent and aesthetic appearance.

JavaScript is used to add interactive and dynamic features to a website. It can be used to manipulate and modify HTML content, handle events such as mouse clicks or form submissions, make HTTP requests to retrieve data in real time, and much more. JavaScript brings the website to life by adding custom behaviours and functionality.

Web applications can offer a wide range of features and interactions, from simple online forms to more complex and interactive applications such as social networking, online banking and more.

They are unique in that they provide a dynamic and interactive user experience. They can be easily updated on the server side, which makes it easier to deploy new features or fix bugs without having to update user devices.

I used the Vue.js framework for this project, which is a popular open-source framework used to develop interactive and responsive user interfaces. It is designed to simplify the process of creating web applications by providing an organised structure.

Vue.js adopts a component-based approach, which means that the user interface is divided into reusable, self-contained components. Each component manages its own logic and can be combined with others to form a more complex application. This makes it easier to maintain, test and reuse code.

One of the main features of Vue.js is its responsiveness. Thanks to its bidirectional data binding system, changes made to the data are automatically reflected in the user interface, and vice versa. This makes it possible to create dynamic applications where users can interact with the data in real time without having to refresh the page.

4.1. Creation of a website

During this project, I had the opportunity to deepen my skills in website creation, based on the knowledge acquired through tutorials available on YouTube. These courses were created by a professor of the university and focused mainly on the use of Vue.Js, the framework I talked about before, combining HTML, JavaScript, and CSS.

Through these instructive videos, I was able to explore various aspects of Vue.Js, such as its structure, its layout system, Bootstrap integration for responsive design, table manipulation, emitters for event management, pop-up display, the creation of sliders (Range), the use of radio buttons (Radio) and checkboxes (Checkbox).

In addition, the tutorials covered more complex topics such as integrating MQTT messaging for real-time communication, streaming video using OpenCV, creating interactive maps using Leaflet and Mapbox, and manipulating markers, polylines, and distance calculations.

These apprenticeships enabled me to develop my technical skills in the field of website creation, familiarising myself with modern technologies while staying within the realms of programming and exploring various advanced functionalities. They provided invaluable inspiration for the rest of the project.

On top of that, I had the role of giving a critical opinion on this tutorial, noting the complexities, what could be simplified or clarified without forgetting the different versions to downgrade of each necessary installation. I've listed my comments and suggestions for changes that might prove useful for future students who will be doing this tutorial.

4.2. Evolution methods

Over time, I developed my website to meet the growing needs arising from the overall progress of the project. This evolution took place gradually and in iterations, allowing me to adapt the site to the new requirements identified through the flights and simulations carried out. My main objective was to simplify the application in order to provide an optimal user experience.

I used various tools to achieve this. To begin with, I exploited specific function libraries that enabled me to implement advanced functionality. These libraries, rich in almost ready-to-use functions, greatly facilitated the development of the site and accelerated its progress.

I also consulted documentations that guided me in the use of the various technologies required for my project. These resources provided invaluable information on the functionalities available and suggested methods for solving certain problems.

However, despite all this information, there can still be things that don't work as expected.

When I was faced with particular challenges or needed specific solutions, particularly during my debugging sessions, I tried to find tricks and use combinations of functions to avoid any more problems. To find answers to my questions and access concrete examples, I looked for online help by exploring the official websites of the libraries, functions, and applications I was using. Then I searched for, and sometimes found, extremely useful online tutorials that gave me detailed explanations and practical solutions for solving problems.

Finally, if I didn't get a satisfactory answer or no answer at all, I took part in question-and-answer forums where I was able to interact with experienced developers. These interactions enabled me to benefit from their expertise and receive valuable advice on how to solve the problems I encountered.

To sum up, the evolution of my website developed gradually in response to the growing needs associated with the progress of the project. I used various tools such as function libraries, documentation, online resources, and question forums. I was able to simplify the application and provide an improved user experience on my website thanks to this iterative approach and my determination to solve the problems encountered.

4.3. Final result

I told you how I learned to work in the Vue.js environment and how to create a website. In this section, I'm going to give you a brief overview of what I've done in this application.

The first page, in Figure 10, only has a "Connect" which turns into a "Disconnect" each time it is clicked, just to enter the application.

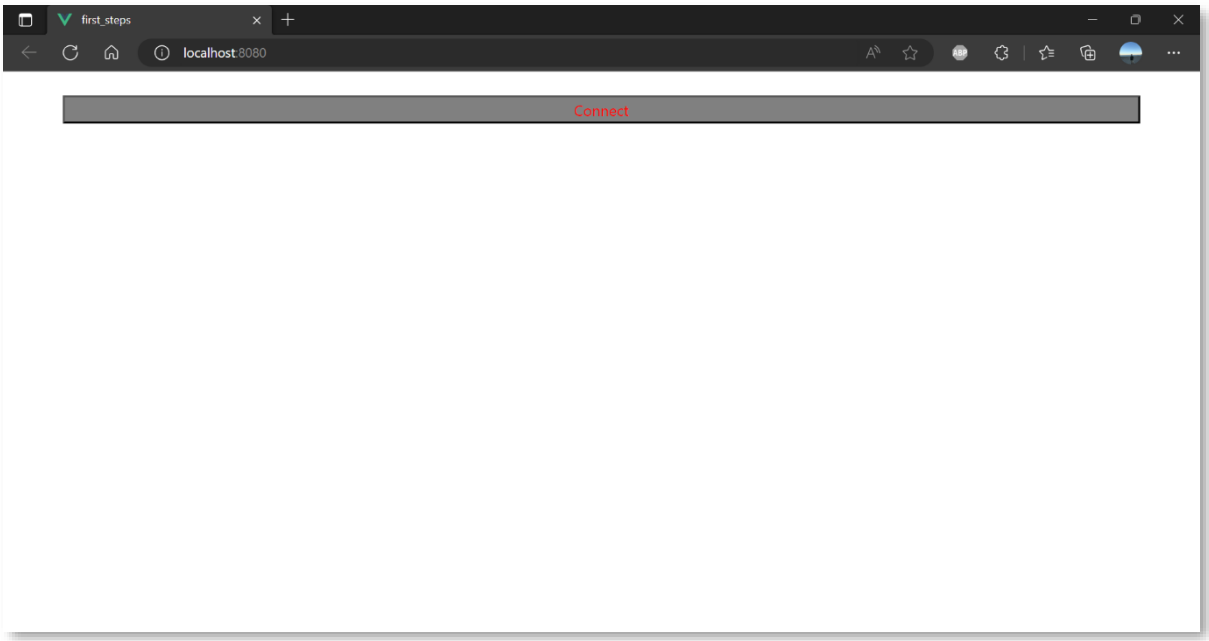

Figure 10: First page of the application: Connect - Disconnect

By clicking, we see the general architecture of the web application, in Figure 11, it still includes the elements of the tutorial that are not directly related to the use for drones, such as the questionnaire and user list, the video screen or a non-interactive map display. These elements have not been removed for two reasons: firstly, to help the website evolve by taking inspiration from what has already been done, and secondly because these elements could prove useful later in the project, for example if we want to add a different interface for each user or be able to film and view live from a drone's camera.

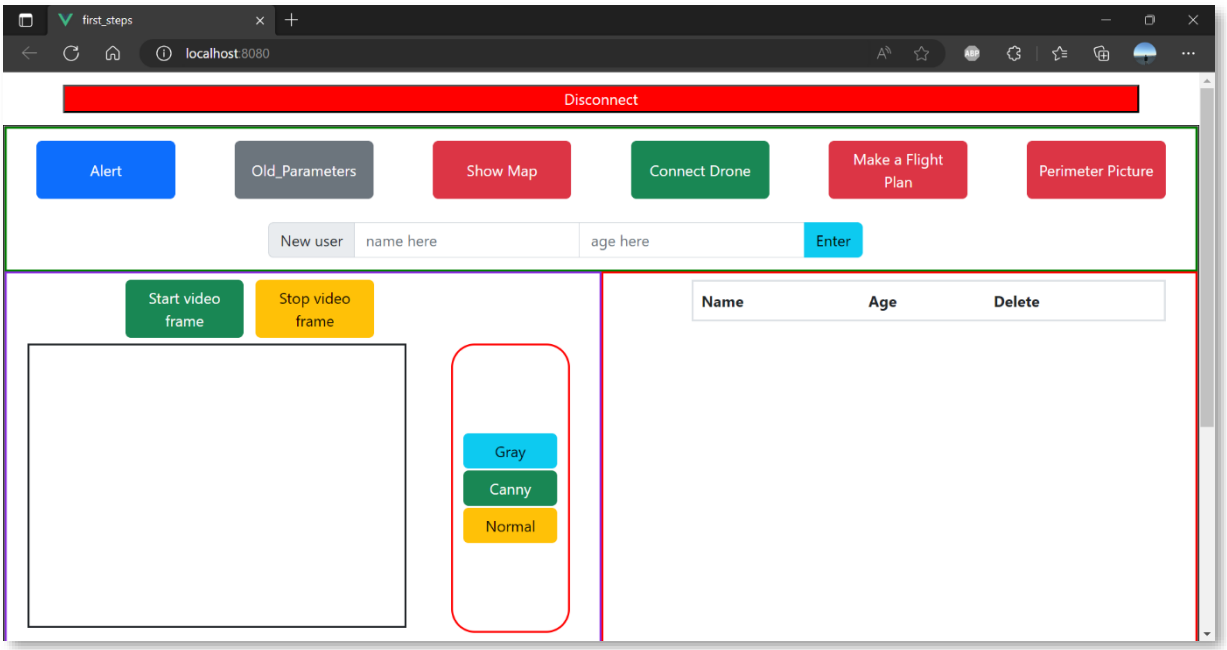

Figure 11: General architecture of the web application

I'm going to introduce you to the main application: creating a flight plan to take photos and stitch them together.

When we open the Pop-up, we can choose between several views of the application, in the Figure 12, the two you won't see are 'Drones and Flight parameters' and 'Photo Testing'. The first allows you to send various parameters to the drone that could be useful. This view was added after a series of flights that involved changing camera parameters in the code. This operation was long and tricky, and this function of the application was simple and quick to implement (at web page level).

The second view, which is also invisible, is used to take a photo and display it on the screen. It is used to test the camera settings before completing the full flight plan, which avoids wasting time by completing the flight plan with unnecessary photos.

The view you see in Figure 12, called "Map and Waypoints", is the main part of the application. I've selected the 'Parameters' view on the right to show you one of the functions. It automatically calculates the distances required between your shots. Depending on the horizontal and vertical openings of your camera (VFOV and HFOV), the desired overlap (horizontal and vertical overlap), and the height at which you are going to fly, you can calculate the necessary distance between images. The application will automatically write them in the box at the bottom left. Next, you need to click on the map and select the area you want to scan and calculate the flight plan that meets this combination.

