Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Autonomous detection of high-voltage towers employing unmanned aerial vehicles

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Ferran Colomé Sanz

DIRECTOR: Miguel Valero Garcia

DATA: 28 de maig del 2023

Títol: Detecció de torres d'alta tensió de forma autònoma empleant vehicles aeris no tripulats

Autor: Ferran Colomé Sanz

Director: Miguel Valero García

Data: 28 de maig del 2023

Resum

El monitoratge i la inspecció de la xarxa elèctrica és extremadament important per tal de prevenir falles i aturades del subministrament elèctric. Tradicionalment, les metodologies més emprades per a la inspecció de torres d'alta tensió i línies elèctriques inclouen inspeccions visuals realitzades per personal qualificat, inspeccions mitjançant helicòpters i, en alguns casos, l'ús de robots especialitzats, d'entre d'altres. L'objectiu d'aquestes inspeccions són: la detecció d'anomalies properes a la infraestructura, anomalies per punts calents als aïllaments o bé defectes visuals dels diferents elements estructurals.

Un dels grans inconvenients d'aquestes tècniques recau en un elevat cost econòmic i la manca de precisió. Com a alternativa, l'ús d'aeronaus no tripulades (UAV) cada cop està esdevenint més popular al mercat i gradualment la tendència comença a decantar-se cap aquesta tecnologia, ja que permeten una reducció important dels costs, una millor mobilitat i flexibilitat, així com un gran potencial per obtenir imatges d'alta qualitat.

En aquest treball de final de grau (TFG) es desenvoluparà un sistema capaç de detectar torres d'alta tensió de forma autònoma mitjançant una aeronau no tripulada i s'estudiarà la seva viabilitat per a utilitzar-se en inspeccions de línies elèctriques. Aquest sistema estarà compost per una aplicació d'escriptori, que permetrà a un usuari programar diversos trams d'un pla de vol, i un dron que s'encarregarà d'executar-los i detectar les torres d'alta tensió mitjançant un model d'intel·ligència artificial.

El sistema desenvolupat en aquest treball forma part d'una contribució al *Drone Engineering Ecosystem* (DEE), plataforma de control i monitoratge de vehicles aeris no tripulats mitjançant aplicacions d'escriptori i/o aplicacions web. L'objectiu principal d'aquesta plataforma és la millora i la continuïtat del desenvolupament per part de futurs alumnes.

Title: Autonomous detection of high-voltage towers employing unmanned aerial vehicles

Author: Ferran Colomé Sanz

Director: Miguel Valero García

Date: May 28th, 2023

Overview

Monitoring and controlling the power grid is extremely important to prevent power outages and blackouts. Traditionally, the most common methods for inspecting high-voltage towers and power lines include visual inspections by qualified personnel, inspections by helicopters, and, in some cases, the use of specialized robots, among others. These inspections aim to detect anomalies near the infrastructure, anomalies due to hotspots in the insulation or visual defects in the various structural elements.

One of the main problems of these techniques is the high economic cost and the lack of accuracy. As an alternative, unmanned aerial vehicles (UAVs) are becoming more popular in the market, and the trend is gradually moving towards this technology, as it offers significant cost reduction, better mobility and flexibility, and great potential for obtaining high-quality images.

This thesis (TFG) studies the feasibility of developing a system capable of autonomously detecting high-voltage towers using an unmanned aerial vehicle and conducting power line inspections. This system consists of a desktop application that allows the user to program legs of a flight plan, and a drone that executes them and detects the high-voltage towers using an artificial intelligence model.

The system developed in this study is part of a contribution to the *Drone Engineering Ecosystem* (DEE), a platform for controlling and monitoring unmanned aerial vehicles using desktop and/or web applications. The main goal of this platform is the improvement and continuity of its development by future students.

Agraïments

En primer lloc, vull agrair als meus pares tot l'esforç, el sacrifici i el costat que m'han brindat al llarg d'aquest camí, així com el suport i la motivació durant els moments més complicats.

Vull donar les gràcies als meus amics, companys i professors per haver-me acompanyat durant aquesta etapa, especialment al professor Miguel Valero, qui m'ha ajudat i assessorat, fent possible aquest projecte.

Finalment, vull dedicar-li aquest treball a una persona molt especial amb qui hagués volgut compartir aquest moment. Allà on siguis, això és per a tu avi.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

UAV	Unmanned Aerial vehicle
RPAS	Remotely Piloted Aircraft Systems
ICAO	International Civil Aviation Organization
ITU	International Telecommunication Union
RF	Radio Frequency
MTOW	Maximum Take-Off Weight
VLOS	Visual Line of Sight
EVLOS	Extended Visual Line of Sight
BVLOS	Beyond Visual Line of Sight
VMC	Visual Meteorological Conditions
VTOL	Vertical Take Off and Landing
ESC	Electronic Speed Controller
DEE	Drone Engineering Ecosystem
EMI	Electromagnetic Interference
CTR	Controlled Zones (Airspace)
ATZ	Aerodrome Traffic Zone
FOV	Field of View
HFOV	Horizontal Field of View
VFOV	Vertical Field of View

1. INTRODUCTION

1.1. Introduction

In today's world, mobile phones, computers, and other electronic devices are indispensable in our day a day. In the course of technological advancement, an increasing number of devices require electricity to operate. In order to accomplish this, power plants generate the necessary amount of electrical energy and distribute it through a network of power lines to their respective destinations.

To prevent power failures and potential blackouts, it is extremely important to monitor and inspect the power grid. The most commonly used methods for inspecting transmission towers and power lines are human inspectors, crawling robots, and helicopters. Nevertheless, these methods may pose a threat to the safety of the staff, require high costs, and may not provide enough precision. Due to their flexibility, low operational costs, high mobility, and potential to obtain high-quality images, unmanned aerial vehicles (UAVs) are becoming increasingly popular as an alternative.

This project involved the development of a desktop application which allows a user to create flight plans for a fully autonomous tower detection by means of UAV. The flight will be carried out in conjunction with an artificial intelligence trained model to detect towers through the drone's camera and take pictures that will be saved in a directory.

1.2. Motivation

My motivation for this study comes from my passion for aviation and the desire to put into practice the knowledge acquired during my college studies to do my bit in the development of technological solutions that can be exploited shortly.

The Telecommunications Engineering bachelor's degree associated with the mention of Systems imparted at *Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels* (EETAC) has mainly provided me with more analytical thinking, as well as the capacity for synthesis and abstraction to deal with technical problems.

During this study, I have put into practice concepts acquired during the degree, especially about RF and programming areas. Moreover, to develop this project from scratch, it has been necessary to learn:

- Python programming environment and library usages, such as *Tkinter*, *CTKinter*, *Dronekit, OpenCV, Torch,* and *Geodesic* among others.

- Extraction and data pre-processing from XML-formatted files.

- Segmentation, pre-processing, and data augmentation of datasets.
- Training and evaluation of object-recognition models.
- Design of 3D-printed parts using CAD software tools.
- Vector graphics software editors to generate flowcharts and illustrations.
- Setting up and data analysing from the EMFACDC simulator.
- Calculation of distances between geographical coordinates.
- GitHub repository management
- Understanding and applying MQTT brokers
- Gantt Project planification software

1.3. Structure

This report is structured as follows:

2. Context. This chapter intends to put the reader in context about the working ecosystem used, the problems of the current methodology, and general descriptions of the elements involved in the study.

3. Objectives and work plan. The objectives set for the development of the system and the planning that has been carried out to achieve them.

4. System Development. This chapter will discuss in detail how the system has been developed for each phase, as well as the problems that have appeared and how they have been solved.

5. RPAS Regulation. This chapter reviews the most relevant aspects of drone regulations and their use in airspace and justifies certain decisions taken in *System Development*.

6. Conclusions. Finally, this chapter will present the conclusions regarding the feasibility of the system, considering all the previous points and interesting future upgrades.

2. CONTEXT

This chapter aims to provide the reader with a comprehensive approach to understand the development that has been carried out in this study. The first section will broadly discuss how power line inspections are carried out, as well as the most common systems used. The second and third sections will delve into detail regarding the hardware and software platforms that have been utilized to develop the project. Finally, the fourth section briefly introduces object detection technology and the model used in this project.

2.1. Overhead power line inspection

There are several common systems used to inspect power lines and also to detect issues such as equipment wear and tear, damage, or vegetation encroachment, which can pose safety risks or disrupt the power supply. Some include:

- Aerial inspections: Involves the use of helicopters or drones equipped with high-resolution cameras or LIDAR sensors to conduct visual or 3D inspections of the power lines and their surroundings.

- Ground-based inspections: These involve using vehicles or foot patrols to inspect power lines and equipment up close. Ground-based inspections are usually conducted to verify issues detected by aerial inspections.

- Thermal inspections: These involve using infrared cameras to detect hotspots on the power lines, which can indicate equipment defects, loose connections, or other issues.

- Ultrasonic inspections: These involve using ultrasonic sensors to detect cracks, corrosion, or other defects in the metal components of power lines.

- Laser scanning: This involves using laser scanners to create a 3D model of the power line infrastructure and identify structural issues such as sagging power lines or bent towers.

Although there are several different methods and systems for the inspection of high-voltage towers, the focus of this section will only be on aerial inspection methods for overhead power lines.

The Spanish electrical transmission grid is made up of more than 44,000 km of high-voltage overhead lines, and more than 6,000 substation positions [7]. It is therefore desirable to use aircraft that have considerable range to carry out inspections in a consistent time. Such inspections are carried out at low speed by following the towers while a camera operator takes pictures of the areas of interest, which will later be processed and analyzed.

2.1.1. Camera

It is necessary to dedicate a section to the importance of the camera to carry out this type of work correctly. Must have several features to ensure they capture accurate and detailed images of the power lines and the surrounding environment.

Some basic key features include:

- High-resolution: Essential for capturing detailed images of power lines and their surroundings. Cameras with at least 20 megapixels are recommended.

- Zoom capabilities: The ability to zoom in and out is crucial for getting close-up shots of equipment and identifying potential issues. Cameras with at least 20x optical zoom are recommended.

- Stabilization: Vibrations or movements can affect the quality of images captured during aerial inspections. A camera with a built-in stabilization system or a gimbal mount can help ensure stable, clear images.

Fig. 2.1 DSLR camera above and thermal camera below (left) and *WESCAM MX-15* [27] camera (right).

Aerospace gyrostabilized cameras can be mounted on helicopters or drones to capture detailed images and video of power lines and surrounding infrastructure, some of the best known are *WESCAM* or *FLIR Systems* among others.

WESCAM cameras feature multiple sensors, including high-resolution cameras, infrared cameras, and laser range finders, which enable them to capture images in a wide range of conditions, including low-light, high-altitude, or extreme temperatures. The cameras also feature advanced image stabilization technology, which helps to ensure clear and steady images even when the camera is moving at high speeds. Despite being technologically very advanced, it also has a high acquisition cost (Fig. 2.1).

There are other possibilities such as Digital Single Lens Reflex (DSLR) cameras. DSLRs are popular among professional and amateur photographers due to their versatility, image quality, and range of features, see Fig.2.2.

Fig. 2.2 Sequence taken from a stabilized DSLR camera.

Alternatively, infrared (IR) cameras are commonly used for power line inspections due to their ability to detect thermal energy emitted by objects. Infrared cameras capture images in the infrared spectrum of light, which is beyond the range of human vision. This allows them to detect hotspots on power lines and electrical equipment, which can indicate potential issues such as loose connections, corrosion, or equipment defects, see Fig. 2.3.

IR cameras require several features to ensure they capture accurate and detailed images. Some key features include:

- High-resolution: Essential for capturing detailed images of the power lines and their surroundings. Cameras with at least 640x480 pixels are recommended for these applications.

- Temperature range: Infrared cameras need to have a temperature range that is suitable for the temperature range of the objects being inspected. Cameras with a temperature range of at least -20°C to 500°C are recommended for power line inspections.

- Sensitivity: Infrared cameras need to be sensitive enough to detect small temperature differences. Cameras with a thermal sensitivity of at least 0.05°C are recommended.

- Stabilization: Vibrations or movements can affect the quality of images captured during aerial inspections. A camera with a built-in stabilization system or a gimbal mount can help ensure stable, clear images.

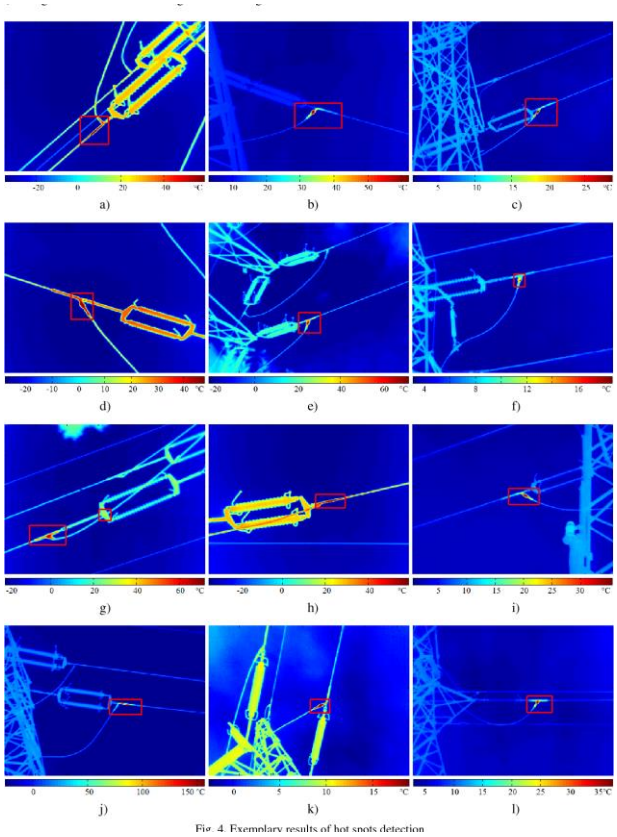

Fig. 2.3 IR hotspot detection [17]

2.1.2. LIDAR

Light Detection and Ranging (LIDAR) technology is used to create highly accurate and detailed 3D images of power lines and their surroundings.

LIDAR works by emitting laser pulses and measuring the time it takes for the pulses to bounce back after hitting the surface of an object (2.1). This enables the creation of high-resolution 3D maps of the surrounding terrain, vegetation, and structures.

$$\mathrm{d} = \frac{c \cdot t}{2} \qquad \textbf{(2.1)}$$

From Equation (2.1), c represents the speed of light and t the time it takes for the laser ray to travel back and forth to the LIDAR.

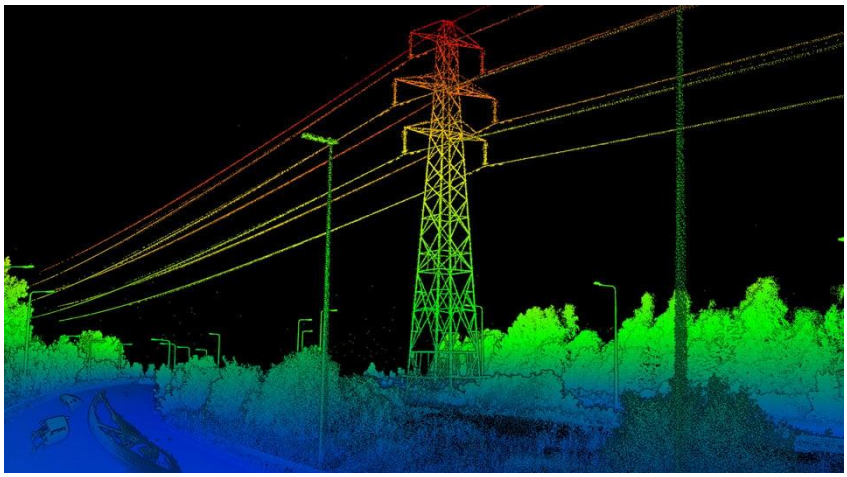

Fig. 2.4 Tower and power line 3D-scanned with LIDAR [16]

The data collected by the LIDAR system can be used to create a 3D model of the power lines and surrounding area, as shown in Fig. 2.4 so that inspectors can identify potential problems such as plant growth, sagging lines, or damage to equipment.

2.2. UAV platform

A drone platform is a type of unmanned aerial vehicle (UAV) that serves as a hardware base or frame for various types of payloads, including cameras, sensors, and other equipment. A drone platform typically consists of a flight control system, motors and propellers, and a payload attachment system.

Fig. 2.5 UAV platform used in this project [8]

Some UAV platforms can be adapted and configured in various ways to suit different applications (Fig.2.5). Some platforms are designed for specific payloads, such as high-resolution cameras or LIDAR sensors, while others are more general-purpose and can be used for a wide range of payloads.

2.2.1. Frame components

This quadcopter consists of a *Hexsoon EDU450* kit (Fig. 2.6) designed specifically for the Pixhawk Cube autopilot, which is also compatible with other flight controllers. The vehicle frame is perfect for development, testing, or just hobby use. The kit includes all the hardware needed to complete the frame (cables, power distribution, motors, ESC...).

Fig. 2.6 *Hexsoon EDU450* kit during its assembly [8]

This quadcopter uses four brushless motors to generate the rotational motion necessary for the propellers to generate enough lift to fly the vehicle. These motors are powered by an electronic speed controller (ESC); these devices allow the motor speed to be adjusted by pulse width modulation (PWM).

The PWM signal is a series of square-wave pulses, and the width of each pulse determines the position of the motor rotor. The PWM signal is usually generated by the flight controller, which determines the motor speed based on inputs from the ground station or automatic flight control algorithms (see Fig. 2.7).

The PWM signals are sent to the ESC, which interprets the width of each pulse and adjusts the voltage and current to the motor accordingly. The ESC converts the PWM signals into a variable voltage and current that drives the motor.

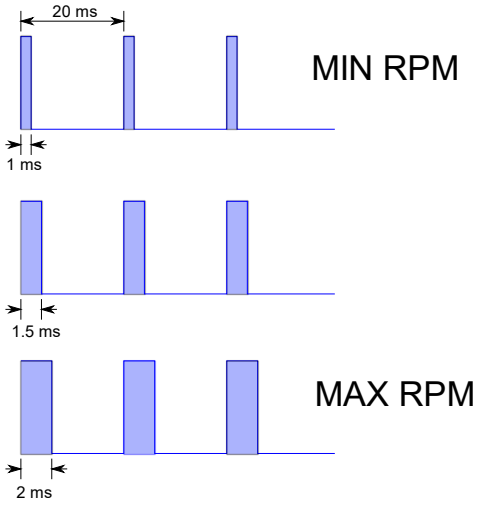

Fig. 2.7 PWM with different pulse width

2.2.2. Flight controller

During the flight, the flight control module generates a series of PWM signals to control operations such as takeoff, landing, forward, left, right, and yaw.

It is also possible for the flight controller to execute a flight plan independently. In this case, a ground station is required where the flight plan can be configured beforehand.

The flight plan can be sent to the controller and then retrieved from telemetry data collected by the controller during the flight. Also, this system can generate the proper control signals for the motors to maintain the stability of the drone.

Fig. 2.8 Pixhawk Orange Cube [15]

The flight control module built into this drone is Pixhawk's Orange Cube (Fig. 2.8), which is designed to run ArduPilot software. It includes a more powerful

processing unit and an improved power supply system for higher energy efficiency and better performance in extreme environments. Some notable improvements include:

- Processing unit: 32-bit ARM Cortex-M7 processor, enabling it to handle complex flight control algorithms and sensor fusion. Also mounts a fail-safe co-processor.

- Memory: 4MB of flash memory and 1MB of RAM, providing ample space for storing flight data and running complex algorithms.

- Improved sensor capabilities: GPS, accelerometers, gyroscopes, and magnetometers.

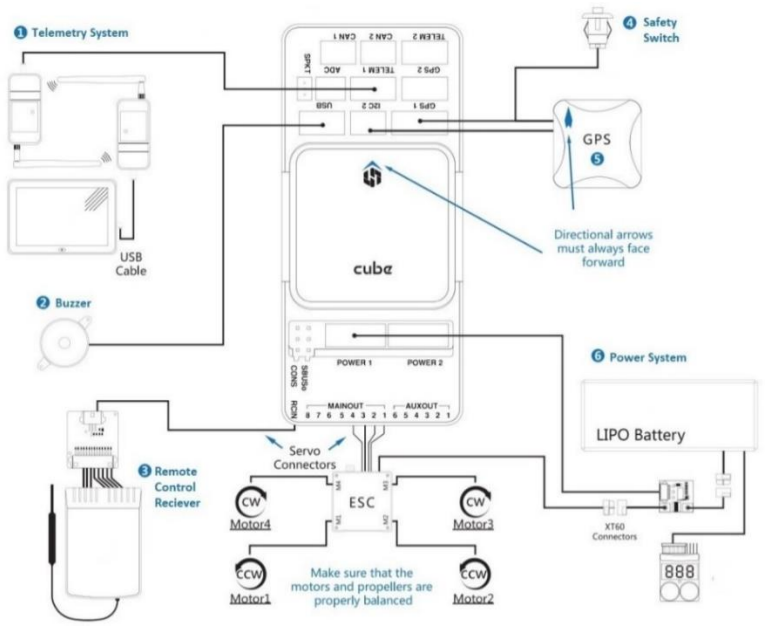

Fig. 2.9 Pixhawk Orange Cube connection scheme [15]

Fig. 2.9 shows the most relevant peripherals connections:

1. Telemetry System: The user can plan/run missions and control and monitor the vehicle in real time. Typically includes telemetry radios, tablet/PC, and ground station software.

2. Buzzer: Provides audio signals to indicate different UAV configuration information.

3. Remote Control Receiver System: Connects via radio to a transmitter that an operator can use to fly the vehicle manually.

4. Safety switch (true purpose): Press and hold to lock and unlock motors. Only required if non-using the recommended GPS with an inbuilt safety switch.

5. GPS, Compass, LED, Safety Switch: The recommended GPS module contains GPS, Compass, LED, and Safety Switch.

6. Power System: Powers Cube and the motor ESCs. Consists of a LiPo battery, power module, and optional battery warning system (audio warning if battery power goes below a predefined level).

2.2.2.1. Mission Planner Ardupilot SITL simulator

ArduPilot is an open-source autopilot software package that can be used to autonomously control drones, ground vehicles, and watercraft (Fig. 2.10). Due to its flexibility, reliability, and user-friendly interface, it is a popular choice for drone enthusiasts and professional developers. Some of these software features are:

- Flight control: *ArduPilot* includes a sophisticated flight control system that can stabilize the aircraft, maintain altitude and speed, and perform complex manoeuvres.

- GPS navigation: The software suite supports GPS navigation, which allows the drone to follow pre-programmed flight paths or fly to specific locations.

- Telemetry and communication: *ArduPilot* supports telemetry and communication systems that allow the operator to monitor the status of the aircraft and control it remotely.

- Autonomous operation: *ArduPilot* supports autonomous operation, allowing the drone to make decisions and adapt to changing conditions without human intervention.

- Mission planning: The software suite includes a mission planning tool that allows users to plan and execute complex flight missions.

Fig. 2.10 *ArduPilot* flight simulation at UPC *DroneLab*

2.2.3. Raspberry Pi

A dynamic flight means that an on-board computer can change the flight course during a mission if certain events have been detected by other on-board devices such as the camera or any sensor.

On the UAV platform used in this project, a Raspberry Pi (RPi) is mounted as the on-board computer, see Fig. 2.11. The programs running on this on-board computer are written in Python. These programs also use the *DroneKit* library to interact with the flight control module and other devices on the platform.

Fig. 2.11 Raspberry Pi board (left) and the RPi V2.1 camera (right)

The Raspberry Pi kit includes a high-quality camera designed specifically for use with the RPi. The camera module itself is small and lightweight, making it ideal for use in small or portable projects, here are some features:

- High-resolution image and video capture: the camera can capture images at up to 8MP resolution and 1080p video at 30 frames per second.
- Interchangeable lenses: The camera module has a standard C-mount lens connector, so users can easily change lenses for different applications.
- Low-light performance: The camera has a large aperture and a high-quality sensor, so it can capture clear images even in low-light conditions.
- Programmable control: The camera can be controlled programmatically via the pins of the Raspberry Pi.

2.3. Drone Engineering Ecosystem

This study is part of a contribution to the *Drone Engineering Ecosystem* (DEE).

DEE is a software platform consisting of different modules that enable a drone to fly from different devices and perform certain operations. This platform began as a proof-of-concept that can be used to perform various flight tests on a UAV model at the UPC *DroneLab*. It is also constantly being developed and improved based on the various contributions of EETAC students.

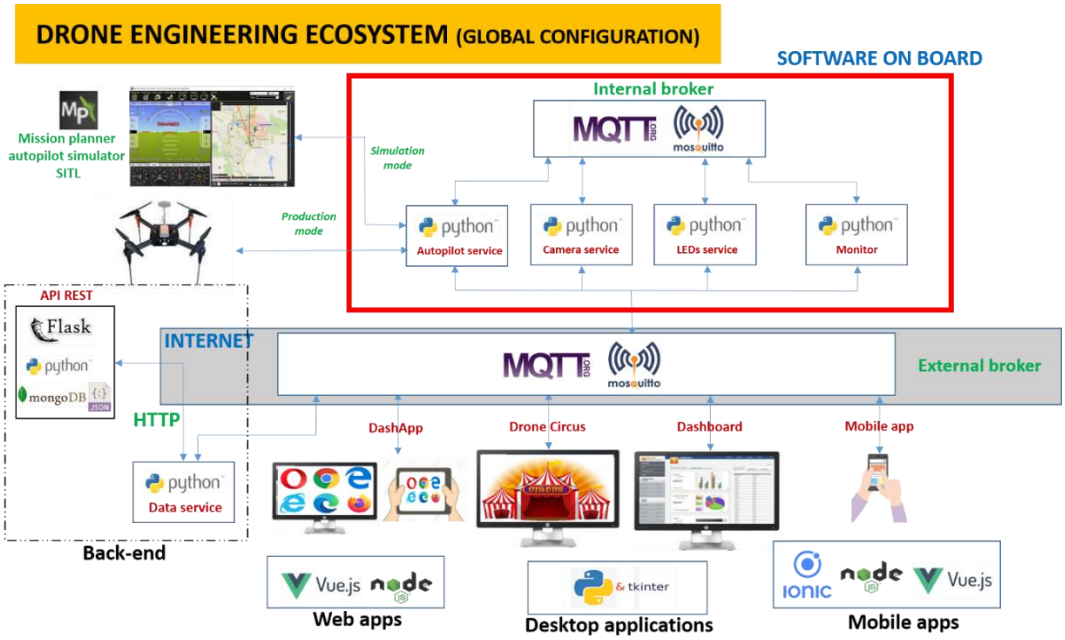

Fig. 2.12 Global DEE scheme [8]

Some modules can be run on the RPi of the drone (red box in Fig. 2.12).

These modules control various devices on the drone platform, such as the autopilot, camera, LEDs, servo, etc. Some others are front-end applications that can be launched from a computer or a cell phone and allow the user to control the vehicle.

The modules of the ecosystem are in constant development. In the following sections, you will find a brief description of the current modules on DEE.

2.3.1. On-board modules

The main modules running on the RPi are the autopilot, the camera service, and the LED services.

2.3.1.1. Camera service

The camera service provides images to the other modules of DEE as needed. Some applications need the camera service to provide a single image or to start/stop a video stream.

This service is used in this study and therefore needs to be modified in its code to make the communication between the drone and the developed application compatible.

2.3.1.2. Autopilot service

The autopilot service is an on-board module that runs on the RPi board and controls the operation of the flight control module.

Dashboard applications need the autopilot service to connect to the flight controller, arm the drone, take off, approach a specific position, or move in any direction, land, stop, etc.

The autopilot is also used to send the same flight commands to the *Mission Planner Ardupilot SITL* simulator when it is in simulation mode, or directly to the drone when it is running in production mode.

Like the camera service, this service is also used in this study and therefore needs to be modified in its code to make the communication between the drone and the developed application compatible.

2.3.1.3. LEDs service

The LEDs service is an on-board module that controls the LEDs and the servo installed in the UAV platform, as required by the rest of the modules at the *Drone Engineering Ecosystem*.

Dashboard or mobile applications will require the LEDs service to light certain LEDs with a given RGB colour of to move the servo to drop an object. This module is not used in this project.

2.3.2. Front-end modules

There are several front-end applications developed in DEE. Below is a brief description of each application:

- **Mobile App**: Web app (Vue + Ionic) with a reduced set of functionalities that can be operated from a cell phone or tablet connected to the Internet.

- **Dashboard**: A desktop application that uses all the services on board the drone.

- **DashApp**: Web application (Vue) with similar functionalities to the Dashboard, but which can be controlled remotely from a laptop connected to the Internet.

- **Drone Circus**: A desktop application that allows the drone to be controlled by voice or body postures. It is designed to allow the audience participating in the exhibition to interact with the drone in a safe and fun way.

