Aerial Drones Missions With 5G And Starlink Support

University of Aveiro

António Almeida, Francisco Ribeiro, Guilherme Vieira, Miguel Vila, Tiago Rodrigues

Aerial Drones Missions With 5G And Starlink Support

Department of Electronics, Telecommunications and Informatics

University of Aveiro

António Almeida, 108250, ant.almeida@ua.pt
Francisco Ribeiro, 107993, francisco.ribeiro@ua.pt
Guilherme Vieira, 108671, guilhermevieira@ua.pt
Miguel Vila, 107276, miguelovila@ua.pt
Tiago Rodrigues, 108324, tiago.m.rodrigues@ua.pt

08-06-2024

Abstract

Unmanned Aerial Vehicle (UAV) are becoming increasingly popular and affordable, making them ideal for various complex tasks such as carrying small loads, scouting areas, and reaching difficult-to-access locations. Leveraging these capabilities, UAV can be invaluable during disaster response scenarios.

This project equips a UAV with object detection and avoidance capabilities using a depth sensor camera and a command framework. Additionally, the project enables the UAV to follow a user via coordinates provided by a smartphone application.

Furthermore, the UAV serves as a mobile 5G cellular network base station, ensuring constant connectivity. The UAV is also connected to Starlink, providing reliable and high-speed internet access. Lastly, we created a live stream service, which allows the camera to transmit its live feed to a client on the internet.

The integration of these technologies enhances UAV functionality and reliability in critical situations, like disaster scenarios.

Acknowledgment

We are grateful to everyone who contributed to making this project a reality. We would like to thank our supervisors Susana Sargento, Pedro Rito and Joaquim Ramos for their immense support and guidance throughout this project, to our Projeto em Engenharia de Computadores e Informática (PECI) professors for always giving us advice on how to improve our work.

Special thanks to Pedro Valente for his incredible help of everything related to communications in our project, to Rodrigo Rosmaninho, without him, we would spend so much time figuring out how to deal with kernel issues, to Marco Mendes for helping us with the computer vision. Lastly, we would like to thank Diogo Correia, our Drone's guy, the one who helped with all mission related work.

To everyone who somehow contributed to the success of the project, our deepest thanks!

Contents

Chapter 1

Introduction

In recent years, UAVs, commonly known as drones, have seen significant advancements and increased affordability, positioning them as valuable tools for various complex tasks. From carrying small loads to scouting areas and accessing hard-to-reach locations, UAVs offer versatile solutions. This project leverages UAVs capabilities to enhance disaster response scenarios by integrating advanced technologies such as object detection, avoidance algorithms, and 5G connectivity.

This project seeks to harness the capabilities of UAVs to enhance disaster response scenarios by integrating technologies such as obstacles detection, advanced obstacles avoidance algorithms, and 5G connectivity. The primary objective is to give the UAVs the ability to dynamically adjust their mission paths in response to obstacles and to create adaptable mission profiles that allow the UAVs to follow individuals as defined.

Moreover, this project aims to establish a private 5G network with internet access via Starlink, providing a robust communication infrastructure. An additional goal is the development of a live feed streaming service, enabling real-time video transmission from the UAVs to remote operators.

In this paper, we will detail the integration and implementation of technologies such as Intel RealSense for depth sensing and object detection, Starlink for satellite-based internet connectivity, and 5G for high-speed communication. We will demonstrate how these technologies bring this project to life, showcasing the potential of UAVs in enhancing operational efficiency in disaster response scenarios.

Chapter 2

State of the Art

2.1 Previous Work

In this section, we will showcase all the previous work that was used in our project. We will begin with the existing research on drone control and then move on to the application used to making available a user's geolocation.

2.1.1 A mission planning framework for fleets of connected drones

On their master's thesis[2] Margarida et al. introduced a drone control framework and mission planning solution made specifically for fleets of aerial drones. Most public, non-commercial solutions offer limited functionality for mission planning, typically consisting of a linear mission flow for a predetermined aerial drone, where actions taken at each step are chosen from a non-extensible set. This framework overcomes those limitations and offers flexible mission planning, with non-linear missions that can change the course of a drone at any time during its sequence of actions while being able to request data from its own sensors and reacting to that data.

All of this was achieved using a groundstation that is responsible for the control and management of every drone. For the mission planning it was developed a Domain-Specific Language (DSL) built on top of the Apache Groovy language, allowing for logical conditional plans and providing user-friendly methods to declare common actions such as taking off, moving, turning, retrieving sensor data, landing, and returning to the home position.

2.1.2 VRU Safety App

On their master's thesis[7], Pedro et al. implemented a mobile application to share VRU Awareness Messaging (VAM) messages from a phone and how to prevent potential accidents between vehicles and vulnerable road users. This

system was based on the computation of risk zones and prediction of collision points between VRUs and vehicles.

Considering that we needed to get a person's geographical coordinates, we decided to use the application already developed on this project, in order to get information about a person in real time.

2.2 Related Work

In this section, we will review some related work that could be beneficial to our project. We will begin by addressing all the existing research regarding networking, followed by a discussion on the mission and drone-related studies.

2.2.1 Open5GS

Open5GS is an open-source project that provides a flexible and comprehensive implementation of the core network components required for 5G and LTE networks. Given its open-source nature, is an accessible way to experiment and understand the internals of 5G network operations.

Having a modular architecture, it allows for deep adaptation to specific use cases. This modularity lets us substitute specific network functions with alternative implementations, for example, we might integrate Open5GS with a custom Radio Access Network (RAN) to investigate how different radio technologies affect overall network performance.

2.2.2 srsRAN

A RAN is a network function that connects individual devices to other parts of a network through a radio link. srsRAN is an open-source software suite that provides a complete implementation of the RAN. The suite includes a full stack of components necessary for both the User Equipment (UE) and the gNodeB/eNodeB (the base station for LTE/5G), enabling the creation of end-to-end communication systems.

srsRAN is also highly compatible with a range of Universal Software Radio Peripheral (USRP) hardware, which allows the physical layer of the network to be adjusted dynamically (critical for studies involving varying radio frequencies and technologies). It is also compatible with many Commercial Off The Shelf (COTS) devices such as our testing device: the Pixel 6a smartphone. [5]

2.2.3 Obstacle Detection and Avoidance Algorithms

UAVs require sophisticated obstacle detection and avoidance systems to navigate safely and effectively in various environments. There are three approaches that dominate this field: reactive obstacle avoidance, vision-based obstacle detection and neural network based.

Reactive Approach

This approach allow for the Rotorcraft UAV (RUAV) to navigate point-to-point trajectories efficiently. The paper Reactive obstacle avoidance for RUAV [3] by Stefan et al. presents a robust approach employing a goal-directed 3D reactive obstacle avoidance algorithm tailored for RUAVs. This method utilizes a cylindrical safety volume projected ahead of the UAV within a 3D occupancy map to detect potential collisions.

The algorithm's key innovation lies in its expanding elliptical search strategy, which identifies collision-free waypoints by checking for occupied voxels using an efficient search. This search approximates the safety volume with spheres, ensuring rapid identification of escape points within 100 ms, even in environments cluttered with up to 20,000 occupied voxels. Such performance metrics underscore the algorithm's suitability for real-time applications in complex terrains.

Vision-Based Approach

Monocular vision-based systems offer another viable approach for UAV obstacle detection and avoidance. These systems leverage a single camera to process visual data, predicting collisions and calculating optimal avoidance paths. The system presented in the Abdulla et al. paper [1] emphasizes real-time image processing techniques such as edge detection, optical flow, and depth estimation from monocular cues to detect obstacles effectively.

Upon detecting potential obstacles, the algorithm dynamically adjusts the UAV's path, taking into account the vehicle's dynamics and sensor inputs to avoid collisions while maintaining the mission objectives. This approach benefits from the simplicity and cost-effectiveness of monocular vision systems, though it faces challenges related to depth perception and processing latency in rapidly changing environments.

Neural Networks Approach

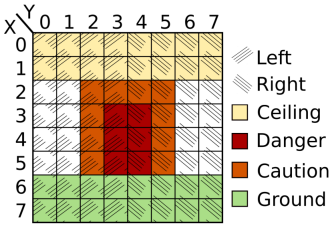
The paper Robust and efficient depth-based obstacles avoidance for autonomous miniaturized UAVs by Hanna et al. [4] demonstrated high-speed flight in complex environments using stereo cameras and neural networks trained on synthetic data. This approach achieved impressive results in terms of speed and obstacle avoidance, but the complexity is unsuitable for our project due to high memory and computational demands.