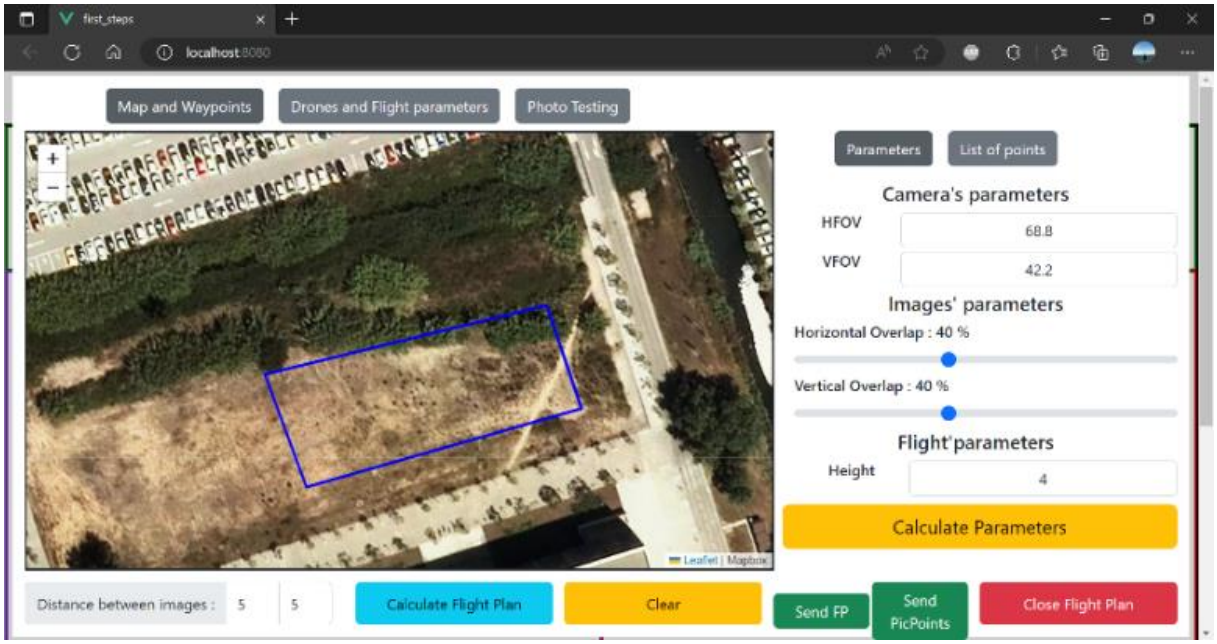

Figure 12: "Make a Flight Plan" Pop-up

On the map, on the same Figure, you can see a blue rectangle, it has no functionality. It serves as a simple visual reference on the map to have the flight limits imposed by our geofence.

Once you have calculated the flight plan that matches your requirements. The flight plan is displayed. In Figure 13, you can see a green line inside your defined area corresponding to the flight plan that is about to be executed. The points on the flight plan are listed on the right-hand side of the map. We can also select what we want to observe. This can be the points selected for the surface to be scanned, the points on the flight plan or the points for picture-taking.

The two green buttons are used to send the photo points to the camera and the flight plan points to the drone's autopilot. They have been separated to make it easier to resolve any errors. Thanks to this separation, if there's a problem receiving one of the two lists, we know quickly and easily which one it is.

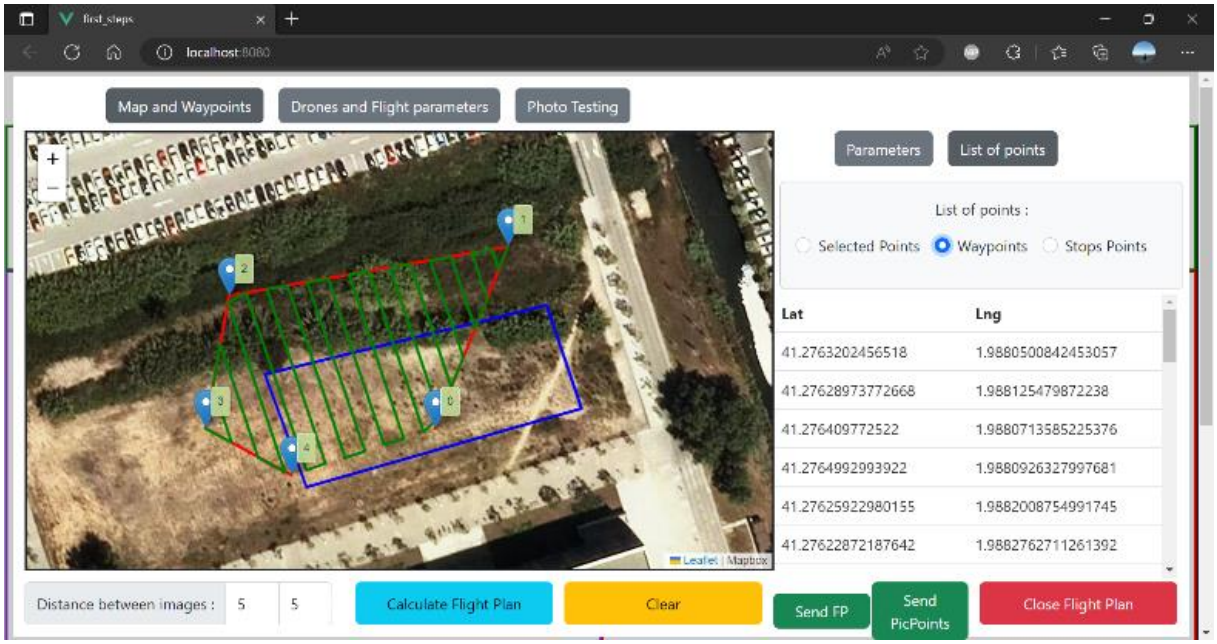

Figure 13: View of an executed Flight Plan in the web-application

4.4. Possible path of future development

This web application is still in its first stages and so offers many possibilities for future development. In particular, this could be to ensure its deployment online. One of the priorities would be to focus on the overall aesthetics of the site, creating an attractive user interface, choosing fonts, colours and images that match the desired vibe. This will give future users a positive and engaging first impression.

Even if the site is currently only accessible locally, it would be wise to ensure compatibility with other devices. This way, when it's ready to be rolled out online, it will be suitable for a variety of devices, such as phones and tablets, and offer an optimal user experience, whatever the device.

It would also be a good idea to improve the website's performance, in particular by optimising the code and compressing the files. This will ensure that future users have a fluid and fast experience when accessing the service online.

Finally, it's important to consider security. It's crucial to ensure that the site is robust in the face of potential threats, particularly if it allows one or more drones to be controlled.

By implementing these potential improvements, the web application will be ready to be deployed online in the future and will be able to offer an optimal user experience from its very first users.

What's more, over time, features can be easily added, just like those already implemented.

5. SERVER AND FUNCTIONALITIES

An application server is software that provides an environment for running application components. It is used to create web applications and a server environment to run them transparently. Application servers are server programmes that provide the execution environment for a programme. Web applications have a client-server architecture, where the server-side script handles data processing and the web application server handles and responds to client requests.

It is this server side and its applications that I am going to present to you in this section.

Firstly, I'll introduce you what was needed and how I decided to handle it. Then I'll present my first ideas, my first problems and show you the first implementations of these applications. I'll go on to show how I added possibilities for our flight plans and describe the problems I encountered and the solutions I found for each of them. I'll finish by explaining how my application server works.

5.1. What was asked for? How did I handle it?

We had to make an application capable of sending back a flight plan as well as a way of knowing where to take the photos to do our image stitching.

I chose the python language to the server and its applications for several reasons. Firstly, because it has a simple, readable syntax, it's a language with easy portability because it's an interpreted language, particularly for implementing it in a Raspberry pi, it's a language that has many usable libraries, and obviously to save time because I'd already learned to code in Python at ENAC and before, and even during this master's degree.

Now I'm going to describe you how I came up with the application, what my first ideas were and how I started this application.

I decided to use the distances between the images to be assembled as a reference value in my code and to establish my algorithms. I made this choice rather than any other. We could, for example, have chosen a time and a speed to trigger when to take a photo rather than at the position. But this solution seemed less precise to me, and it was important to me to have precise GPS points rather than times and speeds that could vary and make our overlap between photos vary without our wanting it to, for example with a wind changing during the flight that would have an impact on the drone's speed, or a speed in a turn that wasn't exactly precise.

So as inputs to the algorithm we had a list of GPS points defining a surface to photograph and two distances: a length and a width, corresponding to the distances between the photos. And as outputs, we had to provide the points of the flight plan, and the points on which we had to take a photo. And it was from these elements that I started to think.

5.2. First ideas, first problems, first implementations

In this part, I'm going to take you through the first stages of my application, the first difficulties and my first implementations.

I'd like to make it clear that I made the hypothesis that I was working with a flat surface for the use of GPS points and that perhaps my distances wouldn't really correspond close to the poles, for example. But we're currently working in Spain and France, so I think my hypothesis is reasonable for latitudes like ours.

First, I quickly realised that I would have to work with changes of scale and changes of reference points. Because firstly my input distances, length, and width, were in metres, while positions of my points were in GPS coordinates, and my outputs (from the algorithm) were in GPS coordinates as well.

The first function I performed was a distance calculation between two GPS points. To do this I used the distance method in the geopy module in python. Thanks to this function and my generic distance between each photo and therefore between each point, I was able to calculate the number of photos needed between 2 GPS points to respect my imposed distance.

Then, thanks to a bit of geometry and knowledge of Python programming, I was able to return the GPS points at a distance d two by two on the straight line defined by GPS points A and B. I've illustrated this in a diagram below, named Figure 14. You can find the details on the code in the ANNEXE 1.

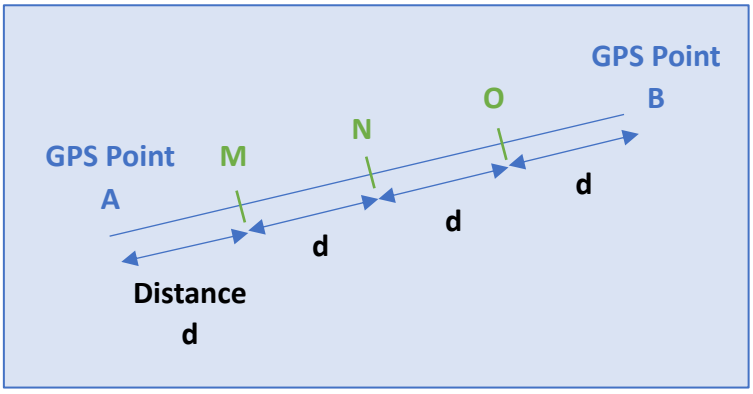

Figure 14: Illustration of my function stops on a line.

The next step was to draw up the first flight plan. The idea was to make a simple flight plan that works, even if it's not very customisable at the moment.

I wanted to be able to make a rectangular flight plan from two GPS points. In other words, from two points **A** and **B** with GPS coordinates **(X_A, Y_A)** and **(X_B, Y_B)** and two distances **d_length** and **d_width**, I could create the desired flight plan. To clarify my point, we could use two points to make a rectangle, defined by the points **(X_A, Y_A)**, **(X_B, Y_A)**, **(X_B, Y_B)**, **(X_A, Y_B)**. Below is a simple diagram to illustrate my thinking, named Figure 15.

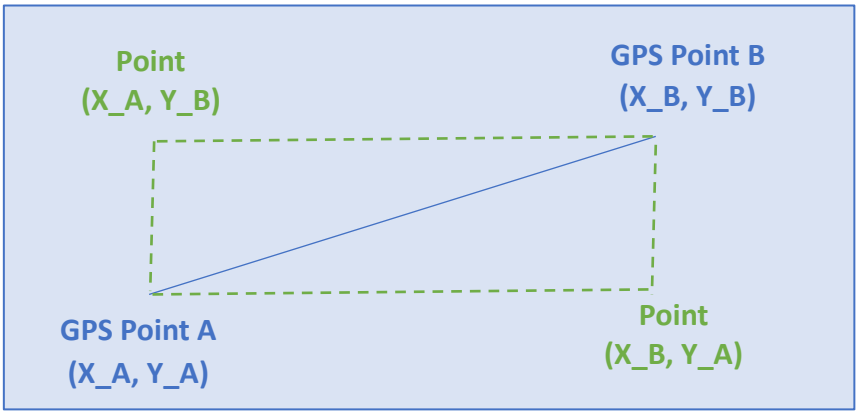

Figure 15: Illustration of creating a rectangle with 2 points.

Then, by adding the two distances **d_length** and **d_width**, we can create a grid where the intersections correspond to the points where we need to take a photo to make the desired stitching.