Finally, there are other modules such as API REST, a server that allows the storage and retrieval of data through HTTP base operations, and the data service that controls the storage and retrieval of data on API REST as required by the other modules.

The desktop application to be developed in this study will provide a new functionality in DEE, that allows the creation of flight plans for the inspection of transmission towers.

2.3.3. Communication mode

Mosquitto is a message broker that implements the MQTT (Message Queuing Telemetry Transport) protocol. It was developed by the Eclipse Foundation and can be used to create a distributed network of devices that communicate with each other using the publish/subscribe messaging pattern.

MQTT uses a binary protocol that is intended to be efficient in terms of bandwidth usage and processing power from the CPU. The protocol is intended to be simple,

flexible, and scalable so that it can be used in a variety of applications. This communication protocol was also chosen because of the low processing power of the RPi.

The *Drone Engineering Ecosystem* can be operated in global or production mode, as shown in Fig. 2.13.

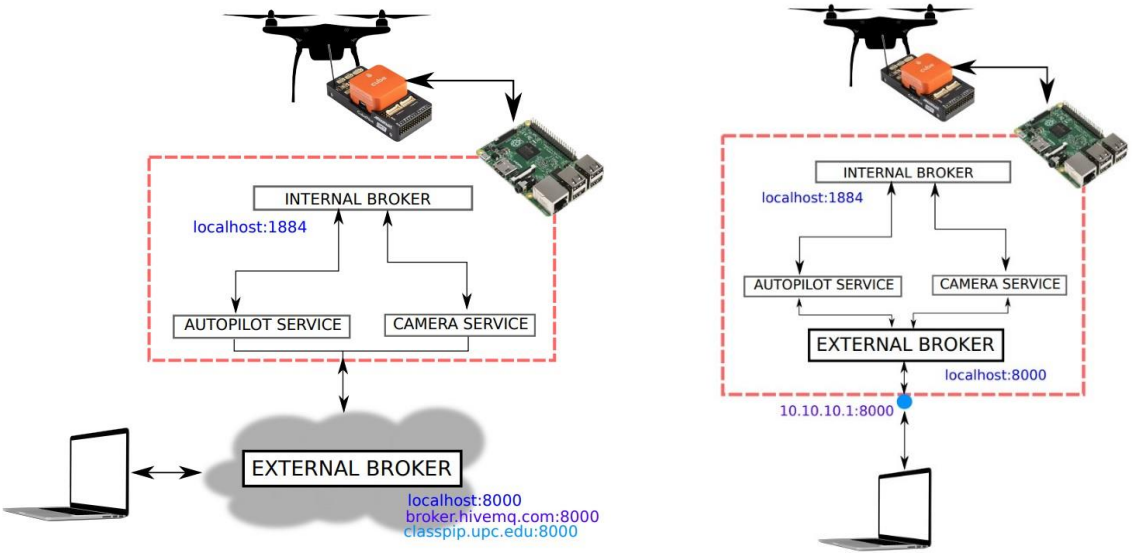

Fig. 2.13 Global mode (left) and production mode (right)

In global mode, it is assumed that the UAV platform's front-end and back-end modules are all connected to the internet and communicate through an external broker. The external broker can be any publicly available broker (*broker.hivemq.com*) or the private broker running on a server in the campus facilities, for which credentials are required.

If there is no internet coverage, production mode should be used.

In this case, the front-end module (e.g., the user application) must connect to a Wi-Fi access point provided by the built-in RPi. The external broker is also running on the RPi, and the on-board modules connect to this broker via *localhost:8000*, but the external modules should connect to the *10.10.10.1:8000* access point.

As shown in Fig. 2.13, an external and an internal broker are needed.

The external broker is responsible for the communication of all external devices with the services running in the RPi of the drone, such as the camera or the autopilot.

The internal broker enables the communication between the different RPi services and always runs on port 1884.

2.3.4. Operation mode

DEE can be executed in production mode or simulation mode.

The production mode corresponds to the actual execution of missions with the UAV. Of course, in this case, the on-board services have to be run on the Raspberry Pi.

In simulation mode, all modules (including the brokers) run on the same computer (e.g. a laptop). In this case, the *Mission Planner SITL* is needed, which contains a simulator that is controlled by the autopilot service exactly as it would run in production mode.

2.4. Object detection

When the drone approaches the exact location of the tower to be inspected, it must use the camera to autonomously identify where the tower in question to take a picture of it. This section describes the basic concepts of image-based object recognition.

Object recognition with machine learning (ML) is a machine vision technique that uses algorithms to identify objects in an input image or video stream and then classify them into predefined categories.

Based on the number of times the same input image is passed through a network, object recognition algorithms fall into two broad categories: Single-shot object detection (YOLO, SSD) and Two-shot object detection (RCNN, Fast RCNN...).

Single-shot object detection traverses the input image in a single pass to make predictions about the presence and position of objects in the image. It processes an entire image in a single pass, making it computationally efficient but less accurate than two-shot object detection models.

YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process images.

2.4.1. YOLO model

One of the most popular families of models for object recognition is YOLO ("You Only Look Once"). YOLO models are much faster than R-CNN (regions with CNN) and allow real-time recognition of image objects.

This model takes an image as input and uses a simple deep convolutional neural network (CNN) to recognize objects.

YOLO splits the input image into an MxM grid. Image classification and localization are applied to each cell of the grid. Given several *C* classes, the label y for each grid is a *5*+*C* dimensional vector, see Fig. 2.14.

A dimensional vector *y* gives all the necessary information about each cell, namely:

- *pc*: Defines if an object is present in the grid or not (it's the probability).
- *bx, by, bh, bw*: Specifies the bounding box position and dimensions.
- $c_1, c_2, \ldots c_C$: Represent the class types.

y =	pc
	bx
	by
	bh
	bw
	c_1
	…
	c_j

Fig. 2.14 Dimensional vector example

If there is no object in a cell, the *pc* field becomes zero and the rest of the attributes are *"?"* unknown.

For each cell containing an object, YOLO predicts a bounding box around the object. The bounding box is described by four coordinates: the x and y position of the centre of the box (*bx, by*) and its width and height relative to the cell dimensions (*bh, bw*), see Fig. 2.15.

To predict the class within a bounding box, a score is assigned to each class in the training dataset. The class with the highest score is considered the predicted class for that object.

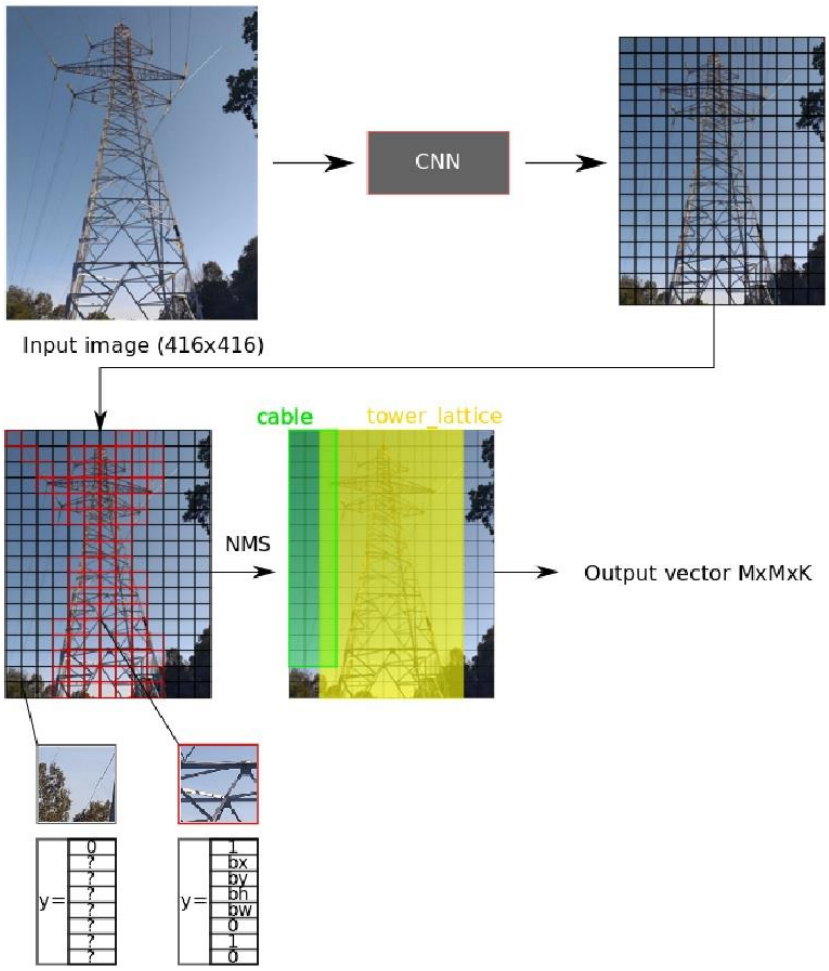

Fig. 2.15 Simplified YOLO working scheme.

Finally, a key technique is used in YOLO models: NMS (Non-Maximum Supression) is a post-processing step used to improve the accuracy and efficiency of object detection. Multiple bounding boxes for a single object in an image are common in object detection.

These bounding boxes may overlap or be at different positions. However, they all represent the same object. NMS is used to identify and remove redundant or incorrect bounding boxes. It also outputs a single bounding box for each object in the image.

In the end, the final output from YOLO is a vector of bounding boxes with object scores, classes, and coordinates. This list is: MxMxK, where MxM is the number of cells in the grid and K is the size of the dimensional vector (*5+C*).

Compared to YOLO, YOLOv5 uses a more complex architecture called *EfficientDet*, see Fig. 2.16. By using a more complex architecture, YOLOv5 achieves higher accuracy and better generalization to a wider range of object categories.

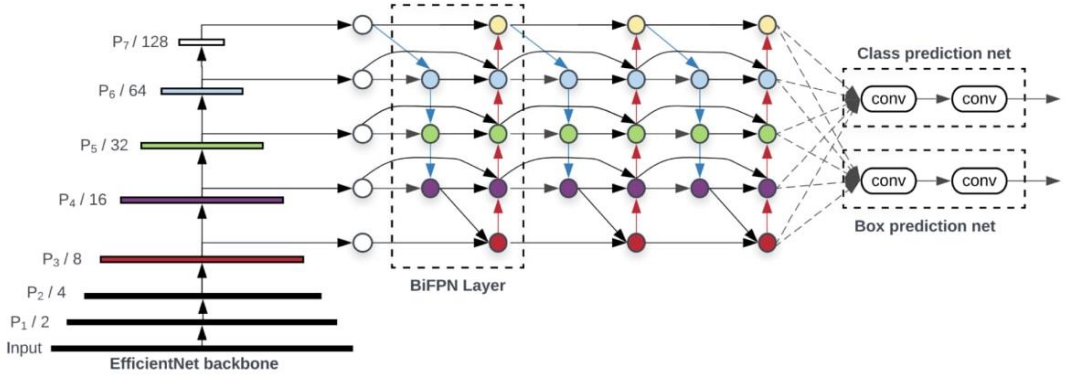

Fig. 2.16 YOLOv5 *EfficientDet* architecture[14]

*EfficientDe*t is based on a backbone network called *EfficientNet*, which is designed to achieve better accuracy and efficiency than existing networks by optimizing the scaling of network depth, width, and resolution.

The architecture creates a feature pyramid by connecting different layers of the feature extraction network. Each level of the pyramid has different image resolutions and different levels of feature abstraction. In this way, objects of different sizes and scales can be detected in one image.

BiFPN (Backbone Feature Pyramid Network) is used to merge features from different levels of the feature pyramid in an efficient way.

Finally, a convolutional network is used to detect objects in the image. This network uses the feature pyramid generated in the previous step and generates a list of recognition candidates with their class and recognition probability.

There are a variety of models within the YOLO family, which are shown in Table 2.1.

Table 2.1 Comparison of YOLO models trained on a COCO (Common Objects in Context) dataset for 300 epochs and an image size of 640 using an NVIDIA A100 GPU [11]

Model	Size (pixels)	mAP50 box (%)	mAP50 mask (%)	Speed A100 (ms)	Params (M)	FLOPs @640 (B)
YOLOv5n	640	27.6	23.4	62.7	1.2	7.1
YOLOv5s	640	37.6	31.7	173.3	1.4	26.4
YOLOv5m	640	45.0	37.1	427.0	2.2	70.8
YOLOv5l	640	49.0	39.9	857.4	2.9	147.7
YOLOv5x	640	50.7	41.1	1579.2	4.5	265.7

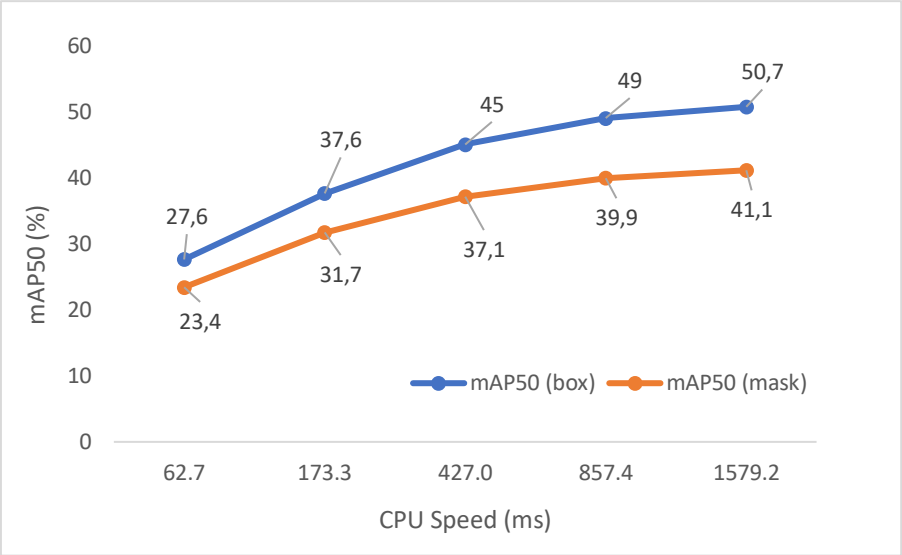

Fig. 2.17 Relationship between CPU and the average precision gain for each model.

Based on the results shown in Table 2.1 and Fig. 2.17, it can be seen that YOLOv5n "Nano" and YOLOv5s "Small" are the models that require the fewest floating-point operations (FLOPS) and therefore may be suitable for execution on embedded systems with limited computational capacity, such as a Raspberry Pi. However, these are the models that have the worst object detection accuracy.

The YOLOv5l "Medium" model requires a significant increase in computational cost and number of operations, but in return allows a significant improvement in accuracy.

The YOLOv5l "Large" and YOLOv5 "XLarge" models have a significant increase in computational cost in exchange for a slight improvement in recognition results.

For applications where a high detection rate is required regardless of computational resources, the latter two models would be the best choice. On the other hand, if a balance between CPU and accuracy is desired, the "Medium" model would be the best choice.

The "Small", "Medium", and "XLarge" models will be taken into consideration for this project. A comparison is done in Annex H to decide which model is best for the system.

3. OBJECTIVES AND PLANNING

This chapter will outline the objectives set out at the beginning of this project, as well as the planning that has been carried out.

3.1. Aim

The aim of this project is the feasibility study and development of a front-end and back-end application to provide a UAV with the necessary capabilities to carry out autonomous inspection of high-voltage transmission towers.

3.2. Scope

The scope of this study is to design, implement and prove that this system applies to a given UAV model and offers advantages over other currently used methodologies, justifying each of the decisions taken and how the problems that have arisen during the process have been solved.

The study will be carried out in two phases:

- Development of the desktop application that will allow the user to generate a flight plan to perform the desired inspection.

- Development of the algorithm with object detection to take pictures of the towers.

To achieve the expected results, the following targets have been defined:

1) Extraction and processing of the information given in the files from the company outsourcing the service.

2) Development and implementation of a user application that allows the generation of flight plans according to the UAV's limitations and the current regulations, using the data extracted from the given files.

3) Redesign of the camera support mount for horizontal focusing

4) Development of an algorithm capable of detecting objects through the UAV camera using machine learning and implementing it in the system.

5) Perform system tests in a real environment, such as at the UPC *DroneLab*, to verify its operation.

6) System integration to the DEE

7) Documentation of all the functionalities so that the system can be further developed in the ecosystem in the future.

3.3. Work plan

This section will explain the classification of the tasks to carry out the project as well as the work plan.

3.3.1. Task identification

The tasks have been classified according to their nature.

Tasks A represent the assignments dedicated to study and information research, tasks B are those performed during the actual development of the system, tasks C are those related with manufacturing and testing, tasks D for documenting the study, and tasks E as follow-up feedback with the professor:

A. Research and study
    1. Requirements for understanding
    2. Python familiarisation
    3. Python libraries
    4. Object-detection model

B. System development
    1. Archer App
        1. KMZ extraction
        2. Basic functionalities
        3. Extra functionalities
        4. Modification option
        5. *Ardupilot* + *Dronekit* implementation
    2. Object-detection model
        1. Transmission tower dataset
        2. Object-detection model validation
    3. EMI Analysis

C. Manufacturing
        1. Laptop camera USB conversion
        2. CAD design development
        3. Camera mount prototype manufacturing
        4. Poster support manufacturing
        5. Simulations and tests

D. Documentation
    1. Writing of the study

E. Feedback
    1. Follow-up meetings

3.3.2. Gantt diagram

The Gantt chart (Fig. 3.1) below details the planning for each of the most important tasks described in the previous section. Different colours are used to symbolize different task types:

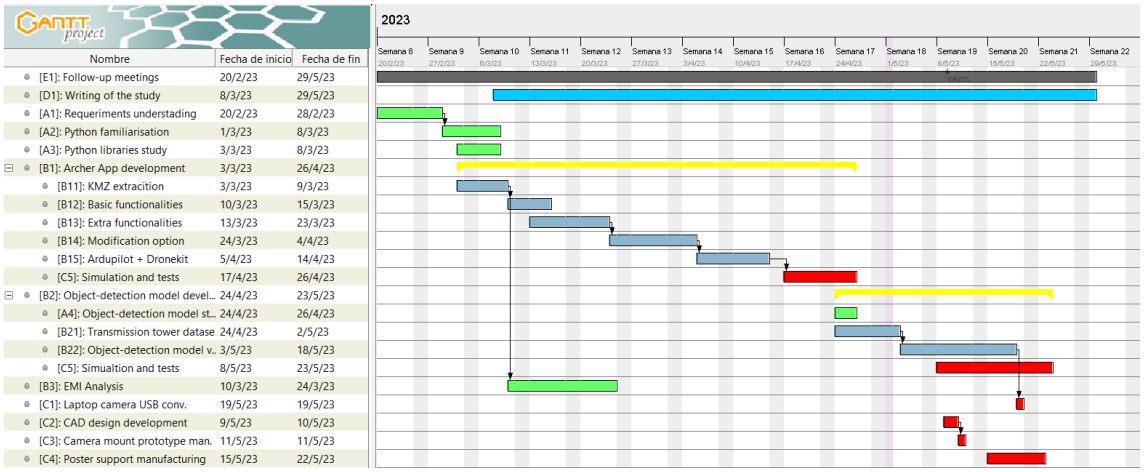

Fig. 3.1 Gantt chart planification

To understand the requirements that the system would need, the first duty for this project was to conduct research and gather data; this took up about one week.

As there was no prior knowledge of this language, it was essential to learn Python and how to use its libraries. Simpler applications like the DEE Dashboard were created to practice for this purpose. An estimated two weeks would be needed.

The development of the desktop application began when the programming language learning had been completed. For this purpose, several tasks were planned, and dedicated to the development of specific parts of the application, with constant feedback with the professor. It was determined to commit at least 1.5 months, including the necessary simulations and tests, as this first part of the study laid the groundwork.

The creation of the object recognition model, its integration with the camera, and the modification of the UAV back-end code to make it compatible with DEE were the main goals of the second stage, which was scheduled after the previous stage's verification. As a result, during this second step, everything relating to camera and image testing was taken into consideration.

4. SYSTEM DEVELOPMENT

The followed steps to design the system, the issues that were found, and how they were resolved are all covered in more detail in this chapter. There are two sections to the chapter. The development of the user application "Archer App" is discussed in the first section, and the development of the object recognition model is described in the second section.

4.1. Archer App

The creation of a desktop user application has been crucial for giving the UAV all the data required to inspect transmission towers autonomously.

The application was called "Archer App", in reference to an archer. In this case, the drone needs to hunt a target, which would be the tower to be detected.

The purpose of the Archer App application is to create a flight plan that complies with current regulations; Therefore, its basic requirements should be:

- Applicable to any UAV model
- Simple and user-friendly interface
- Allowing the creation, removal, and modification of flight plans
- Compliance with current regulations
- Compatibility for future upgrades

This chapter will be devoted to its development, justification, and resolution of problems encountered during this process.

4.1.1. Source data file processing

The company in charge of the service must have at its disposal files with information and the location of each tower and the distribution of the power line to inspect.

This file must be in a *.kmz* format that can be loaded from Google Earth, currently the companies that provide helicopter inspections use these files to mark the waypoints in the GPS. The program developed must be able to read the *.kmz* files and have access to the Internet to load the map images.

4.1.1.1. Keyhole Markup Language (KML)

KML is an XML notation for the expression of geographic annotation and visualization in two-dimensional maps and three-dimensional Earth browsers. KML was originally developed to be used with Google Earth.

KMZ files are often distributed as KMZ files, which are zipped KML files with a *.kmz* extension, see Annex B.

4.1.1.2. Data extraction process

The first step was to convert the *.kmz* file to a .zip extension file, setting apart the file name and format extension, to extract the desired data from these formatted files. Next, locating the *.zip* file and extracting any compressed *.kml* files within.

Once this was achieved, the remaining empty .zip folder was deleted and the *.kml* file was renamed to its original name. By default, it was named *"doc.kml*". This process was done in *KmzToKml.py* [8] file, see Fig. 4.1.

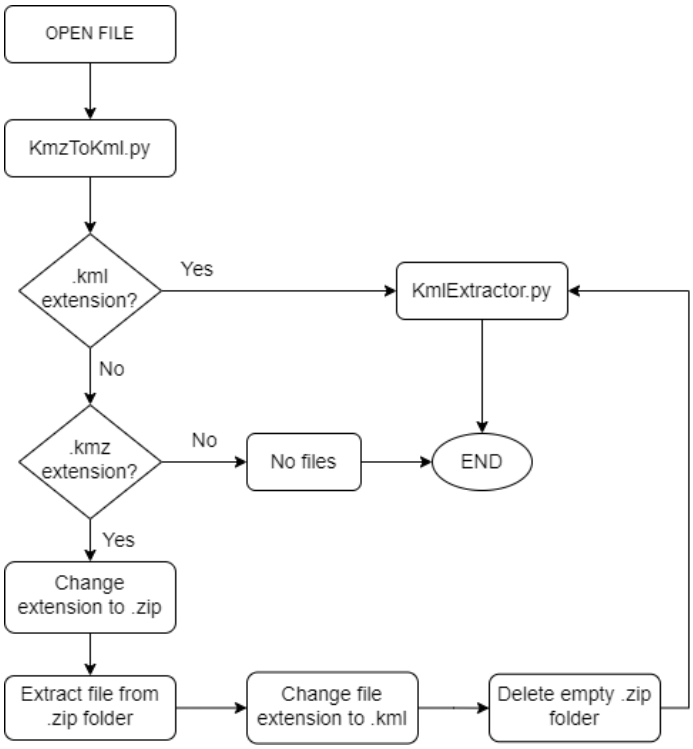

Fig. 4.1 File extension conversion flowchart

With the accomplishment of the first step, what came next was the extraction and processing (parsing) of the *.kml* file. This was done in the *KmlExtrator.py* [8] file, as shown in Fig. 4.2. By taking advantage of the hierarchical structure of XML,

by iterating over each line of the file, and by finding certain namespaces, it was possible to extract the desired data to build this project.

This was done by first setting the *xmlns* (XML namespace) attribute to "*{http://www.opengis.net/kml/2.2}*". The value of this attribute is a string specifying the namespace URI (Uniform Resource Identifier), which allows different elements or attributes with the same name to coexist in an XML document without ambiguity.

Once the *xmlns* was set up, a loop was run to find the identifier of each of the towers within each of the placemark namespaces. Within each placemark namespace opening, two further loops were used to find the coordinates and description. When all the lines in the file have been read, the process ends.

The resulting strings require data processing to clean up the element information used in the XML syntax. The desired values can be extracted and returned to the main program for use in the map using Python's split function.

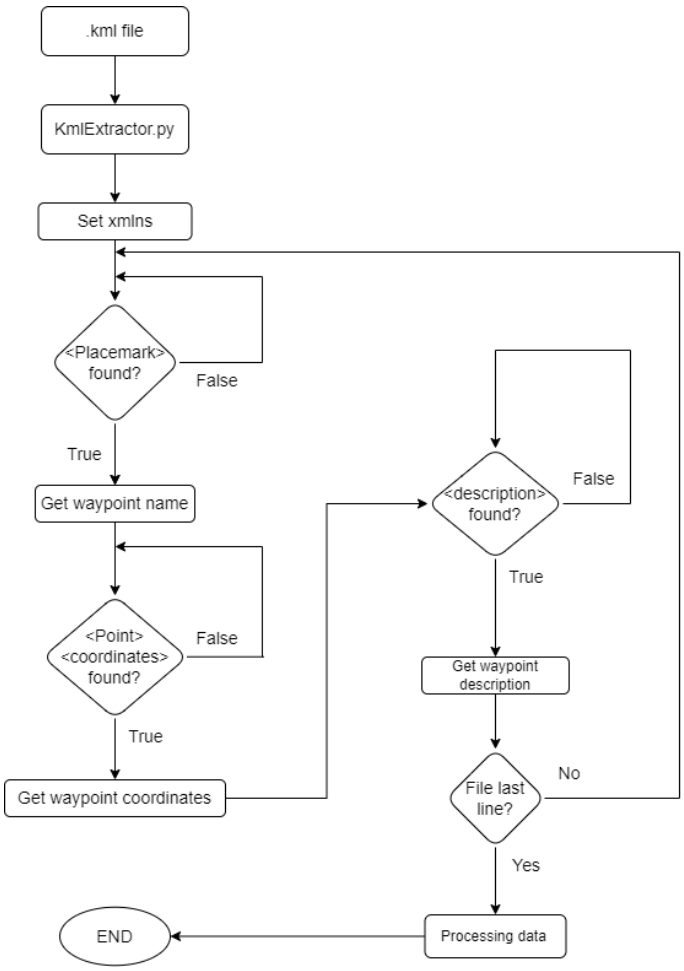

Fig. 4.2 File parsing flowchart

Required extracted data for the Archer App:

- Tower position. Geographical coordinates to represent the tower's location on the map.

- Tower identification. Unambiguous identification for each transmission tower to be shown above.

- Coordinates between towers joined by a power line. Necessary for the *"Suggested WP"* functionality, it will be discussed further below.

There are some important considerations about the *.kml* format to avoid issues, see Annex B for the structure of the files used.

If, for any reason, a file containing the position of the power lines to be surveyed must be constructed, it shall be made from Google Earth and saved as *.kmz*. Another option is to modify the XML file directly from its code.

4.1.2. Main application functionalities

4.1.2.1. New Flight Plan

Once the Archer App has been run, the user must open the *.kml* or *.kmz* file that has the desired path of transmission towers to inspect. The application must have connection to the internet, otherwise a warning message will appear as the map will not be able to download the satellite images from the server, see Fig. 4.3.

When the Archer App is executed for the first time, the programme will need a certain amount of time to locate the icon's directory on the user's computer. Once it has been found, each time a new file is opened, it is immediately uploaded as shown in Fig. 4.4.

