Drone Mesh for Crowd Density Monitoring

41008 - Redes e Sistemas Autónomos

Universidade de Aveiro

Diogo Correia 90327, Tiago Rodrigues 93413

2023/2024

Departamento de Eletrónica, Telecomunicações e Informática

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

API Application Programming Interface

APU Application Processing Unit

B.A.T.M.A.N. Better Approach to Mobile Ad-hoc Networking

CSI Camera Serial Interface

DOM Document Object Model

FC Flight Controller

IBSS Independent Basic Service Set

ITS-G5 Intelligent Transport Systems 5.9GHz

NAP-IT Network Architectures and Protocols at Instituto de Telecomunicações - Aveiro

OBU On-Board Unit

ROS 2 Robot Operating System

SSH Secure Shell

UAV Unmanned Aerial Vehicle

Acknowledgments

Queremos agradecer ao grupo Network Architectures and Protocols (NAP) do Instituto de Telecomunicações da Universidade de Aveiro, mais especificamente aos professores Susana Sargento e Pedro Rito, e aos investigadores Joaquim Ramos, Marcos Mendes e Andreia Figueiredo.

Introduction

1.1 Motivation

This project aims to take advantage of the versatility of UAVs, more specifically drones, when it comes to providing an overview of crowds that would, otherwise, be difficult to monitor from the perspective of an individual. The automation of movements through missions, the reasonable payload capacity for its size and its tolerable presence - being, in this case, short lived and at a fair altitude - make drones a fair choice to perform tasks that push the limits of the human aptitude.

Combined with the power of AI in the field of object detection and networking technologies, such as ad hoc networks through Wi-Fi using B.A.T.M.A.N. and vehicle communication-oriented technologies like ITS-G5, this creates the ideal scenario to setup a fleet of drones to monitor crowd density in a given area.

In this project, we will focus on the implementation of this idea with a fleet of two drones, which is enough for the size of the area that we have available to monitor. Nonetheless, this should be scalable, although scalability will not be taken into consideration here.

1.2 Objectives

The main goal is to have drones, equipped with cameras, to surveil a given area and count the number of people in it and send this information to a ground station that makes sense of the numbers and displays them in a dashboard. To tackle this, we need to handle the following fronts:

- Detect people using cameras and extract the number of detections;
- Utilize the fleet of drones and the Fleet Manager framework provided by Network Architectures and Protocols at Instituto de Telecomunicações - Aveiro (NAP-IT) to take this project from the simulator into the real world;
- Create an ad hoc network between the drones and the ground station. Because it is decentralized in nature, it is possible to maintain communication between the drones and the ground station at a reasonable distance with one node relaying the data from nodes further away;
- Dynamically distribute the drones through the area to be monitored and create missions to be used on Fleet Manager;
- Exchange coordinates between drones and implement a mechanism to prevent collision;

- Create a dashboard to run on the ground station that is able to initiate the monitoring missions and display results;

These objectives will be accomplished always with the intention of using real drones, whereas the eventual usage of a simulator is only intended for testing purposes. For the simulator, we will use the jMAVSim[1], running the PX4-Autopilot firmware v1.10.1[2].

[1] https://github.com/PX4/jMAVSim
[2] https://github.com/PX4/PX4-Autopilot

Related Work

This project is built using frameworks, hardware and technologies that already exist and we find it worth mentioning.

2.1 Unmanned Aerial Vehicles

To be able to leave the simulator, we depend greatly on the infrastructure provided by NAP-IT. We will use drone dev kits built and maintained there, and we will interact with the Flight Controller (FC) using the mission planning framework Fleet Manager also developed there. All the OBUs, either APUs or Raspberry Pis, are also facilitated by them, as well as antennas and energy sources like Powerbanks and LiPo batteries.

The APUs already have a WLAN interface with ITS-G5 ready to use.

The Fleet Manager framework uses ROS 2 to send commands to the drone's FC and provides a layer of abstraction that allows for missions (sets of commands) to be created using a custom user-friendly description language through Groovy. This framework is composed of two modules, the ground station and the drone. Those modules will be running on our ground station and drone's OBUs, respectively.

Architecture

3.1 Network

This project follows an ad hoc oriented approach, in which we use B.A.T.M.A.N. to setup a network between all nodes, and facilitate a relay mechanism. Our ad hoc network has three nodes, two Drones and one Ground station. From those nodes, only the Drones are expected to be moving nodes.

In parallel with this WiFi network, we also take advantage of ITS-G5, also present on all devices.

These two technologies serve distinct purposes due to their characteristics. WiFi is used in situations where we need to send larger payloads, but the latency and range aren't crucial. On the other hand, ITS-G5, being a technology made with vehicle communications in mind, has lower latency and wider range, and is used in this project to assure the drones can always receive commands from the groundstation.

3.1.1 Drone

Excluding specific components, such as the Flight Controller, a drone is made up of two on-board units: an APU and a Raspberry Pi with a camera attached.