To do this, using the previous function, we can create the points on each segment of the rectangle, and by paying attention to the order of these we can create the flight plan lists as well as the positions at which we need to take photos for our assembly. Below is a screenshot of what it looks like when displayed on a plot, named Figure 16. You can find the code of the rectangle flight plan function in the ANNEXE 1**.**

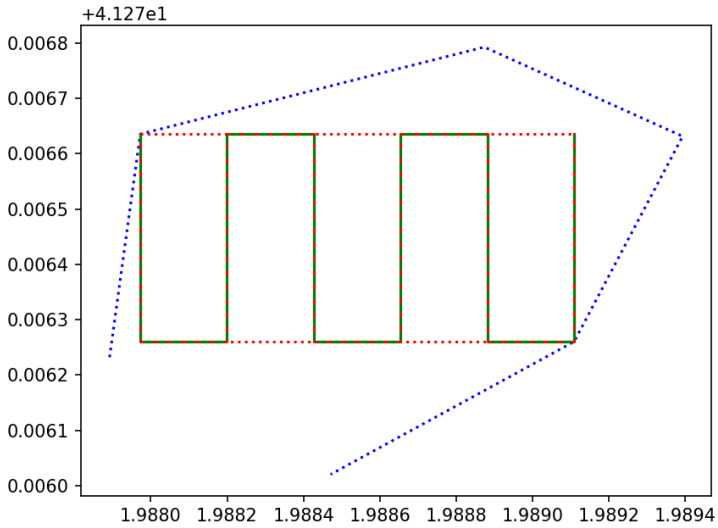

Figure 16: Plot of my function rectangular flight plan.

5.3. Additions of some complexities

So, we know how to make a flight plan with its stopping points for taking photos, but only within the framework of a rectangle. From there I wanted to be able to make a flight plan for any surface.

At this point I decided to divide the problem into two parts, convex surfaces, and non-convex surfaces. This strategy stems from a technical choice that I'll talk about later, but with more hindsight on the project, I realise that it might have been possible to find another solution that would have allowed me to stay within the general case, even though I don't have the answer now.

I have allowed myself to divide the problem in this way, knowing that a non-convex set can be represented as a union of convex sets, according to the property of decomposition of non-convex sets into convex sets. You can access a demonstration in ANNEXE 2. So, in the case of a non-convex surface we could divide the surface into several convex surfaces and continue the process for each surface. In a higher version of the application, we could implement this problem in the Python code generating the flight plan. I've decided to leave this option aside and give an alternative for cases that don't fit into the framework of a convex surface.

So, the rest of this report will deal with convex surfaces, and for the sake of brevity I'll call them any surface.

Then, the technical choice I made was to take the greatest distance between the points of the surface and its perpendicular as the orthogonal base of a reference frame. I imagined doing an iteration of **d_length** metres along my greatest distance, and at each iteration listing the points having this abscissa in my new reference frame and being in my convex surface. (Here came my problem in a non-convex surface, I thought it would be difficult to implement this knowing that for the same abscissa there could be "distant" points, knowing that it was possible to divide these surfaces into convex surfaces, so I didn't look any further). Here is Figure 17 to illustrate my point.

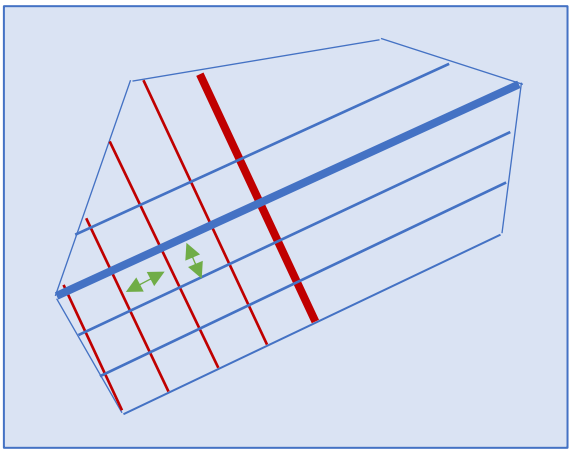

Figure 17: Illustration of my strategy for using long distance to create the flight plan.

To continue, I'm going to define a few words that I've used to simplify what I'm saying. I'm going to define segments as the sides of the polygon and the neighbours of a point as the points to which it is attached by a segment. For example, in Figure 18, A is a neighbour of B and D and forms the segments AD and AB.

Next, I needed to know which points were neighbours. My first implementation contained a major error: I had assumed that the two closest points to a point were its neighbours. As you can see in Figure 18, B is not a neighbour of D.

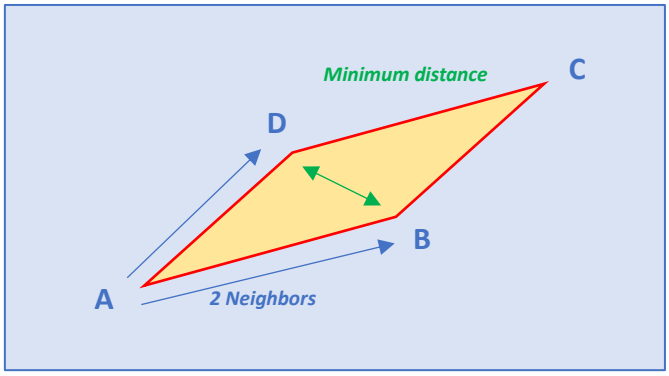

Figure 18: Illustration of nearest points and neighbours

To find the real 'neighbours' of each point I used the assumption that I was working on convex sets, and angle calculation. In fact, in a convex surface, a point and its two neighbours form the largest angle with respect to the other points of the polygon. To illustrate, in Figure 18, angle DAB is greater than angle DAC and angle CAB. For another corner, angle ABC is greater than angles DBC and ABD.

To find out whether it is the neighbour to the left (clockwise rotation) or the neighbour to the right (anti-clockwise rotation) we can study the sign of the angle. The code for calculating the angle and finding the neighbour is in ANNEXE 3.

Thanks to this, I was able to find the two neighbours of each point in my polygon. All I had to do was sort my list carefully. You can also see the code created in the same appendix (ANNEXE 3).

To recap, at this point I had a reference point formed by the greatest distance in my surface and its perpendicular, and I could find out which points are neighbours of each other.

I chose to create unit vectors **u** and **v** with the direction of the greatest distance and its perpendicular respectively, and with norms **d_length** and **d_width**, my desired distances between each photo. I've included the code for creating these vectors in ANNEXE 4.

I'm now going to show you my strategy to calculate this flight plan.

The idea is to start at one of my longest distance points, which we'll call A. Then move by a distance **d_length** along this greatest distance, which amounts to translating point A by the vector **u**. And look for the points (M and N) on the segments (AB and AC) formed by A with its neighbours which are on the perpendicular to the greatest distance at A'. You can follow the logic in Figure 19.

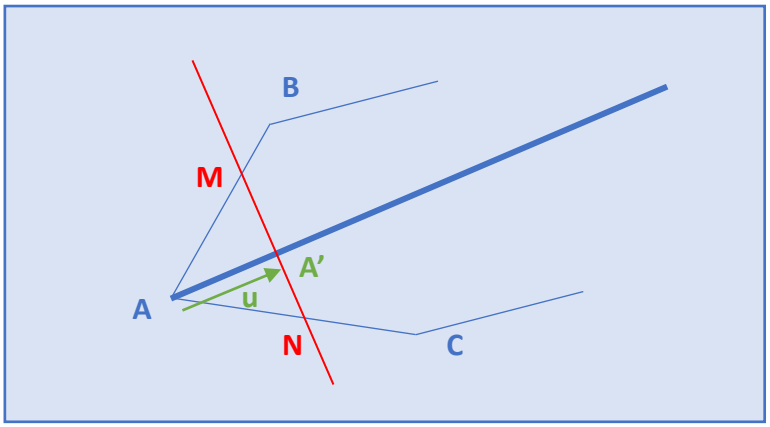

Figure 19: Illustration of my flight plan creation strategy

Finding these points (M and N) with a ruler and a square on a sheet of paper is not complicated but calculating them in Python is not trivial.

To help you follow my reasoning, you can use my instructions in parentheses and the diagrams I have provided in Figure 20 and Figure 21. So, to do this I used orthogonal projections of my segments (Pt1Pt2) onto my longest distance. Then, using the abscissa of my points (Pt1 and Pt2) and the abscissa of their projections onto my longest distance, I was able to find the abscissa of the point I was looking for (X_pt in R0). Finally, the line equation of the points on the segment allowed me to find the y-coordinate of the point (Pt) I was looking for. In this way, starting from a point at my greatest distance, I was able to find the point at the perpendicular to a segment. For more details on the code itself, I put you the code implemented in ANNEXE 5.

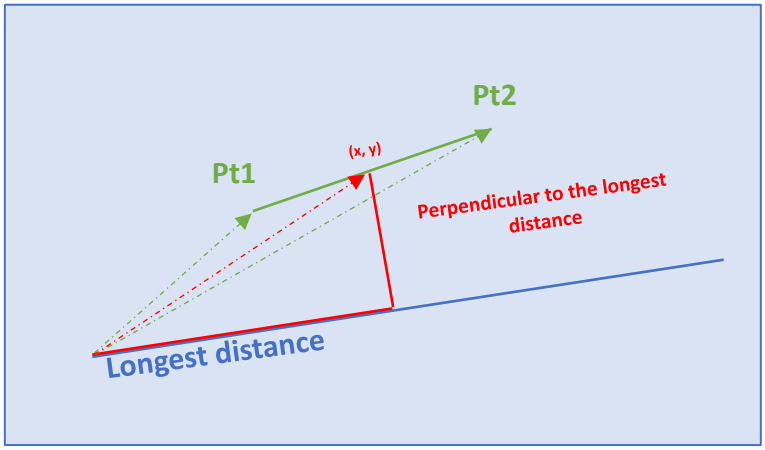

Figure 20: Point formed between the perpendicular to the greatest distance and the segment.

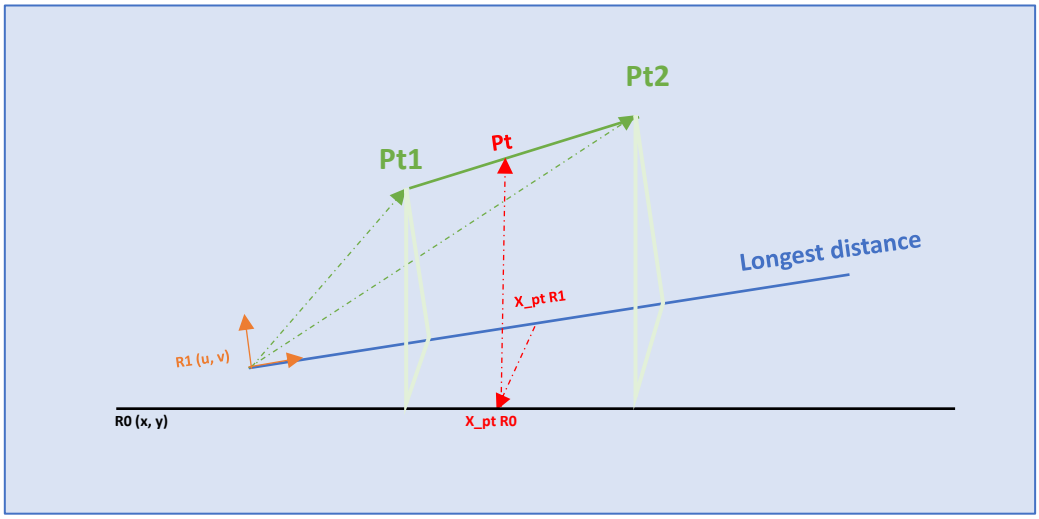

Figure 21: Illustration of my method to find the desired point on the segment.

Now, by implementing a distance **d_length** (advancing a vector **u**) over the longest distance, we can recover the two points on the segments and on the perpendicular for any iteration (advancing over the longest distance). However, as we move forward, the segment may no longer correspond to the perpendicular. Our left and right neighbour's algorithm comes into play here and allows you to change segment when the previous one is exceeded.