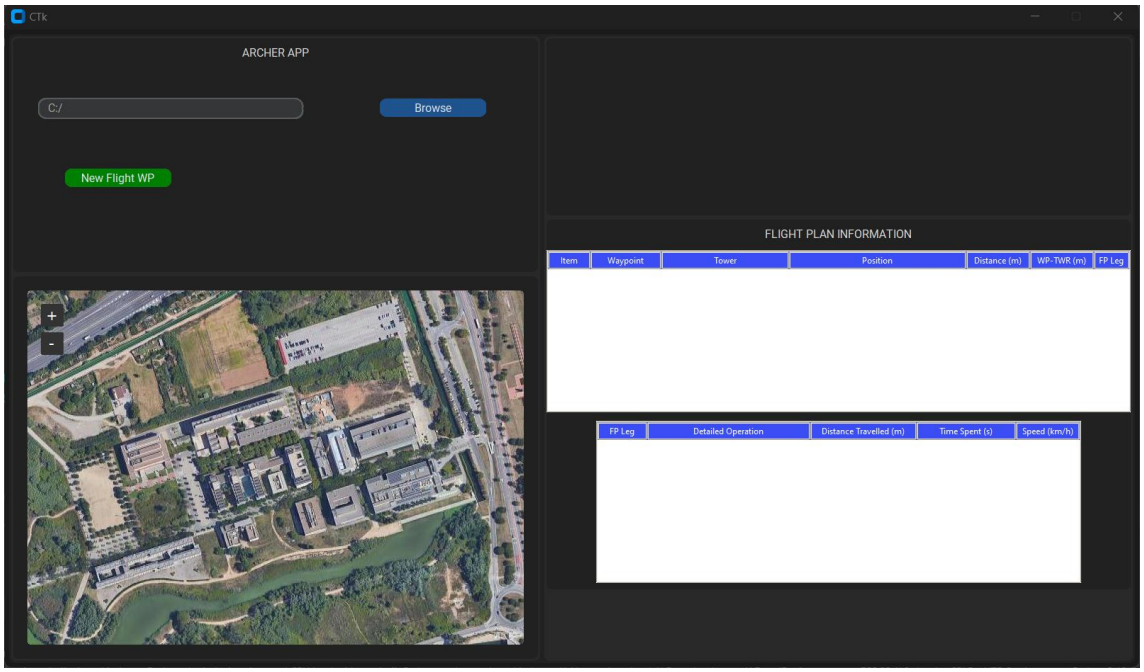

Fig. 4.3 Archer App initialized

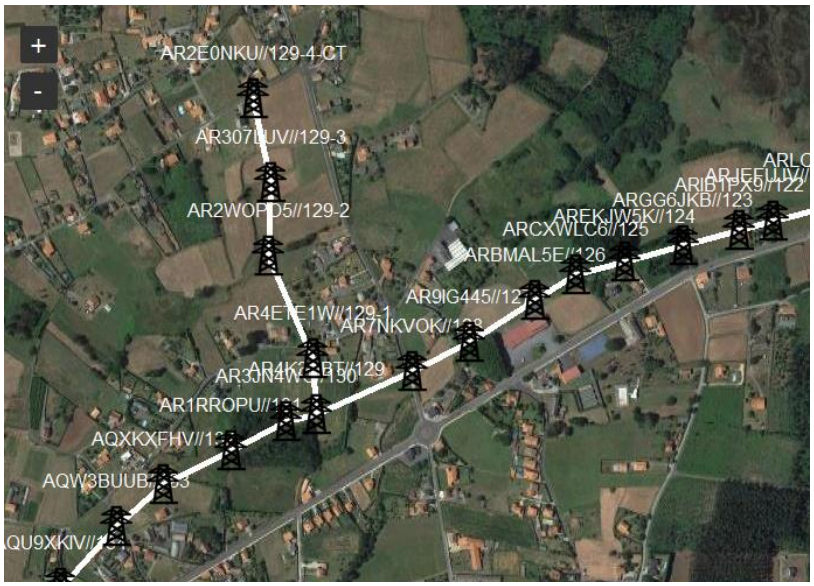

Fig. 4.4 Detail of loaded data from a *kml* file.

Now the user can start designing the flight plan. By clicking on "*New Flight WP*", a menu will appear allowing the user to set the UAV autonomy in minutes, the expected average speed in *km/h* and the flight height in meters. An additional slider will also appear to set the distance from the added waypoint to the nearest towers when the *"Suggested WP*" option is enabled, see Fig. 4.5.

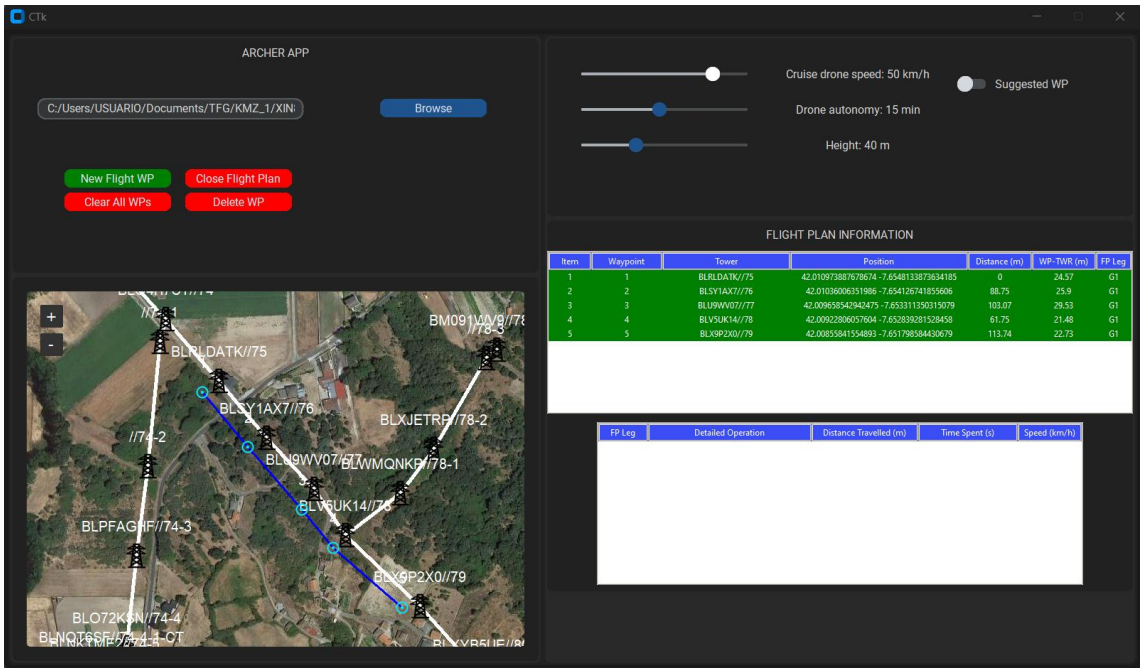

Fig. 4.5 Flight plan leg programming

The following sections describe the *"Suggested WP*" feature with more detail.

Fig. 4.6 shows an example of the different parts of a flight plan, called legs. Each leg is limited by the initial configuration set in the menu displayed when clicking the "*New Flight WP*" button.

Fig. 4.6 Building a new flight plan leg (left) and different legs closed (right)

When a flight plan is being generated, the current path is plot in colour blue, indicating that it has not yet been closed. Each waypoint added is indicated above the icon by an integer, being "1" the first number, following an ascending sequence.

As soon as the user decides to close the leg or the program determines that the UAV's limitations are about to be reached, then the leg is sequentially assigned to a different colour from the previous one and a unique identification code is generated, see Fig. 4.7 and Fig. 4.8.

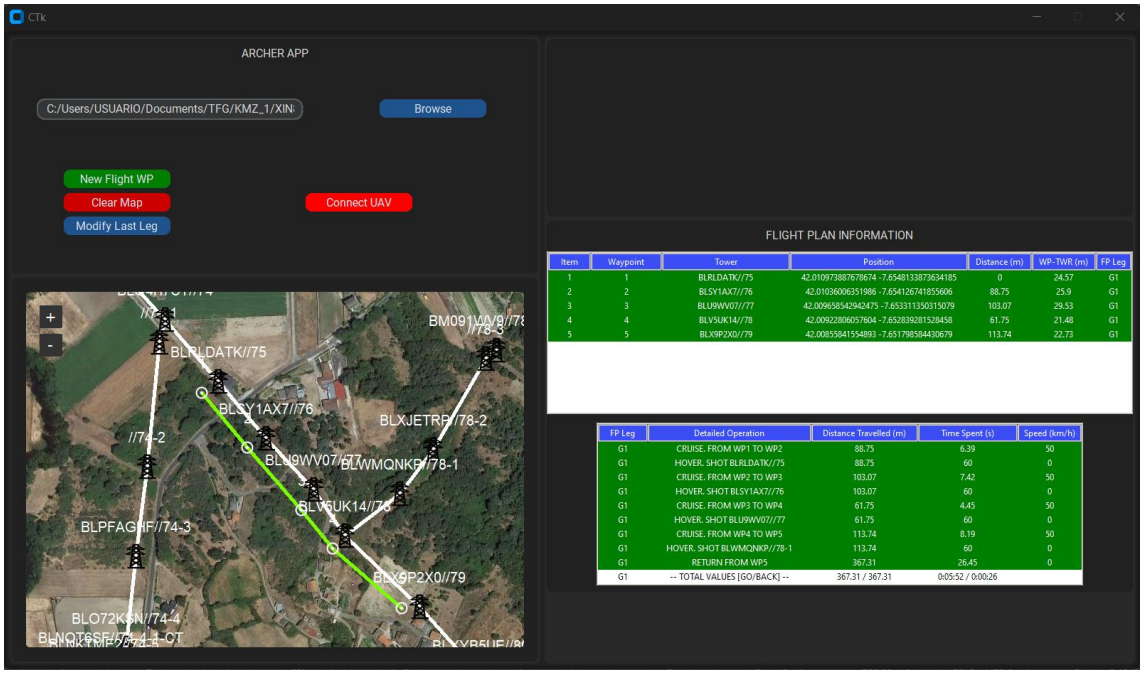

Fig. 4.7 Closed flight plan leg

When the first leg is closed, a "*Connect UAV*" button appears to establish connection with the UAV. Once the connection is done, another option "*Send data*" appears to send the programmed route to the autopilot service.

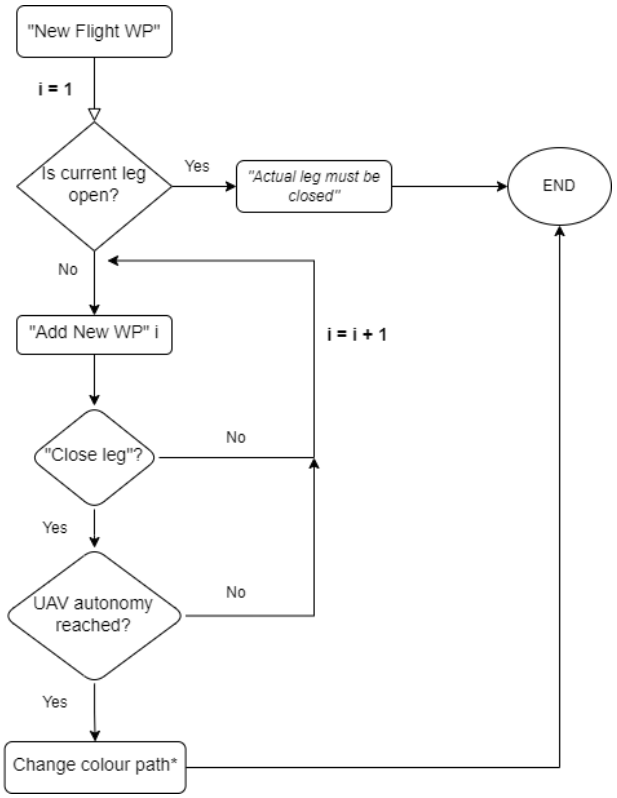

Fig. 4.8 "New Flight WP" flowchart

As the user adds new waypoints, there is an updated table showing the information for each one (Fig. 4.9). The columns of the table are:

- Item. Integer number that determines the number of waypoints in the flight plan; *item = 1, item = 2 ... item = n*. Being *n* the number of total waypoints in the flight plan.

- Waypoint. Integer number that identifies the waypoint within the leg; *waypoint = 1, waypoint = 2 ... waypoint = k*. Being *k* the number of total waypoints in a leg.

- Tower. Identifier of the nearest tower to the added waypoint.

- Position. Geographical coordinates of the nearest tower to the added waypoint.

- Distance (m). Distance in metres from the current waypoint to the previous one. The distance from the first waypoint is always "*0 m"*.

- WP-TWR (m). Distance in metres from the current waypoint to the nearest tower.

Item	Waypoint	Tower	Position	Distance (m)	WP-TWR (m)	FP Leg
1	1	AQBEN6CL//146	43.58386047766344 -8.183447643397471	0	5.58	G1
2	2	AQBODW2O//145	43.58420825598389 -8.18323843109431	42.18	9.01	G1
3	1	AQCPJPIV//144	43.58457141786359 -8.1827092847671	0	5.98	R2
4	2	AQE6WX16//143	43.58513002772852 -8.181879946430598	91.22	5.16	R2
5	1	AQFPBWOL//142	43.58570611135724 -8.181004718075457	0	5.36	O3
6	2	AQHDG5N1//141	43.586318454198796 -8.180089681962784	100.34	5.02	O3
7	1	AQJDJFFK//140	43.58691489757539 -8.178933649877365	0	5.95	Y4
8	2	AQL962VC//139	43.58744705504077 -8.177862223425109	104.63	6.67	Y4
9	1	AQNPP3U5//138	43.58848563524892 -8.176448826161874	0	5.82	G5
10	2	AQOK5VUM//137	43.58891716709125 -8.175963112014927	61.91	6.8	G5

Fig. 4.9 General information for each leg

When a leg is closed, a second table is automatically filled with additional operational information (Fig. 4.10). Such as:

- FP Leg. Leg code
- Detailed Operation. Briefly describes the operation to be performed by the UAV.
- Distance Travelled (m). Distance travelled from the current waypoint to the previous one.
- Time Spent (s). Estimated time to perform each operation.
- Speed (km/h). Average speed of the UAV, set in the initial configuration by the user.

At the end of this second table there appears a white row showing the total values of the distance travelled and the time taken, both for the outward and the return trip. The purpose of this table is to give the user estimated information about the UAV's execution sequence and timing.

FP Leg	Detailed Operation	Distance Travelled (m)	Time Spent (s)	Speed (km/h)
G1	HOVER. SHOT AQBEN6CL//146	42.18	60	0
G1	RETURN FROM WP2	42.18	4.34	0
G1	-- TOTAL VALUES [GO/BACK] --	42.18 / 42.18	0:02:08 / 0:00:04	
R2	TAKE-OFF. MOVE TO WP1	0	0.0	35
R2	HOVER. SHOT AQCPJPIV//144	0	60	0
R2	CRUISE. FROM WP1 TO WP2	91.22	9.38	35
R2	HOVER. SHOT AQCPJPIV//144	91.22	60	0
R2	RETURN FROM WP2	91.22	9.38	0
R2	-- TOTAL VALUES [GO/BACK] --	91.22 / 91.22	0:02:18 / 0:00:09	
O3	TAKE-OFF. MOVE TO WP1	0	0.0	35

Fig. 4.10 Operational information for each closed leg

4.1.2.2. Nearest tower identifier detection

The ability to find the nearest tower to the added waypoint between two coordinates is one of the most important features of this application.

A Geographic Coordinate System (GCS) uses a 3D spherical surface to define positions on the Earth in terms of latitude and longitude.

Latitude is a coordinate that specifies the north-south position of a point on the surface of the Earth or any other body and is given as an angle that ranges from -90º at the South Pole to 90º at the North Pole. Longitude is a coordinate specifying the east-west position of a point on the surface of the Earth or another body, given as an angle ranging from 0º at Greenwich (the prime meridian) to 180º east and west, see Fig. 4.11.

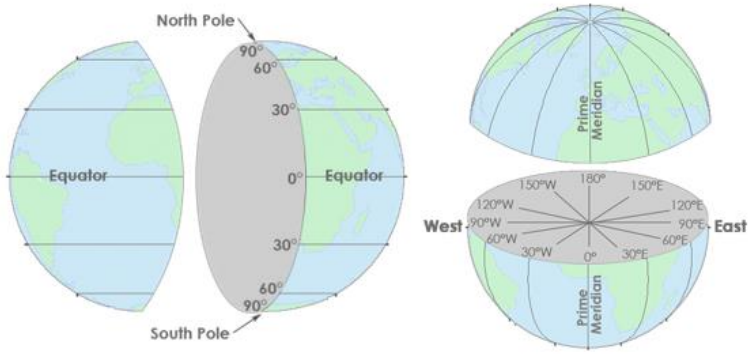

Fig. 4.11 Geographical Coordinates System

Since these are geographical coordinates, distances cannot be computed in Cartesian coordinates. To calculate the distance between two coordinates, the Haversine formula is used. Haversine distance is defined as the angular distance between two spherical surface points [9].

Initially, the distance calculation was done directly by the application of the haversine formula, which is developed in Annex D.

Finding the nearest tower worked correctly until, while developing the "*Suggested WP*" functionality, it was noticed that there was a small error in the decimal places used in the calculations, which became obvious when working with smaller and smaller distances.

In the end, the Python *Geodesic* library was found and allowed for the more accurate calculation of distances between points, in addition to the calculation of the position of a point given the distance and the azimuth with respect to another point.

4.1.2.3. Remove, Delete, Clear and Cancel

Apart from building a flight plan, the programme also allows:

- *Clear All WPs*: This function deletes all generated waypoints and paths for the current leg. Previously closed legs are kept on the map.

- *Delete WP*: The last waypoint and path added to the current leg is deleted. This option is hidden and only appears when the number of added waypoints is greater than one.

- *Clear Map*: This option only appears when there is at least one closed leg, it deletes all waypoints and paths from the map.

- *Cancel FP*. It only appears after clicking the "*New Flight WP*" button and disappears again when the first waypoint is added. This function is used to cancel the creation of a flight plan in case the button is pressed by mistake.

4.1.3. Extra functionalities

The purpose of these extra functions is to provide the user with more tools to adjust the generated flight plans to the characteristics of the UAV and the inspection scenario.

4.1.3.1. Suggested WP

Implementing a feature that allows the user to suggest where to place the waypoint so that the UAV flies at an equidistant distance from all the towers was one of the major challenges in developing the user application.

When the user starts drawing a new leg, a scroll button will appear in the parameter configuration menu to enable the suggestion mode to be activated, see Fig. 4.12.

Fig. 4.12 Extended flight parameters configuration menu detail

Once activated, a new option appears allowing the user to define the distance to be maintained from the tower. The range of values is determined by the camera's capabilities and potential electromagnetic interferences from the power line. As soon as the first waypoint is added, two red waypoints will appear on the next tower. These waypoints appear perpendicular and equidistant to the tower as shown in Fig. 4.13.

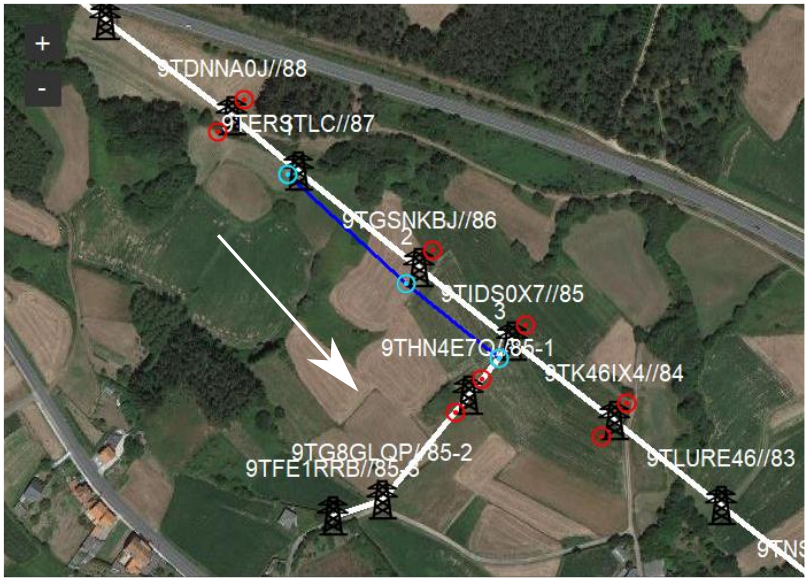

Fig. 4.13 Suggested waypoints are in red, white arrow indicates the user direction of travel.

Implementing the system to offer suggestions even when a path splits off was one of the most challenging difficulties. Additionally, based on the starting position, to offer logical options. Each of the following points will illustrate the issues and solutions offered:

4.1.3.1.1. Logical data pattern

To ensure that each suggestion appeared logically based on the starting point of the waypoint within the map, the data from the *.kmz* file had to be analysed first. It was observed that the towers in the file were not defined in a logical order, so it was decided to extract the description namespace too.

The description specified the name of the initial and final tower of each segment. This information was used to scan the closest tower to the waypoint.

Once found, the next step was to determine if that tower was referenced to other segments, which would mean that it would not be a path's initial or final point. If a tower turned out to be intermediate, it was possible to identify itself and the tower to which it was linked. That is to say, to have the network logic that was being sought.

4.1.3.1.2. User's travel direction

Another important aspect was to find the direction in which the user was building the leg since the first waypoint was added, the aim of this point was to anticipate the suggestion proposal to the upcoming tower.

When the program detected the first nearest tower, while entering the first waypoint, the function listed the closest tower and its subsequent forming the segment, those were then stored into a list to kept track of them.

When the second waypoint was added, it again founds the nearest tower and its subsequent tower which formed that leg. The program then compared the current tower with the one in the list, if there was a match, it means that the user is moving in the direction of that matching tower. Otherwise, if the tower did not match, then the user was moving in the opposite direction and therefore the program would determine the subsequent tower of the first segment detected as the next one.

Later, when adding the first waypoint, it was established that the programme would be able to give all possible suggestion solutions. It is worth noting that when the user chooses a specific direction or branch, the program only continues to suggest in that direction and not in others.

4.1.3.1.3. Orthogonal angle

Once the identification of the next suggestion candidates was achieved, the next challenge was to generate two perpendicular and equidistant waypoints suggestions for the next tower.

The reason why two possibilities are shown is to avoid the UAV crossing the powerline, i.e., if the inspection is being carried out on one side of the tower, do not go to the other side and risk the vehicle crashing into the power line.

Perpendicular positions have been searched to improve the detection of the tower with the algorithm and decrease the area-scanning time.

Depending on whether the azimuth angle α falls into one of the 45º different divisions of the circle, a value shall be added or subtracted to adjust the two possible solutions so that they are perpendicular to the tower, see Fig. 4.14 and Table 4.1.

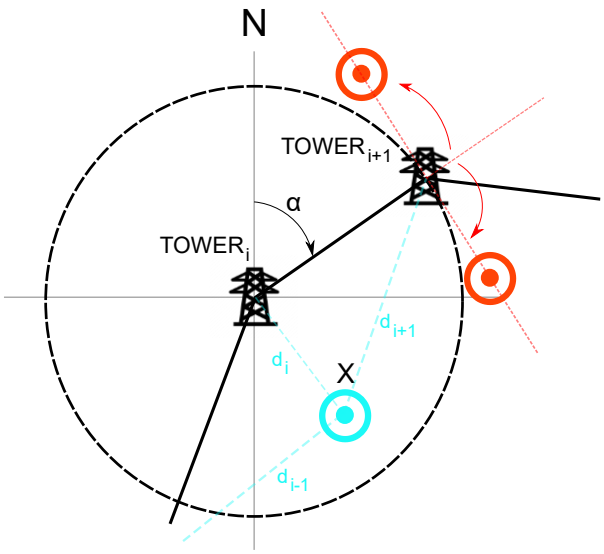

Fig. 4.14 Graphically suggestion process

Table 4.1 Azimuth angle corrections

Azimuth angle:	LH suggestion	RH suggestion
$0° \geq \alpha > 45°$	$\alpha + 270°$	$\alpha + 90°$
$45° \geq \alpha > 90°$	$\alpha + 270°$	$\alpha + 90°$
$90° \geq \alpha > 135°$	$\alpha + 90°$	$\alpha + 270°$
$135° \geq \alpha > 180°$	$\alpha + 90°$	$\alpha - 90°$
$180° \geq \alpha > 225°$	$\alpha + 90°$	$\alpha - 90°$
$225° \geq \alpha > 270°$	$\alpha + 90°$	$\alpha - 90°$
$270° \geq \alpha > 315°$	$\alpha - 90°$	$\alpha + 90°$
$315° \geq \alpha > 360°$	$\alpha - 90°$	$\alpha - 270°$

Once the azimuths were adjusted for each solution, the next step was to obtain the value of each coordinate by passing the angle and the distance established by the user to a function defined in the *Geodesic* library.

4.1.3.1.4. Haversine distance

As discussed above, initially, distance calculations were performed directly from the expression developed in Annex D. When performing calculations for relatively short distances, the program tended to make a noticeable error in the decimals and therefore in the position of the suggested points when a certain distance was fixed.

The best solution was using the Python *Geodesic* library, which also, by means of a function, allowed to obtain coordinates by giving a distance and an azimuth angle.

4.1.3.1.5. Justification of selectable distance settings range

The range of selectable distances set in the initial menu were based on a simple analysis of the effect of electromagnetic interference (EMI) from power lines on the UAV and the field of view calculations of the camera.

From Annex A can be concluded that EMIs should not cause any impact and therefore would not represent a limitation. In any case, it has been determined that performing flights in areas where high-power telecommunication antennas and/or frequency jammers are present could cause interference with the GPS signal and result in potential issues for the flight.

Therefore, the range of selectable distances is set out in Annex F, where the minimum distance shall be 1 m and the maximum distance shall be 45 m.

4.1.3.2. Modification

This section will discuss in detail another feature that has been a challenge during the Archer App development, especially in the numbering sequence of the new waypoints that were added.

One of the main problems detected while developing the application, was that the satellite map images were not usually updated. This implies that if the user was designing a flight plan on old images, it would be possible that, the UAV will follow a collision course with an unexpected obstacle.

When the user closes the first leg, the *"Modify Last Leg"* button appears. If the button is pressed, the previously closed path automatically changes its colour indicating that changes to modification mode.

Now when the user presses the right mouse button to add a waypoint, a new option *"Add New WP"* will appear. The user can create as many points as necessary and, if required, delete them.

When the user desires to finish the modification then should click on *"Save Modification"* and the new route will automatically be changed to the original colours of the leg, see Fig. 4.15.

The main problem came with the new sequence of the newly added points should have. From the beginning it was established that the format would be, for every new waypoint: "$x.c$"

Where x is the index of the waypoint to which it belongs, and c is the subindex numbering for each new added point within the segment as shows expression (4.1).

$$c = \sum_{k=1}^{n} k \tag{4.1}$$

Being *n,* number of total new points added within each index.

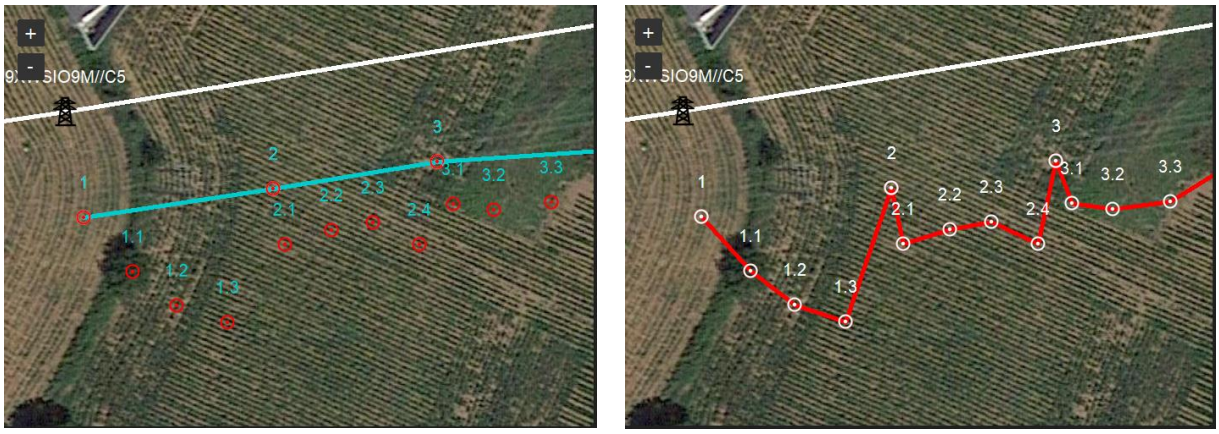

Fig. 4.15 Modification mode (left) and saved modification (right) example**.**

To do therefore, first the program had to be able to detect the nearest waypoint from the new added point. By looping through all the leg waypoints, it was easy to determine shortest distance and therefore its index.

The big problem encountered was when detecting the nearest leg waypoint, if the point was added from the middle-up of the segment, the program tended to interpret the next leg waypoint as the nearest, thereby adopting its index, see Fig. 4.16.

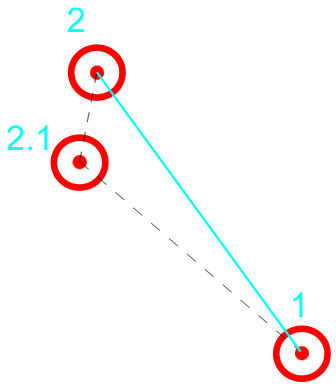

Fig. 4.16 Problem encountered while renumbering waypoints.

To overcome this problem, it was decided to break each of the closed leg segments into smaller equidistant pieces to make the system more sensitive when detecting waypoints over the shortest distance, see Fig. 4.17.