In terms of specific implementations, the document highlights a convolutional neural network deployed on-board a nano-UAV for autonomous navigation. This method shows effectiveness with both static and dynamic obstacles, but struggles in unfamiliar environments not present in the training dataset. Additionally, the integration of a model-free decision tree for obstacle avoidance

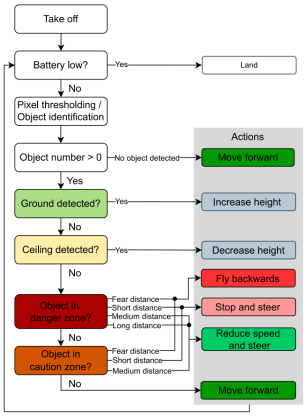
in nano-UAVs demonstrates the potential for real-time, on-board processing without relying on extensive off-board resources.

The most relevant part of this paper is the description of the choice algorithm. The algorithm involves dividing the camera view into two sides (Left and Right) 2.1a, with each side containing three zones (Danger, Caution, Normal) and four different distance categories (Fear, Short, Medium, Long). Depending on the combination of side, zone, and distance, different actions are triggered 2.1b. For instance, if an object is detected in the Danger zone at Fear distance, the UAV will fly backward. Conversely, if the object is in the Caution zone at Medium distance, the UAV will slow down and steer.

Figure 2.1: Algorithm Explained



(a) Camera View of Algorithm



(b) Decision Tree of the Algorithm

Chapter 3

Conceptual modeling

3.1 Functional Requirements

The features or functionalities that the system has to have in order to enable the user to achieve their goals are listed in the functional requirements list below. The needs were split up into several primary parts, including Person Following, Obstacle Avoidance, Network Backhaul, 5G, and Live Feed.

Obstacle Avoidance

Reference	Functional Requirements
FR-1	The UAV must be able to alter the course of the mission while in motion.
FR-2	The UAV must be able to halt the current movement.
FR-3	The system must have the ability to get the distance to objects ahead of the UAV.

Table 3.1: Functional Requirements - Obstacle Avoidance

Network Backhaul

Reference	Functional Requirements
FR-4	The UAV must be able to connect to the Network Backhaul.
FR-5	To access the internet, the Network Backhaul must be connected to the Starlink satellite constellation.

Table 3.2: Functional Requirements - Network Backhaul

5G

Reference	Functional Requirements
FR-6	The UAV must broadcast a private 5G network.
FR-7	There must be a connection from the UAV to the 5G Core.

Table 3.3: Functional Requirements - 5G

Live Feed

Reference	Functional Requirements
FR-8	The UAV must provide a live camera feed.

Table 3.4: Functional Requirements - Live Feed

Follow Person

Reference	Functional Requirements
FR-9	The UAV must be able to follow a person, using their geolocation.

Table 3.5: Functional Requirements - Follow person

3.2 Non-Functional Requirements

Non-functional needs, or system qualities like performance, reliability, and documentation, are listed in the list below.

Performance and Reliability

Reference	Non-Functional Requirements
NFR-1	The obstacle detection and path finding algorithms must execute quickly enough to determine a new course prior to a collision.
NFR-2	Data transmission to the groundstation must be reliable to avoid unexpected obstacles.
NFR-3	The UAV must operate autonomously without user intervention.
NFR-4	The connection to the groundstation must not depend on internet access.

Table 3.6: Non Functional Requirements - Performance and Reliability

Documentation

Reference	Non-Functional Requirements
NFR-5	Clear and comprehensible documentation for the decision-making algorithms must be provided.
NFR-6	Detailed documentation on how to run each component and integrate them must be available.

Table 3.7: Non Functional Requirements - Documentation

3.3 Actors

The main users of the system are the emergency services, including the police, ambulance, and, most crucially, firefighters. Given that one of the system's goals is to provide a private 5G network within the UAV's coverage area, thereby enabling internet access in places without it, particularly in disaster situations there would be:

- **Operator:** The responsible for setting up the mission and activating scripts;
- **On site person:** The on site user who will have connectivity from the UAV and be followed by the same.

3.4 Use Cases

In a disaster scenario, such as an earthquake where communication infrastructure has collapsed, UAVs can play a main role in restoring connectivity and helping in rescue operations. UAVs equipped with private 5G network capabilities are deployed to the disaster area to quickly reestablish communication. These UAVs fly above the affected zone, creating a temporary network that allows emergency personnel to communicate, facilitating efficient coordination of rescue efforts.

The UAVs are programmed to autonomously navigate through the devastated landscape, using obstacle avoidance technologies to dodge around collapsed structures and buildings. They receive and follow coordinates of onsite personnel, while at the same time providing a live video feed to a central command center, which is critical for real-time assessment of the damage and strategic planning of efficient rescue operations.

Chapter 4

Architecture

In this chapter, we will talk about the architecture of our project. Our entire architecture is shown on Figure 4.1.

As shown on the figure, on top of the UAV there is an Accelerated Processing Unit (APU), that controls all flying aspects of the drone. It communicates through ITS-G5 with a Groundstation, which is the module responsible for handling all the mission requests, and sends them to the drone the commands that it has to execute.

The APU is connected through Ethernet to a Jetson, in order for the depth sensor, connected using USB, to send information needed to dodge to the APU. The Jetson is also connected to a camera in order to transmit the Live Feed in real time (although, in our case, this is the same equipment: an Intel RealSense D415). Still on the Jetson, it's here that we have all the communication modules: Wi-Fi (to connect to the network backhaul), a 5G UPF container and a 5G gNB.

Through this 5G gNB we can provide a private 5G network for the UE shown. This UE is constantly sending location data using VAMs to a MQTT broker, located on the Internet, using any available connection: our private 5G network, or a public 5G network, if that is still a possibility.

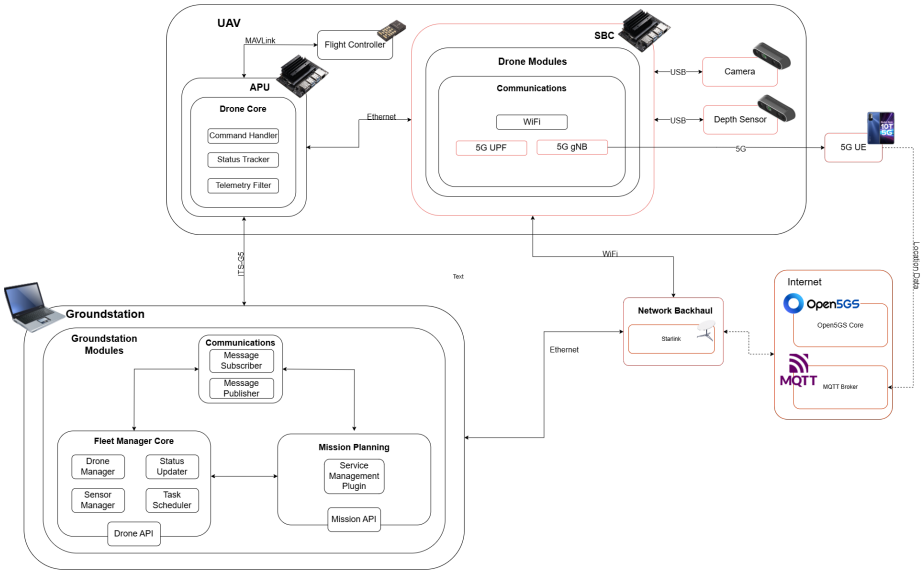


Figure 4.1: Architecture

4.1 UAV

4.1.1 Jetson Orin NX

The NVIDIA Jetson Orin NX is a compact AI platform with 1024 CUDA cores and 32 Tensor cores. Designed for edge devices, it excels in image processing, machine learning, and deep learning, making it ideal for real-time applications in robotics, drones, and autonomous systems.

The NVIDIA Jetson is the central module of this project because all of its functional requirements depend on it: is responsible for retrieving and processing the depth data from the Intel RealSense camera; is responsible for sending the dodge decision to the groundstation; is running the User Plane Function (UPF) and gNodeB to guarantee user's access to the internet; is live broadcasting via WebSockets the colored Intel Realsense's camera feed.

Intel RealSense

The Intel RealSense D415 is a compact depth camera with a 1920×1080 RGB sensor and stereo vision for precise 3D imaging. It delivers accurate distance measurements and high-resolution color images, ideal for robotics, augmented reality, and object tracking applications requiring real-time depth perception.

This camera is connected directly to the Jetson and captures depth values, which are then processed by our obstacle avoidance algorithm on the Jetson. Besides depth data, the RealSense camera also provides a color video feed for live streaming. This stream is transmitted over Wi-Fi via WebSockets.

TP-Link Archer T3U Plus

The TP-Link Archer T3U Plus is a high-gain USB Wi-Fi adapter that provides robust and stable wireless connectivity. Connected to the Jetson Orin NX via USB, it establishes the Wi-Fi connectivity between the drone and the backhaul network. This connection is responsible for transmitting the live video feed as well as data from the UEs connected to the broadcasted 5G network.