From the APU, we take advantage of two main interfaces, a WLAN with ITS-G5, which was already setup beforehand, and an Ethernet Interface, which we use to grant direct communication between the APU and the Raspberry Pi, which is needed for one specific software module. The APU is also connected via USB to the drone Flight Controller, to send commands via MAVLink (which is part of the Fleet Manager Software module that already existed).

The Raspberry Pi is more focused on the image processing part and the distribution of the results to the ground station. We use the Ethernet Interface to, as mentioned above, get a direct communication with the APU and the WLAN interface is used to integrate the drone in the B.A.T.M.A.N. WiFi network. There is also a camera directly connected to the interface video0 which, in the case of a Pi Camera, is using the Camera Serial Interface (CSI).

It is expected, then, that the drone receives commands in the **APU Wireless Interface** through ITS-G5 using ROS 2, sends telemetry data to the Raspberry Pi in the **APU Ethernet Interface**, receives the camera feed in one of the **Raspberry Pi CSI** and sends the YOLO's results from the image processing for detection in the **Raspberry Pi Wireless Interface** through WiFi using HTTP and MQTT.

3.1.2 Ground station

The Ground station has a simpler structure. It is made of a single APU, where we take advantage of two Wireless interfaces, and one Ethernet Interface. For one of the WLAN interfaces, the one that will facilitate the B.A.T.M.A.N. network, we use a WiFi adapter that supports IBSS. The Ethernet Interface is used to SSH into the machine from a laptop.

It is expected, then, that the ground station sends commands to the drone in the **Wireless Interface** setup with ITS-G5, receives video frames and people counter in the other **Wireless Interface** with WiFi, through HTTP and MQTT, respectively and communicates directly via **Ethernet Interface** with an external general purpose computer to serve as the Gateway to all the components of this project, on all nodes. This is possible because the APU of the ground station is part of both wireless networks and can reach all components, both the other APUs (via ITS-G5) and the Raspberry Pis (via WiFi). This is particularly convenient for when we need to setup or debug problems on any component in the system.

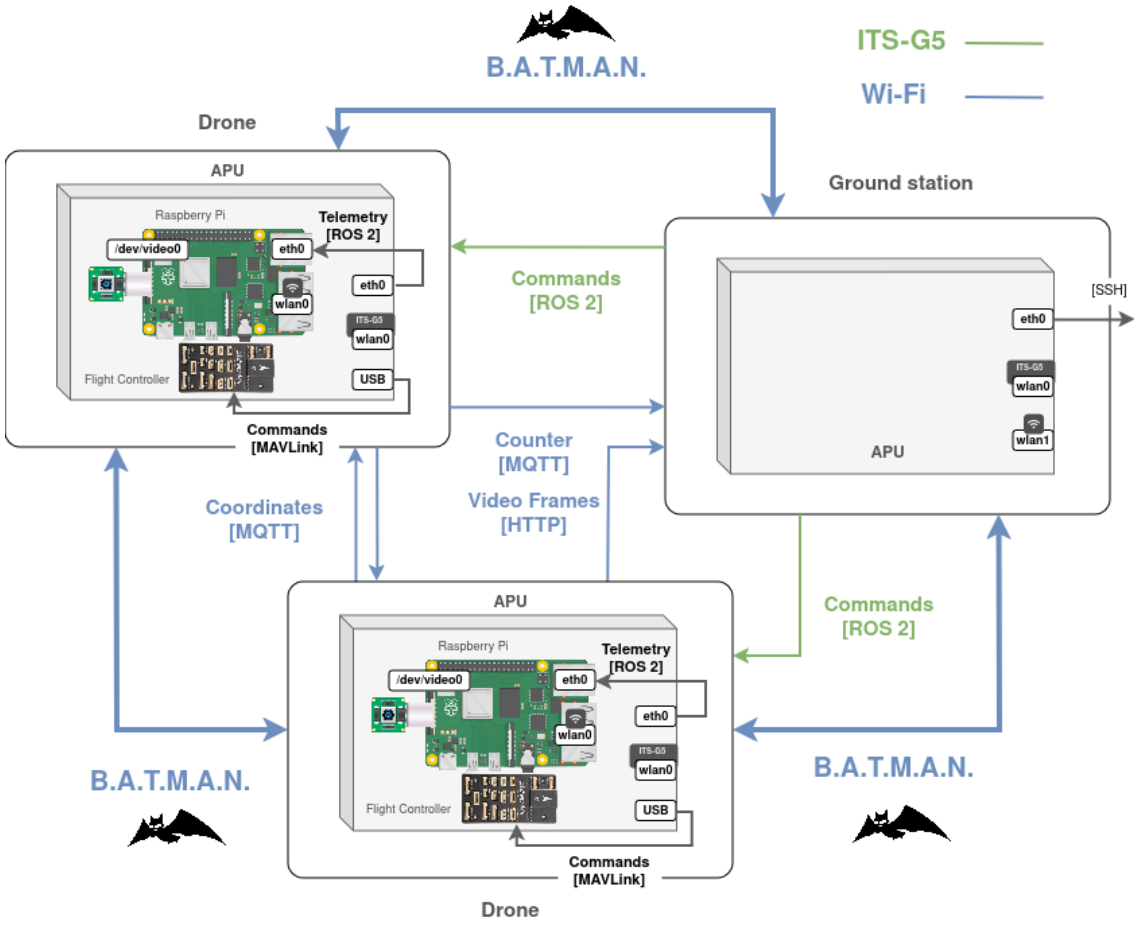

Figure 3.1: Network Architecture

3.1.3 Addresses structure

The IP addresses on each interface can be mapped as follows:

Node	Component	Interface	IP Address	Technology
Drone	APU	wlan0	172.254.0.XXX	ITS-G5
		eth0	192.168.4.XXX	Cabled
	Rasp. Pi	wlan0	10.1.1.XXX	WiFi
		eth0	192.168.4.XXX	Cabled

Table 3.1: Network Interfaces on Drone

Node	Component	Interface	IP Address	Technology
Ground station	APU	wlan0	172.254.0.XXX	ITS-G5
		wlan1	10.1.1.XXX	WiFi
		eth0	192.168.4.XXX	Cabled

Table 3.2: Network Interfaces on Ground station

3.2 Software

To automation of our monitoring system begins in the Ground station. Here, the drone manager uses the **Dashboard**, a web app created with Flask, to outline the area to be monitored, generates the mission with the **Mission Generator** module, sends it with the **Fleet Manager** framework, and watches the outcome while the drones do their job of completing the said mission, while detecting people with their cameras.

This detection is processed in one of the OBUs with the **Detection** module that uses YOLO through the python library Ultralytics.

Because more than one drone is flying at the same time, they share their coordinates through the **Telemetry** module. These coordinates are then used by the **Avoidance** module to detect when two drones are too close to each other and should take immediate action to avoid a collision.

As mentioned before, the Fleet Manager framework already existed and we only had to integrate it in our system, both the Ground station and the Drone modules.

3.2.1 Ground station

The ground station is mainly composed by three modules. The two that we made were developed with Python. The other one, that we are integrating, was made with Java using Spring Boot. There is also a Broker using Mosquitto to handle data coming from the drone's people detection module.

Dashboard

The Dashboard module is a web application made with Flask that makes up both the frontend and the backend. It has only one page divided into three sections: 1. Generate Mission, 2. Send Mission and 3. View Mission.

The first section uses Leaflet[1] to allow the user to define, on a map, the area to be monitored. This area makes up a polygon, that will be used to generate two areas to be distributed by the two drones on section 2.

After the two new polygons are generated along with the mission files, the user can send the two missions to each drone, and jump to section 3, where the feed of each camera is presented with the number of people being counted.

The camera feed is presented by collecting the video frames that are sent by the drones through HTTP POST request to a /upload endpoint in the backend of the web application. The /stream endpoint then displays each frame on the DOM and can be used in the index page, on section 3.

The backend of the web application also has other endpoints such as /station that pings the ground station and checks for its status, the /generate and /mission that implements the Mission Generator module from section 2 and the /avoidance/<drone> that should implement the collision avoidance algorithm as soon as it is called.

To help handling the missions and polygons created, the dashboard has a relational database using MySQL.

Mission Generator

When a user selects coordinates on a map to outline an area, they are essentially creating a polygon. With the help of geometry python libraries, like shapely[2], we can take that polygon and divide it in half, which will result in two missions, one for each drone. The division is done through the following steps on the **geometry.py** script:

- Find the centroid of the polygon
- Calculate a line that goes from one of the vertices to the centroid
- Calculate the direction vector of that line to be able to extend it on the other direction and cross the polygon's perimeter
- Subtract the line to the polygon, originating two new smaller polygons

After obtaining the two smaller polygons, we resize them to give space for the drones to traverse the line that separates the two. The two final polygons are passed to the **mission.py** script that parses the vertices and creates a Groovy file with commands that are compatible with the Fleet Manager framework.

Fleet Manager - GS

We wanted to take advantage of the possibility to automate the drone's movement with the already existing Fleet Manager[3] framework.

The ground station module consists of an API in Java using Spring Boot, that has endpoints that interact with the drone's Flight Controller (via the Fleet Manager drone module) with simple HTTP requests.

From this module, we used three endpoints. The /drone?data=info to get the status of the two drones and display it on the dashboard, the /mission to start the mission with the Groovy files

[1] https://leafletjs.com/
[2] https://pypi.org/project/shapely/
[3] https://link.springer.com/article/10.1007/s10846-023-01820-7

generated on the Mission Generator module and the /mission/<mission_id>/pause to pause an ongoing mission triggered by the Avoidance module running on the drones.

This module communicates with its counterpart on the drone using ROS 2 over the ITS-G5 link.

3.2.2 Drone

The Drone is composed by four main modules. Three modules developed by us using python, and the Fleet Manager drone module that we integrated and was developed in C++ and communicates directly with the drone's Flight Controller with MAVLink[4].