Next, to make the list of stopping points for taking photos, we need to calculate the points between the two points on the opposite segments, using the function in the previous section. To keep the flight plan consistent, we need to remember to alternate the directions of travel. The algorithm stops when we reach the second point of our greatest distance, i.e., the end of our surface.

I illustrated this in Figure 22, and you can see the flight plan creation code that I did in ANNEXE 6.

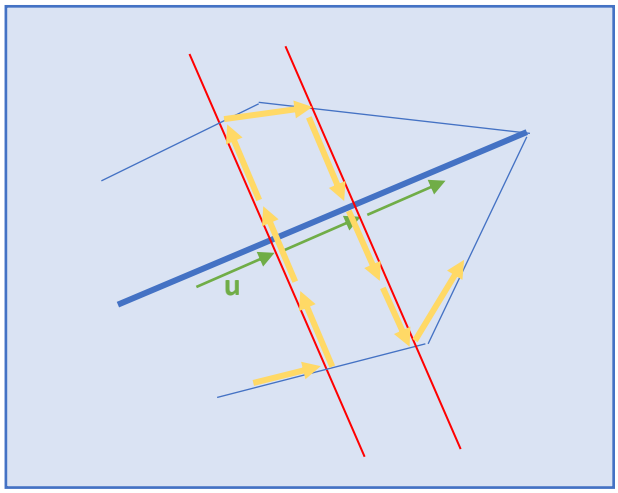

Figure 22: Illustration of the evolution of the flight plan

5.4. Server architecture

Now that we know how to calculate a flight plan with stop points for photos, we need to develop a feature that can receive the data for creating the flight plan, and send the flight plans back to the client that requests them.

This is the role of the application server. First of all, a server is a computer device (hardware and software) that offers services to one or more clients on an Internet or intranet network. It can be hardware or software, and it executes operations in response to requests made by another computer called a "client". Computer servers are involved in the smooth running of networked systems, from connecting devices to managing data. Servers can provide a wide range of services, including file sharing, e-mail, the World Wide Web, firewalls and virtual private networks.

Firstly, I created this server in python as explained above. Then I chose to use the MQTT messaging protocol with the WebSocket communication protocol. I'll tell you more about these later in this report in the Communication section. Figure 23 shows the general architecture of the server.

```
87  if __name__ == '__main__':
88      client = mqtt.Client(transport="websockets")
89
90      client.on_message = on_message_function
91
92      client.connect(global_broker_address, global_broker_port)
93      client.subscribe('webApplication/server/Connect')
94      print("Waiting commands")
95
96      client.loop_forever()
97
98
```

Figure 23: General architecture of the application server

It's quite simple, and I'm going to present it to you briefly. The first line, "client=mqtt.Client(transport="websockets")" is used to create an instance of the Client class of the MQTT module, which is required to establish a connection with the MQTT broker. The transport="websockets" parameter specifies the use of the WebSockets protocol for communication.

Next, "client.on_message" is used to process messages from a client, in my case messages from my web application. So, every time a message is sent by a client, it will be received and the on_message_function will be called. This is where we come in.

"client.connect(global_broker_address, global_broker_port)" establishes a connection with the MQTT broker using the broker's address (global_broker_address) and the broker's port (global_broker_port). These values are defined at the beginning of the code.

We need to subscribe our server to a subject so that it can perform its operations. This is the purpose of the client.suscribe line. I'll talk about how this works in the communication section.

And the last line is used to keep the server awake and waiting for a message.

Leaving aside how communication between the different parts of the project is managed, here's how the server is able to send back a flight plan when it receives input parameters. You can refer to the code below in Figure 24.

```
51  if command == "createFlightPlan":
52      print('Flight plan creation.')
53      msg = json.loads(message.payload.decode("utf-8"))  # print('msg =\n', msg)
54
55      list_pts = []
56      points = msg['points']  # print('points =\n', points)
57
58      for point in points:
59          new_point = point['lat'], point['lng']
60          list_pts.append(new_point)  # print('list_created :\n', list_pts)
61
62      if len(list_pts)==2:
63          flight_points = list_pts
64          stops = FP.stops_on_a_line(list_pts[0], list_pts[1], float(msg['d_length']))
65
66      else:
67          flight_points, stops = FP.create_flight_plan(list_pts, float(msg['d_length']), float(msg['d_width']))
68
69      result = {'flight_points': flight_points, 'stops_points': stops}
70
71      client.publish("server/" + origin + '/createdFlightPlan', json.dumps(result))
72      print('Flight plan created and sent !')
```

Figure 24: Creation of the flight plan in the core of the server

First of all, it has to convert the input data to make it compatible with the flight plan creation program.

Then it manages the execution as we want. For example, I have chosen to run the program with only 2 points by returning the breakpoints on one line. In all other cases, it calls my flight plan creation function in another module and retrieves the output.

When execution is complete, it must return the results to the client that requested execution.

Finally, the server waits for a new message.

6. COMMUNICATION

The effectiveness of a web application depends heavily on its communication. As part of this project, various communication mechanisms have been put in place to enable users to interact with the server in real time. In our project, communication takes place between the web application and the server, and between the web application and the UAV, in particular with the autopilot service and the camera service. This section will highlight the main tools and technologies used to facilitate this communication. I'll start by presenting the MQTT and WebSocket protocols, which were mentioned in the previous section, and explaining how the application server works. Next, I'll explain what a broker is and its different forms, as well as what we use in our application. Finally, I'll show you how I've implemented it in the various entities.

6.1. MQTT and WebSocket

MQTT protocol

In this application, we use the MQTT (Message Queuing Telemetry Transport) protocol. This is a messaging protocol designed for machine-to-machine communications. MQTT is lightweight and based on a publish/subscribe model, and has been specifically designed for low-bandwidth, low-power devices, making it ideal for IoT networks where devices need to transmit data with limited bandwidth. This is particularly the case for our communication with our drone, as the network can be unstable. That's why the MQTT protocol seemed appropriate. It enables asynchronous two-way communication between clients and servers using a message broadcasting mechanism via 'brokers'. Messages are published on specific 'topics', and clients interested in these topics can subscribe to receive updates. MQTT is appreciated for its efficiency and reliability because it works even with high latency and on unstable networks. It also offers different levels of quality of service, making it adaptable, and its publish/subscribe model enables asynchronous communication, avoiding a direct connection between clients.

WebSocket protocol

In addition, we used the WebSocket protocol. This is a bidirectional communication protocol that allows data to be sent in real time from the client to the server and vice versa. Its aim is to create a bidirectional, full-duplex communication channel over a TCP socket for web browsers and servers. WebSocket also offers low latency thanks to the persistence of the established connection, eliminating the need to establish a new connection each time data is exchanged. It also saves bandwidth by eliminating redundant HTTP headers. In addition, WebSocket is compatible with numerous firewalls and proxies, making it easy to deploy in a variety of environments.

MQTT over WebSocket

In our application, we use MQTT over WebSocket to combine the advantages of the MQTT protocol with the flexibility offered by WebSocket. MQTT over WebSocket allows MQTT messages to be sent over a WebSocket connection, instead of using the standard MQTT protocol directly. This allows us to benefit from the lightness and efficiency of MQTT, as well as the real-time, two-way communication offered by WebSocket. With MQTT over WebSocket, MQTT messages are encapsulated in WebSocket messages and transmitted over the WebSocket connection established between the client and the server. This makes it easier to use MQTT in environments where WebSocket is supported, such as web browsers. This allows us to communicate with MQTT brokers and integrate real-time communication capabilities using MQTT, while taking advantage of the benefits of WebSocket connectivity. MQTT over WebSocket requires both client and server to support this functionality, which can be achieved using MQTT over WebSocket-enabled MQTT brokers and appropriate MQTT libraries.

6.2. Mosquitto's broker

An MQTT broker is an intermediary server responsible for distributing messages between MQTT clients. It acts as a mediator between message publishers (clients who publish messages on topics) and subscribers (clients who subscribe to topics in order to receive messages). The broker receives the published messages and forwards them to the subscribing customers, guaranteeing that the data is distributed reliably and efficiently.

In our application, we use Mosquitto brokers to facilitate communication via the MQTT protocol. Mosquitto is a popular open-source MQTT broker that implements the MQTT protocol and allows customers to connect, publish messages and subscribe to topics. Mosquitto brokers provide a robust and scalable infrastructure for MQTT communication, enabling subscription management, secure message exchange and support for multiple simultaneous clients.

Initially, I worked with an online broker: hivemq, but online brokers are dependent on an internet connection, which is much less secure because anyone can connect to the broker, and we are dependent on a third party when using them in our application.

In local mode, I launch the mosquitto server with the appropriate configuration, specifying the WebSocket protocol, on my machine and connect to the desired port. When we're in flight, I connect to the drone's network and choose its IP address as the broker address, then choose the port I want.

As mentioned above, a significant advantage of using Mosquitto brokers is the ability to work in local mode, without an Internet connection. This means that MQTT clients can communicate directly with the Mosquitto broker in a local environment, without requiring Internet access. This can be particularly useful in scenarios where Internet connectivity is limited or unavailable, while still allowing local communication between devices and the server.

In conclusion, the use of Mosquitto brokers in our application enables reliable and efficient communication via MQTT and local protocol. The brokers act as intermediaries for the distribution of messages between MQTT clients, providing robust management of subscriptions and publications, enhanced security for message exchanges, and support for communication in different modes.

6.3. How I implemented it

Configuring the broker

To do this, we downloaded the mosquitto application onto the necessary machines (my computer to work locally, and the drone's Raspberry Pi for the various flights).

I'll only be talking about the 'external' broker (see Figure 8: Drone Engineering Ecosystem's Diagram) in this section, as I'm not working with the 'internal' broker.

To do this, we just need to edit a text file in this way (Figure 25). The first and last lines allow connections to be authorised without authentication. The second is the port affected. The third indicates that the WebSockets protocol should be used for connections.

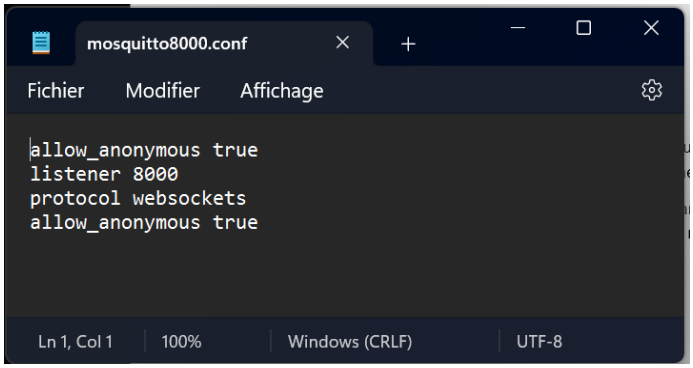

Figure 25: Configuration of port 8000 of the mosquitto broker

Then, to launch the broker, simply go to the application folder and launch the program with the desired configuration. You can see the command I am running in Figure 26. In it, -v is used to display in the terminal everything that is happening on the broker -c is used to choose the desired configuration.

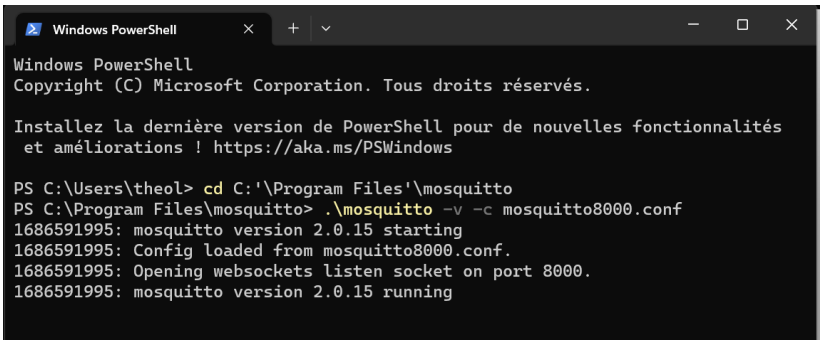

Figure 26: Command in the terminal to launch the mosquitto application.