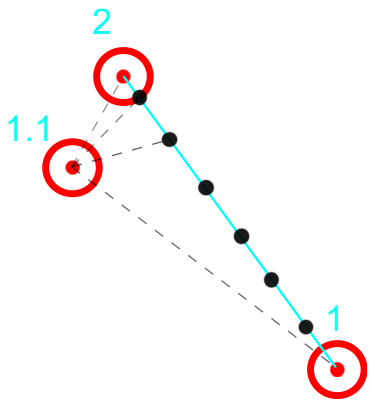

Fig. 4.17 *Suggested WP* fragmentation

To achieve fragmentation, first the shortest distance between segments of each leg was found. This distance was then divided by 10 (depending on the sensitivity required it could be a larger value) and results in a certain parameter.

This parameter was then applied for each segment of the leg, using *Geodesic* library functions, coordinates of the equidistant points were separated by the mentioned parameter.

Once the fragmentation was achieved, it was necessary to indicate that each point belonged to a specific waypoint, by this way the program kept the index and subindex numbering logic.

4.1.4. Simulation

When the Archer application began to take shape, various simulations were gradually conducted with the *ArduPilot Mission Planner SITL* software.

Initially, a certain amount of time had to be devoted to became familiar with the simulation program. For this purpose, manual, automatic, and guided flights were performed on the simulation scope.

The next objective was to transfer all the instructions that were given directly from the simulation tool to Python, using *DroneKit* library. First, a custom script was created to perform tests with different commands and configurations.

When the desired flight operation was successfully executed, the code was adapted to *AutopilotService.py* [8] file as a contribution to DEE.

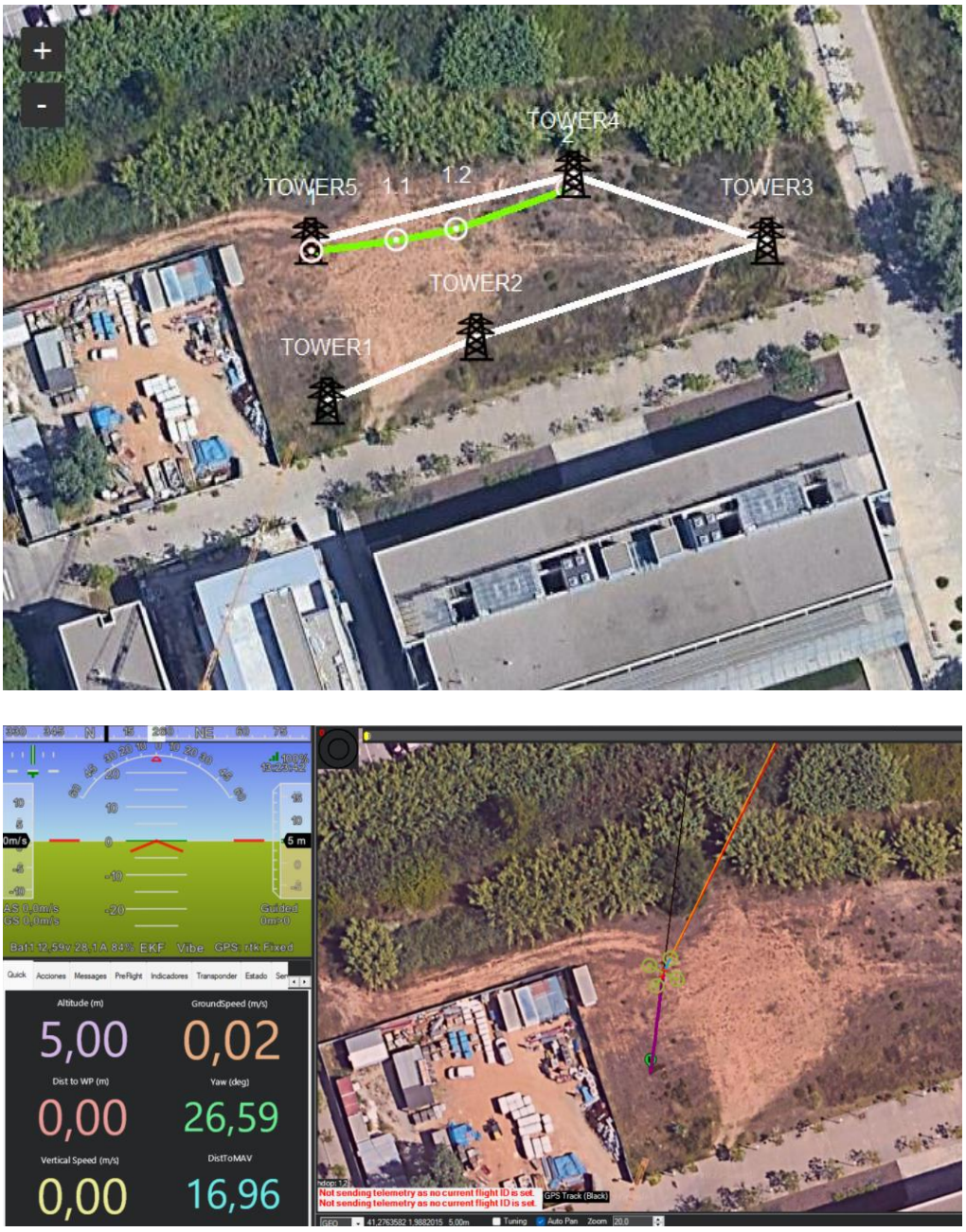

Fig. 4.18 Flight plan leg (upper) and simulated area-scanning process (lower)

The expected flight operation must be as described:

The UAV is initially at rest, receiving GPS signal and disarmed. Once the user has completed the inspection planning, sends the data to the UAV.

Once the flight plan is loaded into the RPi, the motor-arming sequence is initiated automatically. Subsequently, the UAV takes-off and ascends to a height of 5 m (see Fig. 4.18).

When height is reached, the quadcopter proceeds to the first provided point.

If it is a 'T' type point, it means there is a high-voltage tower to inspect, whereas if it is a 'C' type point, it will indicate a collision avoidance point, and the vehicle will fly to the next point upon arrival.

In case of a ‘T’ point, the UAV, after a certain stabilization time, must activate the camera and perform a clockwise yaw rotation to detect the tower.

As soon as the tower is detected, and after a certain time has elapsed, it captures a picture. It then turns off the camera and flies to the next point.

Finally, when the UAV reaches the last point of the flight plan, it returns by the same path it took. Upon arriving at the first point, it then switches to RTL mode (Return to Launch) and performs an autonomous landing at the launch point, see flowchart from Fig. 4.19.

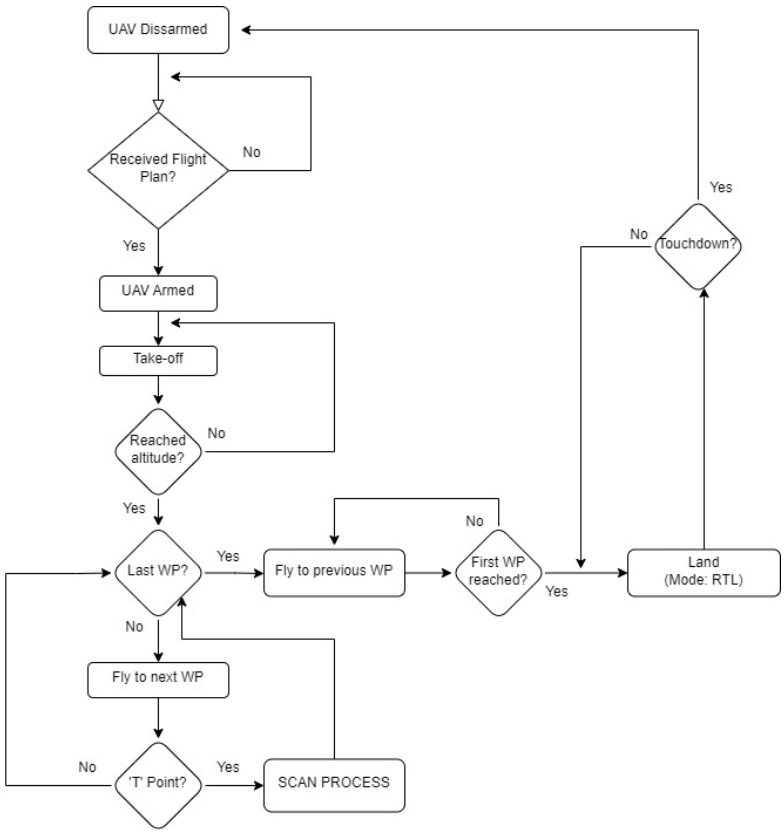

Fig. 4.19 Expected flight operation flowchart.

It should be noted that, at this point, the simulations have been focused more on the flight dynamics, leaving the camera and object detection part for the second development phase. To simulate the UAV detecting the tower, it was programmed to stop yawing when it reached a certain angle relative to the vehicle’s heading.

The biggest difficulty encountered during the simulation process was with "*MAV_CMD_CONDITION_YAW*" command, as there was poor information available on it, and with getting familiar with the MQTT brokers.

4.1.4.1. MQTT brokers

The communication between Archer App, the autopilot, and the camera service was carried out through an internal and an external broker that were running in simulation mode, i.e., on the same computer, see Fig. 4.20.

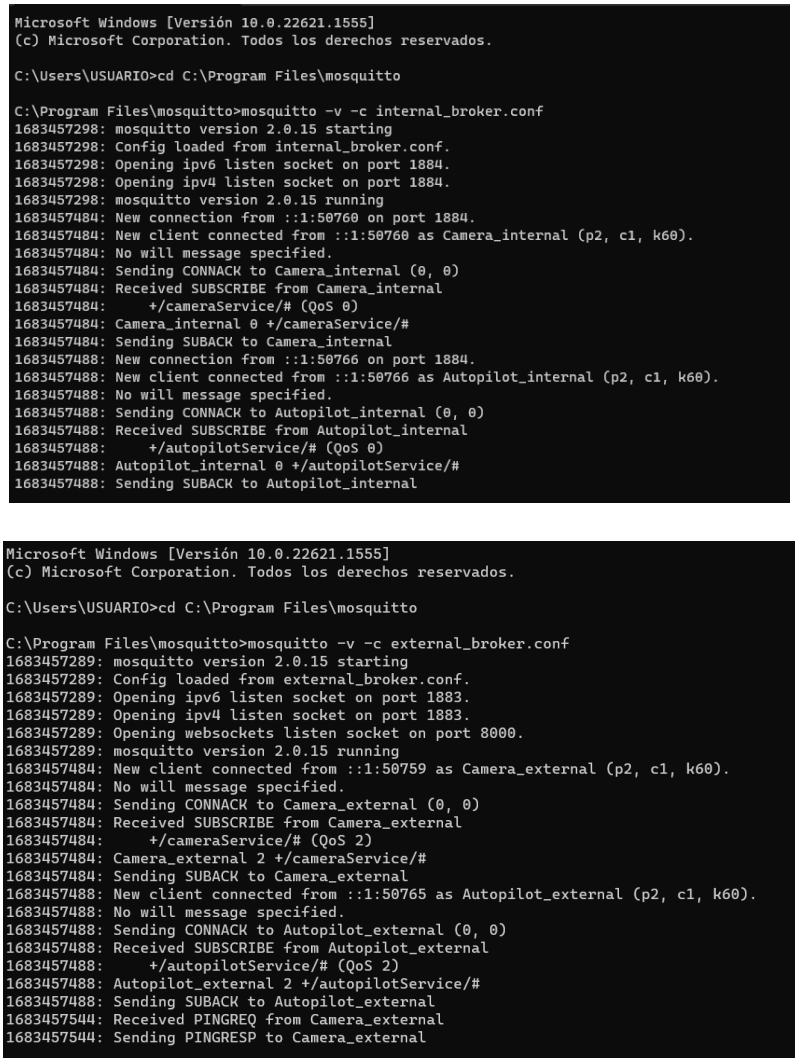

Fig. 4.20 Internal broker (upper) and external broker (lower) running

Once the brokers were configured, the operating mode had to be specified in the parameters of each Python script, which were: *local simulation localhost*.

Finally, by running *AutopilotService.py* [8] and *CameraSrvice.py* [8], they were automatically connected to the internal and external broker and remained on standby until the user application sent the first command.

4.2. Object detection model

This section will deal with the second phase of the development of this project.

It is essential that, once the user has sent all the waypoints necessary to carry out the inspection, the UAV itself flies autonomously to each point and, if there is a tower, performs a yaw rotation to detect where it is to frame it in the camera and take the picture.

A section will be devoted to the used dataset, its training, and tests carried out to evaluate the performance of the object detection capabilities.

4.2.1. Dataset

A dataset is a collection of data used to train and evaluate machine learning models. The dataset typically consists of a set of examples, where each example includes a set of input features and an associated output label or target value. In this case, the dataset will be a collection of images associated to the desired object to be detected.

Fortunately, a sufficiently large dataset of images (1,242) of transmission towers and cables taken from the air was found [10], see Fig. 4.21. This dataset has already been segmented for each of the components to be inspected.

Fig. 4.21 Some images from TTPLA dataset [10]

This dataset contains in a folder the images with their corresponding *.json* files. These files are the ones that provide the information of the segmentation that has been done in each image. Although this dataset is relatively complete and comprehensive, *Yolact* was used as segmentation model and therefore its format is different from the one to be used in the YOLOv5 model.

For this purpose, it was decided to use *Roboflow* website to change the format of the dataset and make some modifications to the images so that the algorithm could be better trained, and therefore objects could be precisely detected.

4.2.1.1. Roboflow

Roboflow website allows users to upload and manage large datasets, pre-process data with a range of tools and data augmentation, also training custom models using a variety of popular deep learning frameworks.

First, the dataset was loaded into a new project. As it had already been segmented before, the classes were already indicated in each image by default. See Table 4.2 for the employed classes in the image segmentation.

Table 4.2 Employed classes.

Class	Description
Cable	Power line cable
tower_lattice	Steel angle sections of a tower
tower_tucohy	Tubular steel and concrete hybrid tower poles
tower_wooden	Wooden tower poles
void	Used for any instance difficult to recognize

It should be noted that, in addition to the images from the used dataset, other pictures were added, taken from the mobile phone itself, to provide different angles and perspectives with the aim of improving the detection of the towers.

Upon completing the segmentation of these images (Fig. 4.22), it was established that 70% would be used to train the algorithm, 20% for validation, and the remaining 10% for testing.

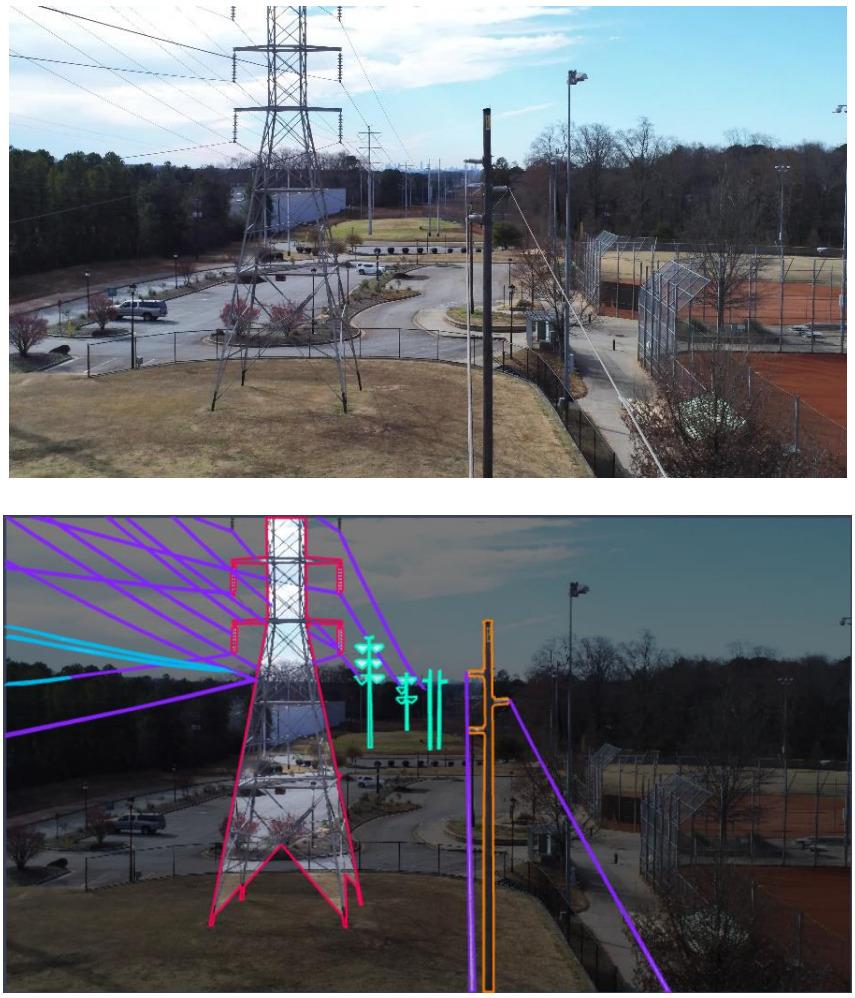

Fig. 4.22 Original image (upper), segmented image (bottom)

Once the images were loaded and segmented into *Roboflow*, data augmentation was performed. This step consisted of applying modifications to the images so that the training of the algorithm was better adapted to real-life conditions.

4.2.1.2. Pre-processing

In the pre-processing phase, modifications such as rotations, contrast changes, and noise were added to better train the neural network. This enabled the model to detect objects even when conditions were not optimal.

The main modifications applied were: 50% crop, 20% noise, and a 1.75px Gaussian blur to make the model more resilient to camera artifacts as shown in Fig. 4.23.

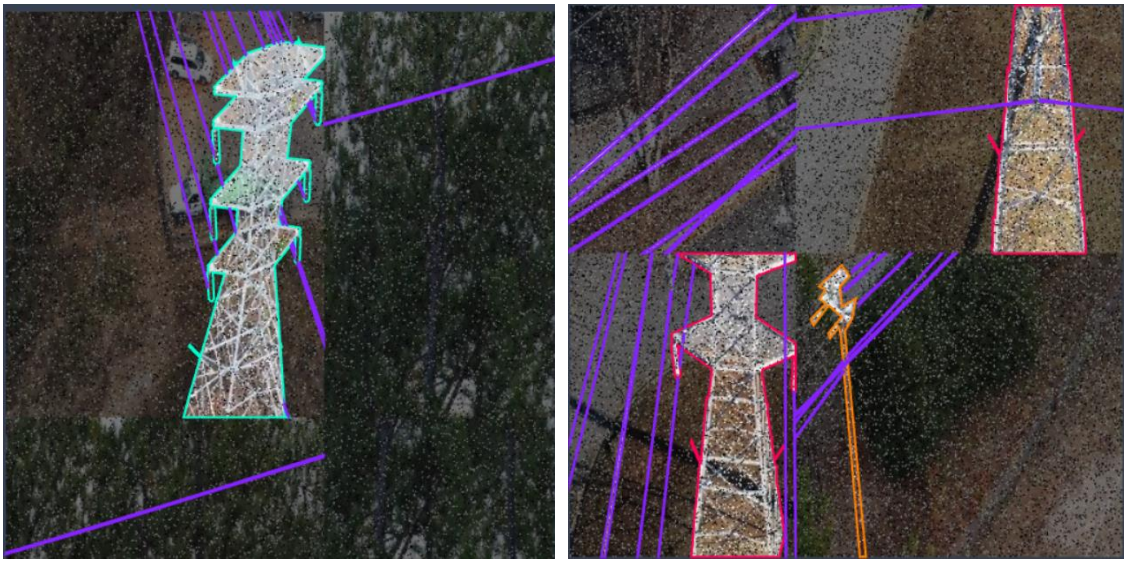

Fig. 4.23 Pre-processed settings applied to training images.

Finally, the dataset was regenerated using the x3 option offered by *Roboflow*. This option allowed a threefold increase in the dataset by applying the pre-processing configuration. As soon as this option was applied, 4,625 images were obtained in the final dataset.

4.2.2. Training YOLOv5x model

Once the dataset had been created with the modifications described in the previous section, the file was then downloaded using the YOLOv5 Torch format and saved at a Google Drive folder.

Refer to Annex G for the used code for training the neural network and Annex H for the justification of YOLOv5x as the chosen model.

The most significant changes made during training were to the batch parameter, image size and number of epochs.

A larger batch size can accelerate the training process, but it can also require more memory and increase the likelihood of the model suffering from overfitting. Otherwise, a smaller batch size can require more training time, but can lead to a more generalizable model and be less prone to overfitting. A good starting point is a batch size of 16 to 64.

The appropriate number of epochs depends on several factors, including the size of the dataset, the complexity of the model, the batch size, and the learning rate. A general rule is to adjust the number of epochs to obtain good accuracy on the validation set without overfitting the model on the training set. A good starting point is to begin with a low number of epochs, such as 20, and then adjust based on the training results.

Also, the recommended image size for training a YOLOv5 model depends on the available hardware, GPU size, and available GPU memory. However, the default image size in YOLOv5 is 640x640. If limited hardware is available, an image size of 416x416 may be more suitable to reduce training time and GPU memory usage.

Generally, it is recommended to use an image size that is large enough to capture the necessary object details for detection, but not so large as to decrease performance and increase training time.

As a starting point, it was decided to begin with 10 epochs to decrease training time, with a batch size of 16. In addition, an image resizing to 416x416 was performed.

Previously, before analysing the results from Fig. 4.24 and Table 4.3, the following concepts must be defined:

- The “Box Loss” graph refers to the anchor box loss during model training or validation. The anchor box (bounding box) is a rectangle that is adjusted around the detected object and is used to calculate the detection error of the object. The anchor box loss refers to the difference between the predicted anchor box and the actual anchor box of the object.

- The "Object Loss" graph refers to the object loss during model training or validation. The object loss refers to the difference between the actual object probability and the predicted object probability by the model. This loss is used to adjust the weights of the neural network and improve object detection.

- The “Classification Loss” graph refers to the classification loss during model training or validation. The classification loss refers to the difference between the actual class label and the predicted class label by the model. This loss is used to improve the accuracy of object classification.

- Precision graph is the ratio of true positive detections to the total number of detections made by the model. It measures the accuracy of the model in identifying true positive instances and avoiding false positives.

- Recall graph is the ratio of true positive detections to the total number of true positive instances in the dataset. It measures the ability of the model to correctly identify all instances of the target object. In other words, it measures the ability of the model to avoid false negatives.

- mAP (mean Average Precision) graph is a widely used metric to evaluate the overall performance of an object detection model. It is the average of the Average Precision (AP) calculated for each class present in the dataset. AP is the area under the precision-recall curve and measures the trade-off between precision and recall for different confidence thresholds. Higher mAP values indicate better performance of the model.

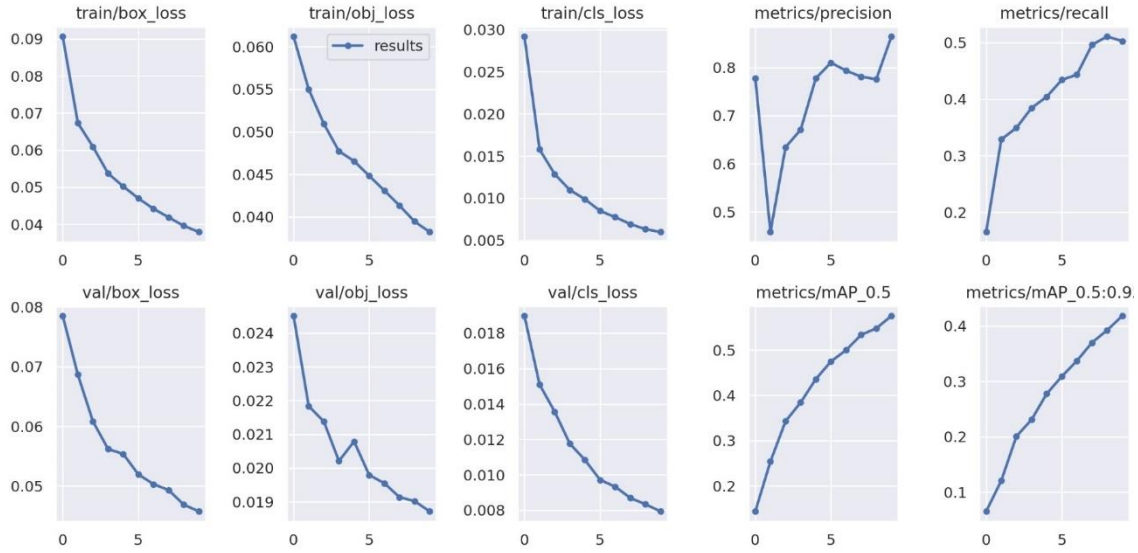

Fig. 4.24 Results from the first training and validation at *Google Colab*

Based on the results acquired, losses during training and validation decrease rapidly, and seems that even by increasing the number of epochs, their value could be further reduced.

The precision, despite suffering some drops, has a positive trend, as shown by the mean average precision (mAP), and it could be higher by increasing the number of iterations, achieving values above 0.866.

As for the recall, it appears to increase rapidly in each iteration until a small drop occurs in the last one.

Table 4.3 YOLOv5x model training with 16 epochs.

mAP50	Precision	Recall
57.5%	86.6%	50.3%

In view of the analysis carried out, a second training was performed by increasing the number of epochs to 25 while keeping the other parameters constant, with the aim of improving the model's prediction. Table 4.4 shows a summarize of the results obtained.

Table 4.4 YOLOv5x model training with 25 epochs.

mAP50	Precision	Recall
64.3%	78.3%	57.1%

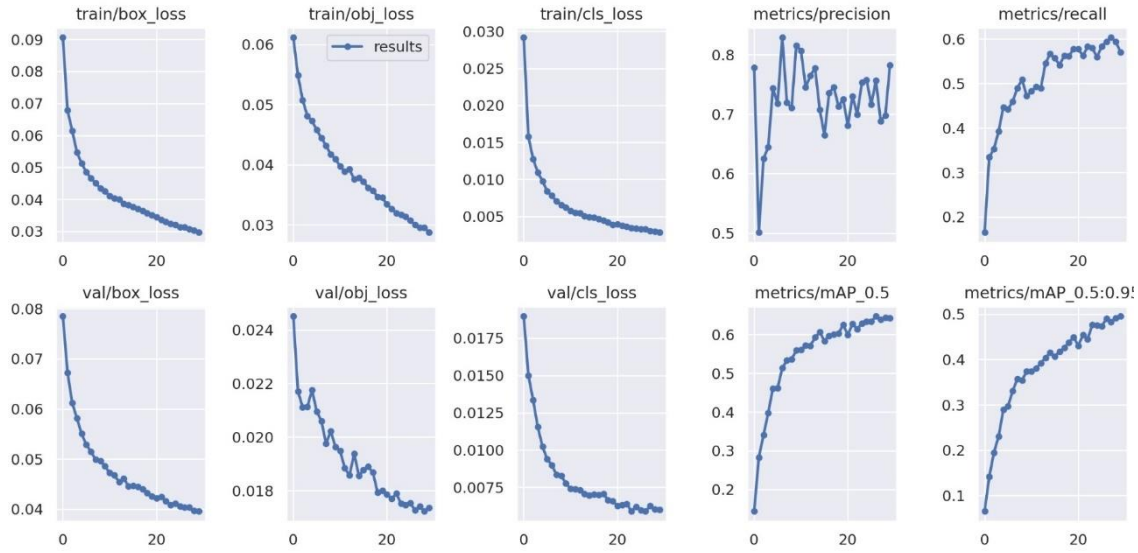

Fig. 4.25 Results from the second training and validation at *Google Colab*

When comparing the results from the second training (Fig.4.25) with the first, it can be observed that the losses decrease rapidly and gradually reduce their rate as the number of epochs increase. The accuracy experiences numerous fluctuations without achieving a significant increase with each iteration, only obtaining slight improvements in recall and the average precision.

Roboflow also allows training the dataset with a custom model (*ROBOFLOW 2.0 OBJECT DETECTION*), which yields the following results from Fig. 4.26. and Table 4.5.