UPF

The UPF is a critical function in the 5G core network as it is responsible for routing user data packets between the gNodeB and external networks. The UPF is deployed within a Docker container on the Jetson Orin NX. This setup enables reduced latency, as the UPF is positioned close to the gNodeB.

USRP

The USRP B210, a software-defined radio device, is connected to the Jetson Orin NX via USB. This device is crucial in establishing a 5G communication link between the UE and the network.

4.1.2 APU

An APU is a type of processor that combines a CPU and a GPU on a single chip. This integration aims to improve efficiency and performance while reducing the power consumption and physical space needed for the components.

In our project, the APU is used as the responsible for the communications between the UAV and the groundstation, as it can communicate through ITS-G5. It's used to control the Drone movement depending on the mission command given by the groundstation. It's also connected to the Jetson, via Ethernet, receiving the avoidance algorithm values relaying the avoidance decision to the groundstation.

4.1.3 Flight Controller

A Flight Controller is a board coupled to the drone which allows it to perform movements. Commands may be sent to the Flight Controller (FC), either through a radio control or programmatically, which the FC then converts to controls in order to manage the motors. To maintain altitude and navigate safely, FCs usually integrate several onboard sensors, which have to be calibrated before flight. The FC needs to be connected to a Single-board Computer (SBC),

on which it will run software capable of sending the commands, to be programmatically controlled by SBC.

In our project, we are using the Pixhawk 4 as our FC, as it was the one used and worked on in all the previous work.

4.2 Groundstation

The Groundstation is the module responsible for managing all the UAVs. It can be a standard computer or an APU; in our project, we opted for the APU. Through the Groundstation, we can send missions or single commands to the UAV and retrieve all of its information, including coordinates, battery level, and sensor data.

To communicate with the drone, we used ITS-G5, since it had already been established in prior work. Despite its low bandwidth, ITS-G5 offers a wider range, which compensates for this limitation. The mission instructions and drone commands are sent to the UAV through Robot Operanting System (ROS).

4.3 Backhaul

The network backhaul is an essential element in our system, acting as the main link between the UAV and the public internet. This connection is vital for the real-time transmission of various data types, including live camera video feeds, and it provides internet connectivity for the 5G private network.

To achieve this, we use the Starlink satellite internet service, which provides a reliable high-speed internet connection even in remote areas. Starlink's low Earth orbit (LEO) satellites ensure that our UAV maintains a continuous and robust link to the internet, facilitating the exchange of data necessary for our operations.

4.4 Internet

4.4.1 5G Core

The 5G Core is the central component of a 5G network, responsible for providing key network functionalities such as authentication, session management, and mobility management. It represents a significant advancement over previous mobile network generations by introducing a service-based architecture (SBA) where network functions communicate through standardized interfaces, allowing for greater flexibility and scalability.

For our project, we have chosen to implement the 5G Core using Open5GS, an open-source project that provides a comprehensive implementation of the 5G Core network. Taking advantage of the flexibility of the 5G Core, we implemented the RAN using srsRAN. Furthermore, we adopted a disaggregated architecture for the User Plane Function (UPF), placing it on the UAV to ensure efficient data handling and reduced latency for our specific use case.

4.4.2 MQTT Broker

An MQTT broker is a key part of an MQTT network, facilitating efficient and reliable message transfer between devices. It manages topics and ensures that messages are delivered to the appropriate subscribers.

In our project, the MQTT broker is used to publish the user's geolocation. This enables the efficient transmission of location data to the UAV, facilitating the follow feature.

Chapter 5

Implementation

In this chapter, we will provide a detailed explanation of how we achieved our project objectives. We will begin by discussing the mission-related aspects, focusing on the development of our final dodge algorithm and the process for enabling the drone to follow a person.

Next, we will explore the communication components of our project. This section will start with an explanation of our Starlink implementation, followed by an in-depth description of how we established a 5G network. Finally, we will describe the implementation of the live feed feature, detailing the methods and technologies used to achieve real-time video transmission.

5.1 Mission

5.1.1 Algorithms to Dodge

We will go into great detail about the development process and application of the dodge algorithm in this part.

First steps

To make the UAV capable of dodging, we first needed to understand how to move it autonomously. We utilized Fleetmanager (subsection 2.1.1), which allowed us to control the drone as needed. After mastering the basics of its movement, we focused on equipping the drone with a camera capable of rapid image processing. For this, we employed an NVIDIA Jetson Orin NX (subsection 4.1.1) due to its high GPU performance, which enabled us to swiftly process images captured by the Intel RealSense D415 (section 4.1.1), a camera integrated with a depth sensor.

With the camera setup complete, our next step was to read its values. We established five different points for distance measurement. These distance readings were transmitted via ITS-G5 using ROS2 to the ground station, functioning

as sensor data.

Having the foundational elements in place, we explored various methods for making the drone dodge obstacles and interrupt its movement mid-mission. Initially, we experimented with pre-existing plugins in the framework, which allowed for continuous event monitoring without altering the main mission.

To implement our concept, we developed a custom plugin that received data from the Intel RealSense camera and attempted to send move commands to the drone. However, this approach resulted in race conditions, as the plugin's commands conflicted with those of the main mission. Despite our efforts to resolve this issue, we were unable to achieve a solution. Consequently, we discarded the plugin approach and began exploring alternative possibilities.

First Algorithm

After discussing various ideas, we decided to adopt a safer approach for the algorithm. Given the issues with interrupting movements mid-mission, we opted to create a mission using the DSL provided with Fleetmanager. This mission was designed so that the dodging algorithm operated entirely within the mission parameters.

Delving into the specifics of the mission formulation, we implemented a step-wise movement strategy. The UAV moves a predetermined distance, X meters, defined by the programmer. After each step, the drone checks if the distance from the depth sensor is less than the minimum safe distance. If it is, the UAV initiates the dodge algorithm.

The dodge algorithm follows a predefined set of instructions laid out in a flow chart (Figure 5.1). If the distance to an object is less than the minimum safe distance, meaning a collision is imminent in the next step, the drone stops, turns 90^{o} to the right, and checks if it can move forward. If it can, it proceeds, then turns 90^{o} to the left and checks the distance again. If the distance is now greater than the minimum safe distance, the drone continues moving forward. If there is still a risk of collision, it turns 90^{o} to the right again and repeats the process. If it encounters another obstacle after turning right, it performs a 180^{o} turn, checks if it can move forward, and, if so, raises a flag to avoid returning to the right, then proceeds forward. This cycle continues until the UAV either dodges the obstacle or finds itself in a narrow passage with walls on both sides and an obstacle in front. In this case, it turns back (rotates left 90^{o} after the right and left checks), checks if it can move forward, and, if possible, moves forward, restarting the cycle. There is an emergency fail-safe: if the cycle repeats five times without movement, the UAV executes a land command.

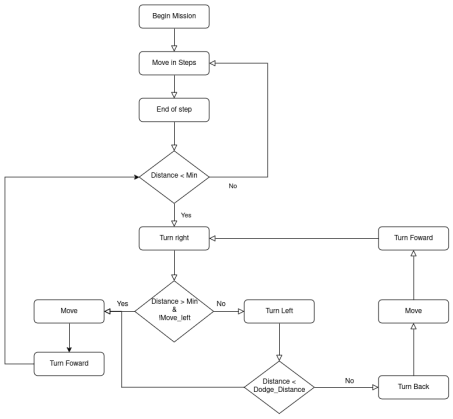


Figure 5.1: Decision Tree of our first algorithm

After testing this algorithm in simulation, we drew several conclusions. Firstly, the algorithm only considers three dodge directions: right, left, and back. This limits our dodging options to only three. Additionally, we observed that the drone moved very slowly throughout the mission due to the stepwise movement approach.

First Algorithm Improved

Taking these issues into account, we began exploring different algorithms. During this search, we discovered that it was possible to cancel a drone's current movement, allowing it to proceed to the next command of the mission. Additionally, we found that the DSL provided supported the use of threads.

With these discoveries, we created a new mission. Although we used the same dodging algorithm from the first mission, everything else was different. Leveraging the capability to use threads, we created two threads: one for handling the dodging and another for running the main mission. The dodging thread continuously monitored the need to dodge, and if necessary, sent a command to cancel the current movement and activated a flag to inform the main mission of the need to dodge. In the main mission thread, as soon as the movement was canceled, it would normally proceed to the next movement. However, before each movement started, it checked the flag for the need to dodge, and if activated, the dodging algorithm described earlier would initiate.

We conducted simulations to test this new approach, and everything worked smoothly. The drone could move faster while still dodging obstacles effectively. However, our current dodging algorithm was still not optimal (Figure 5.2). The algorithm only considered three directions—right, left, and back—when it might have been more efficient to include turns of less than 90^{o}.