In the web application

In this short part, I'm going to show you how to connect the web application to the broker and how to send and receive messages using it.

As before, to connect to the broker, we need to know the IP address on which it is running and the port. The broker sends back a message to validate the connection.

Next, we send messages by publishing to the broker. Of course, we must subscribe to the topic on which the reply will be sent. I'll show you an example in Figure 27. On line 559 you can see the publication to the MQTT client, between the quotation marks, there is the "topic" of the message and dataJSON is the message that is sent. The next line is the subscription to the response we are waiting for. In both topics, you can see the organization 'Origin / Destination / Command'.

```
537     function calculateFlightPlan(){
538         Swal.fire({···
546             .then((result) => {
547             if (result.value) {
548                 // Creation of the JSON object with the list and the parameters needed
549                 let data = {
550                         points: points.value,
551                         d_length: d_length.value,
552                         d_width: d_width.value,
553                 };
554
555                 let dataJSON = JSON.stringify(data);
556
557                 // To send the datas
558                 console.log('Points and distances sent from FP.', data);
559                 client.publish('webApplication/server/createFlightPlan', dataJSON);
560                 client.subscribe('server/webApplication/createdFlightPlan');
561             }
562         });
563     }
```

Figure 27: Publish and subscribe in the Web Application

Then you can see how to receive the messages on the Figure 28. As you can see, we receive the topic and the message, and depending on the command (contained in the topic) we can perform the operations we want with the data.

```
258     client.on('message', (topic, message) => {
259         let splited = topic.split("/")
260         let origin = splited[0]
261         let destination = splited[1]
262         let command = splited[2]
263         if (command != 'telemetryInfo'){···
265         }
266
267         if (command == 'createdFlightPlan') {···
300         }
301
302         if (command == 'calculatedParameters') {···
311         }
312
313         if (command == 'telemetryInfo') {···
317         }
318
319         if (command == 'photoTaken') {···
327         }
328     })
329 })
```

Figure 28: Receiving messages on the web application.

On the server side and the drone side

I'm going to quickly go over the server and the drone autopilot because the communications with the web application are relatively similar to what I explained earlier for the web application. However, on the server side, the only point that really differs is that the server subscribes to all the necessary topics when it connects. And not as in the web application, which anticipates responses each time it publishes a topic. On the drone side, it doesn't respond to any requests; it simply translates what is sent to it into MAVLINK commands and makes trajectory changes. Apart from that, it periodically sends telemetry data that enables the drone to be placed on the map in the application. Then for the two entities, drones and server, it's the same principle of subscriber publications as in the web application.

7. IMAGE STITCHING

I decided to present this part of the project to you even though I wasn't the main player in this area. Even though I wasn't the first person involved in this part of the project, I learnt how to assemble two images, what the parameters are that come into play in the assembly and also in taking the photo, and I did some assembling myself even though it wasn't for the end of the project. What's more, throughout our project, I followed my friend's progress, giving him advice when I could and helping him to find solutions to the various problems he encountered. So, I'll start by explaining the principle of image stitching, what the important elements are and what this technology is based on. I'll then go on to describe some of the problems we encountered that went beyond the theoretical framework of stitching and how we solved them. Then I'll give you some overall progress on the stitching in the project.

7.1. Principles

In this section, I'm going to introduce you to the principles of stitching and show you a method that works quite well in theory. First of all, image stitching is used to create panoramic or high-resolution photos. This technology is used very often without us even realising it. Image stitching is divided into three distinct parts, firstly taking the photo, then transforming the image and merging it.

For this learning process, we had an activity during a course on image stitching, and I'm going to use this activity to show you how it works.

First of all, we use Python and OpenCV for image stitching. I'm going to show you the assembly for just two images for this part and these two images are not included in the whole project to make it easier to explain what I'm saying and to be able to show the different elements easily.

To begin, here are our two starting images in Figure 29, which we want to assemble.

Figure 29: The two photos to stitch

To join two images together, we need to have similar elements in both images. To do this, we need to find "regions of interest" called "keypoints" and descriptors associated with these keypoints. These descriptors are invariant vectors that allow us to match the different scales of the images.

To detect these keypoints and descriptors, we use the SIFT (Scale-Invariant Feature Transform) and ORB (FAST and Rotated BRIEF) methods. The SIFT method is more focused on finding invariants in scales, which means that it can detect features regardless of their size and can be used in the context of rotations. ORB uses a combination of the FAST (Features from Accelerated Segment Test) and BRIEF (Binary Robust Independent Elementary Features) algorithms, so it is not specialised, but its combinations compensate for this non-specialisation and make it even more powerful in terms of calculation speed. The ORB method is often said to be less robust to differences in lighting, but it has the advantage of being free of copyright, unlike SIFT, which is subject to patents.

I've included representations of keypoints detection and their descriptors in the two photos in Figure 30. In this figure, you can see circles representing the keypoints, the size of the circle represents the scale, and the line in the circle shows the orientation of the keypoint.

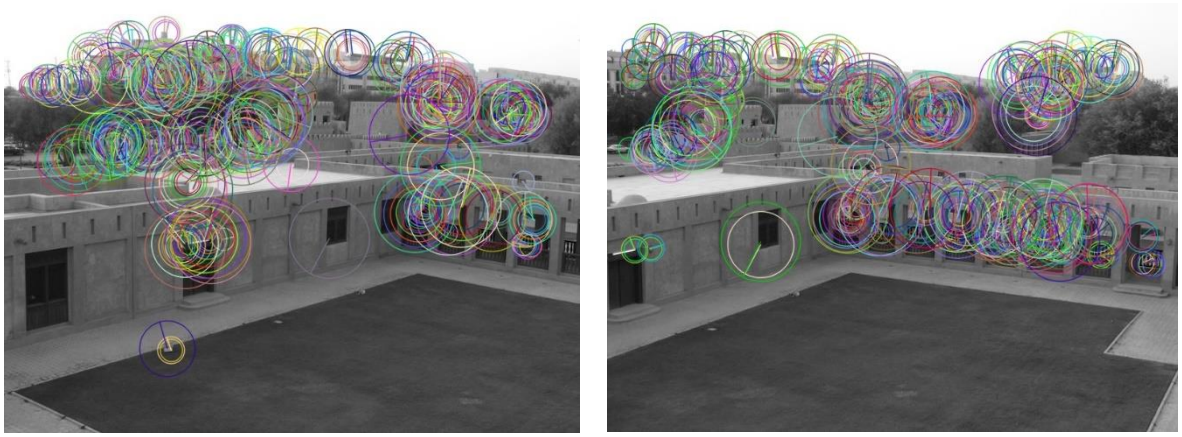

Figure 30: The two photos with representations of their keypoints and descriptors

Then, once we have these keypoints and descriptors, we need to match them together. We can match them according to various parameters, for example the distances between the keypoints. I've included this matching in Figure 31 to help you understand.

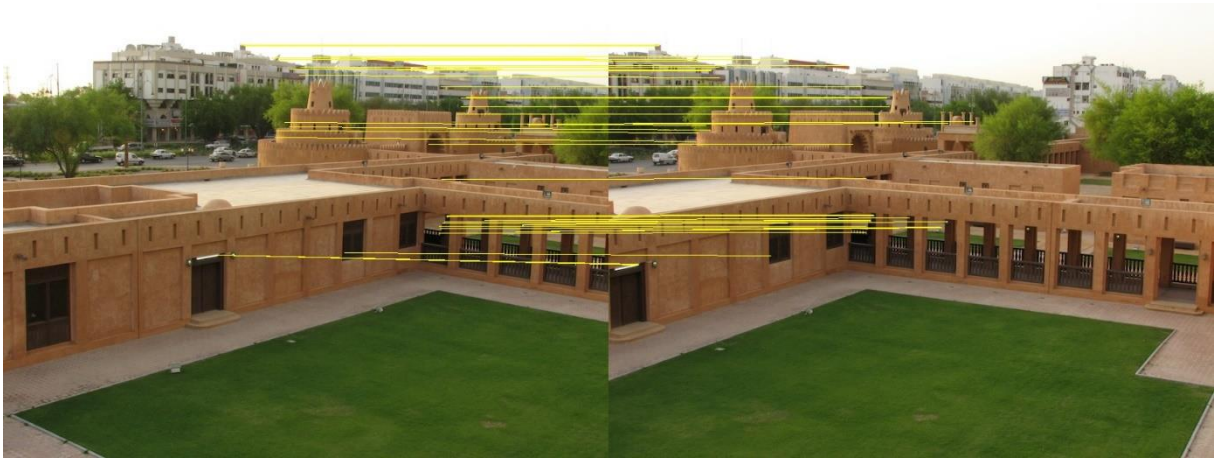

Figure 31: Illustration of matching between two photos

The next step is to change the scales of the images. To do this, we need to estimate a homography matrix. In our case, the two images are at the same scale, so we don't need to do this step.

Next, we need to define which points of the images will be on the corners of our panorama. After these steps we can assemble our images to give Figure 32.

Figure 32: Panorama created by stitching of both images.

7.2. Problems encountered and some results.

In this section, I'm going to tell you about some of the problems we encountered when stitching. I'll also give some ideas we've had to solve these problems.

When taking photos, a lot of parameters come into play, including resolution, exposure, saturation, and others. If one of these parameters is not managed properly, stitching can be impossible, and image details can sometimes even be invisible.

Resolution, or the number of pixels in the image, is very important, but when operating, it is sometimes necessary to reduce this resolution to improve the speed of capture and reduce the storage required. For example, we're using a Raspberry Pi v2.1 camera, which can take photos with 3280x2464 pixels, but this setting results in photos with around 6.5Mb of storage and makes the drone too slow to take photos. We therefore decided to lower the resolution to 1920x1080, which offers better performance.

During a flight, the drone is never perfectly stable, and in addition to displacement, due to the altitude and relative position of the drone, the same object may appear different on two adjacent photos. During stitching, distortions can appear as you can see in Figure 34. The raw images are shown in Figure 34Figure 33.

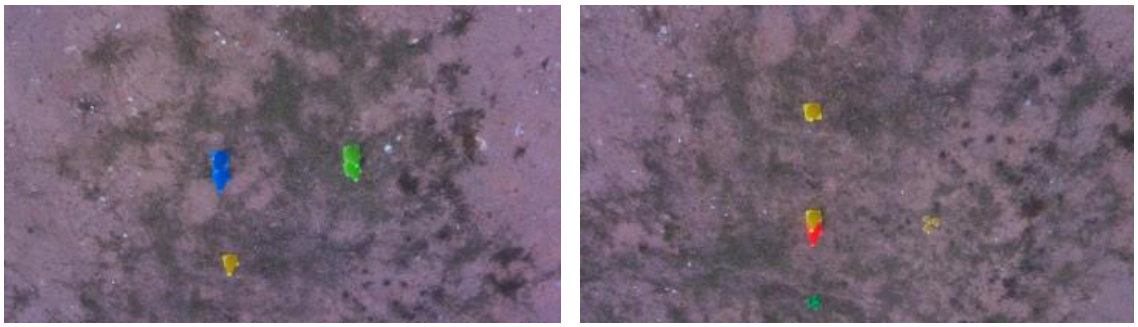

Figure 33: Raw images of the stitching example

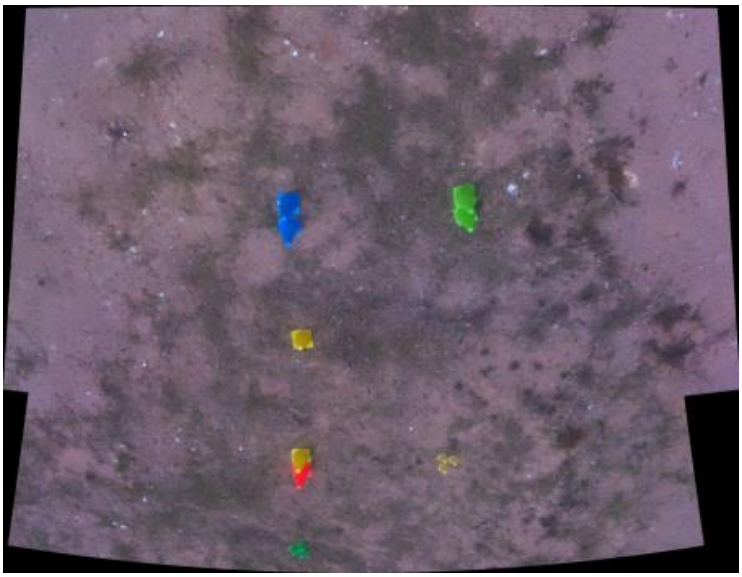

Figure 34: Result of the stitching of the example

These distortions are not a real problem, but they change the shape of the image and therefore impede stitching with an additional image.

