University of Aveiro
Integrated Masters in Computer and Telematics

Informatics Engineering Project

Drone Relay

Aerial Drone platform for scenarios of limitation and emergency

Authors:

Guilherme Amaral Ribeiro Pereira:	93134
José Luís Rodrigues Costa:	92996
Diogo Miguel Rocha Amaral:	93228
João Tiago Lacerda Rainho:	92984

Drone Relay

Technical report of Informatics Engineering Project of Integrated Masters in Computer and Telematics of University of Aveiro

Diogo Amaral (93228) diogomra7@ua.pt
Guilherme Pereira (93134) guilherme.pereira@ua.pt
José Luís Costa (92996) joselcosta@ua.pt
João Tiago Rainho (92984) tiago.rainho@ua.pt

Supervised by professor of Department Electronics, Telecommunications and Informatics (DETI) of University of Aveiro (UA) and researcher on Institute of Telecommunications (IT), Susana Sargento and professor of Superior Institute of Engineering of Lisbon and researcher on Institute of Telecommunications, Miguel Luís. With the support of Margarida Silva and Nuno Ferreira that developed the previous part of our project.

Acknowledgement

We would like to express the deepest appreciation to our supervisors, Dr. Susana Sargento and Dr. Miguel Luís. Their dedication and keen interest to help and orient us was a key point to the completion of this project.

We also would like to thank Margarida Silva and Nuno Ferreira for their availability to help us and give suggestions on keeping the project in the right direction.

And lastly, to the professors of the subject Informatics Engineering Project who always gave us advice on how to improve our work.

Abstract

Nowadays aerial drones are becoming very popular as they also become more accessible, in terms of usability and economics. Drones can have reduced size and good cost-benefit ratio which make them an excellent alternative for some specific objectives/missions. Some scenarios that benefit from the use of aerial drones are the surveillance of important individuals, monitoring public-gathering restrictions, monitoring forest areas or disaster areas, transferring lightweight high value objects, watching live video transmission, etc…

This project was developed in conjunction with other UAVS projects in IT Aveiro, mainly the Fleet-Manager software that provides a grounstation module that controls the drones to perform coordinated missions.

Taking all that into account we present in this report how we developed the main components that were the focus of this project:

A wireless network sensor to measure multiple network parameters and extrapolate the quality between the different drones and the groundstation.

A plugin for the groundstation of a mission algorithm that takes into account that network quality and places relay drones in an optimal position as to extend the range and connection quality of the drones.

A video transmission system that automatically and dynamically changes quality according to the network, displaying the live output stream in a dashboard.

Lastly the telemetry and sensor information sending ratio also dynamically changes according to the network quality in a given moment.

List of Figures

Index

1. Acronyms

FANET	Flying Ad-Hoc Network
IP	Internet Protocol
MAC	Media Access Control
NVENC	Nvidia Encoder
PEI	Informatics Engineering Project
POI	Point Of Interest
ROS	Robotic Operating System
RSSI	Received Signal Strength Integrated
SOA	State Of Art
SPA	Single Page Application
SDK	Software Development Key
UAV	Unmanned Aerial Vehicle

2. Introduction

In this chapter we introduce the context and the objectives of this project that has been developed along the semester in the informatics engineering project. We start by the motivation of this project followed by the goals and contributions that we expected to achieve with this project and to conclude this chapter a description of what each chapter will aboarded is provided.

2.1. Motivation

Nowadays aerial drones are becoming very popular as they also become more accessible, in terms of usability and economics. Drones can have reduced size and good cost-benefit ratio which make them an excellent alternative for some specific objectives/missions.

These drones can be used in a number of scenarios such as monitoring emergency situations, patrolling and surveillance and to monitor many areas that are difficult to access.

A already developed mission planning framework for drone fleets that supports a system for remote drone control has already been developed, however this framework can now be further developed and improved by adding new features.

With that in mind the motivation for this project is to develop features that allow the drones to have a wider range of operation by extending the network with the aid of relay drones based on how the network quality is at certain moments as well as integrating a camera system on the drones that also takes into account the network in order to change quality as to maintain the video feed lag free.

2.2. Goals and contributions

This project aims to develop new features under the framework mentioned previously and those goals are:

- Developing a wireless network sensor that runs on the drones with the objective of indicating the drones if they need a relay drone to extend the network and to change the quality of the video.

- Integrating a camera on the drones capable of changing its quality depending on the network conditions

- Adapt the sending ratio of telemetry messages and sensor messages according to the network quality.

- Develop an algorithm that runs in any mission (plugin) which enables the feature of deploying relay drones to an optimal position depending on the network quality making sure that the drones that are performing a mission don't lose connectivity to the groundstation .

- Integrate the video provided by the camera that will be placed on the drones and other features in the already existing dashboard for displaying and informational purposes.

2.3. Document outline

This document will discuss a wide range of topics which we divide by chapters. In this section we will briefly describe what each chapter mentions.

Chapter 3 we will be aboarding some of the related work and technologies used in this field of study which we used/studied in order to develop this project.

Chapter 4 presents the state of the art, that mentions some papers that have been published in this field of study.

Chapter 5 is the part of the document that discusses the architecture we proposed in order to achieve all goals of this project and also describe the system requirements as well as the actores and the use cases.

Chapter 6 documents the implementation, this is, how each module works, how they can be used, their specific architecture. In some this is the chapter that goes in depth on each big component developed in this project.

Chapter 7 is where we demonstrate the results we have obtained, either in the form of graphics or videos which are more self-explanatory and easy to understand.

Chapter 8 mentions the difficulties that developing this kind of project comes with, which are important to mention in order to better understand the time impact that using some of these technologies have.

Chapter 9 contains a list of skill sets that we acquired in the development process of this project.

Chapter 10 is the conclusion chapter to give some closure to this report

Chapter 11 has all the references that can be useful to better understand this report.

Chapter 12 is the attachments chapter, this chapter contains information such as the weekly work and the project calendar.

3. Background and Related Work

In this chapter we will describe some of the related work that we analysed in order to find ways to develop this project and also some of the technologies related to this field of study.

3.1 Unmanned Aerial Vehicles

An unmanned aerial vehicle (UAV), often known as a drone, is a flying machine that operates without any humans on board. These vehicles are used in multiple real life situations, not only helping with surveillance and recognition missions by the military, but also in agricultural contexts and commercial use.

Most systems in which UAVs are integrated include a groundstation control module and a communication channel between both. UAVs have also become more affordable and available to the population, which lead to the development and deployment of new drone solutions.

3.2 Flying Ad-Hoc Networks (FANETs)

In today's world, where recent technology sensors and communication systems, the production of UAVs became more present, however the capability of a single UAV is not adequate, multiple UAVs can make a much more interesting and richer system.

FANET is the concept of having a network that consists of a group of UAVs connected in an ad-hoc. This mode of connection allows the drones to expand their range of communication at an infrastructure-less area which in scenarios of a catastrophic nature prove to be efficient when ordinary communication infrastructures are not available.