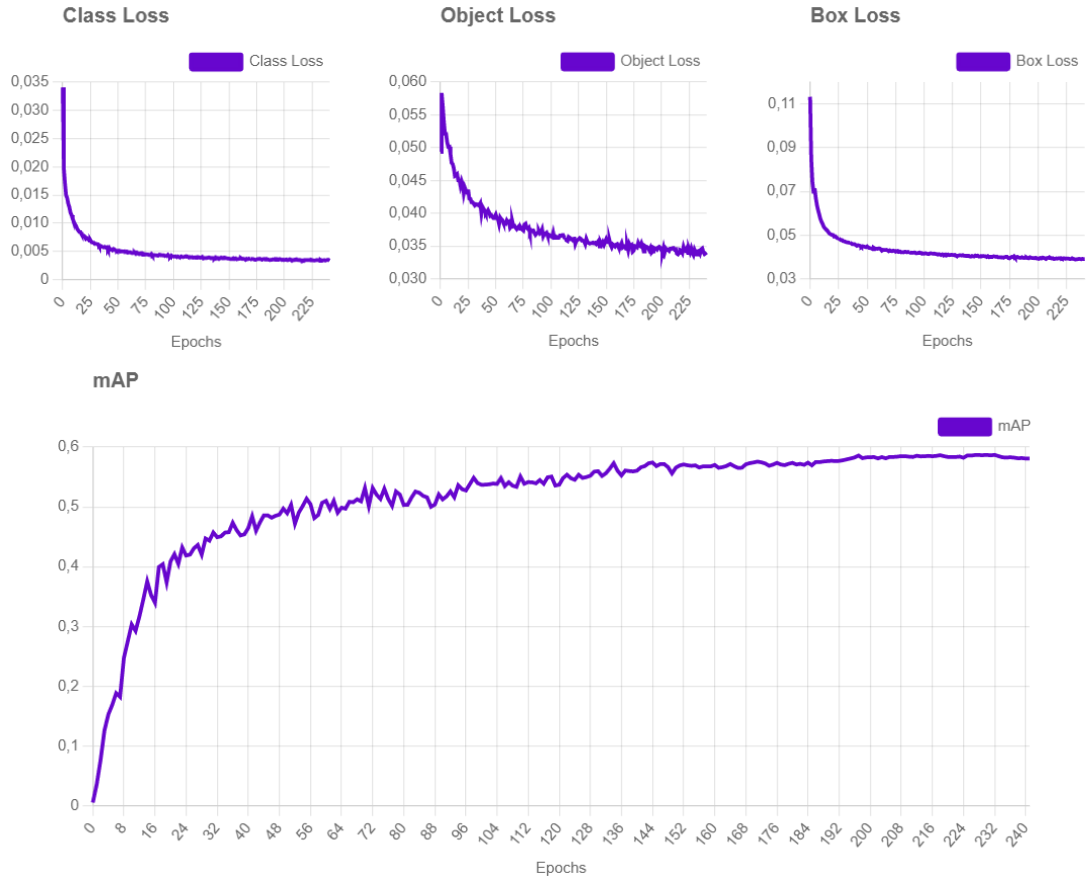

Fig. 4.26 *Roboflow* training results

For the dataset training, 240 epochs had been used with a batch of 16. Losses are rapidly corrected and from around 80 epochs, the classification errors, box positions, and objects tend to stabilize without showing significant improvements as iterations are increased.

On the other hand, the mean average precision increases notably up to 24 epochs, and from there on, it gradually reduces its precision increase until around 80 epochs, where it tends to stabilize with almost no noticeable fluctuations at a value of 0.583.

Table 4.5 *Roboflow* training results using "*Roboflow 2.0 Object Detection*" model.

mAP50	Precision	Recall
58.7%	77.3%	53.4%

Therefore, it was determined that the dataset employed could achieve an average accuracy of somewhat more than 60% while retaining a specific recall value.

It is possible that by using a larger dataset with greater diversity of angles, the detection results could be improved.

Although, the tests carried out with the 10% of the photographs devoted to it were successful.

The next step was to test the model with images that did not belong to the dataset.

4.2.3. Testing the model

A Python file was built to facilitate the model test, enabling the connection of the computer's camera and real-time item classification.

Torch library was used to load the trained model weights, *OpenCv* was used to manage the camera's resources, *Numpy* and *Pandas* to bound and display the detected object in a box.

Since the images in the dataset were captured using a different camera than the one on the UAV, tests were conducted using two other, lower-resolution cameras to ensure that the model continued recognising towers correctly, see Table 4.6.

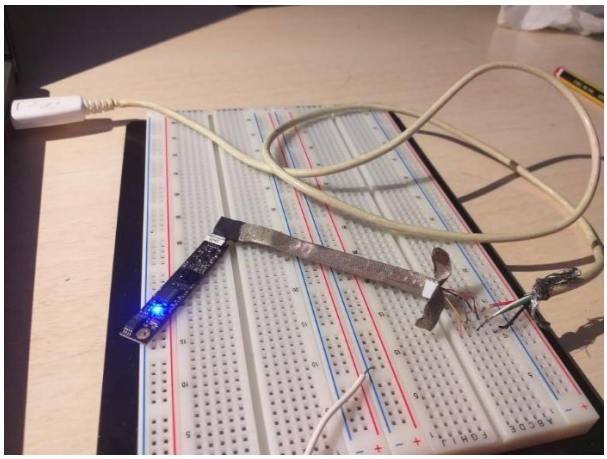

Fig. 4.27 Camera 1 extracted from a scrapped PC and welded to USB port to run the test.

One of the cameras used in the test was from a broken laptop. This involved taking the whole broken computer apart and removing the camera. It was first tested on a breadboard and then the connectors were soldered to a recycled USB port to make it easier to use, see Fig. 4.27 and Table 4.6.

Table 4.6 Main features of used cameras for the test

	Source	Resolution	FPS
Camera 1	Toshiba NB250	640x480	30
Camera 2	Lenovo V15 G2 ITL	1280x720	30

The tests were performed by displaying images of high-voltage towers with metal structures (*tower_lattice* class), which is the type of tower focused on in this project. The detection and classification of the different objects occur satisfactorily for both cameras. A mobile phone was used to display different images from the internet.

As seen in Fig. 4.28, for the same image, Camera 2, having higher resolution, allowed capturing more details in the image and therefore to detect smaller objects or those located at greater distances. It was also observed that both cameras had some issues while trying to detect objects in presence of slight camera movements, where the image appear blurry and distorted.

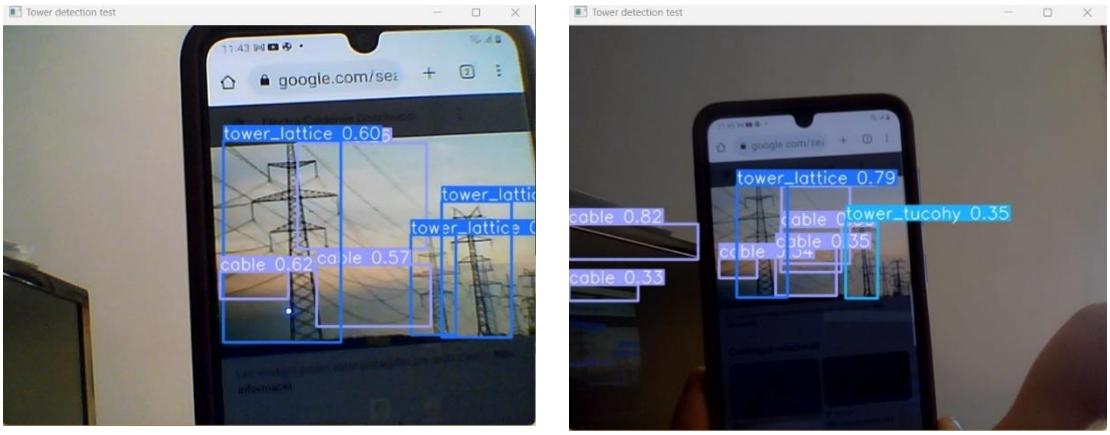

Fig. 4.28 Camera 1 (left) and Camera 2 (right)

Another factor to take into account was that the model had trouble detecting some sorts of structures, presumably because the dataset needed a wider variety of images with various structural topologies.

While the UAV camera had a resolution of 3280x2464, it was expected that the model will have no difficulty detecting towers at greater distances. Yaw rotation speed should also be taken into account to avoid blurry images, as well as a suitable height to guarantee that the tower is at least partially in the centre of the picture.

4.2.4. Simulation

This section describes the process used to achieve the flight simulation using the camera and the trained model, the problems encountered and how they were solved. The two main problems encountered will be presented below in two sections.

4.2.4.1. Processing delay

First it was necessary to modify the DEE's *AutopilotService.py* [8] and *CameraService.py* [8] files. In the first instance, it was considered whether the object detection code should be executed from the RPi or from the user application.

Since the RPi has a low computational cost, it was ultimately decided that the object-detection algorithm should be run from the Archer App, as it was expected that the computer from which it was run had a greater computational capacity and thus more efficiency when processing images, this implied having to send the images from the camera service to Archer App, see Annex H.

The original plan was to turn on the camera, rotate it in a yaw motion, and send video footage to the user application when the UAV reached the location where a tower needed to be examined. The model in the Archer App would then evaluate this video stream until the tower was found.

The problem was that the image-processing time using the model was too high compared to the rate at which the camera service was sending the video frames, the buffer was filled too quickly, and the delay was noticeable, see Table 4.7.

The next commands on the autopilot could not be carried out at the appropriate time until the buffer was cleared, and the UAV lost control until it was.

Table 4.7 Comparison of increase in image processing time with and without YOLOv5x model

Average time without object-detection model (s)	Average time with object-detection model (s)
0.016	1.520
0.038	1.489
0.021	1.511

It was decided to transmit pictures every predetermined number of seconds rather than video streams to address the issue. It was found that delivering images at particular times was sufficient to cover the whole scan rotation in the camera's field of vision and that sending images every few milliseconds was

unnecessary. As a result, the system had more time to process images, despite the overall inspection time also increased.

Taking advantage of the fact that the images were received every few seconds, the scanning operation was also modified so that the UAV would rotate a certain angle and stop before taking the picture. This ensured that the taken images at rest were not blurred or distorted due to the vehicle's motion, which should contribute to a better detection of the object.

4.2.4.2. Threaded loops issue

The communication between the Archer App and the autopilot service was one of the main issues that surfaced during the second development phase of the project.

When a tower was detected in the user application, it was not possible for the UAV to execute the instruction to finish the yaw manoeuvre. Basically, the problem came from the way this part of the code was initially programmed in the autopilot script.

The execution of the program was done through a secondary thread that then entered a loop to control the yaw rotation, and as the program was executing this loop there was no way to listen for new external instructions until the loop was finished, that is, when a certain angle was reached. Let's say the autopilot went into an "asynchronous behaviour" with the Archer App.

The commands were executed too late, which caused problems in the simulated flights.

After discarding several options, it was decided to change the way much of the code in the autopilot was programmed to achieve a more common communication between the user application and the autopilot.

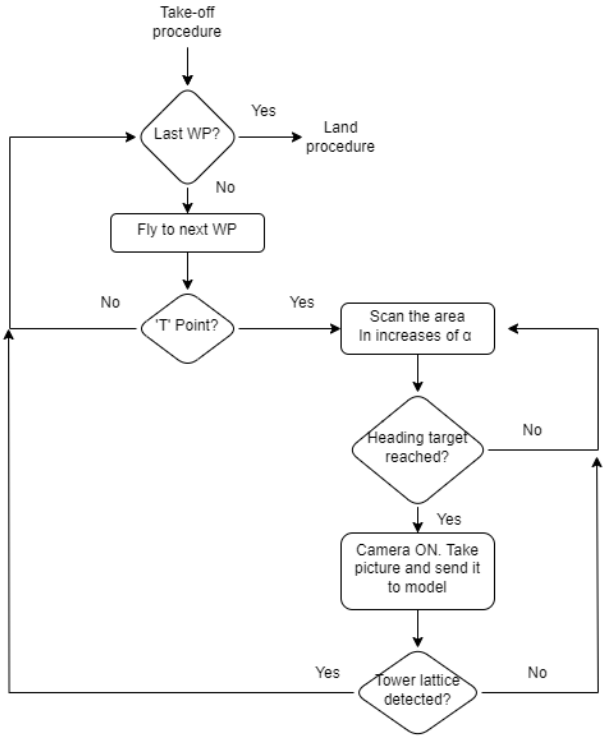

Fig. 4.29 Scan process flowchart

The expected UAV behaviour for transmission tower inspection can be seen in Fig. 4.29.

A schematic of the temporal communication between the user application, the autopilot, and the camera service is shown in Fig. 4.30 below.

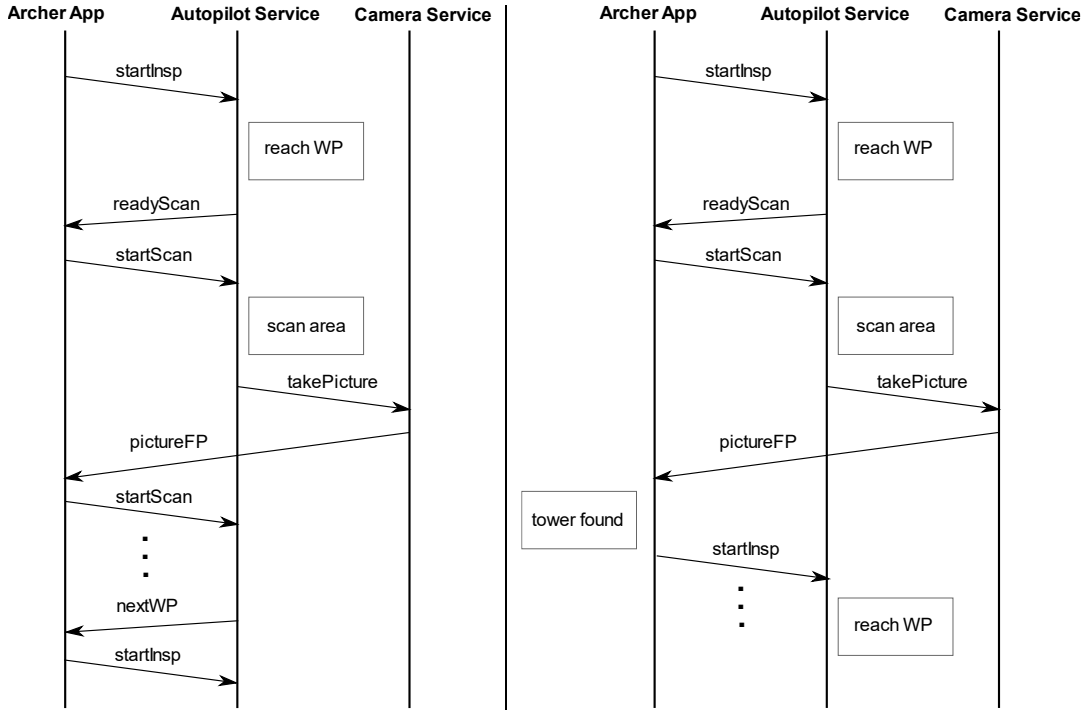

Fig. 4.30 Communicating protocol between the UAV modules and user application.

Once the UAV has taken off and reached its target height, the autopilot service notifies the Archer App with an *'inspTakeoff/OK'* command. At this point, the application sends a *'startInsp'* command to the autopilot indicating that it can proceed to the first designated point.

If the point is of type ‘T’, the autopilot sends a "*readyScan*" command to the Archer App and this one replies with a "*startScan*". Currently, a function is executed enabling the UAV to rotate at specific angle increments.

At each increment, the UAV stops the rotation, and the autopilot sends an internal "*takePicture*" command to the camera service.

The camera service then sends the image taken to the user application via the "*pictureFP*" command. This image is processed by the YOLOv5x model, if no tower is detected, the autopilot is notified again with a "*startScan*" to continue scanning. This process is repeated until the UAV completes a full round, then the scan is aborted and the flight to the next point is notified with a "*nextWP*".

Conversely, if a tower is detected, the Archer App informs the autopilot to fly to and evaluate the next point with the command "*startInsp*".

A *“startInsp”* instruction is issued by the user application in response to the autopilot's *“nextWP”* command, which instructs the UAV to fly to the next point without scanning if the WP it reaches is a 'C' type point.

As soon as the last point of the leg path has been reached, the autopilot should inform the Archer App, and this one will send an instruction to begin the return and land process from the starting point, refer to Annex J for the detailed telemetry results.

Finally, based on the camera's half-field-of-view (HFOV) and a rotation speed of 10º/s, rotational increments of $\alpha = 32º$ were selected for each area scan. The purpose of this was to speed-up the tower-detecting process and achieve camera field of view overlap to decrease the possibility of taking a picture just before and/or after the tower, which would result in it not being detected (Fig. 4.31).

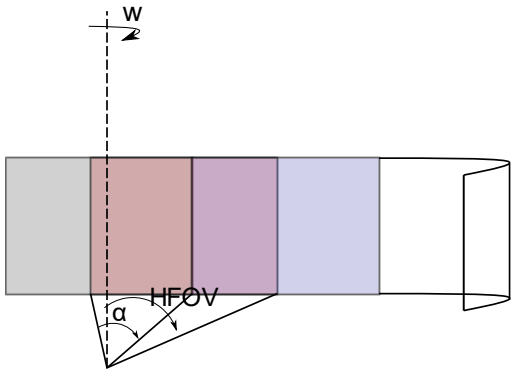

Fig. 4.31 FOV overlap at ***w*** angular speed.

When a tower is spotted, all photos are saved in a directory that is created automatically and has the format name:

"INSPECTION YYYY-MM-DD hh.mm.ss"

Where *hh.mm.ss* is the timestamp and *YYYY-MM-DD* is the date where the folder was created, see Fig.4.32.

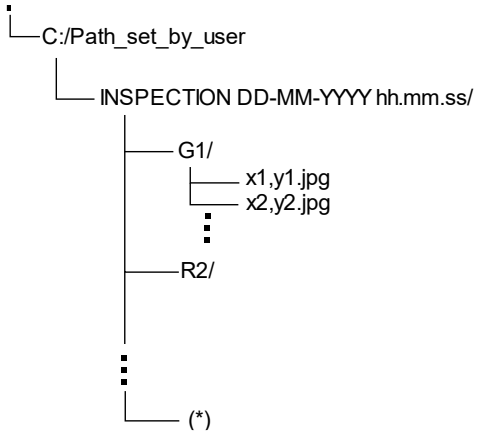

Fig. 4.32 Directory organisation

As additional legs are generated, new sub-folders will be established inside the main directory. The procedure of assigning each leg code follows the process indicated in Fig. 4.33, and the names of these sub-folders will be the same as their respective leg codes.

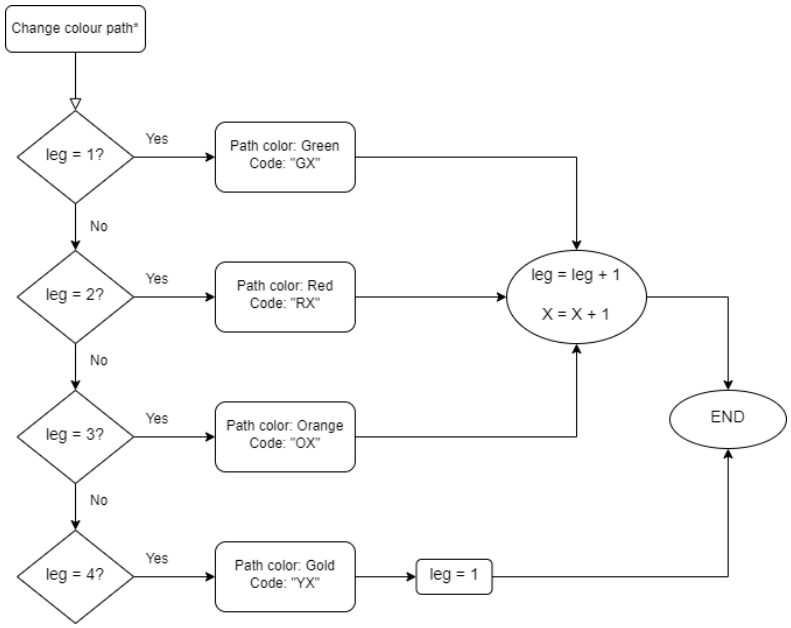

Fig. 4.33 Colour code flowchart (*)

4.3. Verification

The tests performed to validate the system at each stage of development are described in this section.

4.3.1. Flight Test I

Once the Archer app was fully developed, a real flight was performed at UPC *DroneLab,* with the aim of:

- ✓ Verifying the connection between the user application and UAV.
- ✓ Verifying that the flight dynamics were as expected.

The first thing was to perform the final checks in the simulator envelope and a final adjustment was made to the RTL mode height. By default, this mode returns the UAV to the take-off point at 14 m and was set to the same height as the flight (5 m), because the *DroneLab* has a maximum height of 15 m.

Next, the Python *AutopilotService.py* [8] and *CameraService.py* [8] files that run on the RPi were loaded. Once uploaded, a connection test was performed from the user application and was successful.

The next step was to place the UAV inside the *DroneLab* and activate the virtual security fence inside the enclosure to prevent the vehicle from going into the perimeter grid and crashing, see Fig. 4.34.

Fig. 4.34 Propellers assembly (left) and UAV armed (right)

As the time available for the flight test was limited, the most comprehensive flight test possible was carried out. To do this, a simple route with 4 waypoints within the virtual fence was programmed from the user application, the first and last being ‘T’ type waypoints and the two intermediate ones being collision avoidance

'C' points. In addition, the test was conducted with an average cruise speed of 35 km/h at a height of 5 m.

The only inconvenience encountered during the test was the failure to establish a connection between the PC and the UAV via Wi-Fi at the UPC *DroneLab*. This problem was easily solved easily by connecting a Wi-Fi adapter to the USB port to achieve greater signal gain.

The flight lasted approximately three minutes and completed the circuit as expected in the simulations. It was observed that the UAV was very stable against crosswind and always maintained the correct height. In addition, yaw rotations were performed at 10º/s, and the transitions from hover to forward flight were carried out smoothly.

The study continued in the creation of the object detection model and all other features related to the acquired image after the first flight test had been satisfactorily confirmed.

4.3.2. Flight Test II

After the integration of the tower recognition model was accomplished, the second flight test was carried out with the aim of:

- ✓ Verifying the camera mount designed in Annex E.
- ✓ Checking the received images delay.
- ✓ Verifying a complete inspection flight.

For this test, the poster designed in Annex I and its mounting were used to allow the model to recognise a tower-like object and therefore simulate a more realistic scenario. The poster was suspended from the UPC *DroneLab* net at a height of approximately 2.5 m using a hanger made of rods and wire, see Fig. 4.35.

Fig. 4.35 Image of the tower hung on the net.

In order to get the UAV to the right height for the poster, it was first decided to lower the flight height from 5 m to 3 m.

The UAV was flying too high during the first three flights, making it impossible to detect the object, but it was later confirmed that the vehicle was successfully executing the scanning operation and sending photographs.

It was also verified that the new camera mount did not generate vibrations when airborne and the Wi-Fi communication between the vehicle and the Archer App did not cause major delays in the reception and processing of the images that could affect the scanning sequence.

For the fourth flight, the battery was replaced, and the flight altitude was adjusted to 2 m, and the camera's tilt angle was slightly raised. In this case it was noticed that the images were inside the camera’s FOV, but the model was not able to identify the object. The issue was caused by wind gusts that caused the poster to flap, distorting the image that was captured.

It was then decided to keep the vehicle setup for the fifth and final flight and hold the poster from the bottom to prevent it from moving regarding in the wind.

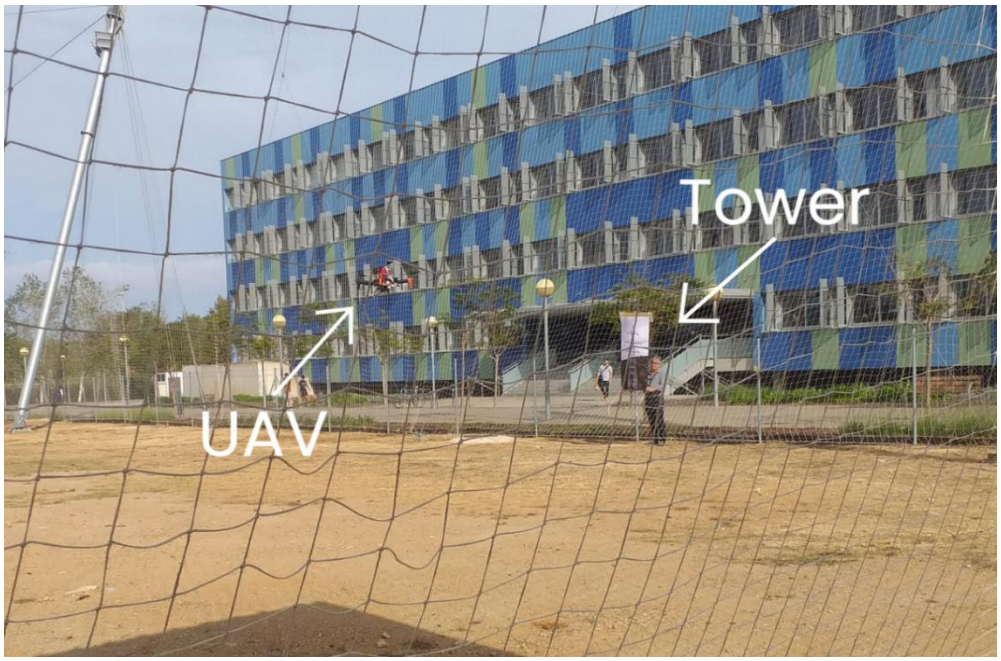

Fig. 4.36 UAV flying through the first ‘T’ point.

Fig. 4.37 Detected image from UAV’s on-board camera.

The last flight began with a 2 m takeoff, after which the UAV flew to the first ‘T’ point and began the area-scanning routine (see Fig. 4.36). At a distance of about 2.5 m from the poster, the UAV did a static hover before starting a clockwise rotation at a speed of 10º/s. It missed the tower during the first capture, but it found it during the second try.

The picture taken, Fig. 4.37, and its coordinates were automatically saved in a folder, the UAV subsequently moved on to the following location. Since this point was a ‘C’ type, it avoided an obstacle and then moved on to the next waypoint, which was another ‘T’ type point. Since there was no poster at this location, the vehicle performed the entire scanning process and, upon finding nothing, turned around and returned along the same route until it landed.

Finally, in view of the results achieved, all the points are verified, and the study is considered completed.

5. RPAS REGULATION

This chapter will highlight the most relevant aspects from the current Spanish regulations contained in the Royal Decree 1036/2017, 2017 applied to Remotely Piloted Aircraft (RPAs).

This regulation has been constantly taken into account during the development phase of the Archer App and serves as a reminder for the user before planning a flight.

"*Every aircraft that flies without a pilot on board, and which can either be fully controlled by the remote pilot, remotely piloted aircraft or are programmed and fully autonomous, autonomous fully autonomous."* [2]., autonomous aircraft in the terminology of ICAO

The amendment of Article 11 from Law 48/1960 on Air Navigation established that these devices are effectively aircraft, and as such, their civil use is subject to civil aeronautical legislation. This regulation applies to every civilian RPA whatever their MTOW is carrying out specialised air operations.

A UAV can be controlled within the visual line of sight (VLOS) or an extended visual line of sight (EVLOS):

- VLOS: Operations in which the pilot has direct visual contact with the RPA.

- EVLOS: Operations in which direct visual contact with the aircraft is satisfied by observers in permanent radio contact with the RPA pilot.

Chapter II. Art.4. General requirements using RPA: *"The remote pilot shall always be responsible for detecting and avoiding potential collisions and other hazards."*

Chapter III, Art.21: Conditions of use of the airspace for the conduct of specialized operations:

- When the operation is conducted within the VLOS, all RPAs without airworthiness certificates may conduct specialized air operations in areas outside cities, towns, or outdoor meeting places of people. If the operation is managed within EVLOS, the maximum horizontal distance between the RPA from the observers (or the pilot) cannot exceed 500 m and a full ceiling of 120 m (400 ft).

- Specialised aerial operations may be conducted over cities, towns, or outdoor meeting places of people only for RPAs with an MTOW < 10 kg in VLOS operation with a maximum horizontal distance of 100 m and a maximum ceiling of 120 m (400 ft). Such procedures may be carried out in areas delimited by the authority, restricting the passage of persons and vehicles. A minimum-security distance of 50 m from buildings must be maintained.

Chapter III, Art.25. Visual flight weather conditions: RPAs must operate in daylight and visual meteorological conditions (VMC).

Chapter III, Section 2. Art.29. Piloting limitations: RPA may not be flown from moving vehicles unless operational planning is in place to ensure that there's an obstacle between the remote pilot station and the aircraft.

Chapter III, Section 2. Art.30: Protection and recovery area: The operator must establish a protected area for take-off and landing so that no person under the direct control of the operator is within a minimum 30 m radius, except Vertical Take-Off and Landing (VTOL) aircraft, in which case, may be reduced to a minimum of 10 m.

Chapter III, Section 2. Art.32. Overflight of installations: The overflight of chemical, transport, energy, water, and telecommunications infrastructures must be, after obtaining authorisation, at a minimum height above them of 50 m and at least 25 m horizontally from their axis (if there are linear facilities) and not less than 10 m of distance from its outer perimeter. Overflying national defense or state security facilities is forbidden.

Finally, it should be noted that the application has been developed around this regulation and therefore the height and range ranges are limited to it.

On the other hand, this application has been developed based on a specific use of the UAV and therefore it is the user's responsibility to have prior knowledge of these regulations as well as all permits and licenses in order before making a flight.

6. CONCLUSIONS

The ultimate conclusions of the study are discussed in this section of the memory. To outline the goals attained, the system's flaws, and potential improvements, the chapter has been divided into three parts.