Detection

The module to detect people interacts directly with the camera using the PiCamera2[5] python library. The frames extracted are used with another python library, Ultralytics[6], that takes advantage of YOLO[7] object detection AI models to extract the number of people in a frame. Both the frame and the number of people are sent, through HTTP and MQTT respectively, to the Dashboard module in the ground station, via the WiFi link.

The messages.py script is used to setup an MQTT Client and publish the data to the Broker in the ground station.

Telemetry

As mentioned already, we wanted to deal with the fact that we have two drones flying at the same time, fairly close to each other.

We chose an approach that involves continuously measuring the distance between the two drones and taking action if the distance falls below a predefined number of meters.

To measure that distance, we needed to exchange coordinates between the drones, and then, for example, calculate the Euclidean distance.

The fetching and exchanging of coordinates is done in the Telemetry module. A straightforward approach would be to get the coordinates directly from the Fleet Manager ground station module, but that would require using the wireless link which would introduce delay, which is not acceptable in a situation of emergency.

We this in mind, the script telem.py, using the python library rclpy[8], fetches the coordinates directly from the Fleet Manager drone module through ROS 2 and publishes those coordinates on a Broker running in one of the two drones. This way, with the Telemetry module running on both drones, the coordinates are accessible to the two nodes.

Avoidance

This module, through the script avoidance.py, fetches the coordinates from both drones from the Broker running in one of the drones and, using the python library geopy[9], calculates, in meters,

[4] https://mavlink.io/en/
[5] https://pypi.org/project/picamera2/0.2.2/
[6] https://www.ultralytics.com/
[7] https://pjreddie.com/darknet/yolo/
[8] https://github.com/ros2/rclpy
[9] https://geopy.readthedocs.io/en/stable/

the distance between the two in real time.

If this distance falls bellow a pre define value, it send an HTTP Request to the Dashboard module on the endpoint /avoidance, to trigger the collision avoidance procedure.

It is worth noting that this module does not implement any avoidance mechanism, it simply tells the ground station when a collision is imminent.

On the side of the ground station, the dashboard, upon receiving the avoidance request, queries the database for the mission id given to the relevant drone and sends an HTTP Request to the Fleet Manager ground station module to pause that mission.

The mechanism that follows the pausing was not implemented in this project.

Fleet Manager - Drone

The drone module from Fleet Manager is what communicates directly with the Flight Controller with MAVLink. It receives commands from the ground station module and parses them.

This module also publishes the telemetry that comes from the Flight Controller, and that is then fetched by both the Fleet Manager ground station module and our Telemetry module.

This module is the bridge between the drone itself, and the software running on the OBU.

It is worth mentioning that this module is the only one running in the APU of the Drone. All other modules are running on the Raspberry Pi, and this is the reason why the APU and Raspberry Pi need to be connected via ethernet.

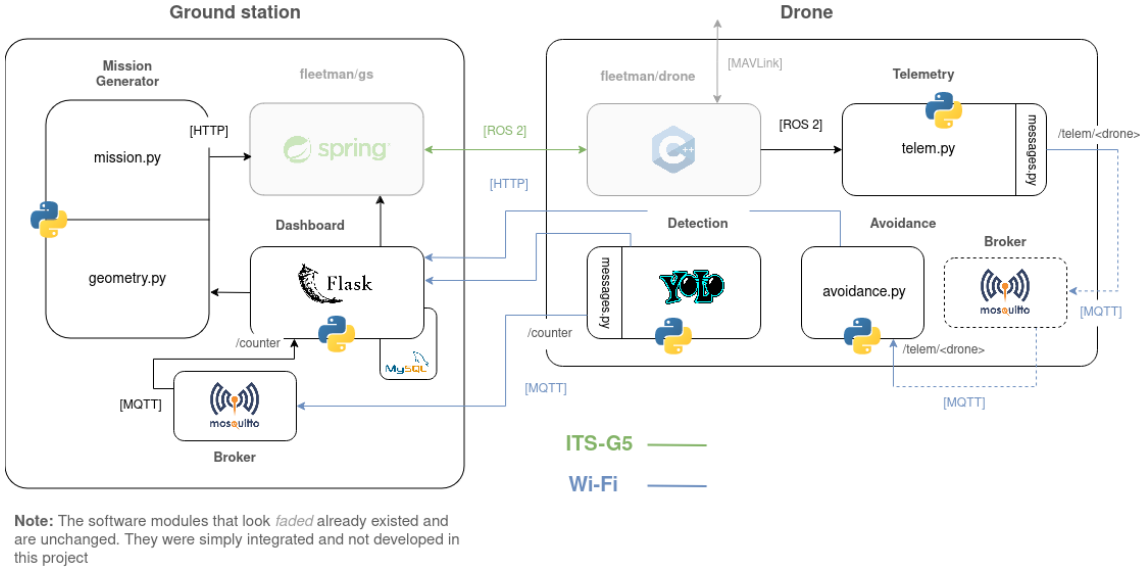

Figure 3.2: Software Architecture

3.3 Real World Scenario

With all the nodes communicating with each other through two distinct wireless links, and all the modules running on the OBUs and ground station, all that is left is to idealize a real world use case scenario to test our system.