The solution to this problem that we used was to predict the size of the assembled image before adding a new image and to fill this new "new size base" of the image with 0s and complete it with the calculated image.

A second problem was the sequencing of the images. It is important to control the order of the images and their orientation. Firstly, because stitching assumes that there is an overlap between each pair of photos. The photos must therefore be in the right order for stitching to work properly. The solution, although very basic, is that we put the time in the name of the photos, and we can use this time to find the chronological order of the photos. Secondly, orientation also has an impact on stitching. To remedy this problem, we need to make sure that the flight plan is correct and ensures vertical and horizontal overlap.

Secondly, we can see that there are significant distortions at the edges of the panorama. These distortions are due to the stitching processing. The solution is to make the flight plan surface wider than necessary.

I've included some results from flights with stitching in Figure 35 and Figure 36.

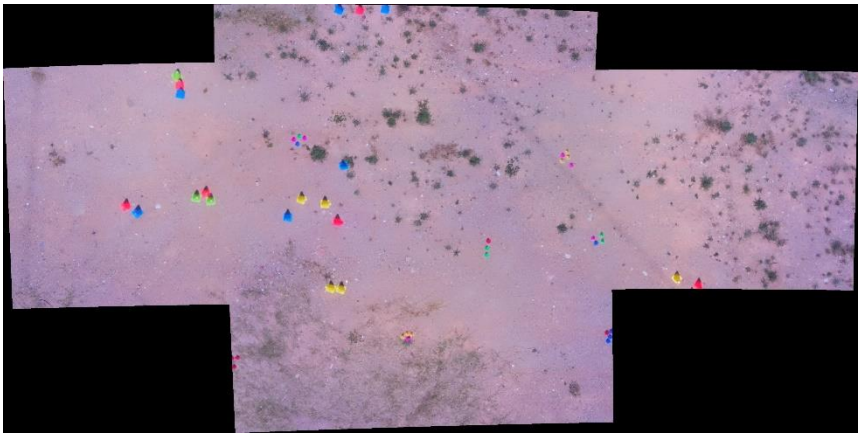

Figure 35: Example A of stitching

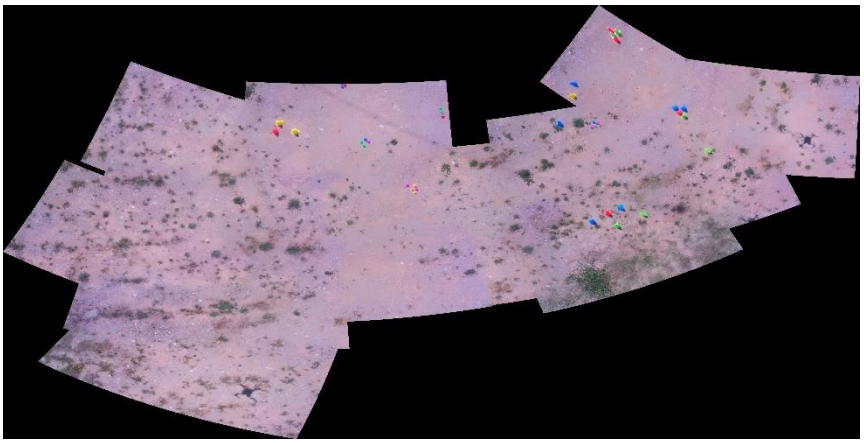

Figure 36: Example B of stitching

8. TESTING AND IMPROVING

In this part, I'm going to tell you about some of the simulations and flights carried out during this project, and the purpose of the flights we carried out. I will also briefly present the architecture of the autopilot service coded in Python.

8.1. Autopilot service

To create our autopilot service, we used code that had already been developed in the ecosystem and modified it to suit our application.

First of all, when we launch the module, we choose whether we want to run a simulation or a "production", working locally or globally. Then the autopilot chooses its connections automatically. For example, for a local simulation, the service will use the broker address and the default port for this configuration. For other simulations, we can for example use a broker that requires password authentication, these parameters will also be put in the function parameters when we call it.

Then, it works in a similar way to a server, it waits for a message, and when it receives one, depending on the message, performs the corresponding operations.

I have included the code for this autopilot in ANNEXE 7.

When messages are received, this service is responsible for translating the list of waypoints in the flight plan into MAVLINK commands that can be interpreted by the autopilot. These commands are presented as follows (Figure 37).

```
200     cmds.add(
201         Command(0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT, mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0,
202                 0, 0, 0, originPoint.lat, originPoint.lon, altitude))
```

Figure 37: Example of a MAVLINK command

Then, when all the commands have been downloaded by the autopilot, the service sends the arming and take-off commands and then runs through the various commands. I've included the code in Figure 38 to help you understand. When all the commands have been completed, the return to launch command is sent.

```
224     arm()
225     state = 'takingOff'
226     take_off(altitude, False)
227     print('Take off realised !')
228
229     state = 'flying'
230
231     vehicle.commands.next = 0
232
233     # Set mode to AUTO to start mission
234     vehicle.mode = VehicleMode("AUTO")
235
236     print('Beginning of the loop while.')
237     while True:
238         nextwaypoint = vehicle.commands.next
239         print('next ', nextwaypoint)
240         if nextwaypoint == len(
241                 waypoints) + 1:  # Dummy waypoint - as soon as we reach waypoint 4 this is true and we exit.
242             print("Last waypoint reached")
243             break
244         time.sleep(0.5)
245         # external_client.publish(sending_topic + "/waypointReached", json.dumps(waypointReached))
246     print('Ending of the loop while.')
```

Figure 38: Chronology of an autopilot flight plan

8.2. Simulations

All along our project and before our flights, we attached great importance to carrying out simulations to test our functionalities as much as possible. So, we took the initiative of adapting our autopilot service so that it could operate in a simulated environment using the DroneKit-Sitl library in Python.

To run these simulations, we used the part dedicated to simulating in the well-known Mission Planner ground station. In addition, we made sure to run the mosquitto broker on our machine to enable communication within the simulation. I would like to add that in order to carry out the simulations, we also need to simulate the drone's internal broker on our machine, i.e. on port 1884 in our project.

Over the course of our project, we ran several simulations at different stages, often independently. This approach proved particularly useful when we were working from home, without access to a real drone or the space to carry out real flights. Thanks to the simulations, we were able to continue to make progress and develop our functionalities, even under constrained conditions.

Here are two screenshots showing what we can see during a simulation, on the web application in Figure 39 and on the Mission Planner application in Figure 39 for the same mission (not at the same time, to have time to take the screen shot). You can find a video of a screen recording of a simulation at this link if you want: https://youtu.be/eFJfKcCq5lo.

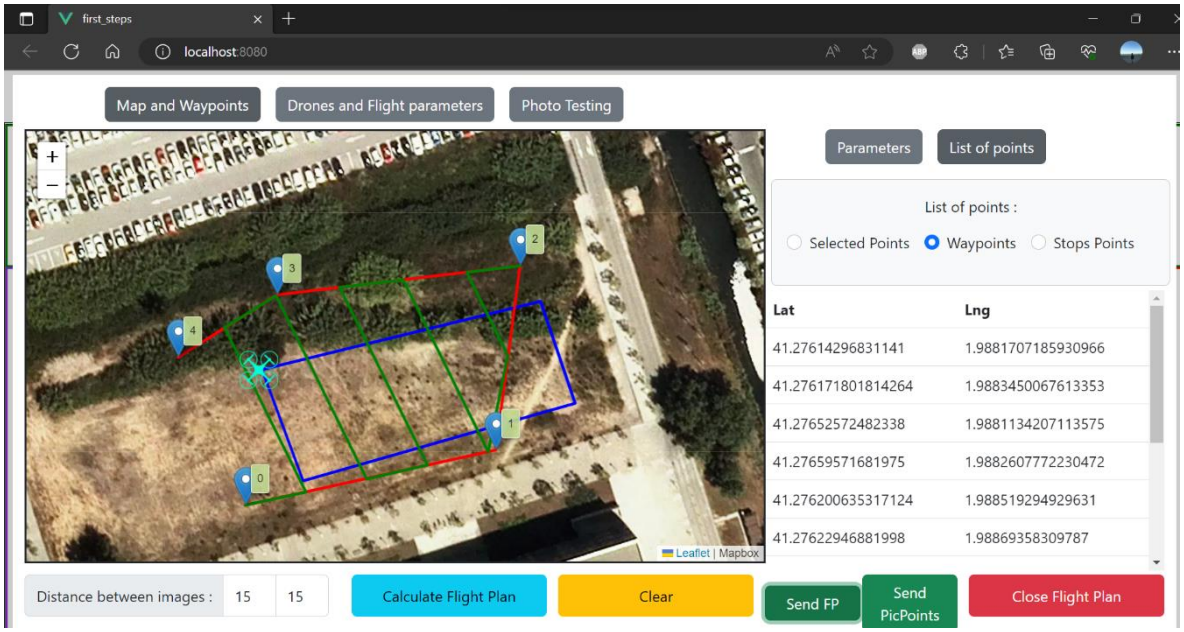

Figure 39: View of a simulation from the web application

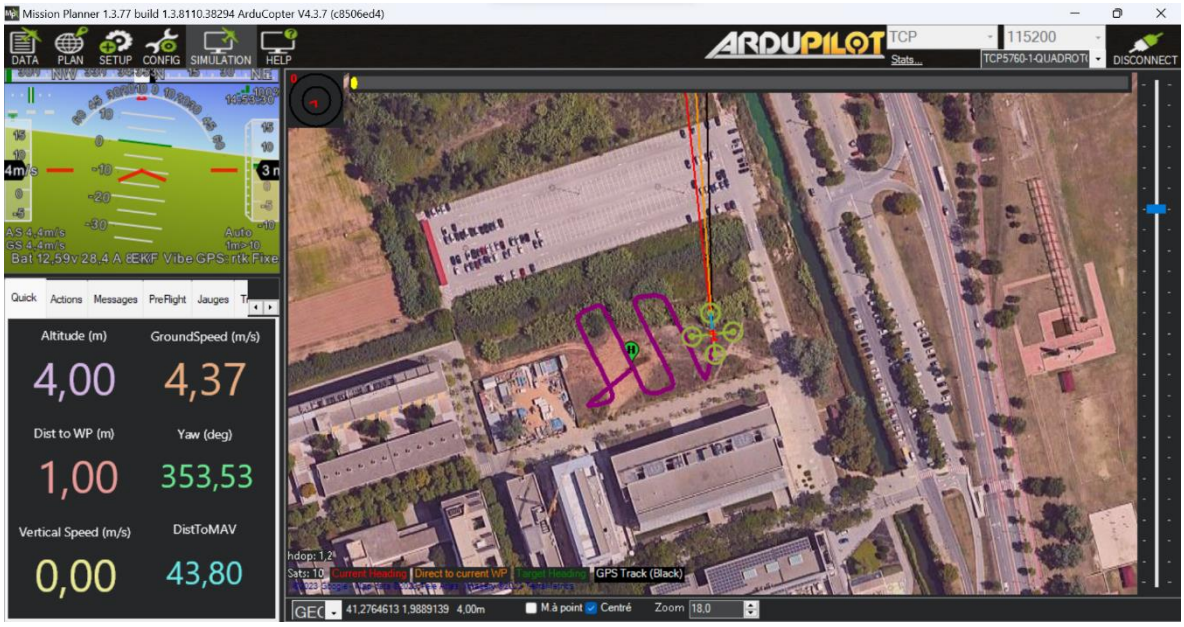

Figure 40: View of a simulation from Mission Planner

8.3. Flights performed

To test our functionalities on a real drone, we had an open-air aviary at our disposal on the university campus. This aviary, called DroneLab, is an open-air space with nets to ensure safety. The DroneLab measures 80 × 40 × 15 metres and is used by many actors from all over Catalonia. We've also been using it, almost every week, for our various tests and to validate our progress.