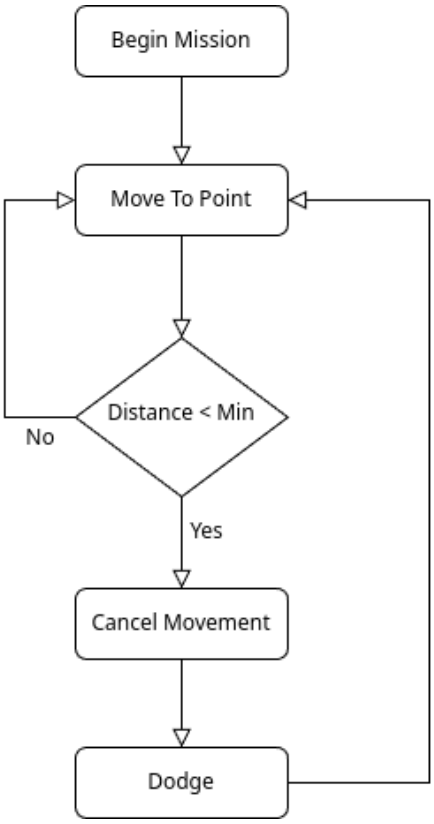


Figure 5.2: Decision Tree of the fluid movement algorithm

Second Algorithm

After being successful in creating a smooth way for the drone to move, we started again our search for an algorithm, we found some candidates, but most of couldn't be implemented with what we had at hands.

One of the candidates that we found and previously talked about was the paper Robust and efficient depth-based obstacles avoidance for autonomous miniaturized UAVs by Hanna et al. [4] even though at first this algorithm looked good, after a throughout read of the paper we found that the results were not that good, on the outside at a speed of 2m/s the drone collided 100% of the times, inside the results were better but for our work we needed an algorithm that worked outside and one that was more reliable, because of this we discarded this one from our options.

Implemented Algorithm

As we continued our search, we found another algorithm: the Escape Points Algorithm (EPA) [3]. As we did a deeper research on this paper, we found that it was exactly what we were looking for, the ability to find a way to dodge in any point of the drone's field of view, and we started to implement it into our work.

During the implementation, we learned that a parallel project using the same Fleetmanager had successfully added the ability to pause and resume missions, a feature that was previously unavailable. With this new capability, we could now make the drone dodge during any mission created. Consequently, we stopped using the provided DSL and began utilizing Python scripts instead.

The `pause` function stores the state at the time the mission is paused and disassociates the drone from the mission, allowing for a new mission to be sent.

The `resume` function works slightly differently, offering three types of resume options. The default, `last_position`, returns the drone to the position where the mission was paused before resuming. The second option, "complete_current," has the drone complete the command that was running at the time of the pause from any position. The final option, `start_next`, begins with the command immediately following the one that was active when the mission was paused.

The algorithm employs a circular mask of points over the Depth Mesh, with an image radius larger than that of the UAV. This circle checks if any point within its area is closer than the minimum distance required for dodging. If such a point is found, the circle must find a new position by adjusting its course, moving left and right until a suitable path is identified. To minimize search time, instead of examining each pixel of the image, the algorithm moves by a pre-calculated distance $d_{\max}$, which represents the maximum distance between two spheres for the optimal gap value:

$$d_{\max} = 2\sqrt{R_{\mathrm{SV}}V - \frac{V^2}{4}}$$

where R_{SV} represents the cylinder radius and V denotes the voxel size (the area not checked by the cylinder). In this project we found the optimum values to be: $R_{\mathrm{SV}} = 5$, V = 0.5m, resulting in value of 4.6. Converting this to pixels, we obtain $d_{\max} = 62$ pixels to move left and right.

A Python script is implemented on the UAV, running continuously to return an array of values based on dodging conditions. The script takes the depth value from the Intel RealSense camera as input. If a potential collision is detected, the script calculates the escape point and sends an array of three values to the UAV's APU via Ethernet, as a Sensor. These values include a flag indicating the need to alter the mission movement, the angle for dodging, and the distance to move. The distance to move corresponds to the minimum dodge distance,

positioning the UAV on the obstacle's side. The dodge angle is calculated by converting pixels to degrees, based on the Intel RealSense's horizontal pixel resolution of 640 and a view angle of 70 degrees:

$$DegreeForPixel = \frac{viewAngle}{HorisontalResolution} = \frac{70}{640} = \frac{7}{64}$$

Multiplying this by the number of pixels the circle moved from the center gives the angle for dodging.

```
    while distance < dodgeDistance.
        move center right
        if distance > minDistance
        then
            send flag,angle,movement
        move center left
        if distance > minDistance
        then
            send flag,-angle,movement
        if border of circunference outside the camera
           view
        then
            send stop flag
```

Listing 5.1: **Chosen Algorithm** : Escape points algorithm

On the APU side, we run a Python script that subscribes to the ROS topic to receive values from the algorithm script and begin to analyze them. Upon receiving the array of values, we immediately check the flag. If the flag is 1, the dodge sequence initiates; if it is 0, the mission proceeds as usual.

For the dodge sequence, we use ITS-G5 to send an HTTP POST request with a `pause` instruction to the Groundstation. We then dynamically create a new mission, now that the previous one has been paused. Using the angle and distance values, we build a mission that rotates the drone in the specified direction and moves it the required distance. Once this secondary mission is completed, we send the `resume` command with the `complete_current` option.

During the development of the dodging algorithm, we encountered several challenges. Initially, it was not possible to pause while the drone was resuming a mission. To address this, we added complexity to our script: we needed to create a new mission for the drone to complete the command that was paused. As soon as this mission started, we had to check for the need to dodge. However, our existing script couldn't handle this, so we began using threads. When one thread indicated the need to dodge, we killed all threads and stopped initiating new ones. Once the resuming mission started, we resumed accepting messages to check for dodging needs. Unfortunately, this approach was unsuccessful. However, a subsequent fix in the Fleetmanager simplified our script,

making what was previously impossible now possible. We could pause during the resume command, allowing us to revert to the old script and remove all the added complexity.

Besides this, we encountered another issue. From the beginning, we wanted the Jetson to handle everything, but unfortunately, it was not possible. We discovered that the communication between the Jetson and the Groundstation was not working as we expected it to: the Jetson and the Groundstation should be communicating through ITS-G5, but while we could ping each other from the other one and send HTTP requests back and forth, ROS2 packets were not being sent or received on either end. This is where the APU comes into play: it acts as the intermediary between the Groundstation and the drone. The APU manages all drone-related operations, while the Jetson handles tasks requiring more processing power.

5.1.2 Follow Person

Introduction

For the UAV to be able to follow a person, it first needs to know where that person is. In order to do this, we are using an application previously developed for another project, that sends VAM messages to a MQTT broker on the Internet. These messages are going to be received by the groundstation, connected to the same broker and subscribed to the same topic, in order to send the person's coordinates to the UAV as a sensor in real time.

Configuration

First of all, the person that wants to be followed needs to install the VRU Safety App (the application developed by another project) on their phone. Inside the app, the person needs to go to the settings and enable the "Publish Decoded VAM" option. This will allow the app to publish decoded VAM messages to the `vam_decoded` topic, instead of publishing encoded ones to the `vam_encoded` topic.

Once the person starts publishing VAM messages through the application, it is shown a Station ID on the display. The person then needs to take note of this ID, as we will use it in the next step of the configuration.

On the groundstation, we need to go to the `person_location_config.yml` file and configure the `droneId` associated with this sensor. We also need to configure the MQTT broker parameters - the host, the port if it is different from the default (1183), the username and the password if needed - and the `stationId`. This `stationId` is the same Station ID as previously stated, so it needs to be filled in with the Station ID of the person you want the UAV to follow.

You can find an example of the `person_location_config.yml` file on Listing 5.2.

Listing 5.2: person_location_config.yml file example

```
droneId: drone_PECI
sensorTopic: /sensors
telemTopic: /telem
rate: 200
mqttHost: mqtt.host.domain.tld
#mqttPort: 1183
#mqttUsername: Username Here
#mqttPassword: Password Here
stationId: Station ID here
```

Method used

The person location sensor is a script in Python that uses MQTT to connect to the broker where the VAM messages are being published, and ROS2 to publish this info as a sensor to the groundstation container, to be able to be read by the missions that depend on this data.