3.3 ROS2

The Robot Operating System v2 (ROS2) consists of a software framework collection designed and targeted at the development of robotic systems.

ROS2 provides a solution for the communication between multiple robots in real-time systems with non-ideal networks and production environments. This system is an updated version of the original ROS that was mainly used for centralized systems. Both versions use a publisher/subscription model, in which multiple nodes can publish and subscribe to a topic with a custom data type, such as integers, strings etc.

3.4 Fleet-manager

The Fleet-manager software was developed by Margarida Silva and consists of a full environment designed for the automatic control and manipulation of drones.

The Fleet-manager mainly consists of a grounstation module and a drone module. The grounstation runs on a local server, is responsible for the drone coordination and control and allows not only the programming of autonomous missions but also the development of new plugins. The drone module runs on a single board computer directly connected to the drone, that receives commands from the grounstation but also sends multiple information back, such as telemetry data.

3.3 GStreamer

GStreamer is a framework used to develop multimedia streaming applications that support video and audio. The framework comprises multiple plugins that have different functionalities, such as scaling, encoding/decoding, changing media type etc. The plugins can be linked and arranged in a pipeline. The pipeline with the plugins is responsible for applying multiple transformations, resulting in a processed data stream.

A popular example of software that uses Gstreamer is The GNOME desktop environment, it has included GStreamer since GNOME version 2.2 and encourages GNOME and GTK applications to use it.

4. Context and state of the art (SOA)

On this section we analysed studies, projects and technologies already developed about this topic, we considered the most relevante the next:

Drone-Based Wireless Relay Using Online Tensor Update :
(https://ieeexplore.ieee.org/document/7823731)

On this paper, drones are advocated to serve as mobile relays to forward data streams but some problems exist with that, for example, data transmission may suffer severe signal attenuation due to obstructions and it is also difficult to find an optimal location for drones due to the dynamic and unpredictable environments. To try to solve this problem the authors developed an algorithm that outperforms existing methods in achieving the trade-off between time cost and estimation accuracy.

Performance Improvement of Drone MIMO Relay Station Using Selection of Drone Placement:
(https://ieeexplore.ieee.org/document/8536637)

The objective of this paper is to evaluate multiple-input multiple-output (MIMO) transmission when a small autonomous unmanned aerial vehicle (drone) is used as a relay station. When using drones the propagation loss decreases but it has the issue of increased spatial correlation due to the direct wave. To solve this they introduce a propagation environment control method (PECM) that selects drones in optimal arrangement from multiple drones.

5. Architecture

In this part of the document we will discuss the architecture we proposed to develop in order to achieve all goals we have previously mentioned.

In this part we will document the system requirements, both functional and non-functional, that were gathered for this project, the actores and the use cases and lastly an overview of the system architecture.

5.1 System requirements

In this section we will present the system requirements specification beginning with a brief description on the requirements gathering process, followed by the functional and non-functional requirements, actors and lastly the use case diagrams.

5.1.1 Requirements Elicitation

To gather all the requirements in need we started by meeting with our supervisors, professor Susana Sargento, professor Miguel Luís and Margarida Silva in order to evaluate the goals and to help define the system boundaries.

Then we analysed the state-of-art, that is, researching for studies, technologies and projects similar to the one that is being developed.

At last, a brainstorming session to get all the requirements collected and debate which ones were important or disposable was held.

All of that resulted as a big help for us to define the most important functional and non-functional requirements.

5.1.2 Functional Requirements

The following list presents functional requirements, i.e. features or functionalities that the system must have to allow the user to accomplish their tasks/objectives. We divided the requirements in three main modules, Dashboard, groundstation and Drone:

Dashboard:

- **FD1**: Should be possible to visualize real time video sent from the drones
- **FD2**: Should display information about the mission
- **FD3**: Must be able to display telemetry information from the drones

Groundstation:

- **FG1**: Must be able to process network characteristics between the groundstation and the drones
- **FG2**: Should be able to place relay drones in a optimal position taking into account communication and services parameters
- **FG3**: The system should be able to automatically adapt telemetry information sending ratio
- **FG4**: The system should be able to automatically adapt video parameters

Drone:

- **FDR1**: Must be able to monitor network characteristics between the groundstation and the drones
- **FDR2**: Monitor the network characteristics between itself and the other drones and transfer that information to the groundstation
- **FDR2**: Should be able to send video with different codecs and quality
- **FDR3**: Should be able to adapt telemetry sending ratio according to the groundstation

5.1.3 Non-Functional Requirements

The following list presents non-functional requirements, i.e. define system attributes such as scalability, reliability, usability, security, maintainability, and performance. We divided the requirements in three main characteristics, Usability, Performance and Documentation:

Usability:

- **NFU1**: Provide a simple, complete and intuitive dashboard
- **NFU2**: The system must have a familiar visualization and interaction
- **NFU3**: New functionalities should be effortlessly added on the system

Performance:

- **NFP1**: The wireless network sensor must provide new data automatically
- **NFP2**: The camera must provide new information automatically
- **NFP3**: Relay should be sent almost immediately to the correct position

Documentation:

- **NFD1**: Must have a documentation easy to understand about the dashboard
- **NFD2**: Documentation about all the sensor and specific characteristics
- **NFD3**: Documentation with information about how to use the system
- **NFD4**: Documentation about how the whole system works together and plugins

5.1.4 Actors

The target users of this project are those in the security force, civil protection and any other individual or organization that may need to perform tasks such as surveillance, disaster prevention, patrolling difficult access zones, etc.

The level of expertise required to use the system varies according to the needed features.

For example, less experienced users may easily retrieve information from the system, such as sensor information, video footage or overseeing a mission, as these are straightforward tasks. However, the user must have some knowledge in order to define the missions or set up the drones.

With that in mind the main actors are described in the following list:

- **Security Guard**: Represents security forces that use drones equipped with video cameras and other sensors to perform tasks such as monitoring public gathering restrictions or surveillance missions.

- **Civil Protection**: Make use of drones to perform missions that may help preventing disasters or aid in an emergency, such as monitoring forest areas or supporting search-and-rescue missions.

- **Drone Administrator**: Users capable of defining the missions, setting up the drones, handling other maintenance issues and overseeing the missions.

5.1.5 Use Case

The use case model consists of one package which is the web application(dashboard) that will provide the exchange of information between the users and the drones.