6.1. Achievements

This section will review all the initial points set out as targets in Chapter 3 and comment on the results achieved:

- ✓ **Target 1: Extraction and processing of the information given in the files of the company subcontracting the service.**

  This project would not have been feasible if this goal had not been accomplished. The procedure required two steps: the first involved converting the KMZ file to ZIP format and subsequently to KML, and the second involved extracting the data using the XML file parsing method and further processing it.

- ✓ **Target 2: Development and implementation of a user application that allows the generation of flight plans according to the UAV's limitations and to the current regulations, using the data extracted from the given files:**

  This point is summarised in the development of the Archer App. The application is essential to generate the flight plans that will be sent to the UAV so that it can carry out the flight autonomously. This point has not only been fulfilled, but the programme has also received additional functionalities to enhance the user experience.

- ✓ **Target 3: Redesign of the camera support mount for horizontal focusing:**

  This objective has been successfully achieved, taking advantage of the mistakes made in the first prototype, the second design allowed a range of 180º rotation to fix the focus position of the camera which can be useful for other applications.

- ✓ **Target 4: Development of an algorithm capable of detecting objects through the UAV camera using machine learning and implementing it in the system**:

  The object detection model selected for the project, YOLOv5x, had to be configured, trained, and evaluated with a pre-processed dataset to detect transmission towers. The implementation consisted of modifying the back-end script of the UAV camera service script and making it compatible with the Archer App communication protocol in accordance with DEE.

The model is capable of detecting towers while the application saves those images in a repository with their respective coordinates.

✓ **Target 5: Perform system tests in a real environment, such as at the UPC *DroneLab*, to verify its operation:**

After the development of each stage and its successful simulation, real tests have been carried out at the UPC *DroneLab* facilities to verify the performance of the system. Two flight tests have been conducted and all of them had been favourable, which means the accomplishment of this target.

✓ **Target 6: System integration to the DEE:**

One of the challenges of this project has been the integration of the Archer App into DEE. Initially there were some problems while using functions already implemented in both the camera and autopilot services, which were finally solved by adding own functions adapted to the DEE communication protocol.

✓ **Target 7: Document all functionalities so that the system can be further developed in the ecosystem in the future.**

Once the development and evaluation of the system was completed, the contribution to DEE was uploaded to a GitHub repository with its corresponding documentation so that future students can continue in its development. In addition, the code used has been carefully commented so that the task performed by each function can be understood.

Therefore, in view of the results, the study can be considered satisfactory with an estimated cost of 18,494.97 € (refer to Annex C for breakdown).

6.2. Feasibility

In view of the results achieved it can be concluded that the study is feasible.

Despite the fulfilment of all the proposed objectives, it must be emphasised that the purpose of this project has been purely academic, although with prospects for commercial application.

During the course of the study, a number of weaknesses have been identified that makes it not applicable for commercial power line inspection. The most relevant are listed below:

- **UAV platform**. The vehicle used in this study is for educational and/or amateur purposes and has significant payload and autonomy limitations that make it incompatible with a professional use. The use of this drone platform was mainly to demonstrate the correct functioning of the system in a real environment.

- In addition, the use of Wi-Fi as the main communication technology with the Archer App makes its range even more limited (about 45 m) because it must keep in mind that the extension of power lines is extended over hundreds of kilometres long, with spacings between towers of around 20 to 100 m.

- **Payload**. In relation to the previous point, payload and limited space do not allow carrying heavier equipment such as a high-resolution camera, IR camera and LIDAR system. This is why the study has focused only on the detection of tower lattice structures only.

- **Interferences and obstacles**. When the system flies autonomously, it needs a constant communication with the Archer App. If an object appears between the UAV and the app that could interrupt this communication, the vehicle will not be able to continue the flight. In addition, as discussed in Annex A, interference to the GPS signal can occur in areas with frequency jammers or when flying near telecommunication antennas with a high transmitting power.

- **Regulation.** While this is not a direct weakness of the system, it affects all unmanned vehicles that are subject to this legislation, the current regulation limits the range to 500 m and the flight ceiling to 120 m. These restrictions make the use of aircraft such as helicopters more suitable for inspecting long distances in a short time.

Regarding all the mentioned points, it is considered that, at this time, the system is not commercially applicable. However, the system might be made suitable for commercial aerial works by making an important investment on purchasing a professional UAV, cameras, LIDAR, and by applying the groundwork built in this study.

By taking this step, power line inspections could be carried out on a lower budget compared to using helicopters, but at the same time the overall time spent would be significantly higher. Therefore, the company contracting the service will have to make a trade-off between cost savings and time savings.

6.3. Further Upgrades

This study, as a contribution to the *Drone Engineering Ecosystem*, opens the door for future students to further develop and improve the system.

This section lists some improvements that have been found to be of interest:

- Improvement of the current dataset and model to allow a better and faster tower detection for a diversity of structure topologies, such as tuchoy and wooden.

- Implement a web-app version of the Archer App to allow users to create flight plans from other terminals connected to the internet. Currently in DEE there are some front-end programs using Vue.js and Iconic.

- Study and analyse other alternative communication protocols between the UAV and the Archer App to reduce latency and delay times and increase the flight range.

- Implement a system that allows the camera to rotate and thus avoid the yaw rotation sequence that the UAV must perform to scan an area. This would possibly reduce battery consumption and allow more flight time.

- Add more functionalities to the application, e.g., an option to enter the coordinates by hand and these will generate the waypoint or an option to print a report of the scheduled flight for the user.

BIBLIOGRAPHY

[1] *Real Decreto 223/2008, de 15 de febrero, por el que se aprueban el Reglamento sobre condiciones técnicas y garantías de seguridad en líneas eléctricas de alta tensión*. Retrieved March 22, 2023, from: https://www.boe.es/eli/es/rd/2008/02/15/223

[2] *Real Decreto 1036/2017, de 15 de diciembre, por el que se regula la utilización civil de las aeronaves pilotadas por control remoto*. Retrieved March 22, 2023, from https://www.boe.es/eli/es/rd/2017/12/15/1036

[3] F. Lewckiki, *Power supply lines. Calculation of the electric and magnètic field strength,* ITU-T, Geneva, 16 April, 2012. Retrieved March 15, 2023, from: https://www.itu.int/dms_pub/itut/oth/06/5B/T065B0000180006PDFE.pdf

[4] Ministerio de Industria, Energía y Turismo, *Guía Técnica de Aplicación ITC LAT 07,* ITU-T, April, 2019. Retrieved March 22, 2023, from: https://industria.gob.es/Calidad-Industrial/seguridadindustrial/instalacionesindustriales/lineas-alta-tension/Documents/guia-itc-lat-07-rev-2.pdf

[5] J. Viña Fernández, *Línea Aéres de Transporte a 400 KV,* URV, September, 2003. Retrieved March 22, 2023, from: http://deeea.urv.cat/public/PROPOSTES/pub/pdf/358pub.pdf

[6] J. Ignacio Huertas, R. Barraza, J. Mauricio Echeverry, *Wireless Data Transmission from Inside Electromagnetic Fields,* IMPI, February 15, 2010. Retrieved from: https://www.researchgate.net/publication/224105778_Wireless_data_transmission_from_inside_electromagnetic_fields

[7] REE, *Gestor de la red y transportista*. Retrieved March 24, 2023, from: https://normas-apa.org/referencias/citar-pagina-web/comment-page-4/

[8] M. Valero, *DroneEngineeringEcosystemDEE*, March 24, 2023. Retrieved from: https://github.com/dronsEETAC/DroneEngineeringEcosystemDEE

[9] scikit learn, *sklearn.metrics.pairwise.haversine_distances* March 27, 2023. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine_distances.html#:~:text=The%20Haversine%20(or%20great%20circle,the%20longitude%2C%20given%20in%20radians.

[10] R. Abdelfattah, X. Wang, S. Wang, *TTPLA: An Aerial-Image Dataset for Detection and Segmentation of Transmission Towers and Power Lines*. Retrieved April 4, 2023, from https://github.com/R3ab/ttpla_dataset

[11] G. Jocher, *v7.0 – YOLOv5 SOTA Realtime Instance Segmentation*. Retrieved May 3, 2023, from https://github.com/ultralytics/yolov5/releases

[12] PyTorch, *YOLOV5*. Retrieved May 3, 2023, from: https://pytorch.org/hub/ultralytics_yolov5/

[13] v7labs, *YOLO: Algorithm for Object Detection Explained*, April 3, 2023. Retrieved from: https://www.v7labs.com/blog/yolo-object-detection#:~:text=YOLO%20v5%20uses%20a%20new,clusters%20as%20the%20anchor%20boxes.

[14] M. Tan, *EfficientDet: Towards Scalable and Efficient Object Detection*, April 4, 2023. Retrieved from: https://ai.googleblog.com/2020/04/efficientdet-towards-scalable-and.html

[15] PX4 User Guide, *Cube Wiring Quick Start*, April 4, 2023. Retrieved from: https://docs.px4.io/main/en/assembly/quick_start_cube.html

[16] CYIENT, *LiDAR*, April 4, 2023. Retrieved from: https://www.cyient.com/geospatial/lidar

[17] Semantic Scholar, *Approach to automated hot spot detection using image processing for thermographic inspections of power transmission lines*, April 4, 2023. Retrieved from: https://www.semanticscholar.org/paper/Approach-to-automated-hot-spot-detection-using-for-Wronkowicz/221d8eff732037c605db2a60ad78a208bf0af267

[18] Yehia Sayed M, Abdel A, *Mathematical calculations of EM field in high-voltage substations to treatment its effect on the protective equipments*, March 16, 2023, Faculty Engineering Hunedoara, November 2016. Retrieved from: https://annals.fih.upt.ro/pdf-full/2016/ANNALS-2016-4-20.pdf

[19] INELCO, *Armónicos en las redes eléctricas*, May 18, 2023. Retrieved from: https://inelco.info/blog/?p=626

[20] IR-Lock, *Melopero Raspberry Pi 3 Model B, CPU Quad Core 1,2GHz Broadcom BCM2837 64bit, 1GB RAM, WiFi, Bluetooth BLE, plata*, May 18, 2023. Retrieved from: https://irlock.com/products/hexsoon-edu-450-v2

[21] Amazon, *Hexsoon EDU-450 V2*, May 18, 2023. Retrieved from: https://www.amazon.es/Raspberry-Pi-Modelo-Quad-Core-Cortex-A53/dp/B01CD5VC92/ref=asc_df_B01CD5VC92/?tag=googshopes-21&linkCode=df0&hvadid=195264428792&hvpos=&hvnetw=g&hvrand=6886367963442601832&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=20270&hvtargid=pla-195196633035&psc=1

[22] Amazon, *Raspberry Pi Camera Module V2 8MP*, May 18, 2023. Retrieved from: https://www.amazon.es/Raspberry-Pi-Camera-Module-8MP/dp/B01ER2SKFS/ref=sr_1_5?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=240YH0ELEO9J3&keywords=v2.1+camera

+raspberry&qid=1684446503&s=computers&sprefix=v2.1+camera+raspberry%2Ccomputers%2C104&sr=1-5

[23] Amazon, *Impresora 3D Creality Ender 3 V2 con Tablero silencioso de 32 bits Plataforma de Vidrio de carborundo Diseño de Estructura integrada e impresión de currículum 220x220x250mm*, May 18, 2023. Retrieved from: https://www.amazon.es/Creality-Impresora-impresión-reanudación-doméstico/dp/B07FFTHMMN/ref=sr_1_1_sspa?keywords=Ender-3+V2&qid=1684446845&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1

[24] Microsoft, *Microsoft 365 Personal*, May 19, 2023. Retrieved from: https://www.microsoft.com/es-es/microsoft-365/buy/compare-all-microsoft-365-products-b

[25] Autodesk, *Fusion 360: More than CAD, it's the future of design and manufacturing*, May 19, 2023. Retrieved from: https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR&tab=subscription#buy

[26] Talent.com, *Average Software Engineer Salary in Spain*, May 19, 2023. Retrieved from: https://www.payscale.com/research/ES/Job=Software_Engineer/Salary

[27] L3HARRIS, *WESCAM MX-15, AIR SURVEILLANCE AND RECONNAISANCE*, May 21, 2023. Retrieved from: https://www.l3harris.com/all-capabilities/wescam-mx-15-air-surveillance-and-reconnaissance

ANNEX A: EMI ANALYSIS

In this annex a short analysis of electromagnetic interference (EMI) near high voltage power lines will be carried out to determine if there is a potential risk of affecting the flight of the UAV during inspection.

EM field analysis will be performed for a simple two-dimensional model to facilitate calculations and then a series of assumptions will be made to draw conclusions.

A.1. Physical Approach

These is a simple physical approach of the electric and magnetic field under a power line from an $P(x,y)$ observer point. In these analysis conductors are considered infinite line charges and only one-phase power line conductor.

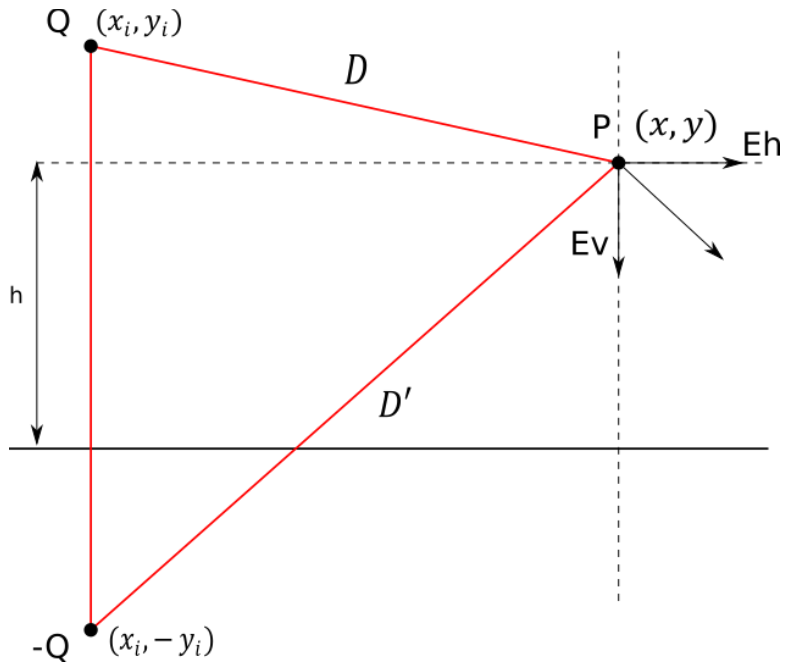

Fig. A.1 Situational scheme

From Fig. A.1.1 can be seen point Q and -Q corresponding to the power line section and its image.

It is important to take into account the image of an electric charge Q when analysing its behaviour in the presence of conductive surfaces (the ground is considered electrostatically a good conductor), as it can have a significant effect on the electrical forces acting on the charge.

Both points are at a distance D and D' respectively from the observer point P.
In electromagnetism, the electric flux is the amount of electric field lines passing through a given area S:

$$\Phi = \int \vec{E} \cdot d\vec{S} \tag{A.1}$$

Considering that the surface surrounding the conductor has a cylindrical shape, where $h \to 0$ (two-dimensional study):

$$S = 2\pi r^2 + 2\pi r h \tag{A.2}$$

$$S = 2\pi r^2 \tag{A.3}$$

The expression of the electric flux then will be:

$$\Phi = \int \vec{E} \cdot d\vec{S} = E \int d\vec{S} = E \cdot S = E\ (2\pi r^2) \tag{A.4}$$

Redefining $r_i \to D_i$ for each i-contribution and applying Gauss Law:

$$\Phi = \frac{Q}{\varepsilon_0} \tag{A.5}$$

It is possible to calculate the electric field:

$$E\left(2\pi {D_i}^2\right) = \frac{Q}{\varepsilon_0} \tag{A.6}$$

$$E = \frac{Q}{2\pi\varepsilon_0 {D_i}^2} \tag{A.7}$$

Now considering the electric field as a vector and applying superposition for each contribution:

$$\vec{E} = E\,\vec{u} \tag{A.8}$$

$$\overrightarrow{E_h} = \sum \vec{E}_i = \vec{E} + \vec{E}' \tag{A.9}$$

To obtain the horizontal and vertical components expression of the electric field by summing the contribution of each charge at a distance D and D':

$$E_{hi}(x) = \frac{Q}{2\pi\varepsilon_0}\left[\frac{1}{D^2} - \frac{1}{D'^2}\right](x - x_i) \tag{A.10}$$

$$E_{vi}(y) = \frac{Q}{2\pi\varepsilon_0}\left[\frac{(y - y_i)}{D^2} - \frac{(y + y_i)}{D'^2}\right] \tag{A.11}$$

The same can be done for the magnetic field:

$$B_{hi}(x) = \frac{\mu I}{2\pi}(x - x_i)\left[\frac{1}{D^2} - \frac{1}{D'^2}\right] \tag{A.12}$$

$$B_{vi}(y) = \frac{\mu I}{2\pi}\left[\frac{(y-y_i)}{D^2} - \frac{(y+y_i)}{D'^2}\right] \tag{A.13}$$

Finally, both electric and magnetic resultant fields can be expressed as follows:

$$E(x, y) = \sqrt{E_{hi}(x)^2 + E_{vi}(y)^2} \tag{A.14}$$

$$B(x, y) = \sqrt{B_{hi}(x)^2 + B_{vi}(y)^2} \tag{A.15}$$

A magnetic field B will be created by the current going through the conductors.

$$I = \frac{P}{\sqrt{3} \cdot U \cdot cos\theta} \tag{A.16}$$

Where:

Q: Charge of the conductor resultant of horizontal and vertical components
ε_0: Relative permittivity of air
μ: Air relative permeability
P: Active power
U: Voltage applied.
$cos\theta$: Power factor. Being θ the angle between voltage and current

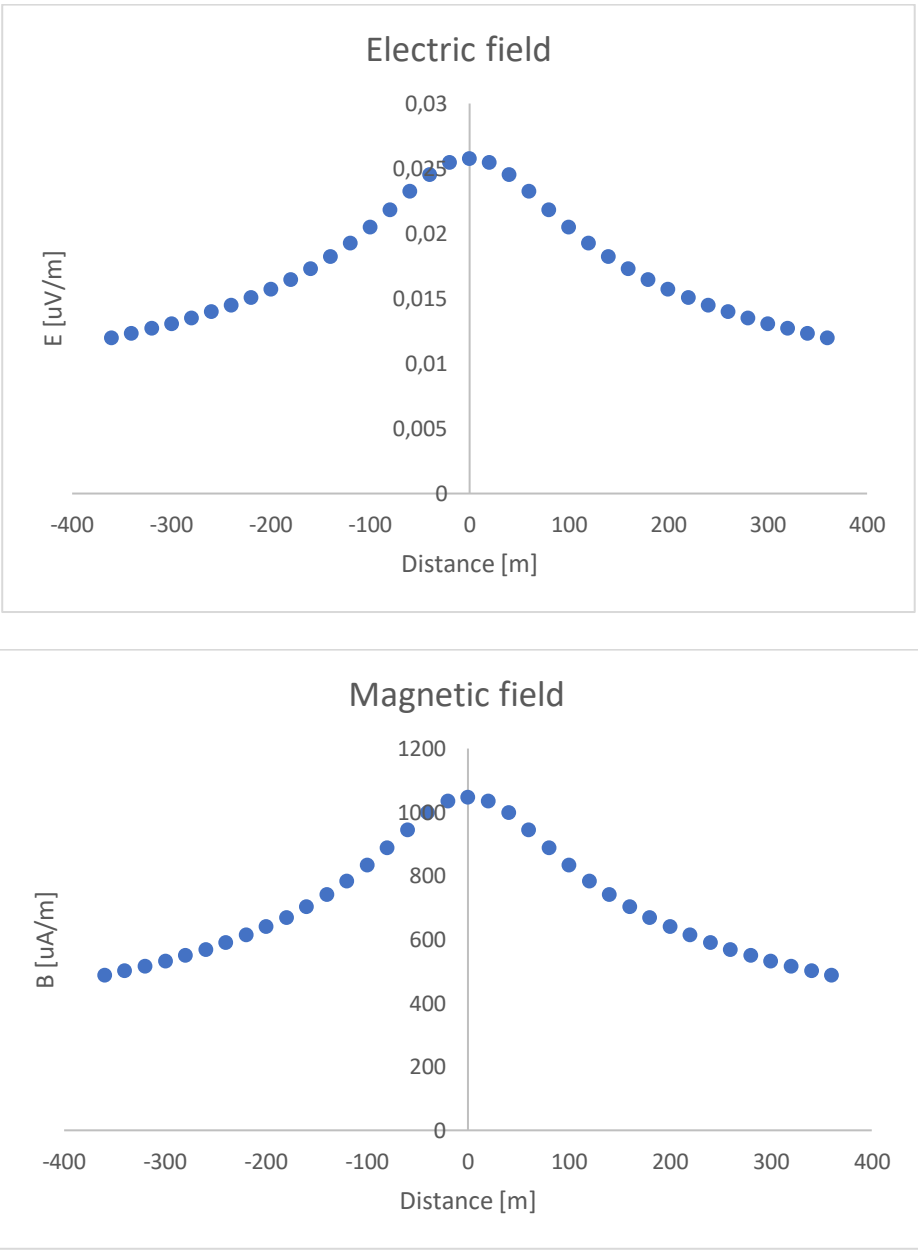

Fig. A.2 Plots of the electric and magnetic field in presence of a single electric charge (A.14),(A.15) along x-axis

According to the resultant expressions it can be deduced that the EM fields will be dependent of:

- The number of conductors on the power line
- Distance observer-conductor
- Distance observer-image, directly related with tower's height above ground.
- Type of power line. Depending on the power that can deliver, these implies using certain materials and specific radius for the conductors, the amount of current and voltage that can hold.

A.2. Simulation and results

To get accurate calculations about the presence of EM fields around complex infinitesimal-length power lines, it has been used ITU's *EMFACDC* software tool.

This software is used to evaluate the strength of electric and magnetic fields around power supply lines.

Table A.1 Spanish nominal and highest network voltages [1]

Nominal Network Voltage [kV]	Highest Network Voltage [kV]
3	3,6
6	7,2
10	12
15	17,5
20	24
25	30
30	36
45	52
66	72,5
110	123
132	145
150	170
220	245
400	420

Table A.1 shows the nominal voltage for overhead power lines and its highest allowed value. Blue indicates the most used voltage used by electrical companies.

So, the following analysis will consider a 400kV hypothetical power line as the worst-case scenario with the following aspects (Fig.A.3):

- Type of conductor ACSR (Aluminium Cable Steel Reinforced) with a cross section of 546.1 mm^2 (r = 3.18 mm) [5].
- For 546.1 mm^2, the current density for an aluminium alloy conductor is 1.55 A/mm^2. Therefore, will flow 930 A of current approximately.
- The average height of a 400 kV transmission tower is between 20 and 55 m. There isn't any relation between height and voltage, indeed, depends on the potential risk of surrounding obstacles. It will be considered 25 m.
- 3 conductors/circuit (typical configuration).
- Frequency 50 Hz (EU).
- Three-phase system (120º)

Fig. A.3 Similar 400 kV transmission tower model

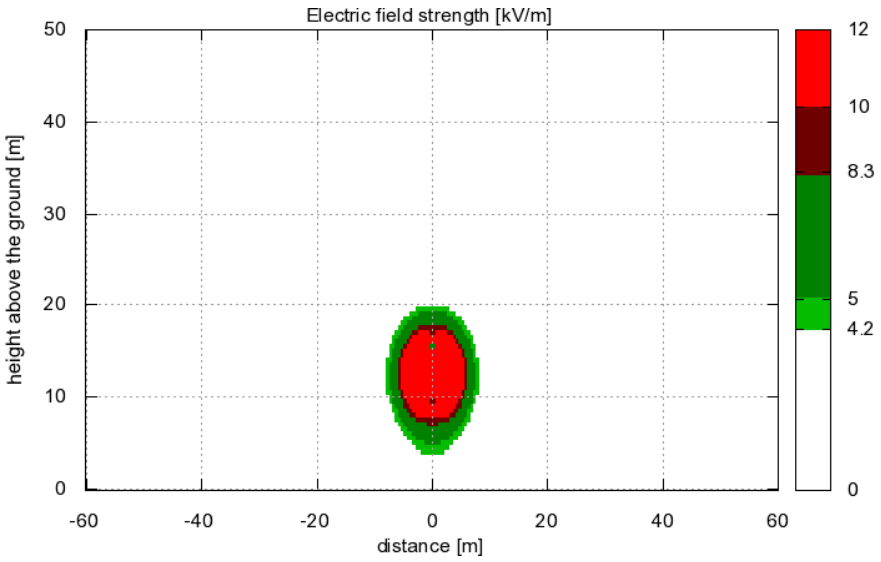

Fig. A.4 EMFACDC, electric field distribution around the conductors

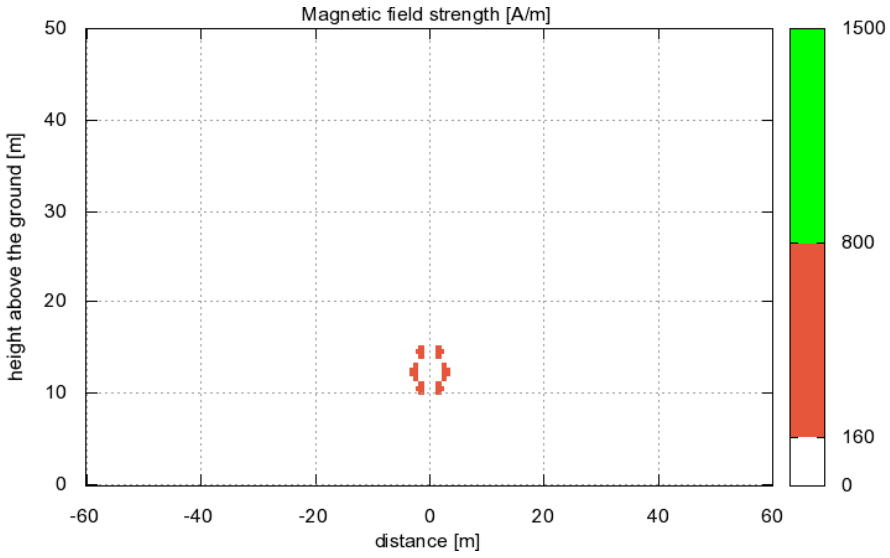

Fig. A.5 EMFACDC, magnetic field distribution around the conductors

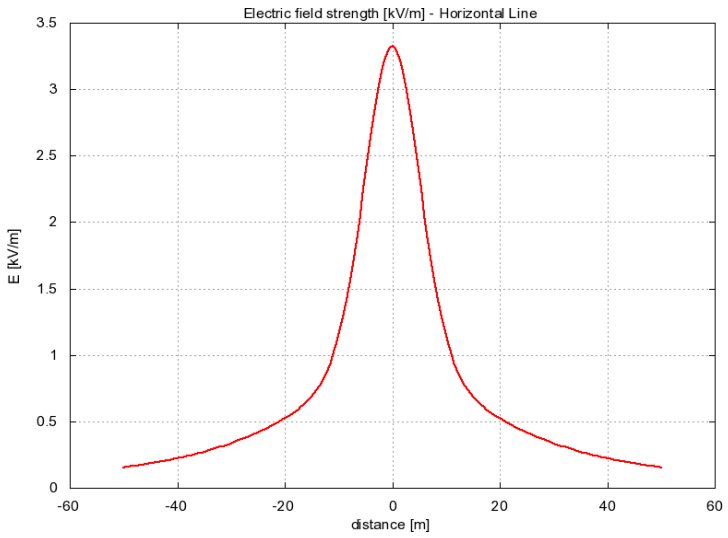

Fig. A.6 EMFACDC, electric field distribution along x-axis

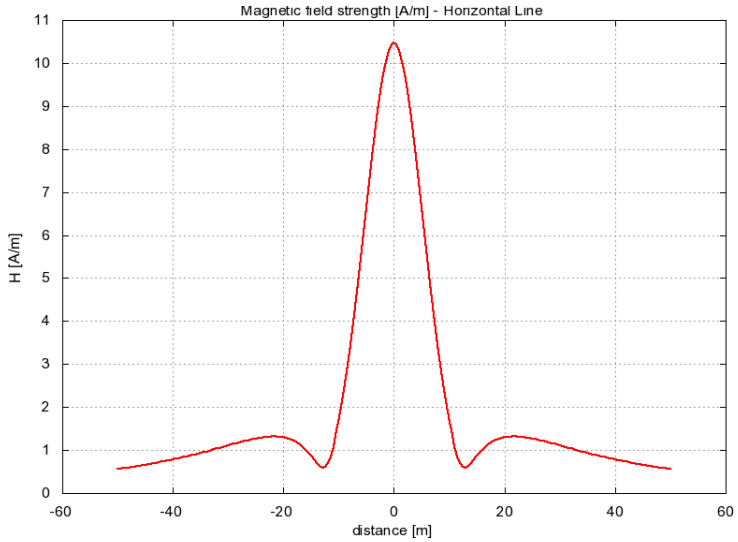

Fig. A.7 EMFACDC, magnetic field distribution along x-axis

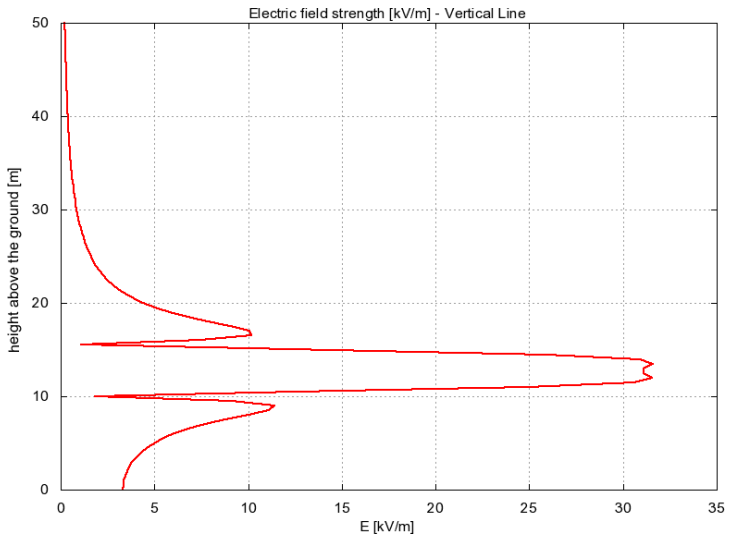

Fig. A.8 EMFACDC, electric field distribution along y-axis

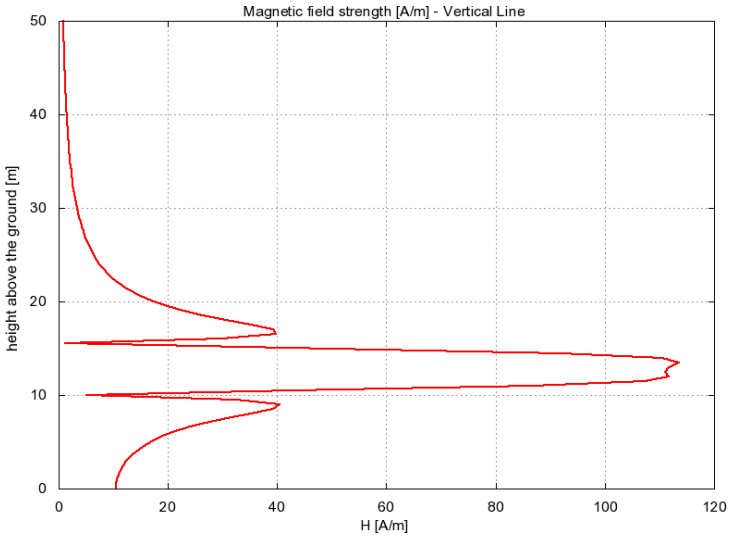

Fig. A.9 EMFACDC, magnetic field distribution along y-axis

In the first instance the results for the hypothetical model appear to be similar to those obtained in the physical approximation of section A.1, the difference lies in the contributions of the other conductors.