The most obvious use case is to count the number of people in a given area, for example a public park, a festival, the beach, basically anywhere that you can plant the ground station and send the drones into the air.

A situation of emergency where the reaching is limited, because of the limitation of WiFi in terms of range, we would need to use specialized drones just to relay the communications between the monitoring drones and the ground station. This would be possible thanks to B.A.T.M.A.N.. We would always be limited to the ITS-G5's range, but it is larger than WiFi's.

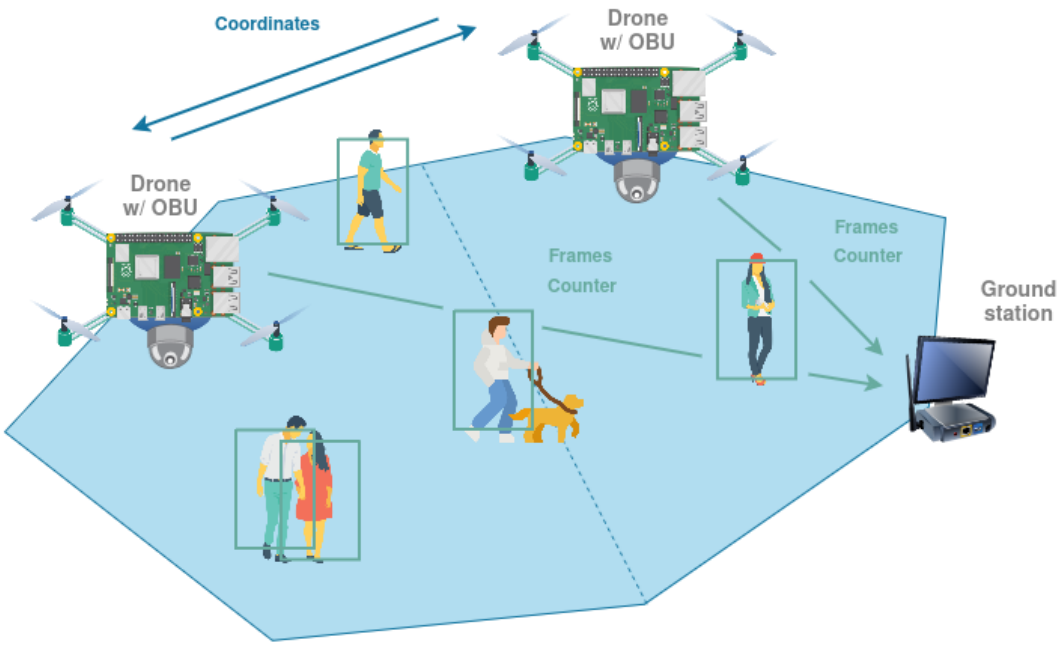

Figure 3.3: Real World Scenario

Implementation

We need to setup three different machines to have the system working, which are the APU in the ground station and the APU and Raspberry Pi in the drone (repeating this last two machines twice).

The first step is to clone the project repository on every machine (see chapter 8).

Inside the repository, create a python virtual environment and install the requirements.

```
$ python3 -m venv --system-site-packages venv
$ source venv/bin/activate
$ pip install -U pip
$ pip install -r requirements.txt
```

4.1 Ground station

In terms of hardware, we need an APU, a WiFi adapter that supports IBSS, a 30W power bank and a ethernet cable to connect directly to a laptop.

In terms of software, the setup of the Fleet Manager module is done with scripts provided by Instituto de Telecomunicações.

```
$ cd uavs
$ ./run.sh -t groundstation
```

Setup the B.A.T.M.A.N. interface (you will need batctl and batman-adv). Make sure to choose the WLAN interface that is not using ITS-G5.

Inside the repository and the virtual environment, do the following:

Run the broker

```
$ cd src/broker
$ ./broker.sh <port>
```

Run the dashboard

```
$ cd src/groundstation/dashboard
$ docker-compose up --build
```

Make sure you have the .env file with the following content:

```
TILE_URL=<MAP_TILE_URL>

DBMS = 'mysql'
USER_NAME = 'root'
PASSWORD = 'password'
HOST = 'localhost'
PORT = 3307
DATABASE = 'groundstation'
```

4.2 Drone

In the drone, we need to setup the Fleet Manager drone module in the APU and the python modules from the project in the Raspberry Pi/Jetson Nano.

The APU and the Raspberry Pi need to be directly connected through ethernet.

4.2.1 APU

To run the Fleet Manager drone module, do the following:

```
$ cd fleet-manager/deploy
$ ./launch_drone_container.sh <drone_id> -c ../configs/
    drone_cfg_serial.yml
```

Check the name of the serial port the drone is connected to, and change it, if needed, in the configuration file:

```
port: serial:///dev/ttyPixhawk # change this if needed!
telemetryTopic: /telem
...
```

In Linux, to find the name of the port the Drone Flight Controler is connected to, you can use the following command:

```
$ dmesg | grep tty
```

4.2.2 Raspberry Pi

Setup the B.A.T.M.A.N. interface (you will need batctl and batman-adv).