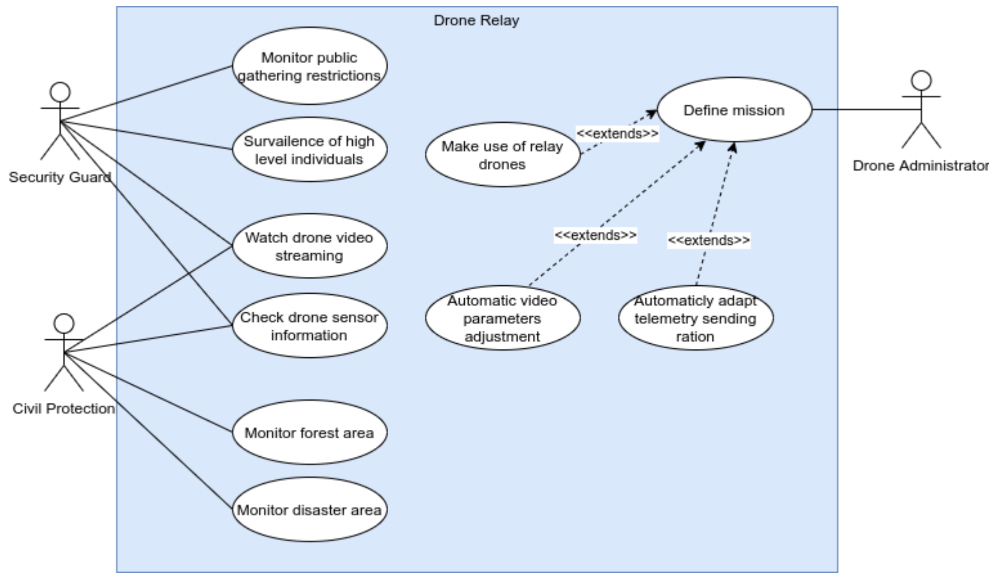

Drone Relay
Security Guard
Civil Protection
Drone Administrator
Monitor public gathering restrictions
Survailence of high level individuals
Watch drone video streaming
Check drone sensor information
Monitor forest area
Monitor disaster area
Make use of relay drones
Define mission
Automatic video parameters adjustment
Automaticly adapt telemetry sending ration
<<extends>>
<<extends>>
<<extends>>

Figure 1: Use case

Drone Administrator

- **Define mission:** program the mission which the other users will be able to use and provide ways for them to provide the information for the program to execute (eg: area of the search).
- **Automatic video parameters adjustment:** configure how the video parameters change for the given network.
- **Automatically adapt telemetry sending ratio:** configure how the telemetry will change accordingly with the given network.

Security Guard

- **Monitor public gathering restrictions:** allow the security guard to check if public gathering restrictions are being followed by, making it easier and faster for the officer to find and attend such events.

- **Surveillance of high level individuals:** provide easy access for officers to follow and watch high level interest individuals.

Civil Protection

- **Monitor forest area:** allow the civil protection user in charge to check a forest area after a fire or a similar event.

- **Monitor disaster area:** allow the civil protection user in charge to check a specific area after/during a disaster or a similar event.

User

- **Watch drone video streaming:** allow the user to get access to the drone live footage or in passed missions.

- **Check drone sensor information:** show current sensor information and last readings from the current mission as well as past missions.

5.2 System architecture

5.2.1 Deployment Diagram

This Figure 3 illustrates a simplified deployment diagram of this project. The aim of this diagram is to capture main components of the deployment, using a simple, easy to understand language.This diagram is further detailed in Figure 2, that elaborates on the internal interconnects of each main component.