From Fig.A.4, Fig.A.5, Fig.A.6, Fig.A.7, Fig.A.8 and Fig.A.9 can be seen that the EM field increases quadratically in strength as an observer approaches the power lines. The EM field distribution along the x-axis shows that flying too close to the tower could expose the UAV to high values, as well as at a height near to the power lines.

In other words, depending on the air conditions between the conductors and the UAV, there will be a greater probability of ionising the air and causing an electrical discharge on the vehicle.

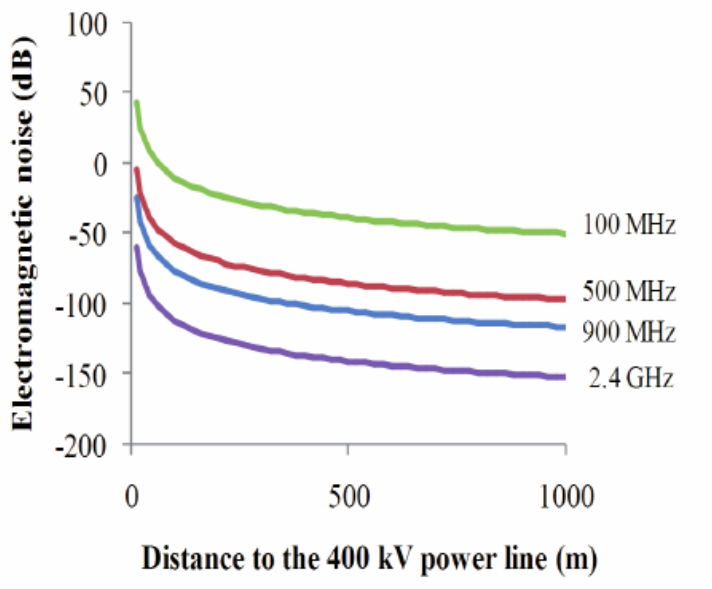

Fig. A.10 Estimation of the radio frequency interference from a 440 kV power line [6]

Fig. A.10 estimates the radio frequency interference noise generated by the electric transmission lines of 400 kV as a function of the distance to the transmission line and frequency [6].

From a frequency point of view, the conductor is considered to be a non-linear element that generates harmonics in integer multiples every 50 Hz. The distorted signal at the output is also considered to have cubic characteristics to simplify the calculations [19], see Fig.A.11.

Where Vin is a sinusoidal signal at the input of the element and $Vout$ is the resultant signal passing through the element.

$$V_{in}(t) = A \cdot \sin(w_0.t) \quad \textbf{(A.17)}$$

$$V_{out} = a_1 V_{in} + a_1 {V_{in}}^2 + a_1 {V_{in}}^3 \quad \textbf{(A.18)}$$

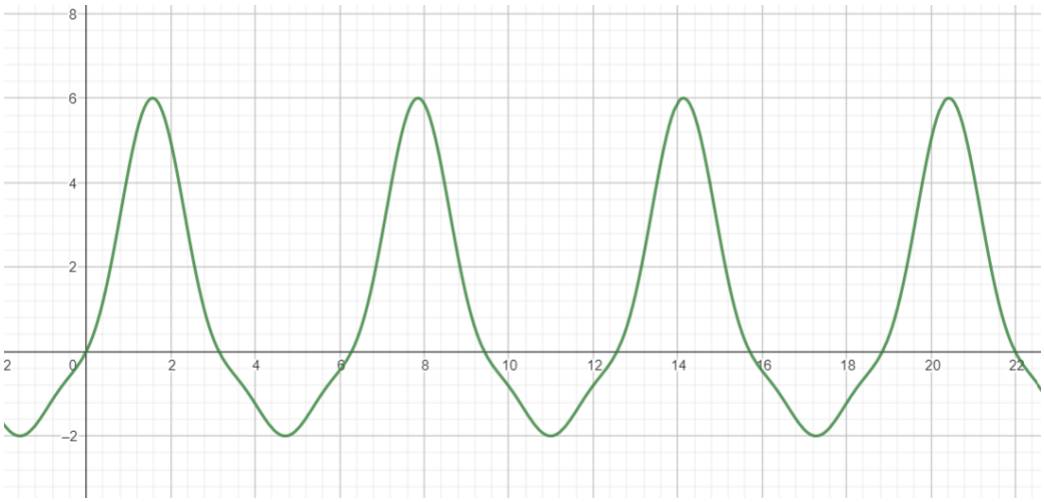

Fig. A.11 $\boldsymbol{Vout}$ with fundamental component, 2nd, and 3rd harmonic (A.19)

$$V_{out}(t) = a_1(A \cdot \sin(w_0.t)) + a_2(A \cdot \sin(w_0.t))^2 + a_3(A \cdot \sin(w_0.t))^3 \quad \textbf{(A.19)}$$

Developing (A.19) with the trigonometric relations for the double angle and the triple angle:

$$V_{out}(t) = a_1 \cdot A \cdot \sin(w_0.t) + a_2 \cdot A^2 \cdot \frac{1}{2}(\sin(w_0.t))^2 + a_3 \cdot A^3 \cdot \frac{1}{4}[(3 \cdot \sin(w_0.t) - \sin(3 \cdot w_0.t)] \quad \textbf{(A.20)}$$

Applying the Fourier transform of the previous function, it is obtained the frequency components of $Vout$:

$$V_{out}(w) = a_1 \cdot A \cdot \pi[\delta(w + w_0) - \delta(w - w_0)]j + a_2 \cdot A^2 \cdot \pi \cdot \delta(w) + \frac{a_2 \cdot A^2 \cdot \pi}{2}[\delta(w + 2w_0) - \delta(w - 2w_0)]j + \frac{3 \cdot a_3 \cdot A^3 \cdot \pi}{4}[\delta(w + w_0) - \delta(w - w_0)]j + \frac{a_3 \cdot A^3 \cdot \pi}{4}[\delta(w + 3w_0) - \delta(w - 3w_0)]j \quad \textbf{(A.21)}$$

It is observed that as the frequency of the harmonics increases, the value of their coefficient decreases.

While UAV's *RX8R PRO* receiver operates at 2.4-2.48 GHz, telemetry radio emitter at 433 MHz band and the Spanish power grid at 50 Hz, due to frequency separation, EMI should not affect the vehicle while inspecting a tower.

In addition, as shown above in Fig. A.10, as the UAV increases the distance to the power line there is a reduction in EM noise.

Alternatively, the presence of telecommunication towers nearby could potentially interfere with the received GPS signal.
Knowing that:

- Terrestrial television uses the UHF frequency bands (470-790 MHz)

- Mobile communications operate in several frequency bands, including 800 MHz, 900 MHz, 1800 MHz, 2100 MHz, 2600 MHz, and 3500 MHz (these bands are used for mobile technologies such as 2G, 3G, 4G and 5G)

- FM radio uses the VHF frequency band between 87.5-108 MHz

Generally, television and radio bands do not overlap with the frequency bands used by GPS, which operate on frequencies L1 (1575.42 MHz) and L2 (1227.60 MHz). Therefore, TV and radio signals generally do not significantly affect the GPS signal.

Oppositely, mobile phone signals can interfere with the GPS signal. This is because mobile phone signals may use frequency bands close to the frequency bands used by GPS, especially in the L1 frequency bands. Mobile phone interference can cause a reduction in GPS position accuracy as well as a decrease in GPS signal quality, being a potential safety risk while flying the UAV.

To minimise mobile phone interference, modern GPS receivers use advanced signal processing techniques, such as filtering and removal of interfering signals, and differential correction to improve the accuracy of the GPS position. In addition, the location of the GPS antenna can also be important, as an antenna placed in a position with a clear line of sight to the sky can help minimise interference from nearby signals.

GPS signal processing in the Pixhawk Orange Cube receiver involves several techniques such as: Kalman filtering, differential correction, moving average filtering and cross-correlation filtering.

A.3. Flight Test

To verify the above theoretical analysis, a helicopter flight was made near telecommunication towers to check on-board tower inspection equipment and the effect of interference on GPS signal reception, see Fig.A.12.

Fig. A.12 Taking-off (left) and last equipment checks (right)

It was observed during the flight that there was indeed no effect of the high-voltage towers on both the GPS signal and the 2.4 GHz receiver.

On the other hand, there was occasional interference in areas with frequency jammers or near telecommunication towers. The same test was carried out with the equipment inside a Faraday cage-like container made of aluminium about 3 cm thick (Fig. A.13), and it was observed that the interference decreased.

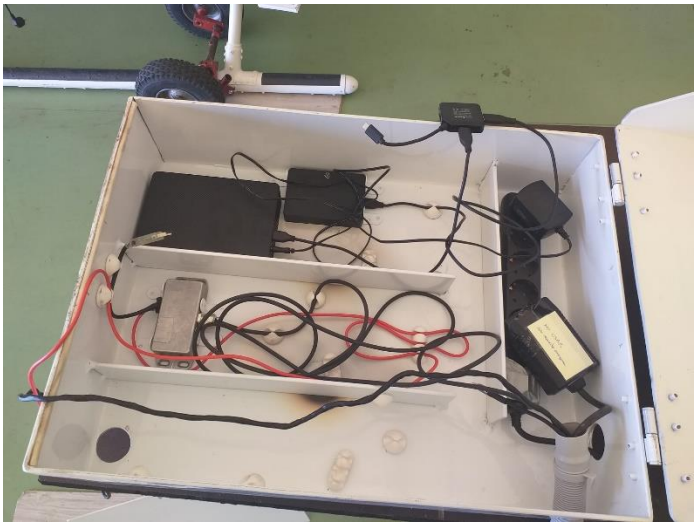

Fig. A.13 Faraday cage used

A.4. Final conclusions

The inspection of high-voltage towers does not affect the communication between the UAV and the base station at 433MHz/2.4 GHz band, nor does it affect the GPS signal. However, special caution should be taken when flying close to high voltage power lines due to the possibility of the formation of electric arcs that can affect electronic equipment.

Although the interference problem is solved to a greater extent, when jammers and telecommunications antennas are present, for the UAV considered in this study, the weight of a Faraday cage would imply a considerable overweight that may affect its flight performance. It would be advisable to carry out tower inspections outside the range of frequency jammers and telecommunication towers or alternatively, use another vehicle with larger load capacity.

ANNEX B: KML FILE FORMAT

This annex shows an example of a *.kml* file used in this project to perform flight test at the UPC *DroneLab* facilities.

Fig. B.1 and Fig. B.2 shows in white colour the virtual fence to prevent the UAV from flying too close to the perimetral net and in yellow a designed power line path with its XML code below.

Fig. B.1 Situational view of UPC *DroneLab* (Satellite image not updated)

Fig. B.2 Aerial view of UPC *DroneLab* (Satellite image not updated)

```
<?xml version="1.0" encoding="UTF-8"?>
<kml
  xmlns="http://www.opengis.net/kml/2.2"
  xmlns:gx="http://www.google.com/kml/ext/2.2"
  xmlns:kml="http://www.opengis.net/kml/2.2"
  xmlns:atom="http://www.w3.org/2005/Atom">
  <Folder>
    <name>DRONE_LAB_FLIGHT_TEST</name>
    <Document>
      <name>EETAC</name>
      <Style id="style13">
        <IconStyle>
          <Icon>
            <href>http://maps.google.com/mapfiles/kml/shapes/place-
mark_circle.png</href>
          </Icon>
        </IconStyle>
      </Style>
      <Style id="style22">
        <IconStyle>
          <Icon>
            <href>http://maps.google.com/mapfiles/kml/shapes/place-
mark_circle.png</href>
          </Icon>
        </IconStyle>
      </Style>
      <Style id="style32">
        <IconStyle>
          <Icon>
            <href>http://maps.google.com/mapfiles/kml/shapes/place-
mark_circle.png</href>
          </Icon>
        </IconStyle>
      </Style>
      <Placemark>
        <name>DLT-1</name>
        <description>
          <![CDATA[<p>Height: 5 m</p><p>In-
stal: DroneLab</p><p>Loc: EETAC</p>]]>
        </description>
        <styleUrl>#style22</styleUrl>
        <Point>
          <coordinates>1.988380,41.276214,0</coordinates>
        </Point>
      </Placemark>
      <Placemark>
        <name>DLT-2</name>
        <description>
```

```
          <![CDATA[<p>Height: 5 m</p><p>In-
stal: DroneLab</p><p>Loc: EETAC</p>]]>
        </description>
        <styleUrl>#style22</styleUrl>
        <Point>
          <coordinates>1.989048,41.276463,0</coordinates>
        </Point>
      </Placemark>
      <Placemark>
        <name>DLT-3</name>
        <description>
          <![CDATA[<p>Height: 5 m</p><p>In-
stal: DroneLab</p><p>Loc: EETAC</p>]]>
        </description>
        <styleUrl>#style22</styleUrl>
        <Point>
          <coordinates>1.988370,41.276447,0</coordinates>
        </Point>
      </Placemark>
    </Document>
    <Document>
      <name>PATH_DL</name>
      <open>1</open>
      <Style id="style62">
        <LineStyle>
          <width>5</width>
        </LineStyle>
      </Style>
      <Style id="style91">
        <LineStyle>
          <color>ff00ffff</color>
          <width>5</width>
        </LineStyle>
      </Style>
      <Placemark>
        <name>PL1</name>
        <description>
          <![CDATA[<p>APOINI_MAT: DLT-1</p><p>APOFIN_MAT: DLT-
2</p>]]>
        </description>
        <styleUrl>#style91</styleUrl>
        <LineString>
          <tessellate>1</tessellate>
          <coordinates>
          1.988380,41.276214,0 1.989048,41.276463,0
        </coordinates>
        </LineString>
      </Placemark>
      <Placemark>
```

```
        <name>PL2</name>
        <description>
          <![CDATA[<p>APOINI_MAT: DLT-2</p><p>APOFIN_MAT: DLT-
3</p>]]>
        </description>
        <styleUrl>#style91</styleUrl>
        <LineString>
          <tessellate>1</tessellate>
          <coordinates>
          1.989048,41.276463,0 1.988370,41.276447,0
        </coordinates>
        </LineString>
      </Placemark>
    </Document>
  </Folder>
</kml>
```

ANNEX C: ECONOMICAL MANAGEMENT

An important aspect of this project is the economic management, which consists in the elaboration of a budget where all the necessary expenditure items are identified together with the cost and justification of each one.

C.1. Measurements

This section lists and classifies the different tasks performed during the study as well as the time spent on each of them, see Table C.1.

Tasks A are related to the research and familiarisation stage, tasks B are those performed during the development process of the system, tasks C are those dedicated to the manufacturing of components and finally task D is dedicated to the documentation of all previous tasks.

Table C.1 Study measurements

Number	Code of task	Task	Duration	Measure
1	A-1	Requirements understanding	15	h
2	A-2	Python familiarisation	33	h
3	A-3	Python libraries study	33	h
4	A-4	KMZ and KML parsing study	25	h
5	A-5	Object-detection model study	42	h
6	B-1	Archer App development	140	h
7	B-2	Transmission tower dataset	10	h
8	B-3	Object-detection model validation	160	h
9	B-4	EMI analysis	40	h
10	C-1	Laptop camera USB conversion	3	h
11	C-2	CAD design development	20	h
12	C-3	Camera mount prototype manufacturing	32	h
13	C-4	Poster support manufacturing	18	h
14	C-5	Simulations and tests	18	h
15	D-1	Writing of the study	120	h

C.2. Unitary prices

C.2.1. Researching Time (Task A)

This section provides a breakdown of the costs per hour spent on each type of task.

Table C.2 Researching time breakdown

Code	Description	Units		
		€/month	h/month	€/h
	Engineer [26]			23.32
	Microsoft Office 365 [24]	5.75	60	0.09
	Python + PyCharm	0.00	100	0.00
01	**Study time**			**23.41**

C.2.2. Engineering time (B and D Tasks)

Table C.3 Engineering time breakdown

Code	Description	Units		
		€/month	h/month	€/h
	Engineer [26]			23.32
	Microsoft Office 365 [24]	5.75	60	0.09
	Inkscape	0.00	4	0.00
	Autodesk Fusion360 [25]	73.00	10	0.13
	EMFACDC	0.00	3	0.00
	Python + PyCharm	0.00	120	0.00
	Ardupilot Mission Planner SITL	0.00	24	0.00
	GitHub	0.00	8	0.00
02	**Engineering time**			**23.54**

C.2.3. Manufacturing time (C Tasks)

Table C.4 Manufacturing time breakdown

Code	Description	Units
		€ / h
	Engineer [26]	23.32
03	**Manufacturing time**	**23.32**

C.2.4. Manufacturing time (C Tasks)

Table C.5 and Table C.6 shows a detailed breakdown of all tools and materials used in this study to manufacture key components for the study and its unitary cost.

Table C.5 Tools price breakdown

Code	Description	Price	Measure
T-1	Einhell TE-CD 18 Li-i	157.00	€ / unit
T-2	Bosch X - Line	16.49	€ / unit
T-3	Stanley 450 mm 18”	22.49	€ / unit
T-4	Wisent Set	17.99	€ / unit
T-5	Standers 4x12	1.59	€ / unit
T-6	Imperial CRD 150	1.29	€ / unit
T-7	Standers ⌀5 mm	2.99	€ / unit
T-8	CON:P 8 – 17 mm	6.99	€ / unit
T-9	Preciva 60W 220V	21.24	€ / unit
T-10	HUAREW HR990307	9.99	€ / unit
T-11	Eventronic 560pcs	8.99	€ / unit

Table C.6 Material price breakdown

Code	Description	Price	Measure
M-1	*Hexsoon EDU-450* v2 [20]	221.90	€ / unit
M-2	Raspberry Pi 3 model B [21]	114.90	€ / unit
M-3	V2.1 8MP[22]	43.45	€ / unit
M-4	PLA	15.00	€ / unit
M-5	Ender-3 V2 [23]	296.00	€ / unit
M-6	Ceys Special Rigid Plastics (30 ml)	2.79	€ / unit
M-7	Wooden pine boards (800x180x18 mm)	10.49	€ / unit
M-8	High-resolution images (700x1400 mm)	55.86	€ / unit
M-9	Wolfpack 2.8 mm	5.99	€ / unit
M-10	INOFIX 9694997	5,49	€ / unit
M-11	Lenovo V15 G2 ITL	559.00	€ / unit
M-12	HP wireless 1JR31AA	59.65	€ / unit
M-13	BenQ GL2250 21.5”	104.00	€ / unit
M-14	TP-Link TL-WN722N	10.99	€ / unit

C.3. Budget

Table C.7 Study budget estimation

Concept	Code	Description	€/unit	Amount	Cost
Drill	T-1	Einhell TE-CD 18 Li-i	157.00	1	157.00 €
Drill bits pack	T-2	Bosch X - Line	16.49	1	16.49 €
Saw	T-3	Stanley 450 mm 18”	22.49	1	22.49 €
Screwdriver set	T-4	Wisent Set	17.99	1	17.99 €
Screws	T-5	Standers 4x12	1.59	1	1.59 €
Sandpaper	T-6	Imperial CRD 150	1.29	1	1.29 €
Nuts pack	T-7	Standers ⌀5 mm	2.99	1	2.99 €
Spanner set	T-8	CON:P 8 – 17 mm	6.99	1	6.99 €
Tin welder	T-9	Preciva 60W 220V	21.24	1	21.24 €
Tin roll	T-10	HUAREW HR990307	9.99	1	9.99 €
Heat shrink tube	T-11	Eventronic 560pcs	8.99	1	8.99 €
UAV	M-1	*Hexsoon EDU-450* v2	221.90	1	221.90 €
Electronic board	M-2	Raspberry Pi 3 model B [21]	114.90	1	114.90 €
Camera	M-3	V2.1 8MP	43.45	1	43.45 €
Filament	M-4	PLA	15.00	1	15.00 €
3D printer	M-5	Ender-3 V2	296.00	1	296.00 €
Glue	M-6	Ceys Special Rigid Plastics (30 ml)	2.79	1	2.79 €
Poster support	M-7	Wooden pine boards (800x180x18 mm)	10.49	2	20.98 €
Posters	M-8	High-resolution images	55.86	2	111.72 €
Rope	M-9	Wolfpack 2.8 mm	5.99	1	5.99 €
Hangers pack	M-10	INOFIX 9694997	5.49	2	10.95 €
Laptop	M-11	Lenovo V15 G2 ITL	559.00	1	559.00 €
Mouse	M-12	HP wireless 1JR31AA	59.65	1	59.65 €
Display	M-13	BenQ GL2250 21.5”	104.00	1	104.00 €
Wi-Fi antenna	M-14	TP-Link TL-WN722N	10.99	1	10.99 €
Direct cost					**1,844.37 €**
Task A	01	Study time	23.41	148	3,464.68 €
Task B	02	Engineering time	23.54	350	8,239.00 €
Task C	03	Manufacturing time	23.32	91	2,122.12 €
Task D	02	Engineering time	23.54	120	2,824.80 €
Indirect cost					**16,650.60 €**
TOTAL BUDGET					**18,494.97 €**

The direct costs of a project are those that can be directly attributed to the execution of the project. These costs are specifically related to the resources and activities necessary to carry out the project, such as personnel, materials, equipment and outsourced services.

On the other hand, indirect costs are those that cannot be directly attributed to a specific project but are related to the environment or infrastructure in which the project takes place. These costs are usually associated with shared resources and general project administration and management costs, such as administrative staff costs, facility rental, utilities, office supplies, etc.

C.4. Summary budget

Finally, Table C.8 shows a summary from Table C.7.

Table C.8 Summary budget estimation

Concept	Code	Description	Cost
Tools	T	Tooling required for the project	248.03 €
Materials	M	Materials required for the project	1,596.34 €
Direct cost			**1,844.37 €**
Task A	01	Study time	3,464.68 €
Task B	02	Engineering time	8,239.00 €
Task C	03	Manufacturing time	2,122.12 €
Task D	02	Engineering time	2,824.80 €
Indirect cost			**16,650.60 €**
TOTAL BUDGET			**18,494.97 €**

The estimated budget of this project has been **18,494.97 €** (eighteen thousand four hundred and ninety-four point ninety-seven euros). State taxes, 21% VAT, are not included.

C.5. Conclusions

The conclusion is that the indirect costs, mainly engineering costs, represent 90% of the budget, see Figure C.1.

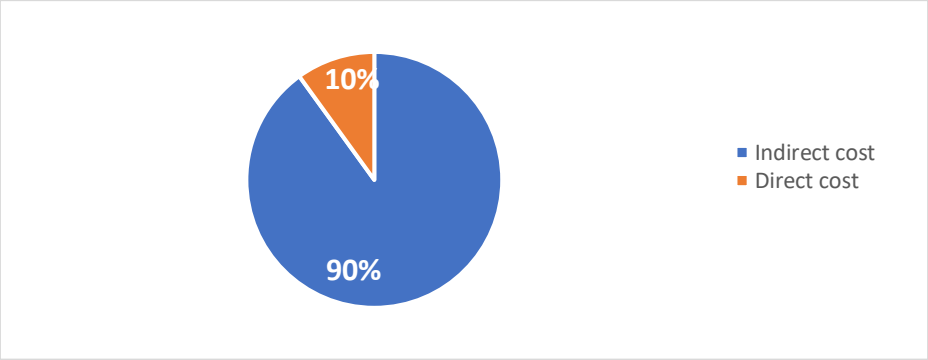

Fig. C.1 Percentage of type of expenditure budget

In order to make the Archer App more economically competitive, the focus should be on optimising development time, bug fixes and system improvements.

ANNEX D: HAVERSINE DISTANCE

This annex details the development of the formula used in the first phase of the project, which was later replaced by the Python *Geodesic* library.

The haversine is a mathematical formula used to calculate the distance between two points on a sphere, such as the Earth. This formula takes into account the curvature of the Earth's surface and is especially useful for calculating aerial distances between two nearby points.

Given two points: $P(\varphi_1,\ \lambda_1);\ Q(\varphi_2,\ \lambda_2)$

Where φ is the latitude and λ is the longitude in radians for each point.

The relation between two points and the haversine is:

$$hav(\theta) = hav(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot hav(\ \lambda_2 - \ \lambda_1) \quad \textbf{(D.1)}$$

The central angle θ between two points on a sphere is:

$$\theta = \frac{d}{r} \quad \textbf{(D.2)}$$

Where d is the distance between two points along the great circle of the sphere and r is the sphere's radius.

The haversine is defined mathematically as:

$$hav(\theta) = \frac{versin(\theta)}{2} = sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos\,(\theta)}{2} \quad \textbf{(D.3)}$$

$$\cos(\theta) = 1 - 2 \cdot hav(\theta) = 1 - 2 \cdot sin^2\left(\frac{\theta}{2}\right) \quad \textbf{(D.4)}$$

And the archaversine:

$$archaversine(\theta) = 2 \cdot \arcsin\left(\sqrt{\theta}\right) = arcos(1 - 2 \cdot \theta) \quad \textbf{(D.5)}$$

Applying (D.5) to extract d from θ:

$$2 \cdot \arcsin\left(\sqrt{h}\right) = sin^2\left(\frac{2 \cdot \arcsin\left(\sqrt{\theta}\right)}{2}\right) = \left(\sqrt{\theta}\right)^2 = \left(\frac{d}{r}\right) \quad \textbf{(D.6)}$$

Where $h = hav(\theta)$

$$d = 2r \arcsin\left(\sqrt{h}\right) \quad \textbf{(D.7)}$$

Developing (D.7) by applying (D.1), (D.3) and (D.4):

$$d = 2r \arcsin\left(\sqrt{hav(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot hav(\lambda_2 - \lambda_1)}\right) \quad \textbf{(D.8)}$$

$$d = 2r \arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\left(\varphi_1\right) \cdot \cos\left(\varphi_2\right) \cdot sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \quad \textbf{(D.9)}$$

The final the expression used to calculate distance between tow coordinates is given in Equation (D.9).