Make sure you have all the dependencies installed to use libcamera and picamera2.

Get inside the project repository and the virtual environment, and do the following:

Run the broker (in one of the drones)

```
$ cd src/broker
$ ./broker.sh <port>
```

Run Detection module

```
$ cd src/drone/detection
$ python3 app.py
```

Run Telemetry module

```
$ cd src/drone/telemetry
$ ./telemetry.sh <drone_id>
```

Run Avoidance module

```
$ cd src/drone/avoidance
$ python3 avoidance.py <drone_id>
```

4.2.3 Jetson Nano JetPack 4.6 (L4T R32.6.1)

If you have a Jetson Nano, instead of a Raspberry Pi (which means better performance on the object detection module), you can follow this setup to make sure you use the GPU:

Export Paths for CUDA

Add the following lines to /.bashrc

```
export PATH=/usr/local/cuda-10.2/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-10.2/lib64:$LD_LIBRARY_PATH
export FORCE_CUDA="1"
```

Build OpenCV with GStreamer support

Check if your OpenCV supports GStreamer:

```
$ python3
>>> import cv2
>>> print(cv2.getBuildInformation())
```

If the GStreamer flag is 'NO', then you will need to recompile OpenCV[1] with the GStreamer flag as 'YES' (this should take a couple of hours).

Create pip package for Compiled OpenCV

Create a dummy package for OpenCV installed from source (to prevent pip from trying to install another version of OpenCV)

```
$ cd ~
$ mkdir opencv-pip
$ cd opencv-pip

$ touch setup.py
```

[1] https://qengineering.eu/install-opencv-on-jetson-nano.html

Add the following content to setup.py:

```
from setuptools import setup

setup(
                  name='opencv-python',
                  version='4.10.0-pre',
                  description='Dummy package for OpenCV installed from source',
                  install_requires=[],
)
```

Install dummy package:

```
pip install --no-deps opencv-python
```

Clone ultralytics

```
$ git clone https://github.com/ultralytics/ultralytics
$ cd ultralytics
```

Setup Python 3.8

Jetson Nano comes with Python 3.6.9, but we need Python 3.8.

```
$ sudo apt-get update
$ sudo apt install -y python3.8 python3.8-venv python3.8-dev python3-pip libopenmpi-dev libomp-dev libopenblas-dev libblas-dev libeigen3-dev libcublas-dev
$ python3.8 -m venv venv --system-site-packages
$ source venv/bin/activate
```

Install PyTorch 0.11.0 and TorchVision 0.12.0

Versions that work with CUDA 10.2

```
$ pip install -U pip wheel gdown
$ gdown https://drive.google.com/uc?id=1
   hs9HM0XJ2LPFghcn7ZMOs5qu5HexPXwM
$ gdown https://drive.google.com/uc?id=1
   m0d8ruUY8RvCP9eVjZw4Nc8LAwM8yuGV
$ python3.8 -m pip install torch-*.whl torchvision-*.whl
```

Install ultralytics

```
pip install .
```

Go back to the previous section and follow the Raspberry Pi setup.

Results

On Figure 5.1, 5.2, 5.3, 5.4 we show the process of generating and sending a mission to the drones.

The drones are setup with OBUs and power banks on Figure 5.5, 5.6. The Ground station setup is on Figure 5.7.

The implementation of B.A.T.M.A.N. protocol was successful and we tested the relay between the nodes on Figure 5.8, 5.9.

Lastly, we implemented the collision avoidance detection between the drones which we demonstrated using the simulator in Figure 5.10, 5.11, 5.12.