I'm going to show you some of the tests we carried out during this project. The first tests were basic, though not always easy. In particular, security tests by testing the geofence applied, which sometimes wasn't, as in Figure 41.

Figure 41: Photo of the drone stuck in the nets.

We tested the flight plan via the web application without applying the photo capture and then with it to see if the two functions worked correctly, independently, and together. You can see our working environment in Figure 42, as well as the drone in mid-flight.

Figure 42: Working environment.

Many flights were necessary to find the right camera settings, in particular to adjust the luminosity and improve the quality of the photos to make stitching easier.

We added coloured dots on the ground to have easily recognisable elements, which helped a lot with the stitching but also to check it. We could see the position of the cones and compare them with the image created by stitching. You can see one configuration in Figure 43.

Figure 43: Photo of a flight plan with a random configuration of coloured cones

Another experiment we carried out was to fly in a straight line, knowing the distances between the cones. As shown in Figure 44 and Figure 45.

Figure 44: Photo of a flight configuration with known cone distances

Figure 45: Photo of a configuration of a straight-line flight

This experiment was useful for several reasons, in particular to calculate the best overlap but also to analyse the rendering according to the drone's speed. For example, we noticed that the drone was going too fast to take photos, which sometimes caused us to 'miss' a shot.

This speed parameter evolves in conjunction with the trigger distance for taking photos, known as the threshold. This is a distance at which the camera takes a photo if it is at a shorter distance from the GPS point and waits if it is further away. A minimum distance is preferable to have a precise photo position, but if the drone passes too quickly, the camera may be out of sync and take photos at an undesired position, or it may not have time to take the photo, which would restrict the rest of the operation.

We spent a lot of time adjusting many small parameters to get acceptable renderings. Sometimes we couldn't understand why stitching wasn't working.

CONCLUSION

This 6-month internship in the UPC research team was very gratifying. I think I gained a lot of autonomy with this end-of-studies project, as we were in control of our project and could move forward as we wished and in the direction we wanted. This project fully met my expectations, because we really had a global vision of the application, and we created the whole chain from A to Z and tested and used it for the various tests and simulations. What's more, the team was international, which really brought a lot out of me, in the sense that I learned to be clear, and to make myself understood in a language other than my own, and with a language I'd known for less than 6 months. I've also learnt to be more patient, as this can slow down communication.

I hope that my report as a whole has interested you and that I've been clear. I'm aware that there are some parts that are a bit technical for people who don't work in the field, but I've tried to provide several levels of reading so that everyone can find something of interest. I also hope that the choice of writing this report in English has not hindered your interest in and understanding of the overall project.

This project was marked by a lot of learning at the beginning, in particular how to create a web application with Vue.Js, or understand how image stitching works, and throughout the project in the workings of each entity.

I think I've met all the objectives set at the beginning of the course, since I've offered a web application that lets you connect a drone easily via the Internet, without having to install a third-party application. What's more, I created the applications that make it possible to use it, in particular the creation of flight plans in Python that can be accessed from a server. Not forgetting that the whole project is fully integrated into the existing ecosystem, which was a necessary requirement. My partner and I are now able to create the flight plan and stitch a predefined surface, which was our first and main application to implement. We're already working on vertical stitching for panoramas and group photos. And we plan to work on the simultaneous operation of several drones. We had a few ideas, such as dynamic geofences, but I'm afraid we don't have the time to bring this idea to fruition.

I think the project could be improved by adding some security features, particularly in the web application and in the communications broker, since they enable drones to be controlled, which is a sensitive area. In addition, we could improve the overall aesthetics of the web application by adding colours that go together and incorporating a world of its own. Finally, we could add a range of applications to this web application, and implementation would be fairly easy, without disturbing the various users, since that's one of the great qualities of working on a web server.

We ran into several minor difficulties throughout the project. But we were able to resolve each one calmly and intelligently. Some problems remain unexplained but do not affect the way the application works.

Finally, all my work will be used by my host university, since it will be integrated into the ecosystem presented. I will also be putting my research online on a GitHub server so that everyone can draw inspiration from my work.

GLOSSARY

EETAC: “Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels”, Castelldefels School of Telecommunications and Aerospace Engineering

ENAC: « Ecole Nationale de l’Aviation Civile », National Civil Aviation School

HFOV: Horizontal Field of View

ICARUS: Intelligent Communications and Avionics for Robust Unmanned Aerial Systems

UAV: Unmanned Aerial Vehicle

UPC: “Universitat Politècnica de Catalunya”, Polytechnic University of Catalonia

VFOV: Vertical Field of View

LEXICON

CAN, Controller Aera Network: Communication protocol used in embedded systems.

CSS, Cascading Style Sheet: Coding language used for describing the presentation or the look and formatting of a document written in HTML.

GPS, Global Positioning System: Satellite-based radio navigation system.

HTML, Hypertext Markup Language: Standard markup language for documents designed to be displayed in a web browser.

HTTP, Hypertext Transfer Protocol: Communication protocol used on the Web to transfer data between a client and a server.

IoT, Internet of Things: A concept that refers to the connectivity of physical objects to the Internet. It refers to a network of intelligent objects, such as electronic devices, sensors, vehicles, household appliances, etc., which have the capacity to collect and exchange data via Internet connections.

LiDAR or LIDAR, Light Detection and Ranging: Method for determining ranges by targeting an object or a surface with a laser.

MAVLINK: Messaging protocol used for communicating with drones and between onboard drones' components.

PWM, Pulse Width Modulation: Technique used in electronics and electrical engineering, based on modifying the duration of a pulse signal while maintaining its frequency constant. It makes it possible to control the power or voltage delivered to a device by adjusting the cyclic ratio, i.e., the proportion of time during which the signal is in the high state relative to the total period of the signal.

BIBLIOGRAPHY AND WEB REFERENCES

Agile Alliance. (2023, April 28). *What is Agile? | Agile 101 | Agile Alliance*. Agile Alliance |. https://www.agilealliance.org/agile101/

Digitiz. (2022, August 8). *Application web : définition, utilité et avantages*. Digitiz. https://digitiz.fr/blog/definition-application-web/

dronsEETAC - Overview. (n.d.). GitHub. https://github.com/dronsEETAC

Duval, V. & Ecole doctorale SDOSE. (2022). *Faces and extreme points of convex sets for the resolution of inverse problems: Signal and Image processing.* HAL Theses. https://theses.hal.science/tel-03718371/document

Falquy, I. (2023, February 8). *Méthode agile : de quoi parle-t-on vraiment ?* https://www.cegos.fr/ressources/mag/projet/agilite/methode-agile-de-quoi-on-parle-t-on-vraiment

Idéematic. (2023, June 18). *Définition d'une application web*. https://www.ideematic.com/dictionnaire-digital/application-web/

Introduction · MAVLink Developer Guide. (n.d.). https://mavlink.io/en/

MQTT - The Standard for IoT Messaging. (n.d.). https://mqtt.org/

Number. (2022). Kanban Versus Scrum in Agile Software Development. *Number8*. https://number8.com/kanban-versus-scrum/

Parnois, E. (2021, April 27). *Top principaux avantage agilité vs méthodes cascade cycle V*. Harington. https://harington.fr/2020/09/23/methodes-agiles-avantages/

RÉDaction, L. (2019). Serveur (informatique) : définition, traduction. *www.journaldunet.fr*. https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203337-serveur-informatique-definition-traduction/

Serveur Web et serveur d'applications | IBM. (n.d.). https://www.ibm.com/fr-fr/topics/web-server-application-server

Stevewhims. (2022, October 20). *WebSockets - UWP applications*. Microsoft Learn. https://learn.microsoft.com/en-us/windows/uwp/networking/websockets

Syloé. (2017, September 4). *Qu'est-ce qu'un serveur informatique ? - Expert Linux*. Syloe. https://www.syloe.com/glossaire/serveur-informatique/

The UPC and the Castelldefels City Council inaugurate DroneLab, a unique laboratory and flight facility for UAVs. (n.d.). UPC Universitat Politècnica De Catalunya. https://www.upc.edu/en/press-room/news/the-upc-and-the-castelldefels-city-council-inaugurate-dronelab-a-unique-laboratory-and-flight-facility-for-uavs

Valero, M. (n.d.). *Vue Tutorial*. https://www.youtube.com/playlist?list=PL64O0POFYjHoeq8dfP-XYPCoNlehSiR_B

Vue.js - The Progressive JavaScript Framework | Vue.js. (n.d.). https://vuejs.org/

Wikipedia contributors. (2022). Main Page. *en.wikipedia.org*. https://en.wikipedia.org/wiki/

Wikipedia contributors. (2023). Convex set. *Wikipedia*. https://en.wikipedia.org/wiki/Convex_set

ANNEXES

ANNEXE 1: Code of the first implementations

```python
def stops_on_a_line(point_1, point_2, dist):
    """Calculate points on a line between pt1 and pt2 with a distance of dist between each stop. From pt1 to pt2."""
    '''...'''
    stops = [point_1]
    if point_1 != point_2:
        dist_between_points = distance_in_meters(point_1, point_2)
        number_of_stops = math.ceil(dist_between_points / dist)
        # print(number_of_stops)
        delta_latitude = (point_2[0] - point_1[0]) / number_of_stops
        delta_longitude = (point_2[1] - point_1[1]) / number_of_stops
        for i in range(1, number_of_stops):
            stops.append((point_1[0] + i * delta_latitude, point_1[1] + i * delta_longitude))
        stops.append(point_2)
    return stops
```

```python
def rectangle_flight_plan(point_1, point_2, d, d_width=None):
    """..."""
    '''...'''
    d_length = d
    if d_width is None:
        d_width = d
    corners = [point_1, (point_1[0], point_2[1]), point_2, (point_2[0], point_1[1])]  # The 4 corners.
    flight_points = []
    stops = []
    if distance_in_meters(corners[0], corners[1]) > distance_in_meters(corners[0], corners[3]):
        length_points = stops_on_a_line(corners[0], corners[1], d_length)
        length_points_bis = stops_on_a_line(corners[3], corners[2], d_length)
    else:
        length_points = stops_on_a_line(corners[0], corners[3], d_length)
        length_points_bis = stops_on_a_line(corners[1], corners[2], d_length)
    i = 0
    while i < len(length_points):
        if i % 2 == 0:
            stops += stops_on_a_line(length_points[i], length_points_bis[i], d_width)
            flight_points.append(length_points[i])
            flight_points.append(length_points_bis[i])
        else:
            stops += stops_on_a_line(length_points_bis[i], length_points[i], d_width)
            flight_points.append(length_points_bis[i])
            flight_points.append(length_points[i])
        i += 1
    return flight_points, stops
```

ANNEXE 2: Decomposition of convex sets

To demonstrate that any non-convex set is the union of convex sets, we can use the following proof:

Suppose that X is a non-convex set. This means that there exist two points x1 and x2 in X such that the line segment connecting them is not entirely contained in X. In other words, there exists a point x3 on the line segment connecting x1 and x2 that does not belong to X.