The script starts by reading the configuration file. If the `droneId` or the `mqttHost` parameters are not defined on this file, the script will throw an error and exit, because these are mandatory fields. After that, the script will launch a thread to subscribe to the UAV telemetry data, received through ROS2, to be able to constantly receive information about the drone status. This is important because we do not want to be publishing location data for a drone that doesn't exist on our fleet. Simultaneously, the script will launch two other threads: one as an MQTT subscriber and another as a ROS2 publisher. These threads serve the purpose already described above: the MQTT subscriber will be listening to VAM messages published on the MQTT broker, saving the person's latitude and longitude to a shared variable. Meanwhile, the ROS2 publisher will read that same variable and send it to the groundstation container, as a sensor.

Now that the drone recognizes the location as a sensor, we created a mission that reads those values and sends a move command to that place, the next move is only sent when the UAV is finished with the previous one.

One important thing to note is that, when the script is started, if the person is not publishing VAM messages yet, the published coordinates will reflect the UAV coordinates, replacing them with the person's coordinates the moment they start publishing the location data.

5.2 Starlink

Introduction

We used the Starlink satellite internet service as the network backhaul of our project. Our main goal is to provide internet access for our UAV. Since our Jetson lacks a built-in Wi-Fi card, we used a TP-Link Archer T3U Plus USB Wi-Fi adapter to enable Wi-Fi capabilities on our UAV.

Configuration

Firstly, we had to install the driver for the TP-Link adapter, Realtek RTL88x2B. For that, we had to clone the corresponding repository, build the driver and install it. After restarting the Jetson, we now had wireless connectivity. Now was the time to connect to the actual Starlink network.

The Starlink satellite has two modes: a default mode using the Starlink's integrated router, and a bypass mode, which allows for the configuration of a personal router. For our project, we decided to use the default mode. After setting up the Starlink Satellite on the ground, we had to access the Starlink application to configure it. This involves setting up the SSID and Password for the private network being transmitted by the Starlink Access Point.

After that, on the Jetson we used `nmcli` to list the available Wi-Fi networks and connect to the network using the previously configured SSID and Password. Not only do we use the Wi-Fi to provide internet access to the UAV, but we also have the 5G Core directly connected to the router via Ethernet.

5.3 5G

This section provides a detailed overview of the implementation of a 5G network using a combination of Open5GS for the 5G Core and srsRAN for the RAN functions.

5G Core

The 5G Core is implemented using Open5GS, an open-source project that provides a full suite of 5G Core network functionalities.

The deployment was conducted on a MINISFORUM EM680 NUC using its Docker-based installation [6]. The core network is composed of several key network functions, each running as individual Docker containers. This modular approach allows for flexibility and scalability.

Taking advantage of this modularity, we opted to deploy the UPF in a disaggregated manner to enhance network performance and scalability. We adjusted

the IP addresses in the `.env` files to ensure the communication between the Core and the UPF.

We took steps to register the UE devices in the Open5GS database manually. This manual registration ensures that each UE can connect to the network and access the services provided by the 5G core.

In a 5G network, each UE must authenticate itself to gain access to the network services. So we registered register our testing device in the Open5GS database. This involved creating an entry for the UE, specifying its International Mobile Subscriber Identity (IMSI), keys, and other necessary details to ensure it was recognized and authenticated by the network.

Radio Access Network (RAN)

The RAN for this project is implemented using srsRAN, which is deployed on a NVIDIA Jetson Orin. The gNodeB is responsible for managing radio communications with the UE and utilizes the USRP B210 as the RF front-end.

The gNodeB configuration is crucial for establishing radio connections and ensuring efficient data transmission. To provide a clear example of how we configured the gNodeB, consider the example `gnb.yml` file, shown on Listing 5.3, where we specify the settings for the srsRAN deployment, including parameters for the RF front-end and cell configuration such as the gains of the antennas and the channel bandwidth.

In our implementation, we opted for a straightforward configuration, employing a 20 MHz channel bandwidth with a single pair of antennas. This choice represents a modest setup designed to balance performance with resource constraints.

```
amf:
  addr: AMF_IP
  bind_addr: SRS_GNB_IP
ru_sdr:
  device_driver: uhd # The RF driver name.
  device_args: type=b200, (...)
  clock: # Specify the clock source used by the RF.
  sync:
  srate: 23.04 # Sample rate according to bandwidth.
  otw_format: sc12
  tx_gain: 80 # Transmit gain of the RF.
  rx_gain: 40 # Receive gain of the RF.
cell_cfg:
```

```
14    dl_arfcn: 632628
15    band: 78 # The NR band.
16    channel_bandwidth_MHz: 20 # Bandwith in MHz.
17    common_scs: 30 # Subcarrier spacing in kHz.
18    plmn: "PLMN" # PLMN broadcasted by the gNB.
19    tac: 1 # Tracking area code.
20    pci: 1 # Physical cell ID.
21 #    nof_antennas_ul: 2 # MIMO (2x2) to use all the
22 #    nof_antennas_dl: 2 # antennas of the USRP.
23    pcg_p_nr_fr1: 23
```

Listing 5.3: gnb.yml file example

This is a simple yet functional deployment of a gNodeB using srsRAN which is sufficient for demonstrating the core functionalities of our network.

User Plane Function (UPF)

The UPF is also running on the NVIDIA Jetson Orin, and in this project it is mainly responsible for routing and forwarding data packets between the UE and external networks.

For internet connectivity, the UPF is linked to a TP-Link Archer T3U Plus USB Wi-Fi adapter, which connects to a Starlink access point. This setup ensures reliable, high-speed internet access.

5.4 Live Feed

First steps

Since we were already using the depth frames from the Intel RealSense, we decided to also use the color frames to establish a live feed from the UAV camera. Initially, we retrieved the color frames from the camera, connected to our computer, and displayed them using OpenCV. Following that, we chose to use WebSockets, so we set up a socket server on the Jetson Orin NX to transmit the color frames to all connected clients. Each client then receives the frames and can display them with tools like OpenCV.

Implementing

In our first attempt, we tried to set up the live stream script on a different file from the avoidance algorithm, during the integration process, we encountered an issue where the dodge script and the live feed script could not simultaneously access the Intel RealSense frames, even if they were from different feeds internally. To solve this access problem, we had to merge the two scripts together into a single script, grabbing both the depth and color frames in the same file. As we still needed them to run concurrently, we had to use threads in order to

segregate the two main functions: one for getting the depth frame and calculating everything necessary for the dodge script, and the other getting the color frame, processing it, and sending the data to all the clients connected to the socket server. On the client-side, to receive the frames, the client must connect to the socket server using UAV's IP Address.

5.5 Jetson Orin NX

Choosing the hardware

For this project, we chose to use a Jetson Orin NX due to its excellent performance in image processing. We opted for this because initially, when thinking about the architecture, we thought about using You Only Look Once (YOLO) (a real-time object detection algorithm that would take advantage of the CUDA cores present on the Jetson) for our object detection and avoidance service. Furthermore, we wanted to have all the UAV modules to run on a single device, so that we could have less weight on top of the drone, as well as reducing the moving parts.

Problems found

As we stepped into actually implementing all our modules on the drone, we were having a lot of issues with the CUDA libraries on the Jetson, and we couldn't get YOLO running on it. While some of us continued trying to solve this issue, others started looking for alternatives.

One of the alternatives that we found was using the Intel RealSense as a depth sensor. That way, we would have the information about the distances to the object without relying on YOLO.

Implementing

Now we no longer depended on YOLO, so we would not need the image processing capabilities of the Jetson anymore, but changing the hardware at this point was not feasible, because other work had already been done on it.

To try to aggregate all the modules on top of the UAV, we had to install drivers for both the TP-Link Wi-Fi adapter and the Intel RealSense. In addition, we also had to rebuild the Jetson kernel with Stream Control Transmission Protocol (SCTP) support, because the 5G gNB communicates using this protocol to register itself on the 5G Core. This protocol is also needed to register the UE on the 5G Core, and other signaling messages.

Chapter 6

Results

In this section, we will showcase some of the results we obtained to better illustrate the functionalities developed throughout this project. These results highlight the effectiveness and practicality of our implemented features.

6.1 Mission

6.1.1 Algorithm

While developing the algorithm, we also conducted tests to verify its functionality and made adjustments to ensure it would not collide with any objects.

We used the simulator to test different scenarios, including dodging a single obstacle in one movement, evading multiple obstacles in different movements, and dodging multiple times within the same movement.