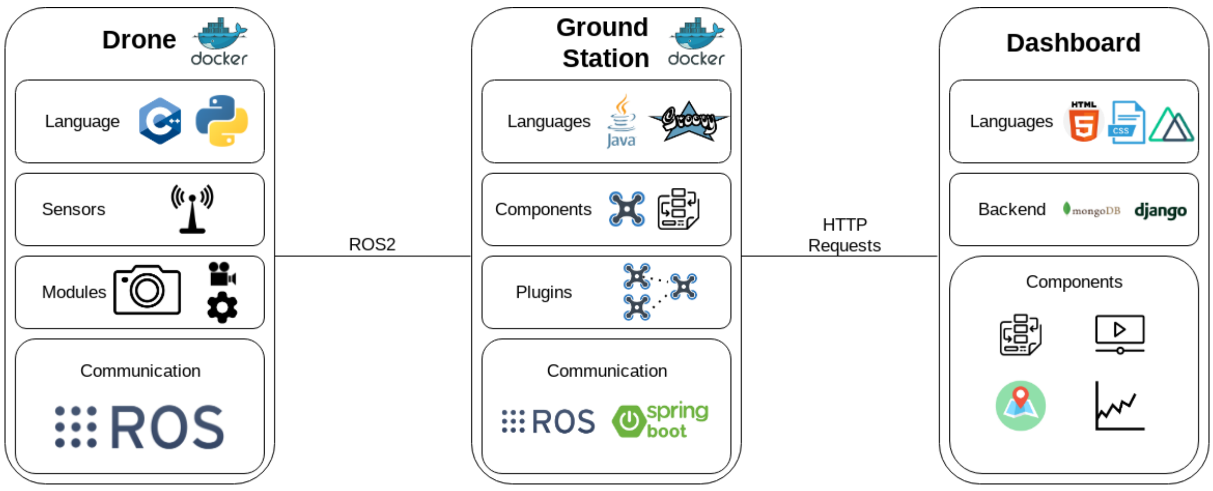


Figure 2: Simplified deployment architecture

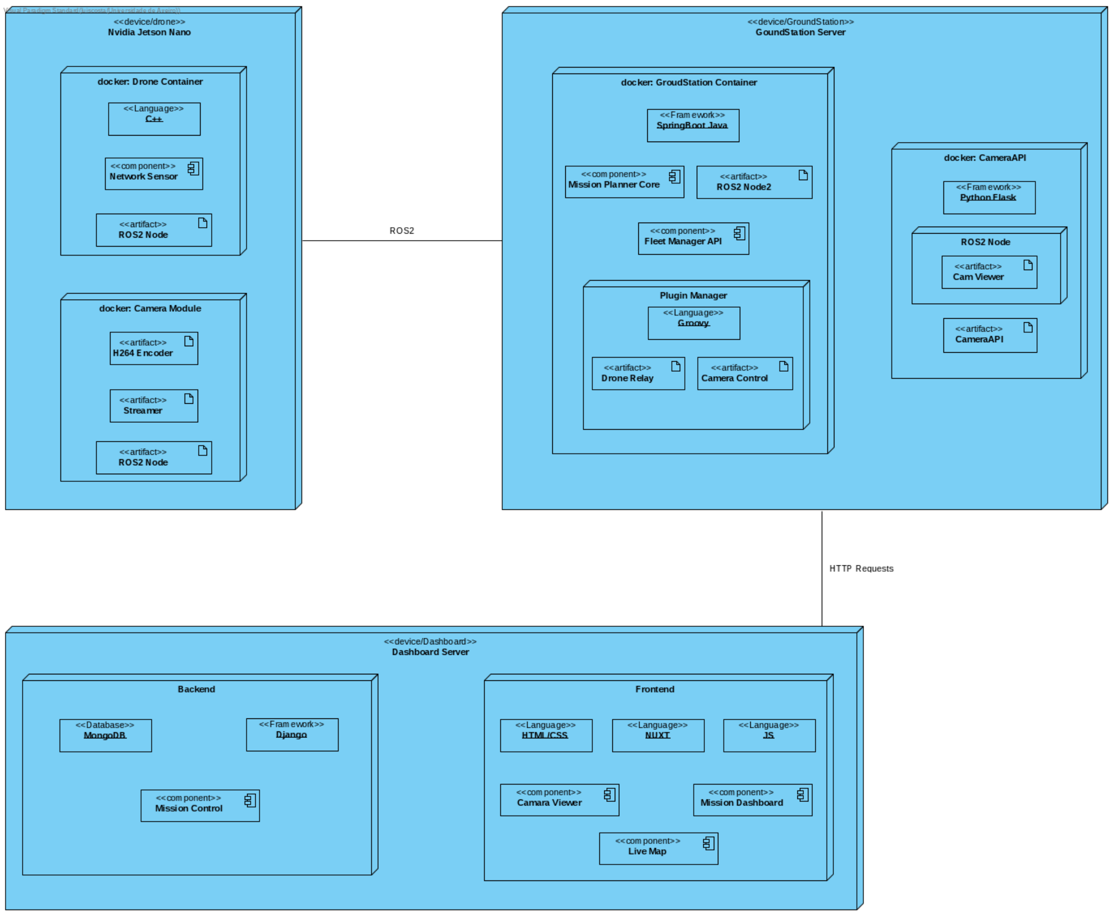


Figure 3: Deployment architecture

The system architecture presented in the figures above, show the three main components of this project:

- **Drone (Nvidia Jetson Nano):** This module consists of 2 docker containers, the first is responsible for the control of the drone main board with the use of mavsdk, a C++ library, and the second responsible for the camera capture, transmission and dynamic update of the video stream quality.

The communication is provided via a ROS2 node that communicates (via subscriptions and publishes) to the groundstation. The wireless network sensor runs via a python script that transmits information about the network signal and its status.

- **Groundstation:** This module consists of a docker container which contains a Flask Server and a SpringBoot Server.

The SpringBootServer consists of a previously developed fleet-manager that is used to control, organize and coordinate drone missions. The system allows for the development and use of plugins. The standard plugins created are the drone relay, which using artificial intelligence automatically positions relay drones during missions, so that the mission drones never lose connection to the groundstation and optimize the number of relay drones.

The flask server is used to transmit video feed and control quality from multiple drones to be displayed in the dashboard.

- **Dashboard**: This module consists of a developed frontend and a backend, that the user will use to interact with the mission, sensors and camera system.

The backend uses mongodb as a database and django to connect to the groundstation for control and management.
The frontend is made with Nuxt, which is a framework that extends vuejs made to create interactive SPA's, and allows the upload and tracking of missions, connect to multiple drones and groundstations, displaying a map for easy interpretation.
This dashboard also contains a section dedicated to the display and control of the drone camera quality settings, and multiple charts that illustrate the continuous status of the attached drone sensors.

5.2.2 Domain model

The domain model presented in Figure 4 describes the entities, roles and relationships of the system. This model will help to better understand the problem.

To begin with, there is a Drone administrator which, as described in the previous chapters, is capable of setting up the missions and configuring the groundstation. The groundstation then through the configuration will make use of one or more drones to coordinate the missions defined by the drone administrator.

As for the users represented by the security guard and civil protection,they will oversee the mission in order to, for example, retrieve information from the system, such as sensor information or video footage.

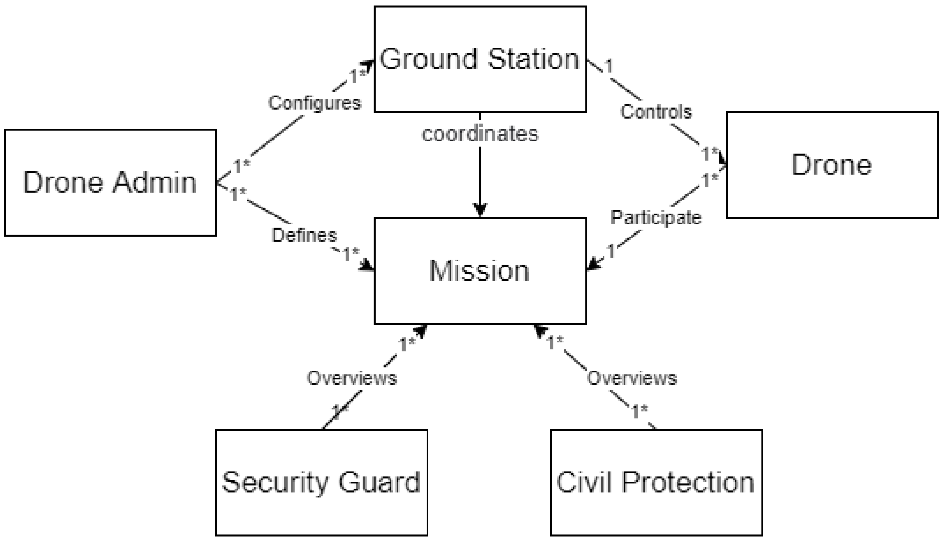


Figure 4: Domain Model

6.Implementation

In this part of the document we will explain and demonstrate how each component of this project was built, how it can be deployed and how they interact with each other.

We will start by documenting the mission algorithm used to place the relay drones in a optimal position taking the network quality into account, followed by the camera integration and its quality adaptation, the wireless network sensor used to retrieve the network parameters and lastly the telemetry module which changes the sending ratio of the sensor messages and telemetry messages according to the quality of the network at a given moment.

6.1 Mission Algorithm Module

6.1.1 Introduction

This module aims to create an algorithm to find the close to optimal position for relay drones in order to expose mission drones (specific drones that are performing a programmed mission) to the best network connection possible with the least amount of drones. To achieve that goal, drones will use a wireless network sensor, also developed and explained further ahead to analyze the quality of the network for the surrounding drones, then the algorithm finds the best position (detailed further ahead).

6.1.2 Simulator

In the beginning of this project a new simulator was developed to be able to simulate more complex environments because the old simulator, even though it is more simulation accurate, lacks performance because of all the physics simulation.

For that reason, in the new simulation we assume all the physics is already secured and also all the controls that maintain the drone operational. In other words, the only focus of this simulator is to simulate as many drones as possible with the least amount of overhead possible.

The simulator was created in python and made with pygame.
In order to operate the drones there are 2 scripts which will be running in the groundstation, one by the programmer and the other by the relay algorithm:

- **Load mission**: programs the mission drone movimentations, can also assign which drones contain self sustaining connection
- **Relay**: performs the algorithm to extend the network for all mission drones which need faster/better connection.

6.1.3 The First Algorithm

This strategy was composed of 3 major steps, as this algorithm is not the final one, it was opted a more superficial analysis:

1. Calculate the Points Of Interest (POI), this is useful because with this technique it is possible to choose where the relay drones should be placed, for example, if we only want the relay drones above streets, we can program the function that gets those points of interest that way.
2. Analysing how the network is performing and estimate the best positions possible based on the POI calculated before, then a tree search algorithm will evaluate which coordinates to choose to create the best network possible.
3. After getting the coordinates to have the best possible network, we use another tree search to assign each drone to one of the coordinates chosen before

Advantages and Limitations of the First Algorithm

The main advantage of this algorithm is the intrinsic flexibility and creativity. It has the capabilities to create more robust and creative network configurations when never thought of situations occur because of its nature of searching through states until an optimal condition is met. For performance reasons, this strategy only works when using A Star algorithms or similar because of the exponential growth of possibilities. It's a powerful technique and if well implemented and optimized can bring immense value to a project like this.