We can then decompose X into two convex sets: X1 and X2, where X1 is the set of all points in X that are on the same side of the line as x3, and X2 is the set of all points in X that are on the same side of the line as x1 and x2, but not x3.

It is easy to verify that X1 and X2 are convex sets. If y1 and y2 are two points in X1, then the line segment connecting them is entirely contained in X1, since all points on this segment are on the same side of the line as x3. Similarly, if z1 and z2 are two points in X2, then the line segment connecting them is entirely contained in X2, since all points on this segment are on the same side of the line as x1 and x2, but not x3.

Finally, we have $X = X1 \cup X2$, since all points in X are either on the same side of the line as x3 (and thus in X1), or on the same side of the line as x1 and x2, but not x3 (and thus in X2). Therefore, X is the union of two convex sets, as desired.

By repeating this process of decomposition for each non-convex set obtained, we can decompose any non-convex set into a union of convex sets.

ANNEXE 3: Code for calculating angles, searching for neighbours and sort the list.

```python
def angle_BAC(point_A, point_B, point_C):
    """Return the angle BAC. Using the cos theorem (c²=a²+b²-2.a.b.cos(alpha))."""
    if point_A == point_B or point_A == point_C or point_B == point_C:
        return 0
    a = distance_in_meters(point_B, point_C)
    # math.sqrt((point_C[0] - point_B[0]) ** 2 + (point_C[1] - point_B[1]) ** 2)
    b = distance_in_meters(point_A, point_C)
    # = math.sqrt((point_A[0] - point_C[0]) ** 2 + (point_A[1] - point_C[1]) ** 2)
    c = distance_in_meters(point_B, point_A)
    # math.sqrt((point_B[0] - point_A[0]) ** 2 + (point_B[1] - point_A[1]) ** 2)
    alpha = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c))
    return alpha
```

```python
def neighbours_of(ind_pts, list_pts):
    if len(list_pts) <= 2:
        return 'No neighbours'
    two_neighbours = [0, 1]
    angle_neighbours = angle_BAC(list_pts[ind_pts], list_pts[two_neighbours[0]], list_pts[two_neighbours[1]])
    for i in range(len(list_pts)):
        '''We're searching for the greatest angle. To have one certain neighbour.'''
        for j in range(len(list_pts)):
            angle = angle_BAC(list_pts[ind_pts], list_pts[i], list_pts[j])
            if angle > angle_neighbours:
                angle_neighbours = angle
                two_neighbours = [i, j]
    vectorial_product = (list_pts[two_neighbours[1]][0] - list_pts[ind_pts][0]) * \
                        (list_pts[two_neighbours[0]][1] - list_pts[ind_pts][1]) - \
                        (list_pts[two_neighbours[1]][1] - list_pts[ind_pts][1]) * \
                        (list_pts[two_neighbours[0]][0] - list_pts[ind_pts][0])
    if vectorial_product > 0:
        two_neighbours[0], two_neighbours[1] = two_neighbours[1], two_neighbours[0]
    return two_neighbours
```

```python
def all_neighbours(list_pts):
    neighbours = []
    for i in range(len(list_pts)):
        neighbours.append(neighbours_of(i, list_pts))
    return neighbours
```

```python
def sorted_points(list_pts):
    """Return the list_pts sorted. Rotational sorting."""
    '''...'''
    # list_pts = sorted(list_pts)
    nb_pts = len(list_pts)
    neighbours = all_neighbours(list_pts)
    sorted_list = [list_pts[0], list_pts[neighbours[0][0]]]  # the first point and the closest neighbour
    i = 1
    ex_index = [0, neighbours[0][0]]
    # print('Sorted =', sorted_list)
    # print('Ex_ind =', ex_index)
    while i < nb_pts - 1:
        """We look the neighbours of the point."""
        ex_index.append(neighbours[ex_index[i]][0])
        sorted_list.append(list_pts[ex_index[i + 1]])
        # print('Ex_ind =', ex_index)
        # print('Sorted =', sorted_list)
        i = i + 1
    return sorted_list
```

ANNEXE 4: Code of vectors creations

```python
def vectorise(point_1, point_2, d, d_width=None):
    """Return 2 vectors, u and v to form our orthogonal grid."""
    '''...'''
    d_length = d
    if d_width is None:
        d_width = d
    distance_max = distance_in_meters(point_1, point_2)
    # print('d_max =', d_max)
    # print('d_length =', d_length)
    # print('d_width =', d_width)
    dx = (point_2[0] - point_1[0])
    dy = (point_2[1] - point_1[1])
    norm = math.sqrt(dx ** 2 + dy ** 2)
    theta = math.atan(dy / dx)
    # print('theta : ', theta)
    vect_u = d_length * norm * math.cos(theta) / distance_max, d_length * norm * math.sin(theta) / distance_max
    vect_v = d_width * norm * (- math.sin(theta)) / distance_max, d_width * norm * math.cos(theta) / distance_max
    return vect_u, vect_v
```

ANNEXE 5: Code of the method for finding the point between the perpendicular at the greatest distance and the segment.

```python
def point_on_with_vectors(point_0, vect_u, point_1, point_2, x_on_ld):
    """O, P the 2 points from the longest distance segment. Pt1 and Pt2 the segment of the side. x the abscissa of
    the point on the longest distance."""
    '''...'''
    'Creation of the segments [pt0, pt1] and [pt0, pt2].'
    seg_pt0_pt1 = vect(point_0, point_1)
    seg_pt0_pt2 = vect(point_0, point_2)
    'Definition of the norm of the vector of the plan.'
    u_norm = math.sqrt(np.dot(vect_u, vect_u))
    'Proportion of the distance of the segment on the scalar product with the vector, with the norm of the vector.'
    pt1_u = np.dot(seg_pt0_pt1, vect_u) / u_norm ** 2
    pt2_u = np.dot(seg_pt0_pt2, vect_u) / u_norm ** 2
    'Creation of orthogonal projection point of pt1 and pt2.'
    pt1_on_ld = add_vect(point_0, vect_u, pt1_u)
    pt2_on_ld = add_vect(point_0, vect_u, pt2_u)  # plot_pts([O, pt1_on_ld, pt1, pt2_on_ld, pt2], color='y', wait=True)
    'Use of this relation : y = y1 + (x-x1)(y2-y1)/(x2-x1).'
    if pt2_on_ld[0] - pt1_on_ld[0] != 0:
        directing_coefficient = (x_on_ld - pt1_on_ld[0]) / (pt2_on_ld[0] - pt1_on_ld[0])
    else:
        directing_coefficient = np.inf
    # directing_coefficient = (x_on_ld - pt1_on_ld[0]) / (pt2_on_ld[0] - pt1_on_ld[0])
    x_on_segment = point_1[0] + directing_coefficient * (point_2[0] - point_1[0])
    return point_on_with_x(point_1, point_2, x_on_segment)
```

```python
def point_on_with_x(point_1, point_2, x):
    """Return the point on the segment [pt1, pt2] with the abscissa x."""
    on_this_segment = True
    if not min(point_1[0], point_2[0]) < x < max(point_1[0], point_2[0]):
        on_this_segment = False
        # print('No point with this abscissa on this segment.')
    m = (point_2[1] - point_1[1]) / (point_2[0] - point_1[0])
    b0 = point_1[1] - m * point_1[0]
    y = m * x + b0
    point = x, y
    # print(on_this_segment)
    return on_this_segment, point
```

ANNEXE 6: Flight Plan Creation Code

```python
def create_flight_plan(list_pts_unsorted, d, d_width=None):
    """Return two lists, the first one with the different points of the flight plan and the second is the list of all
    stops needed for the completed photo."""
    '''...'''
    d_length = d
    if d_width is None:
        d_width = d
    list_pts = sorted_points(list_pts_unsorted)
    neighbours = all_neighbours(list_pts)
    (start, stop), max_dist = longest_distance(list_pts)
    # print('start : ', start, 'stop : ', stop)
    plt.text(list_pts[start][0], list_pts[start][1], 'start')
    first_point = list_pts[start]
    last_point = list_pts[stop]
    vect_u, vect_v = vectorise(first_point, last_point, d, d_width)
    same_way = not point_on_with_x(list_pts[start], list_pts[stop], add_vect(list_pts[start], vect_u)[0])[0]
    up_segment = start, neighbours[start][0]
    down_segment = start, neighbours[start][1]
    flight_points = [first_point]
    stops = [first_point]
    for i in range(1, math.ceil(max_dist / d_length)):
        x = add_vect(first_point, vect_u, i, same_way)[0]
        up_on, point_up = point_on_with_vectors(first_point, vect_u,
                                                list_pts[up_segment[0]], list_pts[up_segment[1]], x)
        down_on, point_down = point_on_with_vectors(first_point, vect_u,
                                                    list_pts[down_segment[0]], list_pts[down_segment[1]], x)
        # print('Loop step on_up : ', up_on)
        if not up_on:
            '''If the point isn't  on the side of the area. We need to change the up_segment.'''
            # print('In the loop because False')
            a, b = up_segment
            if neighbours[b][0] == a:
                # print('if')
                up_next_neighbour = neighbours[b][1]
            else:
                # print('else')
                up_next_neighbour = neighbours[b][0]
            up_segment = b, up_next_neighbour
            up_on, point_up = point_on_with_vectors(first_point, vect_u,
                                                    list_pts[up_segment[0]], list_pts[up_segment[1]], x)
            # print('Test step on_up : ', up_on)
        if not down_on:
            '''If the point isn't on the side of the area. We need to change the down_segment.'''
            a, b = down_segment
            if neighbours[b][0] == a:
                down_next_neighbour = neighbours[b][1]
            else:
                down_next_neighbour = neighbours[b][0]
            down_segment = b, down_next_neighbour
            down_on, point_down = point_on_with_vectors(first_point, vect_u,
                                                        list_pts[down_segment[0]], list_pts[down_segment[1]], x)
        if i % 2 == 0:
            stops += stops_on_a_line(point_up, point_down, d_width)
            flight_points.append(point_up)
            flight_points.append(point_down)
        else:
            stops += stops_on_a_line(point_down, point_up, d_width)
            flight_points.append(point_down)
            flight_points.append(point_up)
    flight_points.append(list_pts[stop])
    stops.append(list_pts[stop])
    return flight_points, stops
```

ANNEXE 7: Autopilot service

```
609  def AutopilotService(connection_mode, operation_mode, external_broker, username, password):
610      global op_mode
611      global external_client
612      global internal_client
613      global state
614
615      state = 'disconnected'
616
617      print ('Connection mode: ', connection_mode)
618      print ('Operation mode: ', operation_mode)
619      op_mode = operation_mode
620
621      # The internal broker is always (global or local mode) at localhost:1884
622      internal_broker_address = "localhost"
623      internal_broker_port = 1884
624
625      if connection_mode == 'global':
626          external_broker_address = external_broker
627      else:
628          external_broker_address = 'localhost'
629
630      print ('External broker: ', external_broker_address)
632      # the external broker must run always in port 8000
633      external_broker_port = 8000
634
635      external_client = mqtt.Client("Autopilot_external", transport="websockets")
636      if external_broker_address == 'classpip.upc.edu':
637          external_client.username_pw_set(username, password)
638
639      external_client.on_message = on_external_message
640      external_client.connect(external_broker_address, external_broker_port)
641
642      internal_client = mqtt.Client("Autopilot_internal")
643      internal_client.on_message = on_internal_message
644      internal_client.connect(internal_broker_address, internal_broker_port)
645
646      print("Waiting....")
647      external_client.subscribe("+/autopilotService/#", 2)
648      internal_client.subscribe("+/autopilotService/#")
649      if operation_mode == 'simulation':
650          external_client.loop_forever()
651      else:
652          external_client.loop_start()
653      internal_client.loop_start()
```