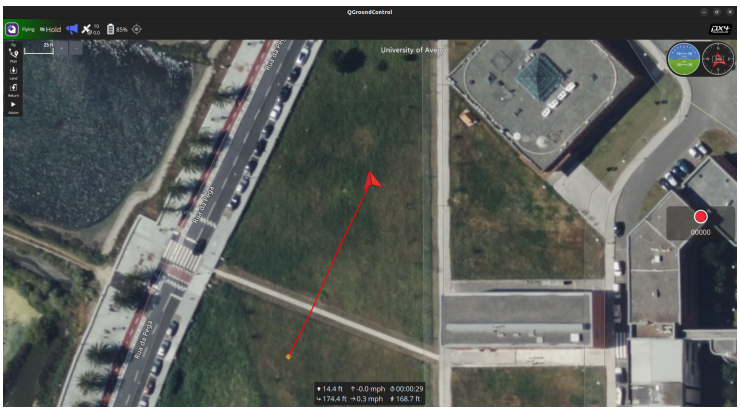


Figure 6.1: Mission without avoidance algorithm

The (Figure 6.2) demonstrates the drone dodging multiple times within the same movement. Each time, it received the data to create a Dodge mission (Figure 6.3) and sent a request to the Groundstation to initiate the dodge (Figure 6.4). After dodging, the drone resumes its planned movement.

Figure 6.2: Mission with object detection and avoidance algorithm

```
[20-05-2024 19:11:31.797] INFO  - dronePECI - Start takeoff with altitude 3.0m
[20-05-2024 19:11:37.689] INFO  - dronePECI - Finish takeoff
[20-05-2024 19:11:37.700] INFO  - Current thread name: mission-thread-14
[20-05-2024 19:11:37.805] INFO  - dronePECI - Start go to 40.634619, -8.660221 at 33.94m
[20-05-2024 19:11:38.728] INFO  - Received request [POST /mission/iHq9xoyK/pause]
[20-05-2024 19:11:38.733] INFO  - Current thread name: http-nio-0.0.0.0-8001-exec-1
[20-05-2024 19:11:38.761] INFO  - dronePECI - Cancel go to 40.634619, -8.660221 at 33.94m
[20-05-2024 19:11:40.682] INFO  - dronePECI - Success on cancel
[20-05-2024 19:11:40.688] INFO  - dronePECI - Withdrawing from mission
[20-05-2024 19:11:40.694] INFO  - Mission "iHq9xoyK" paused
[20-05-2024 19:11:45.720] INFO  - Received request [POST /mission]
[20-05-2024 19:11:45.724] INFO  - Created mission with id "BUcS7QcA"
[20-05-2024 19:11:45.739] INFO  - Mission "BUcS7QcA" started
[20-05-2024 19:11:45.937] INFO  - Assigning drone dronePECI to mission BUcS7QcA
[20-05-2024 19:11:45.941] INFO  - dronePECI - Assigned to mission "BUcS7QcA"
[20-05-2024 19:11:45.954] WARN  - Drone dronePECI is already in the air. Cannot arm.
[20-05-2024 19:11:45.957] WARN  - Drone dronePECI is already in the air. Cannot takeoff.
[20-05-2024 19:11:45.962] INFO  - Current thread name: mission-thread-15
[20-05-2024 19:11:46.709] INFO  - dronePECI - Start turn by 9.78 degrees
```

Figure 6.3: Groundstation logs on starting dodge algorithm

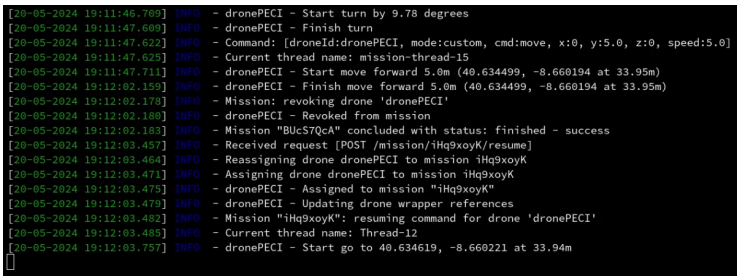


Figure 6.4: Groundstation logs on ending dodge algorithm

6.1.2 Follow

For the follow person mission, the application sends the coordinates to the broker (Figure 6.5) in the form of VAM messages. These messages are then relayed to the Groundstation, which uses the received coordinates to create a mission with the final point set to those coordinates, as shown in Figure 6.6.

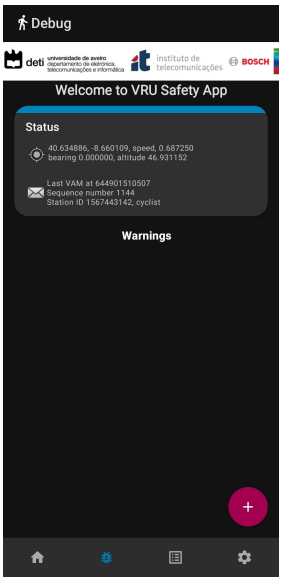


Figure 6.5: App sending coordinates to the broker

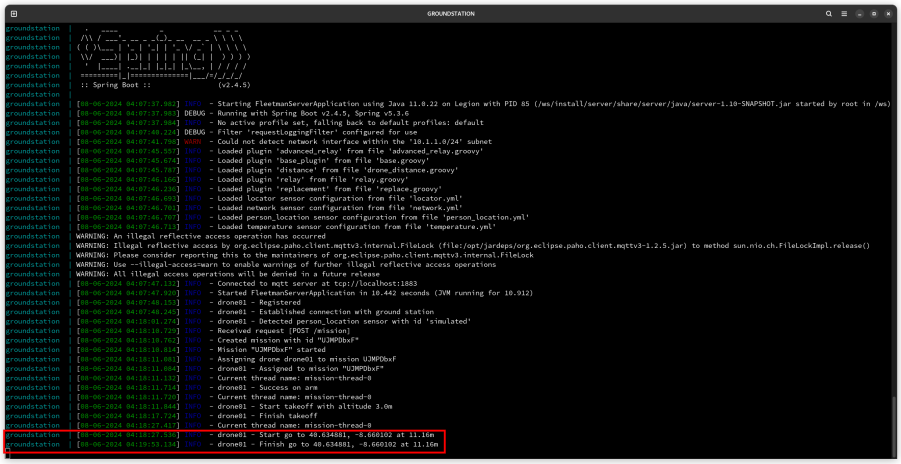


Figure 6.6: Groundstation receiving those coordinates and creating a mission

6.2 Network Performance

To test our connectivity, we conducted a series of tests to evaluate the system's performance. These tests involved testing the connection to the Starlink Satellite Constellation, the UAV connection to the Network Backhaul via Wi-Fi, the connection between the UE and the 5G UPF and last but not least, the connection from the UE, connected to the private 5G network, to the external world using Speedtests (meaning, a test that aggregates all the other ones).

6.2.1 Starlink Performance

The following results were obtained from running multiple speed tests using the `speedtest-cli` tool, conducted on a MINISFORUM EM680 NUC directly connected to the Starlink access point via a 1Gbps Network Interface Card (NIC).

	Download (Mbps)	**Upload (Mbps)**	**Latency (ms)**
Min	165.04	26.26	28.17
Avg	255.63	41.09	30.03
Max	331.84	53.02	33.90

Table 6.1: Starlink Performance results

Metric	Mean	Standard Deviation	Confidence Interval (90%)
Download Speed	255.63	30.912	(189.738, 321.536)
Upload Speed	41.09	4.684	(189.738, 321.536)
Latency	30.03	1.572	(26.681, 33.385)

Table 6.2: Summary of Starlink's internet performance metrics.

6.2.2 Backhaul WiFi

To perform our tests effectively, we established specific test points across the university's campus to measure the distance from each point to the access point.

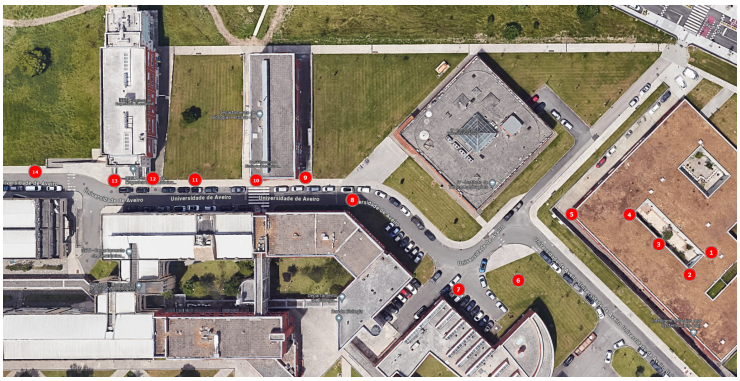
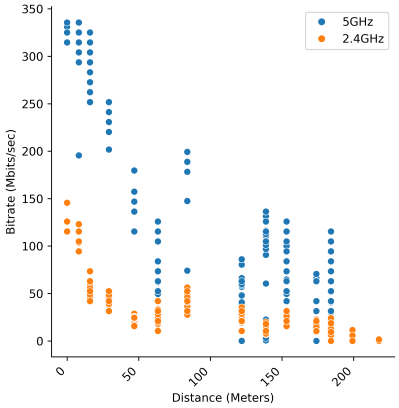


Figure 6.7: Test points

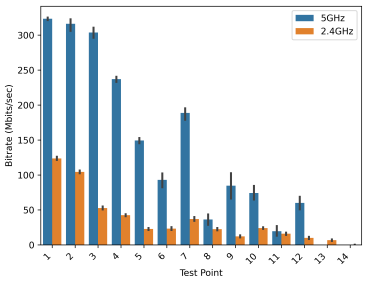
For our Backhaul WiFi tests, we used `iperf3` to generate traffic and measure the performance of the network link. The tests were conducted with a MINISFORUM EM680 NUC acting as the `iperf` server, which was connected directly to the Starlink access point via a 1Gbps Network Interface Card (NIC). The client device was an NVIDIA Jetson Orin NX, connected to the same access point using a TP-Link Archer T3U Plus USB wireless adapter.