However, even though the first step is reasonably fast, this strategy is very time consuming in the second step and for that reason is not able to efficiently calculate the close to optimal network for real time applications with more than 8 drones with a laptop with a Intel i5 8300H processor and a 4Ghz clock. Even though this strategy could be perfected, there would always be a substantial delay when considering more than a few drones because the complexity of such an algorithm would be $O(n^{n})$ which means that with just 8 mission drones in the worst case would have to consider $8^{8} = 16777216$ configurations just to run the second step.

Strategies were used to optimize the search for example A star searches and tree trimming, a great deal of progress was made in the optimization steps, however this problem proved hard to overcome even with strategies developed specifically for this type of issues and for that reason, this strategy was dismissed.

6.1.4 The Second and Final Algorithm

A new algorithm was developed taking into account the previous attempt and for that reason, the new version does not have complex tree searches although some inspiration came from the last version, for example the tree like organization and child-parent relations between the abstractions mentioned ahead.

This algorithm has a complexity of O(2N) which is extremely fast and useful for real time applications because even low performance computers would keep track of thousands of relay and mission drones without a relevant delay in the output of instructions, essentially in the practice there is no limit to the number of drones to manage both in the mission drones and relay drones. It was tested successfully in a simulator and in real life experiments.

6.1.5 File structure

There are two types of files:

- **Mission:** set of instructions programmed by the user, the drones used in this script will be called mission drones, eg: mission.groovy
- **Plugins:** code executed in a looping thread which receives the objects of all active drones and can alter the drones movimentations or proceed to monitor drone sensors and state, eg: RelayPlugin.groovy.

6.1.6 How to setup

When creating a mission, plugins can be enabled anywhere in the file:

```
enable plugin_in_question [ , argument: value ]*
```

and disabled by:

```
disable plugin_in_question
```

In the case of the RelayPlugin when the relay is needed the plugin can be enabled by:

```
groundstations = [
    [
        id:'GroundStation',
        position: [
            lat: 40,
            lon: -8.6
        ]
enable 'relay', rate: 1.s, groundstation: groundstations, simulation: true
```

Note: both the simulation and the rate have default values

- Simulation: false
- Rate: 1s

We use the description below to program the interface to use in the call of the plugin. The relay algorithm maintains global variables added in the *plugin.vars* to continue the optimization of the network between thread calls, from the user script these variables cannot be accessed or changed.

```
plugin {
    id       'relay'
    type     scheduled
    input    rate: Time,
             groundstation: [Map, [
                 id: "groundStation",
                 position: [lat: 40.633874, lon: -8.660311]]
             ],
             simulation: [Boolean, false]
    vars     network: [:],
             groundstations: [:]
    callback relay
    init      init_state
}
```

6.1.7 Explanation and modules

All explanations will be in a high level format in a way not to overwhelm the reader.

We will start by presenting the abstractions used in this algorithm, after that we will explore how the algorithm works.

Finally, how the network keeps the drones efficient when they need to move.

Abstraction of objects

Abstractions were used to provide programmers the ability to have a general idea of what the problem is and how to solve it in a high level way of thinking. The removal of specific details which were not needed to keep track when thinking about the solutions means that the solution will be more general and easy to understand and maintain.

- **Entities:** any object that can extend the network connection.