Where *r* is the Earth radius, 6356.752 km at the poles and 6378.137 km at the Equator.

ANNEX E: CAMERA MOUNT DESIGN

When the UAV reaches the position indicated from the user application, if it is an electric tower, it will start to perform a yaw rotation to scan the area and detect the tower in question and then take the expected images.

To do so, it is necessary for the UAV's camera to focus on the horizon in its line of flight.

Currently, the UAV in question has a single mount that focuses the camera lens downwards, see Fig. E.1.

Fig. E.1 Original fixed-position camera mount

The first step was to remove the support and carry out a dimensional analysis in order to come up with a new design. Fig. E.2 shows a hand-drawn sketch of the original support.

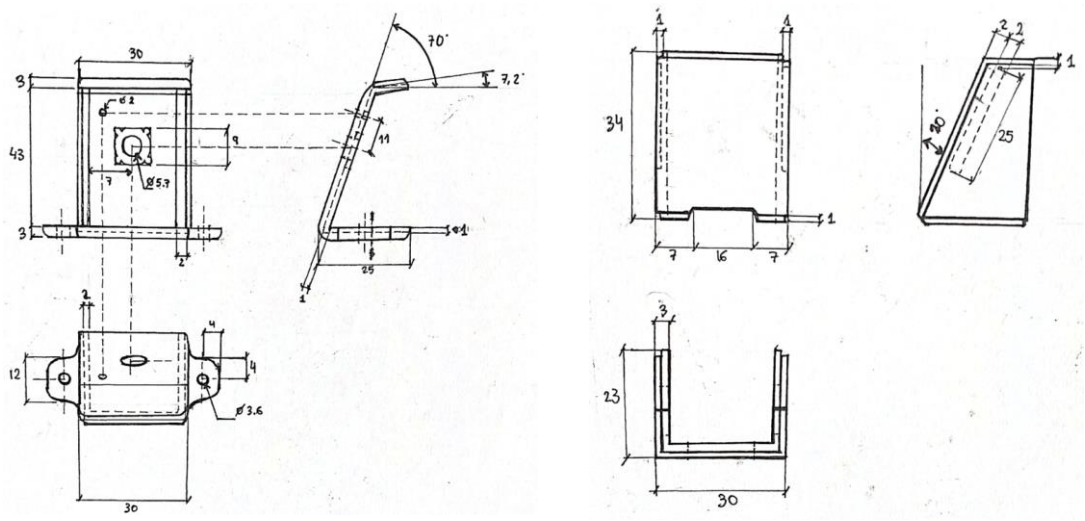

Fig. E.2 Both parts forming the camera container

The new design was made to satisfy the following points:

- ✓ Allowing the camera to focus to the horizon.
- ✓ Simplicity and ergonomics
- ✓ Reducing weight addition
- ✓ Rotational adjustment for different angles

It was decided to use the existing parts instead of redesigning it from scratch, as it would require more time and effort to adjust small tolerances.

For this purpose, the first prototype consisted of two parts, see Fig. E.3. One attached to the original mount and the other to the UAV structure. These pieces would be joined by a screw that would allow the desired rotation.

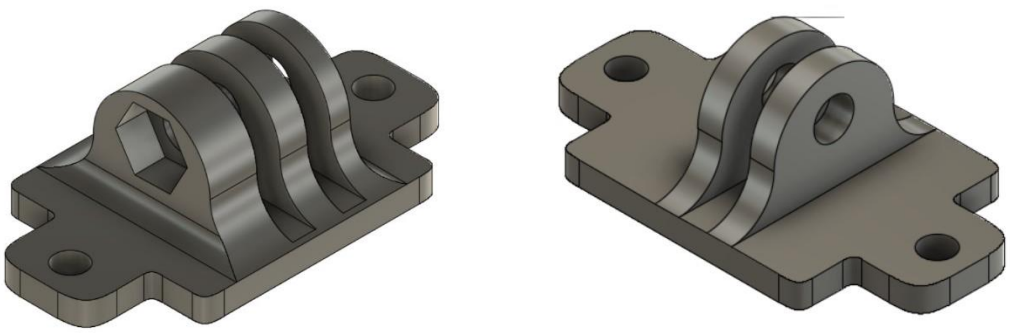

Fig. E.3 First prototype render using Fusion360 CAD tool

Once the design was finished, they were exported to a *.stl* file and printed on a 3D printer. The result is shown in Fig E.4.

Fig. E.4 Different views of the problem found (right)

When the first prototype was assembled, it was observed that there was insufficient clearance between the heads of the screws holding the base and the cross bolt joining the two parts. In addition, it was also observed that the rotation of both parts did not reach 90º as expected due to the limited travel because the bases of both parts were in contact.

For this purpose, both parts were redesigned again, ensuring a greater distance between the bases and the hole axis through which the cross bolt was threaded, see Fig. E.5 and Fig.E.6.

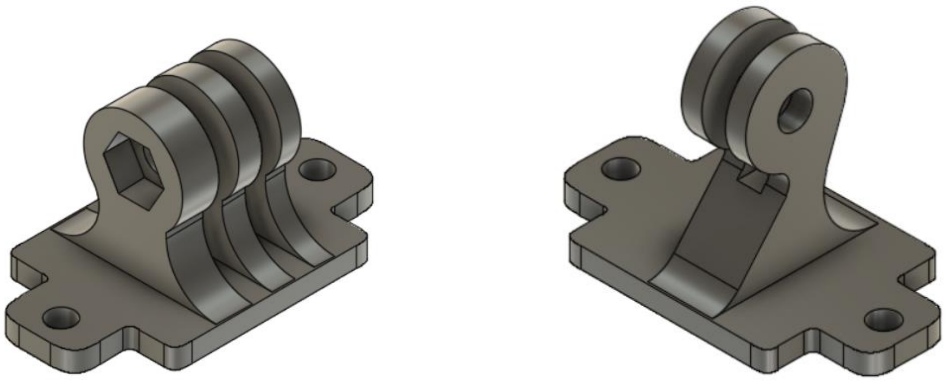

Fig. E.5 Second prototype render using Fusion360 CAD tool

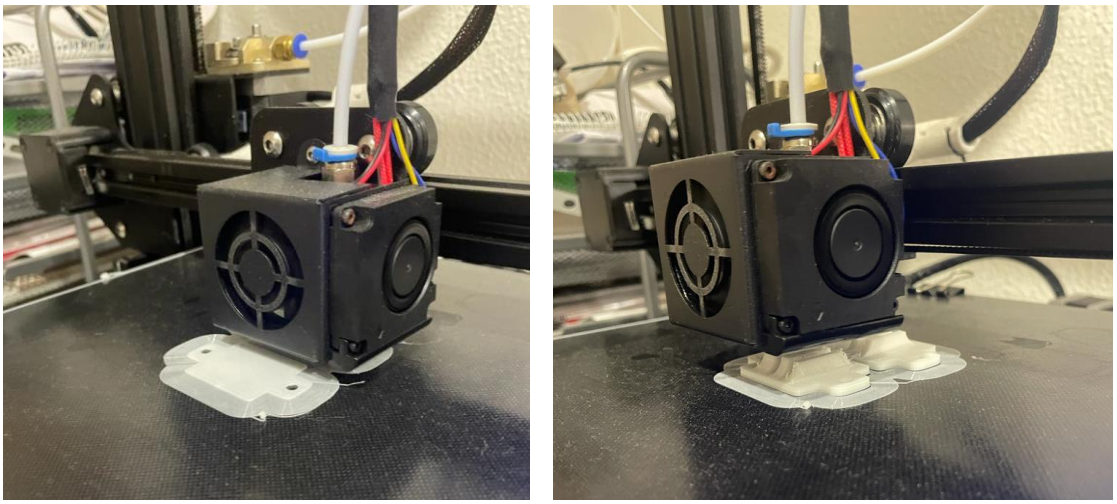

Fig. E.6 3D-Printing process

With the new design, it has been achieved the correction of the separation between both pieces in such a way that it allows the threading of the transverse screw that joins them, and it has been increased the rotation in both clockwise and counter clockwise directions to over 90º as expected.

Finally, a nut has been inserted and sealed into the structure with plastic welding adhesive to maintain its position when subjected to stress. The final result is shown in Fig. E.7.

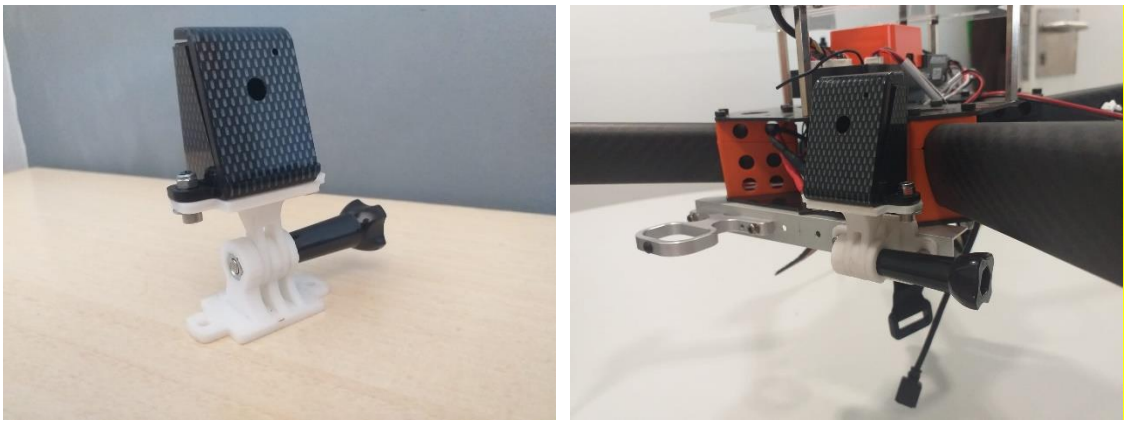

Fig. E.7 Final assembly (left) and mount to the UAV frame (right)

ANNEX F: CAMERA FOV

This annex is intended to justify the distance range established between the UAV and the tower in order to correctly detect the structure and obtain photographs that provide information on its condition.

According to the *Royal Decree 614/2001,* on minimum requirements for the protection of the health and safety of workers from electrical hazards, it is not possible to work within less than 3 m of any overhead power line, 5 m for voltages greater than 66 KV, and 7 m for voltages greater than 220 KV.

In general, it is considered that a distance less than 1 m between a conductive object and a high voltage line can be dangerous and cause electric arcs. However, this distance may vary depending on the voltage of the line and the humidity in the air. For example, under high humidity conditions, the air becomes more conductive, and the safety distance may be smaller.

As a UAV is used for the inspection, the minimum distance established at the *Royal Decree 614/2001* does not apply and therefore the smallest distance not affected by an electric arc will be taken. To find the upper limit of the distance, the characteristics of the camera shall be considered.

A picture taken by the RPi camera covers a rectangular area. The dimension of this area depends on the altitude (height) of the drone when taking the picture and on the field of view (FOV) of the camera [8], see Fig. F.1.

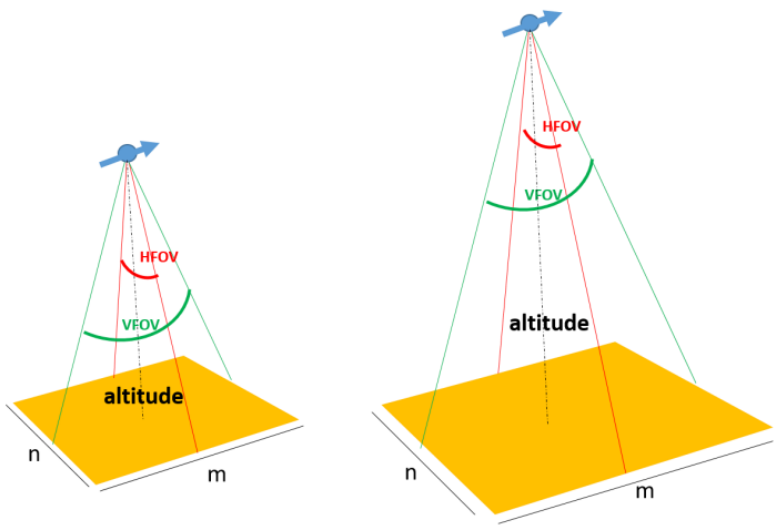

Fig. F.1 Area covered in a taken picture depending on the height, HFOV and VFOV [8]

The FOV is characterized by two values: the horizontal field of view (HFOV) and the vertical field of view (VFOV). In the case of the RPi camera, these values are shown in Table F.1.

Table F.1 Horizontal and vertical FOV of the V2.1 RPi camera

HFOV (α)	VFOV (β)
62.2º	48.8º

By trigonometry it can be obtained the relation between the distance (h) and the dimensions of the rectangle (m,n):

$$\tan(\alpha/2) = \frac{n/2}{h} \quad \textbf{(F.1)}$$

$$\tan(\beta/2) = \frac{m/2}{h} \quad \textbf{(F.2)}$$

$$n = 1.206 \cdot h \quad \textbf{(F.3)}$$

$$m = 0.907 \cdot h \quad \textbf{(F.4)}$$

Keeping in mind that, in general, a 400 kV power transmission tower in Spain can have an average height of around 40 meters and an average width of 8 meters at the base.

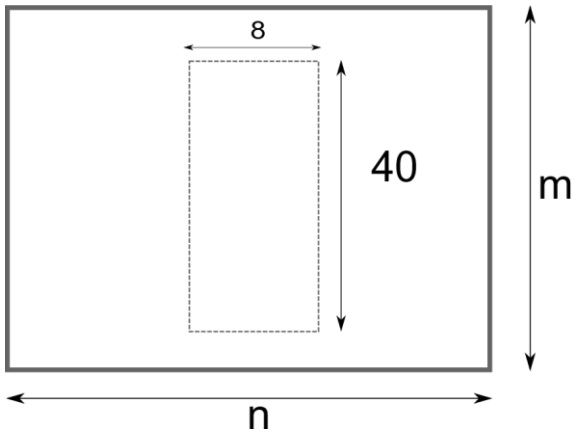

Fig. F.2 Approximate dimensions

By fixing the greatest distance from the tower, its height, it is possible to find the distance (h) between the UAV and WP to capture the whole structure:

$$h = \frac{40}{0.907} = 44.10\ m \approx 45\ m \quad \textbf{(F.5)}$$

Finally, the following table summarises for the range of distances, the dimensions of the rectangle that the camera should capture.

Table F.2 Range of dimensions that the camera will be able to capture.

Distance WP-Tower (m)	m (m)	n (m)
1	0.907	1.206
45	40.815	54.27

It should be noted that the user must adjust the distance value in the application according to the characteristics of the towers to be inspected and the desired detail of the structure to be captured.

ANNEX G: OBJECT-DETECTION TRAINING CODE

```
#clone YOLOv5 repo
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow
import torch
import os
from IPython.display import Image, clear_output
# to display images
print(f"Setup complete. Using torch {torch.__version__} ({t
orch.cuda.get_device_properties(0).name if torch.cuda.is_av
ailable() else 'CPU'})")

#Connect to Google Drive
from google.colab import drive
drive.mount('/content/drive')
from google.colab import drive
drive.mount('/content/drive')

#Unzip dataset
!unzip '/content/drive/MyDrive/TFG/TTPLA.v1i.yolov5pytorch'

#Model training
!python train.py --img 416 --batch 16 --epochs 10 --
data /content/yolov5/data.yaml --weights yolov5x.pt -cache

# Plot training metrics
%reload_ext tensorboard
%tensorboard --logdir runs

# Test of the model using the test images from the dataset
!python detect.py --
weights /content/yolov5/runs/train/exp/weights/best.pt --
img 416 --conf 0.1 --source /content/yolov5/test/images

# Show detected objects bound boxes on images
import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/runs/detect/exp
/*.jpg'):
    display(Image(filename=imageName))
    print("\n")

#Finally download file with network weights
from google.colab import files
files.download('/content/yolov5/runs/train/exp/weights/best
.pt')
```

ANNEX H: MODEL SELECTION

This annex aims to justify the choice of the YOLOv5 model that better fits with the system's requirements: YOLOv5s "Small", YOLOv5m "Medium", and YOLOv5x "XLarge".

Each model was trained in *Google Colab* using the pre-processed TTPLA dataset, 416x416 input images, 10 epochs and a batch of 16.

Google Colab is a free tool provided by Google that offers many features to facilitate data analysis and machine learning tasks, including free access to a GPU or TPU for faster computations, integration with Google Drive to easily store and access data, collaboration with other users in real-time, pre-installed libraries for data analysis, visualization, and machine learning such as *Pandas, Matplotlib*, *TensorFlow, Keras*, and more.

Table H.1 YOLOv5s training metrics results

Class	Instances	Precision	Recall	mAP50	Params (M)	GFLOPs @416
tower_lattice	138	0.86	0.758	0.777	7.026	15.8
tower_tuchoy	67	0.796	0.407	0.56	7.026	15.8
tower_wooden	107	0.654	0.402	0.434	7.026	15.8

Table H.2 YOLOv5m training metrics results

Class	Instances	Precision	Recall	mAP50	Params (M)	GFLOPs @416
tower_lattice	138	0.886	0.732	0.807	20.873	47.9
tower_tuchoy	67	0.597	0.463	0.5	20.873	47.9
tower_wooden	107	0.711	0.483	0.504	20.873	47.9

Table H.3 YOLOv5x training metrics results

Class	Instances	Precision	Recall	mAP50	Param (M)	GFLOPs @416
tower_lattice	138	0.934	0.768	0.827	86.207	203.9
tower_tuchoy	67	0.837	0.567	0.688	86.207	203.9
tower_wooden	107	0.886	0.561	0.634	86.207	203.9

First, it should be pointed out that in the project there are only two possibilities to run the model, one in the user application (a) and the other in the RPi on-board the UAV (b).

The table below lists the main advantages and disadvantages considered for each option.

Table H.4 Comparative table for both options

(a)		(b)	
Pros	Cons	Pros	Cons
✓ Best object detection capabilities ✓ Best image storage capacity ✓ Less time for area scanning ✓ Less UAV battery consumption (more flight range)	× Higher CPU cost × Large processing time × More dependency of WIFI signal	✓ Fastest image processing ✓ Less WIFI signal dependency	× Worst object detection capabilities × Less flight range × Less image storage capacity × More time needed for area scanning × Needs an RPi reconfiguration

As would be expected, YOLOv5s model requires fewer operations and therefore takes less time to process, while its average accuracy is the lowest. Comparing the results obtained with YOLOv5m, despite a threefold increase in computational cost and parameters, there is only a slight improvement in the precision and recall metrics.

It is logical to think that the choices to be considered should then be between YOLOv5s and YOLOv5x, one for its advantage in processing time and the other for its high accuracy. This is why YOLOv5m is discarded.

The pros that have been taken into consideration when running the model from the user application (a) are basically to take advantage of the high computational and memory capacity offered by an average computer compared to an RPi.

On the disadvantage, this implies a longer image processing time and therefore high computational costs. In addition, as the images must be constantly received from the UAV, a high dependency on the WIFI signal between the UAV and the user application is required.
By detecting objects more accurately, scanning time is also reduced and the RPi's processor is prevented from consuming more power and consequently the UAV's battery.

Alternatively, by running the model on the RPi (b), it would simply not be necessary to send images periodically to the user application as they would be processed directly on the UAV itself, thus reducing the dependency on the WIFI signal.

In view of Table H.4, it is finally decided to use the YOLOv5x model in the user application as it has less disadvantages than the other option.

ANNEX I: POSTER HOLDER SUPPORT DESIGN

The purpose of the last flight test was primarily to verify the detection of towers with the UAV in flight. To simulate the presence those structures, it was decided to hang images of towers onto the perimeter net of the *DroneLab*. The initial idea was to design a support that would allow hanging and removing the image from the network without using a ladder every time.

From the results obtained in Annex F, it was determined to use an image of 70x140 cm. With these dimensions, it is estimated that at a minimum distance of 1.5 m, the UAV camera should already frame the image.

The desired transmission tower image should be taken from the air, to keep a more realistic scenario. Thanks to *AGFOTODRONE*, it was possible to obtain UAV aerial high-quality pictures of some of the towers and print them in a very short time.

To accomplish this, permission had to be requested from the competent authorities about 6 days before the flight, as the area where the towers were located was within the CTR and ATZ of Barcelona.

Once permission was authorised, several photographs were taken both from the air, see Fig, I.1.

Fig. I.1 UAV aerial images taken from different point-of-view.

The other reason for taking own pictures of the towers was the resolution of the image. The company recommended that for printing 70x140 sizes it was advisable to use images of about 10 Mb.

A Hasselblad L1D-20c camera with a resolution of 4k was used for this purpose, giving images of 5464x3640 (11 Mb).

Fig. I.2 Image printing process

Once the images were ready, the next step was printing. The company prints on two types of material, paper roll or textile. Due to the purpose for which the images would be used, was opted for the textile material, which, despite being slightly more expensive than the paper roll, allows it to be reused several times and to make holes in it to tie strings, if necessary, see Fig. I.2.

An important point to note is that the images cannot be exposed to water under any circumstances. This means that the tests can only be carried out indoors or outdoors on days with no chance of rain.

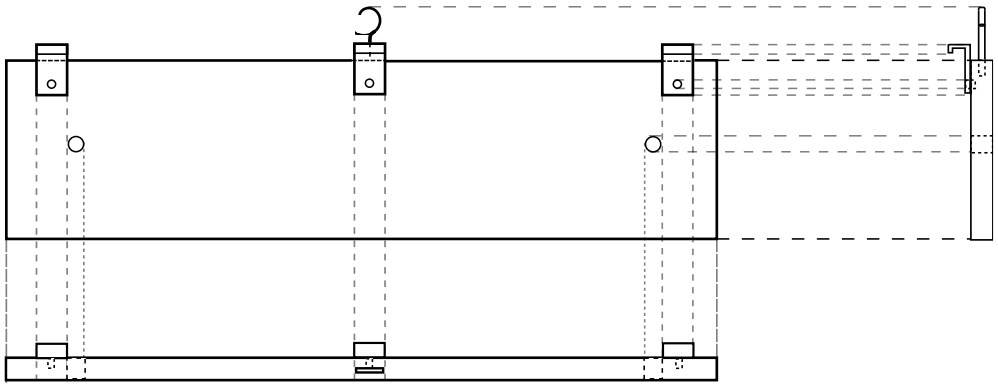

Fig. I.3 Initial sketch for the poster holder

Despite having 6 holes in the poster, it was necessary to design a support to hold the image at the UPC *DroneLab* net n a more solid way. For this purpose, a sketch has been made of how the component should look like, see Fig.I.3.

The item consists of a pine wood slat measuring 80x18x2 cm. Then three wall hooks are placed equidistant to distribute the weight between them and finally a threaded steel hook at the top to raise and lower the set with a hanger without using ladders. Finally, two holes will be used to tie the image to the support using rope or cable ties.

Fig. I.4 Drilling holes (left) and final result (right)

To get an approximation of the distance at which the model is able to detect the object in the image, one of the posters was hung and a test was performed using the laptop camera with a 1280x720 resolution.

For each 1.5 m increment, a clockwise rotation, a counter clockwise rotation, an ascent and a descent were made.

An estimate of the probability of detecting a tower as a function of distance is obtained for each distance increment by counting the total number of images taken with respect to only those in which a detection occurs, see Fig. I.5.

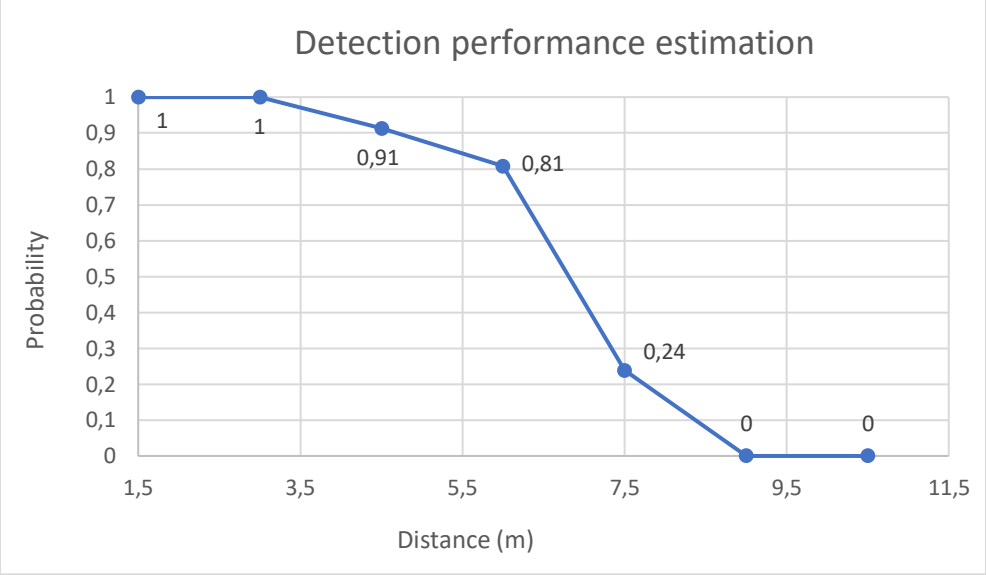

Fig. I.5 Detection probability as a function of distance

It is concluded that for the final flight test at UPC *DroneLab*, the UAV should fly at a distance range of 1.5 m to 7 m. It should be mentioned that the UAV camera has a higher resolution, so the maximum distance is likely to be somewhat greater.

Finally, although a lower resolution camera was used, it should be noted that the image taken by the UAV to create the poster was approximately 12 m. This means that in a test with real towers, the range would be approximately 13.5 m to 19 m or more.

ANNEX J: TELEMETRY DATA RESULTS

This annex is intended to show in more detail the results of the last successful simulation before the flight test at the UPC *DroneLab*.

Fig. J.1 Tower inspection flight path (upper) and RTL slope (lower)

Fig. J.1 shows graphically the flight simulation path performed. The vehicle is armed and takes off up to 5 m.

It starts the flight at the first point where there is DLT-1 tower and performs a clockwise rotation at 10º/s in 32° increments. At each increment it takes an image that is analysed by the machine learning model in the user application.

Once the tower has been detected, it proceeds to the next point. The next one belongs to tower DLT-2, the same procedure as in DLT-1 is carried out. To avoid

crossing the imaginary power line, the UAV must fly towards waypoints 2.1 and 2.2 to make the turn and face the last tower, DT-3.

Finally, on arrival at DT-3, it performs the same scan of the area and when the tower is detected, the UAV returns along the same path it has travelled. When it reaches the first point (DT-1), it switches to RTL mode and makes a vertical-descent landing.

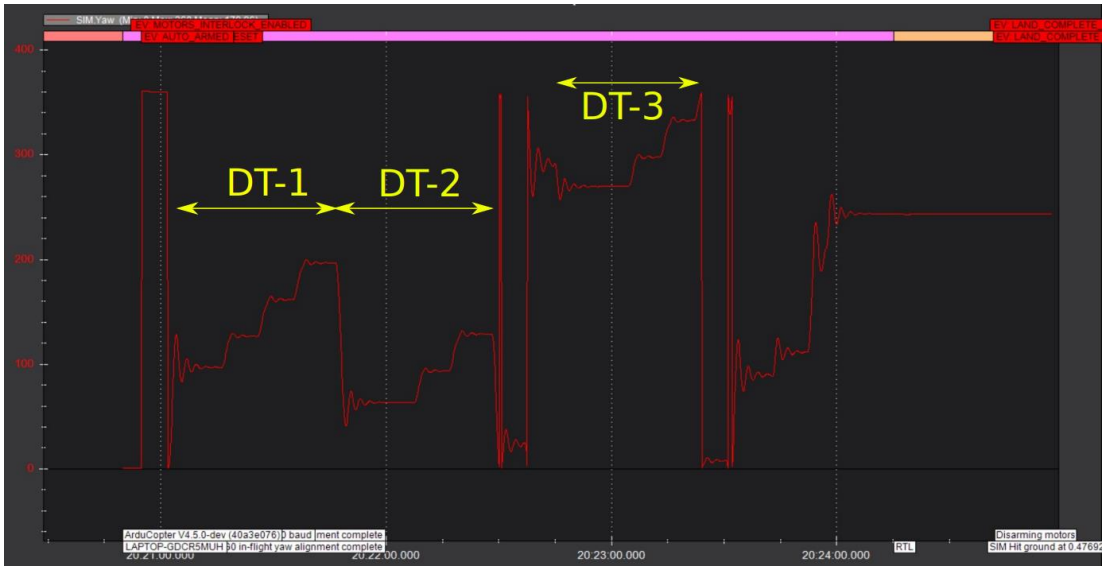

Fig. J.2 Yaw angle increase to scan each tower

Fig. J.3 Height above ground, being around 4.93 m