Using `iperf3`, we generated TCP traffic to fully occupy the bandwidth of the tested link. Each test session lasted for 30 seconds. We conducted these tests locally without requiring external internet access, ensuring that the measurements were consistent and isolated from external network influences. The access point was always the Starlink access point to maintain hardware uniformity.

Figure 6.8: Backhaul WiFi - Bitrate

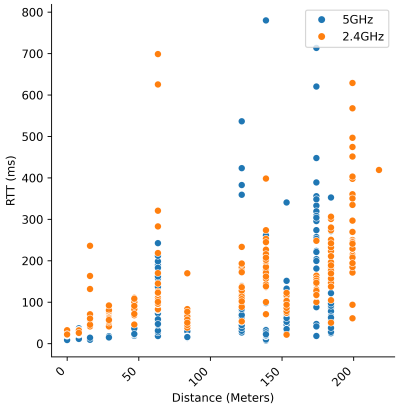


(a) Bitrate values in distance

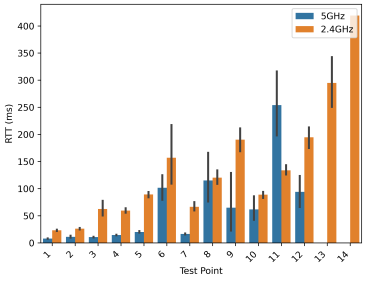


(b) Bitrate values in the test points

Figure 6.9: Backhaul WiFi - RTT



(a) RTT values in distance



(b) RTT values in the test points

6.2.3 5G UPF

To evaluate the performance of the radio link, we conducted a series of local tests to measure the download and upload speeds between the UE, specifically a Google Pixel 6a, and the UPF container running on a MINISFORUM EM680 NUC.

The tests were performed using `iperf3`, which allowed us to fully occupy the bandwidth of the tested link. The `iperf3` server was running on the UPF

container, while the Google Pixel 6a acted as the `iperf3` client.

Our tests involved generating TCP traffic to measure both download (from server to client) and upload (from client to server) speeds. Each test lasted for 30 seconds, during which we collected detailed metrics such as the bitrate. The tests were conducted at various distances from the gNodeB to analyze how distance impacts the performance of the 5G network.

For the server side, the following command was used to initiate the `iperf3` server on the UPF container, binding it to the specific network interface associated with the 5G network:

```
iperf3 -s -B <ogstun ip>
```

Listing 6.1: Server Command for `iperf3`

To measure upload speeds, on the Google Pixel 6a we executed the following command on the Termux terminal emulator to start the `iperf3` client, specifying the server IP address, test duration, and logging the output to a file:

```
iperf3 -c <server ip> -J --logfile <filename> -t 30
```

Listing 6.2: Client Command for `iperf3`

To measure download speeds, we added the `--rev` flag to the `iperf3` client command. This flag reverses the direction of the traffic, allowing the server (UPF Container) to send data to the client (Google Pixel 6a).

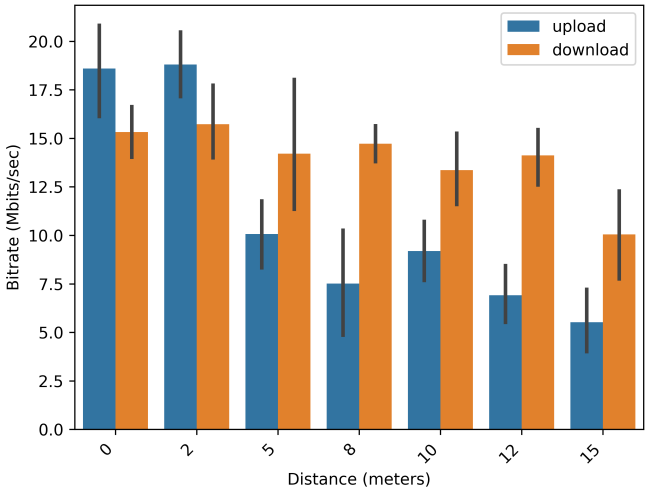
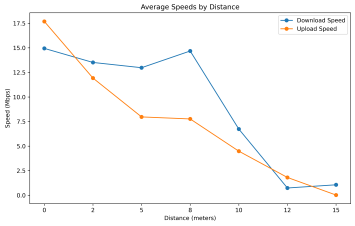


Figure 6.10: Bitrate values in distance

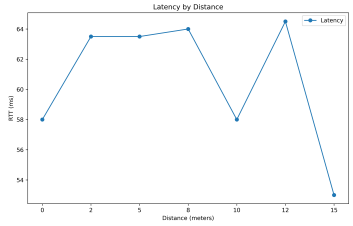
6.2.4 5G Speedtest

We performed multiple speedtests using the `speedtest-cli` tool from the UE (Google Pixel 6a) that was connected to our private 5G network, being broadcasted by the NVIDIA Jetson Orin's USRP.

Figure 6.11: 5G Speedtest - Speeds



(a) Speed values in distance



(b) Latency values in distance

6.2.5 Analysis

Starlink and Backhaul Analysis

The results from our tests indicate that the connectivity provided by the Starlink Satellite Constellation and the Wi-Fi backhaul were not the limiting factors in our 5G network performance. The Starlink connection demonstrated robust download and upload speeds, as well as low latency, which ensured reliable and high-speed data transmission.

Similarly, the Wi-Fi backhaul, tested at various distances, consistently provided sufficient bandwidth and stable performance. These findings confirm that the high-speed links from Starlink and Wi-Fi did not pose any constraints, allowing us to focus on other potential areas within the 5G network setup.

5G Analysis

We know that the goal of 5G is to achieve a high throughput and low latency connection. However, the results we obtained regarding the 5G connection do not reflect this, which was somewhat expected given that:

- Our USRP is among the most basic models available;
- We are not feeding the USRP with an external clock signal;
- While the Jetson platform excels at graphics-intensive tasks requiring acceleration, it is easily outperformed by a NUC (Next Unit of Computing) for tasks that are CPU-bound, such as packet forwarding;

- Nevertheless, our objective was to utilize the Jetson platform exclusively for all these functions on the drone to evaluate its limitations. Consequently, we were unable to employ more advanced configurations such as MIMO (Multiple Input Multiple Output), nor could we stably use signal frequencies above 20 MHz.

The situation worsens when we add our obstacle avoidance system and live video transmission system to the machine where the UPF is running, as both tasks are heavily reliant on CPU resources.

6.3 Live feed

As this was not one of our primary focus, the results are somewhat lacking in detail.

With that in mind, we still needed to verify the feasibility of a live feed. Figure 6.12 demonstrates the capability of providing a live feed. Meanwhile, Figure 6.13 confirms that the live feed is indeed being transmitted by the drone. This visual evidence ensures that the system is functioning correctly, showing real-time footage directly from the drone.