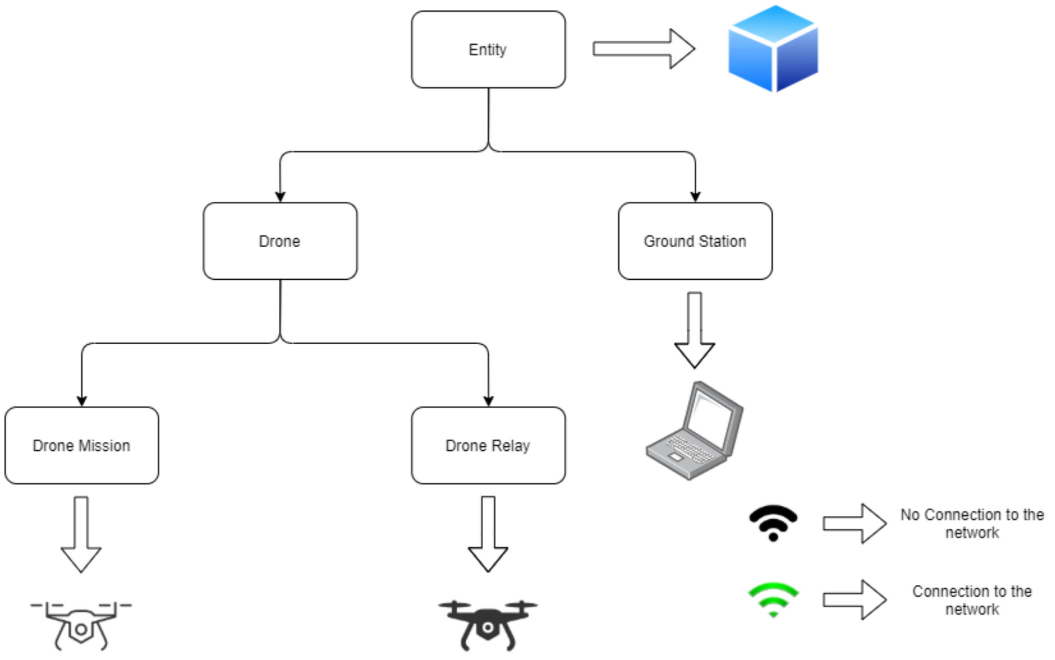


Figure 5: Entity Abstractions

- **Relay Link:** connection between one entity and another entity

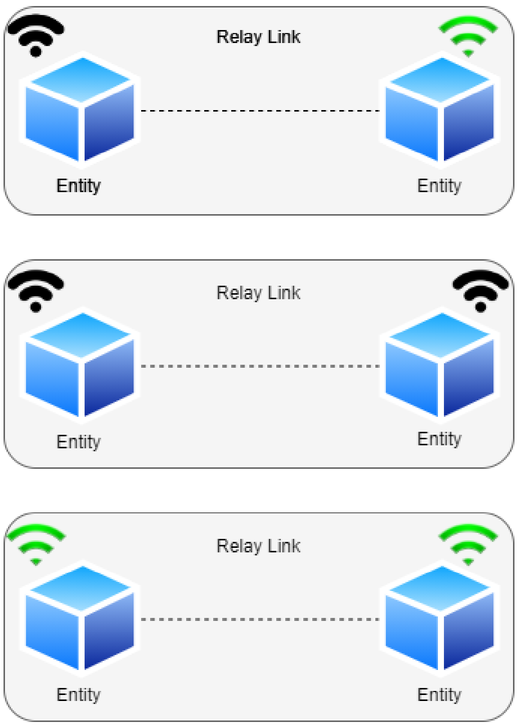


Figure 6: Relay Link

- **Drone Relay Bridge:** connection between one mission drone and one entity

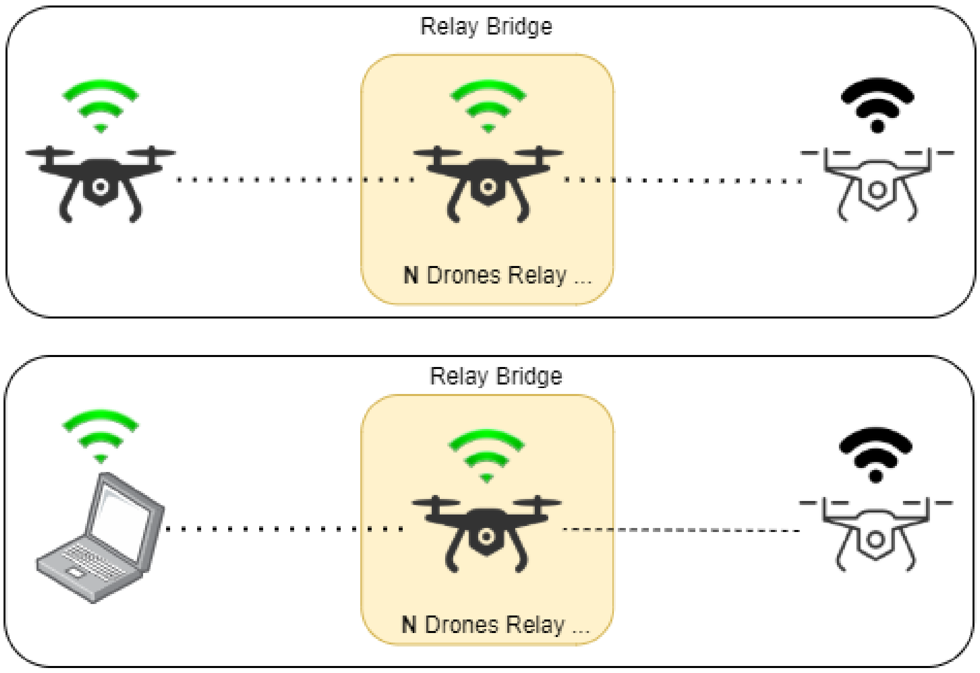


Figure 7: Relay Bridge

- **Network State:** represents the whole network with multiple trees, which each root of a tree would be the generic entity with connection.

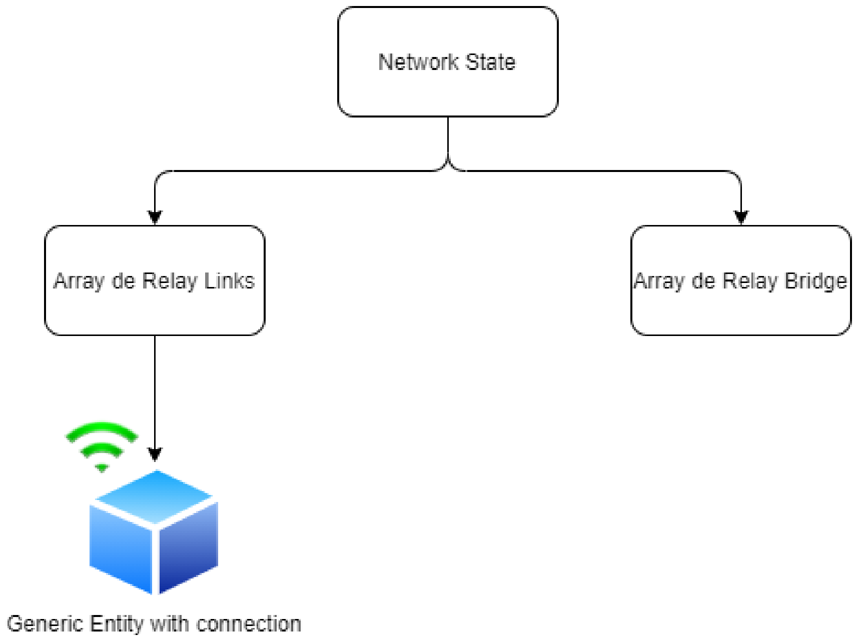


Figure 8: Network State

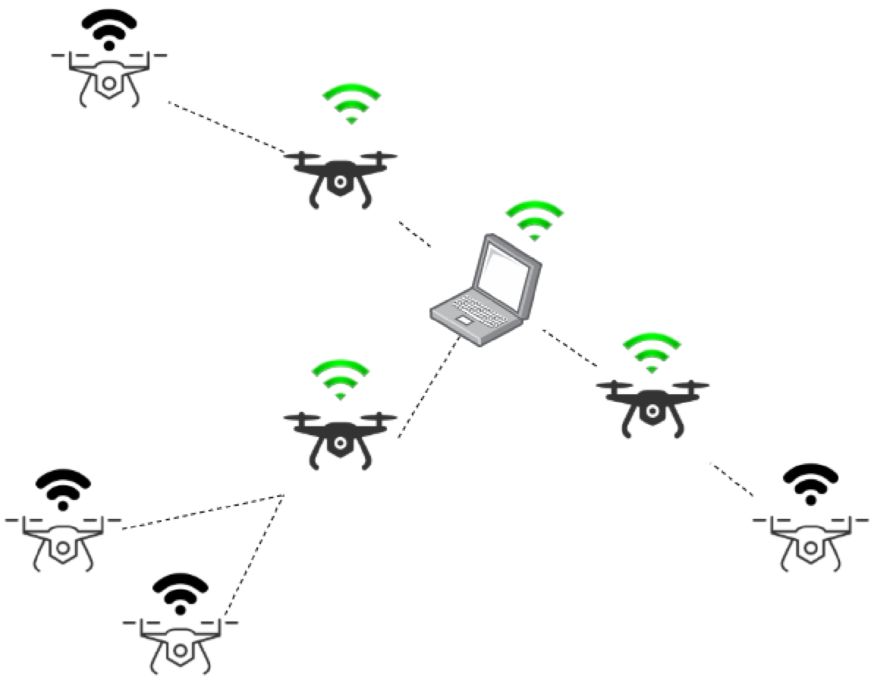


Figure 9: Overview Mission Algorithm

As seen above, the Network State can be compared to multiple Trees which have the purpose of expanding the network so that mission drones maintain their connectivity. These trees are continually maintained and optimized in a way that their leaves get the best network possible. The root of each tree is a self-sustaining connected entity.

Mission algorithm

There are two main software modules which are the functions called by the Plugins Manager:

- **Initialization**: function used to prepare the variables to use in the algorithm for the first time.
- **Relay Loop**: function running in a looping thread, actively monitoring and managing the fleet of drones.