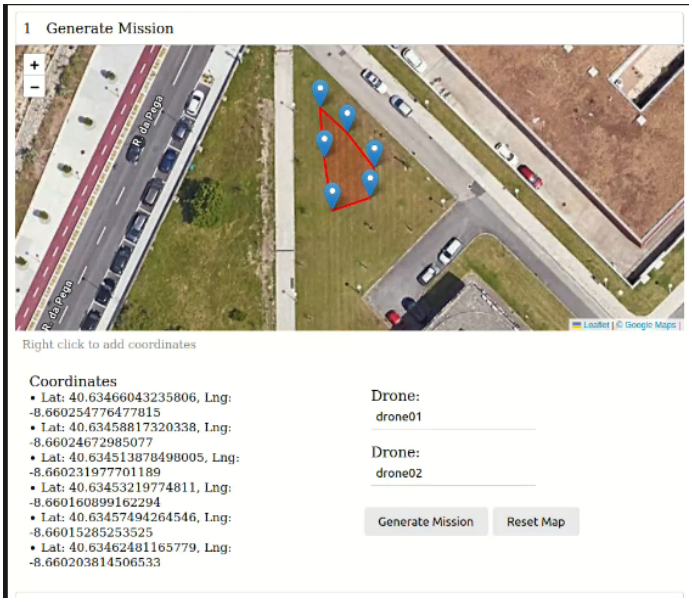

Figure 5.1: Mission setup - step 1

Figure 5.2: Mission setup - step 2

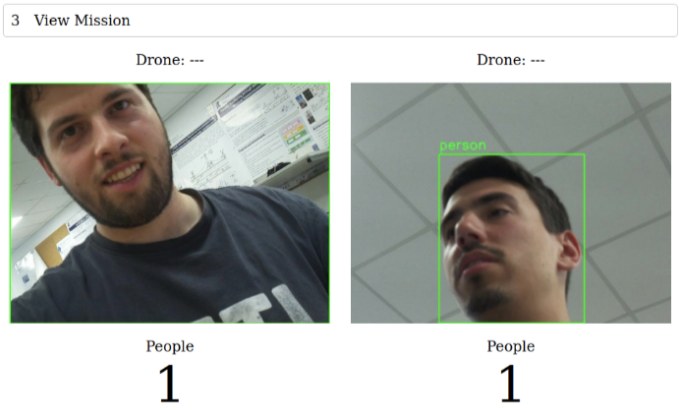

Figure 5.3: Mission setup - step 3

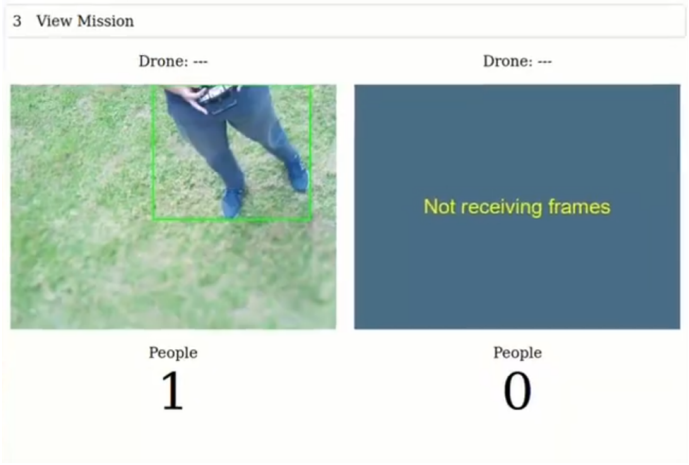

Figure 5.4: In flight camera detection

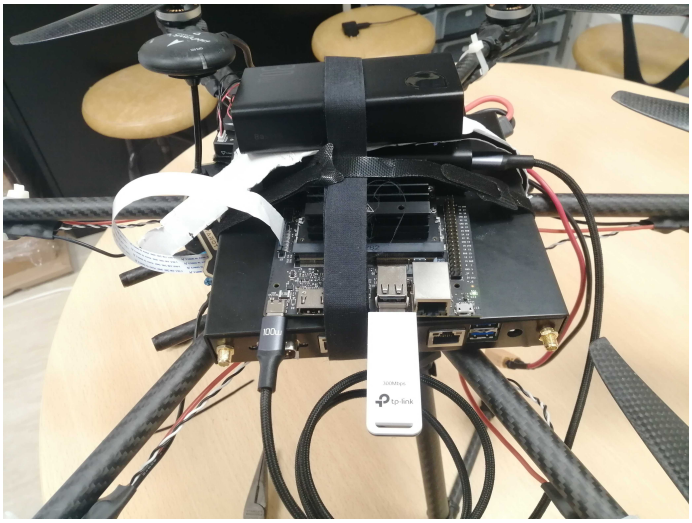

Figure 5.5: Drone Setup - View 1

Figure 5.6: Drone Setup - View 2

Figure 5.7: Ground station Setup

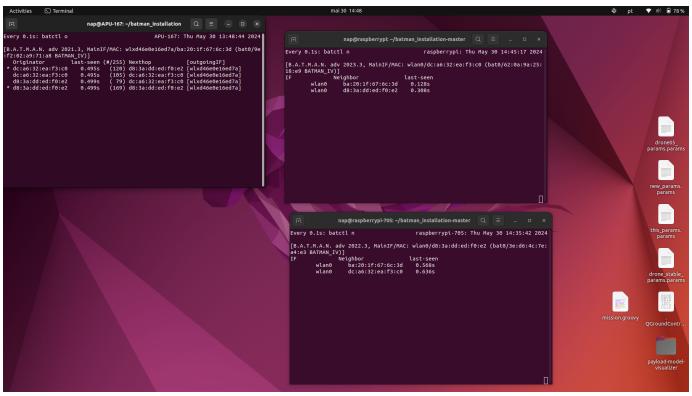

Figure 5.8: B.A.T.M.A.N. relay testing

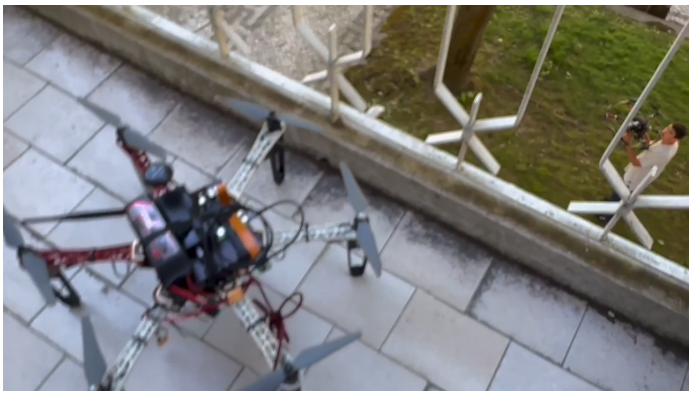

Figure 5.9: B.A.T.M.A.N. testing with the drones

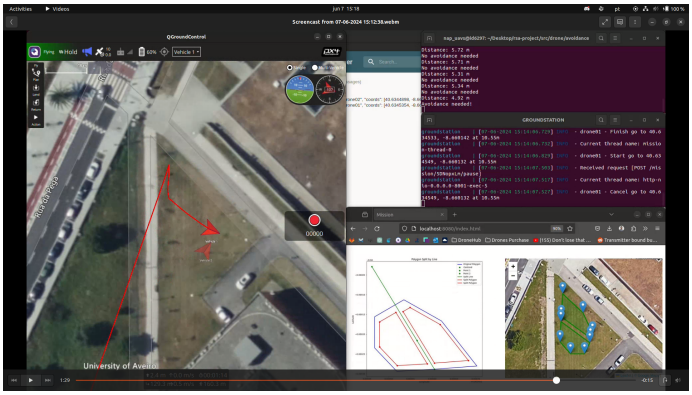

Figure 5.10: Message of avoidance needed

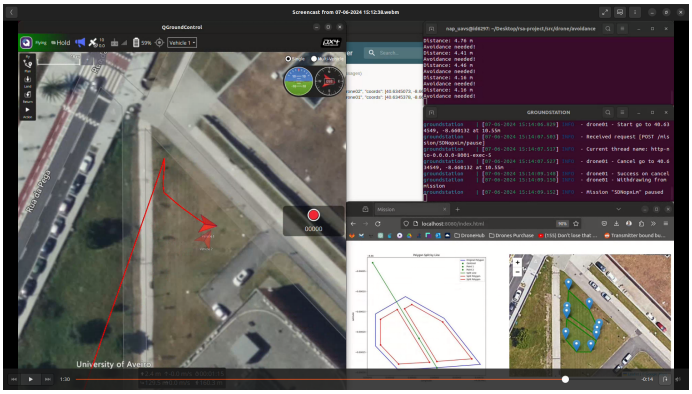

Figure 5.11: Mission paused due to risk of collision

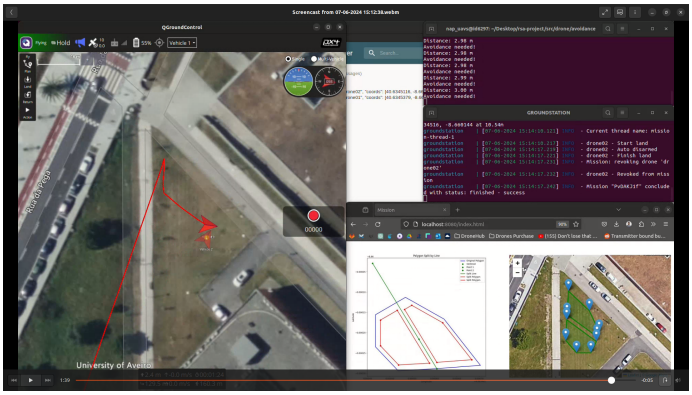

Figure 5.12: Mission concluded

Conclusion

In this day and age, automation is key. With this project, we demonstrated that a couple of drones are able to achieve a task that would be demanding for a couple of individuals. All of this, without a centralized infrastructure, allowing for lower costs and complexity in the implementation.

In summary, we think we have managed to make our vision come to life using real hardware which gave us a bit of trouble, overcoming it nonetheless. We were faced with overheating Raspberry Pis due to the object detection algorithm and solved it using active coolers and older and lighter versions of the detection models. Other inconveniences like the unstable connection of the WiFi adapter connected to the ground station proved a stable communication and the use of ssh tools via a B.A.T.M.A.N. network rather hard.

In the end we believe we tackled all the problems the best we could, having all the main modules implemented in hardware, which is always more challenging than doing it on a simulator, due to problems that, at times, were unrelated to our work.

Future Work

Due to the complexity of implementing this project on real hardware, some work was left unfinished and/or could be polished. That is as follows:

- Implement the mechanism that follows the mission pausing upon the avoidance request made by the Avoidance module to the Dashboard
- Get a complete Demo of all the monitoring being done with two real drones
- Upgrade the OBU that is doing the image processing for people detection to a device with a GPU
- Improve the overall look of the Dashboard, or integrate it on one that is more complete

Resources

- **Repository:** https://github.com/digas99/rsa-project
- **Demo:** https://www.youtube.com/watch?v=9UTSuK-eEUw