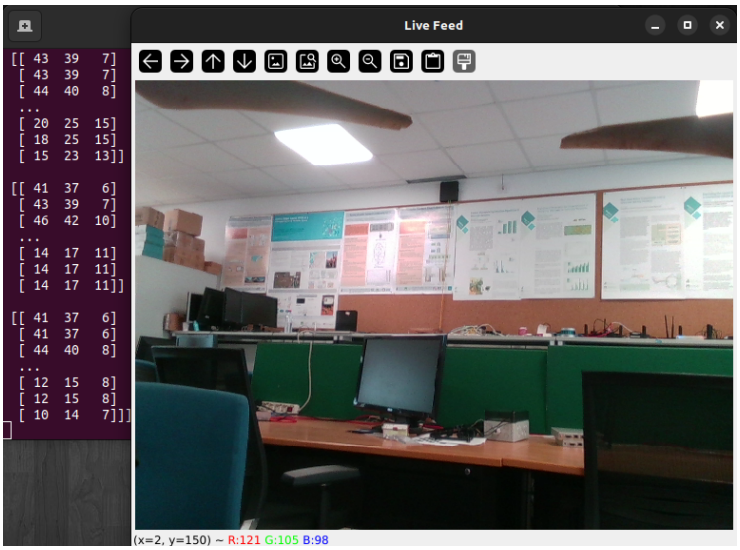


Figure 6.12: Receiving live feed from the UAV

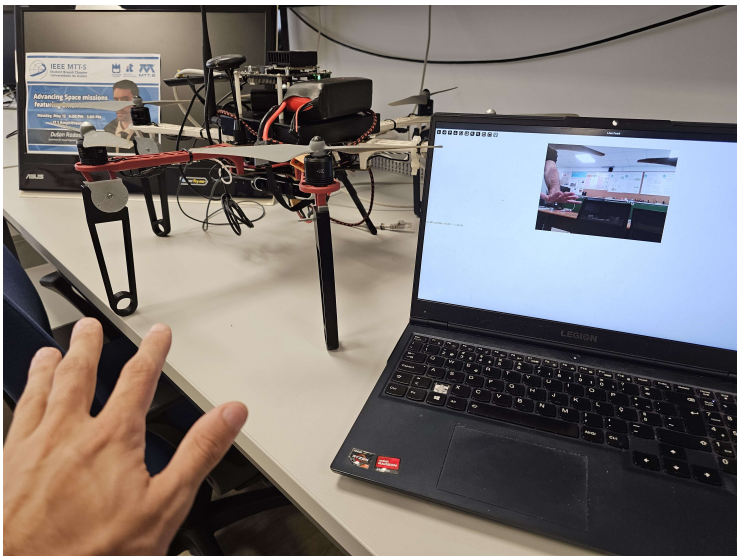


Figure 6.13: Showcasing the drone live streaming its feed

Chapter 7

Conclusion

This project has successfully demonstrated the potential of drones to significantly enhance emergency response efforts in catastrophic scenarios.

By developing and implementing an Escape Points Algorithm, we enabled the UAV to autonomously navigate and avoid collisions, changing its course every time it finds an object inside its cylinder closer than the minimum distance to dodge, ensuring safe and efficient flight. Furthermore, we complemented this ability with the possibility of following a person, by changing the final point of the movement to the coordinates received by a mobile app.

The establishment of a private 5G network further underscores the utility of drones in emergency situations. This setup allows emergency personnel, including police, firefighters, and civil protection teams, to maintain critical communication links even when traditional communication infrastructure is compromised, using the Starlink to achieve this objective. The innovative approach of using a drone to dynamically follow a designated user ensures that connectivity is maintained in a targeted and flexible manner, enhancing the effectiveness of emergency response teams.

Moreover, the integration of live feed capabilities on the drone opens up a wide array of applications, from real-time situational awareness during disasters to potential uses in fields like tourism, where live aerial views could enhance the visitor experience.

Looking forward, future work should focus on expanding the network coverage by deploying a fleet of UAVs. This would address the current limitation of a single drone's coverage range. Additionally, exploring the implementation of object detection using standard camera vision rather than relying solely on depth cameras could further enhance the drone's ability to autonomously navigate and interact with its environment.

In summary, the project's successful integration of advanced UAV technologies and communication networks has demonstrated a viable path towards improving disaster response capabilities. Continued development and scaling of these technologies hold promise for even greater contributions to emergency services and beyond.

Appendix A

Architecture Diagram

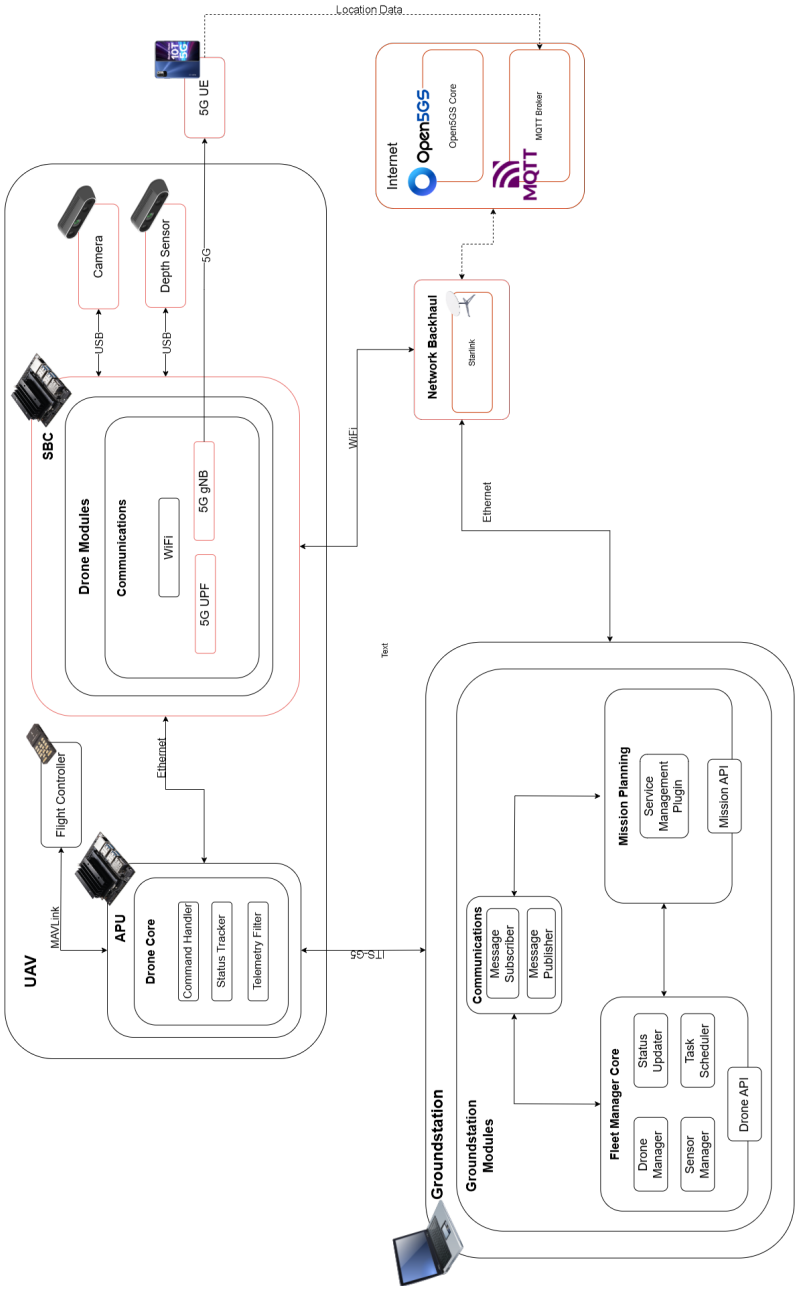


Figure A.1: Complete architecture diagram.

Acronyms

PECI Projeto em Engenharia de Computadores e Informática

UAV Unmanned Aerial Vehicle

SBC Single-Board Computer

VRU Vulnerable Road User

VAM VRU Awareness Messaging

EPA Escape Points Algorithm

RUAV Rotorcraft UAV

USRP Universal Software Radio Peripheral

ROS Robot Operanting System

FC Flight Controller

SBC Single-board Computer

MQTT Message Queuing Telemetry Transport

APU Accelerated Processing Unit

RAN Radio Access Network

UE User Equipment

COTS Commercial Off The Shelf

DSL Domain-Specific Language

UPF User Plane Function

NIC Network Interface Card

YOLO You Only Look Once

SCTP Stream Control Transmission Protocol

IMSI International Mobile Subscriber Identity

Bibliography

[1] Abdulla Al-Kaff, Fernando Garcia, David Martín Gómez, Arturo de la Escalera, and J.M. Armingol. Obstacle detection and avoidance system based on monocular camera and size expansion algorithm for uavs. *Sensors*, 17:1061, 05 2017.

[2] Ana Margarida Oliveira da Costa e Silva. A mission planning framework for fleets of connected drones. Master's thesis, University of Aveiro, 2021.

[3] Stefan Hrabar. Reactive obstacle avoidance for rotorcraft uavs. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4967–4974, 2011.

[4] Hanna Müller, Vlad Niculescu, Tommaso Polonelli, Michele Magno, and Luca Benini. Robust and efficient depth-based obstacle avoidance for autonomous miniaturized uavs. volume 40, pages 1234–1245. IEEE, 2024.

[5] Software Radio Systems. Commercial off-the-shelf user equipment, 2024. Available at `https://docs.srsran.com/projects/project/en/latest/knowledge_base/source/cots_ues/source/index.html`. Accessed: 2024-06-08.

[6] Herle Supreeth and contributors. Dockerized open5gs. `https://github.com/herlesupreeth/docker_open5gs`, 2024. Accessed: 2024-06-08.

[7] Pedro Veloso Teixeira. Awareness of potentially dangerous situations for vrus in a smart city. Master's thesis, University of Aveiro, 2021.