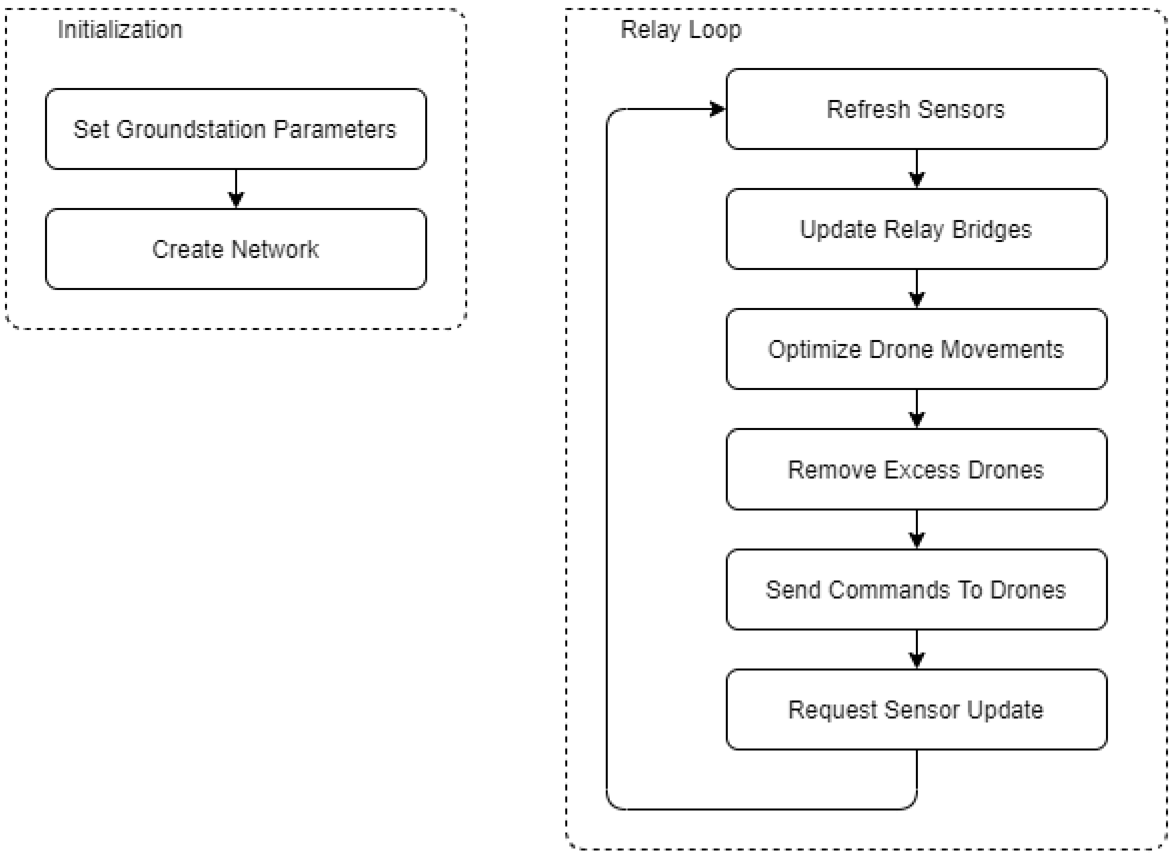


Figure 10: Main Software Modules

Initialization

In this step we should take into account that the code might be running in the simulation or in real life, for that reason we should prepare the parameters for both situations.

In the case of the simulation we check if the user added one ip and mac, if they did then we use it, otherwise one random is fine because it will not be used anyways and will only be helpful for debug purposes.
In the case that is in real life then the user must insert both the mac and ip of the network interface of the groundstation.

```
if(simulation) {
    groundstation.params = [
        ipAddress: groundstation.containsKey('ip') ? groundstation.ip:
'0.0.0.0',
        macAddress:groundstation.containsKey('mac') ? groundstation.mac:
'0.0.0.0',
        tags: []
    ]
}
else {
    groundstation.params = [
        ipAddress: groundstation.ip,
        macAddress: groundstation.mac,
        tags: []
    ]
}
```

The information of the ip and mac can be found in linux with the command:

```
ifconfig
```

and in windows with:

```
ipconfig
```

Relay Algorithm

Note: All code is thoroughly commented for an easy understanding.
The Relay Algorithm will be in this function and only be stopped when the mission is complete or some critical error occurs.

All modules will be analysed in order to better explain how the algorithm handles generic objects:

1. **Refresh Sensors**

Loop through all the available drones in the network and get the last requested sensor information taking into account that if it could be a simulation.

```
for(relay_bridge: network.relay_bridges) {
    for(relay_link: relay_bridge.relay_links) {
        if(relay_link.child is not null)  {
            network_info = sensor.(relay_link.entity.id).network
            if(network_info is not null) {
                if(simulation) network_info = network.simulated
                else network_info = network.real
                network.networks[drone_id] = network_info
            }
        }
    }
}
```

2. **Update Relay Bridges**

Add new mission drones which could have been assigned

```
while(mission_drones_to_update.size()>0) {
    // get closest drone to connected drones, returns the drone and the relay
link in a dictionary
    closest = get_closest_mission_drone_to_connected(connected_relay_links,
mission_drones_to_update)
    // update relay bridge of closest drone
   relay_bridges[closest.drone.id].update_relay_bridge(closest.relay_link,
networks)
    // add relay bridge to the stack
    bridges_stack.add(0, relay_bridges[closest.drone.id])
    // update connected coords
    for(relay_link: relay_bridges[closest.drone.id].relay_links)
        connected_relay_links.add(relay_link)
    // remove drone to eventually end the while loop
    mission_drones_to_update.remove(closest.drone)
}
```

The function update_relay_bridge is a method of the DroneRelayBridge and has the following functionality:

1) Predict where the drone will be in the next seconds (determined by the prediction time attribute in the relay link abstraction)
2) Calculate the unitary vector from the drone to the connected entity
3) Change bridge positions based on the network which alters the distance from the parent to the child entities, as explained in the subsection Drones Network Adjustments.

4) Signal the drones which are not needed anymore because the child drone already has a better connection to a parent node than to their own parent or has a connection to a parent better enough to remove the parents in between.
5) In the case that the last drone of the bridge has a connection to its parent worse than the network_threshold attribute in his RelayLink object, then a new drone will be requested. This drone will only be assigned at the end of the algorithm.

3. **Optimize Drone Movements**

Reposition drones in order to reduce time of adjustment.
We already have the optimal location of drones and which drone should move, the calculations in this step focus on improving how the drones will move to the coordinates, because we could have multiple positions to occupy and for that reason the ground station will run from the leafs to the core assigning and moving drones to get the best results, in the default case the parent will be selected

```
while(network.bridges_stack.size() > 0) {
    for(relay_link: network.bridges_stack.pop().relay_links) {
        // if the child is not null, it means the child has already a drone
assigned to that position
        if(relay_link.entity is not null) continue
        temp_relay_link = relay_link.child
        // get closest drone relay returning base
        returning_base_relay_drone =
network.get_closest_returning_base(relay_drones, relay_link.coordinates)
        // go through the relay links until reaching the links which have
connection
        while(not connected_entities.contains(temp_relay_link.entity)) {
            // if the child relay link has a relay drone ready to be used
            if(temp_relay_link.entity is not null and
relay_drones.contains(temp_relay_link.entity)) {
                // place returning drone relay
                if(returning_base_relay_drone.drone is not null) {

distance_to_final_position=calculateDistance(temp_relay_link.entity.position,
relay_link.coordinates)
                    // and relay drone returning is closer that the child
                    if(returning_base_relay_drone.distance <
distance_to_final_position) {
                        // add returning drone relay to the network

relay_link.update_entity(returning_base_relay_drone.drone)
                        break
                    }
                }
                // switch drones positions with child
                relay_link.update_entity(temp_relay_link.entity)
                temp_relay_link.update_entity(nothing)
                break
            }
            temp_relay_link = temp_relay_link.child
        }
    }
}
```

4. **Remove Excess Drones**

   If the network is no longer using resources (drones), then those drones can return to the landing coordinate where they started

```
for(drone: relay_drones) {
    if(not network.contains(drone) ) {
        return_home(drone)
    }
}
```

5. **Send Commands To Drones**

   Send drones to their respective positions, when drones have not been dispatched yet we should do one of the following: assign a new drone if there is not a close enough drone to carry on the mission or get another drone close enough that is returning home. Otherwise, the parent drone (network side) will be selected to move.

6. **Request Sensor Update**

   Simply loop through all drones requesting new network information to all drones but itself, this is useful because in the next iteration we can't be sure which drone will be used to make the relay link (because anything can happen).

```
 for(relay_bridge: network.relay_bridges) {
    for(relay_link: relay_bridge.relay_links) {
        if(relay_link.child is not null)  {
            entity_ids = []
            entity_ips = []
            entity_macs = []
            for(entity: drones + groundstations) {
                if(entity.id is not relay_link.entity.id) {
                    entity_ids.add(entity.id)
                    entity_ips.add(entity.params.ipAddress)
                    entity_macs.add(entity.params.macAddress)
                }
            }
            send droneId: relay_link.entity.id, connectionsId: entity_ids,
connectionsIp: entity_ips, connectionsMac: entity_macs
        }
    }
}
```

It is easy to modify network parameters simply by changing the attributes in the RelayLink object.

```
double optimal_network = 65;
double network_threshold = 78;
double maximum_distance = 70;
double minimum_distance = 10;
double target_desire = 1/100;
double prediction_time = 2;
double acceptance_radius = 5;
double offset = 10;
```

Drones Network Adjustments

The relay drones ought to reach their goal destination as fast and battery efficient as possible in order to maintain both the maximum time in the air and provide fast connection. For that purpose there was a need to use a method to provide the drones with a perception of how they must move to maintain their optimal network parameter as dynamic as possible while also maintaining efficiency so a function returning the distance to which the drone must deviate from the current distance to the parent was created, using the Gauss Error Function to provide a PID like behaviour with hard limits.

$$offset \times \frac{2}{\sqrt{\pi}} \int_0^x e^{-(x)^2} dx$$

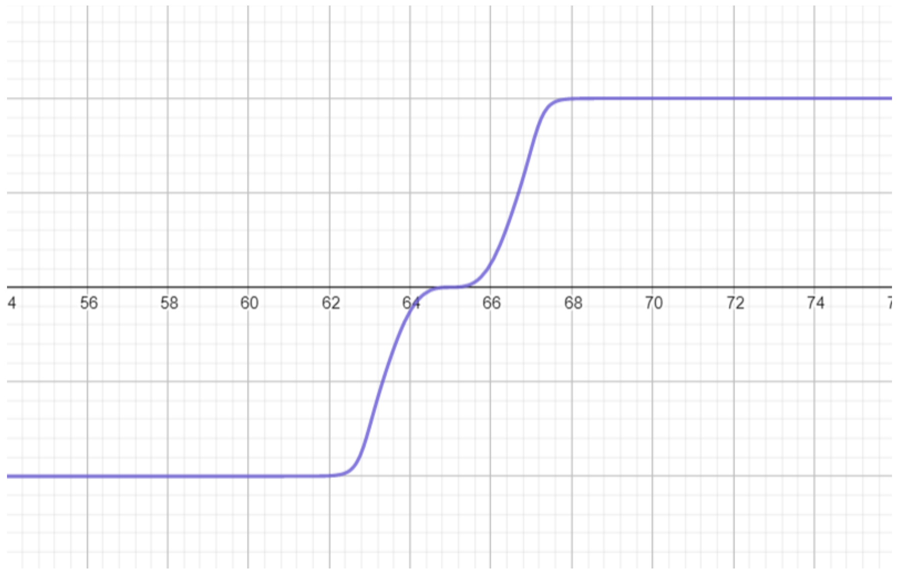
Then substitute the variable to the cubic function below

$$x = t \times (c - o)^3$$

$$\begin{gathered} where \\ o = optimal\ signal \\ c = current\ signal \\ t = target\ desire \end{gathered}$$

Applying some approximations in order to use this in our algorithm

$$offset \times \frac{2}{\pi} arctan(2 \times t \times (c-o)^{3} \times (1 + (t \times (c-o)^{3})^{4}))$$



Note: Taylor Series were not used because the sum wouldn't converge and also because it's faster to compute one function than to compute a sum with multiple terms.

The formula was applied in order to rapidly adjust the drone in a changing environment without sacrificing efficiency because it isn't moving pointlessly trying to get small gains in the network signal in a trade off for lowering battery efficiency. The distance to his child entity is maintained when the target network signal is reached and bigger changes happen when more drastic network changes occur.

6.1.8 Future Work

In order to improve even more efficiency to the network, each relay drone should be able to better adjust to multiple drones connection if it's worth to connect multiple drones to the same relay drone, that way we would create a dictionary with the the relay drones as keys and add an array of drones to the object and then would be calculated how much the relay drone would be moving from their previously calculated point to also provide better connection to their neighbors while maintaining a good connection to their previous parent and only child (network side entity).

This work would provide faster transitions so the overall network maintains the flow of information because the requested drone would already be moving in the desired direction (battery efficient) and also closer to their destination while the child or new assigned drone would also have a closer destination (lower travel time) because the network is maintained from the leafs to the core and the assignment of the drones are from the core to the leafs.

6.2 Video Quality Codec Module

6.2.1 Introduction

This module is responsible for the capture and transmission from the camera that is connected to the drone controller (Nvidia Jetson Nano), to a local grounstation that runs the dashboard software. It is also responsible for dynamically changing the video quality feed, according to the network state.

When the wireless network sensor is active, the quality will be selected from 3 presets (High, Medium, Low), changing automatically the adequate preset in order to maintain the best quality/reliability ratio.

The encoder chosen for this module was the x264 Encoder, it's compression efficiency, popularity and previous work developed by other colleagues are the main reasons for this choice.

6.2.2 File structure

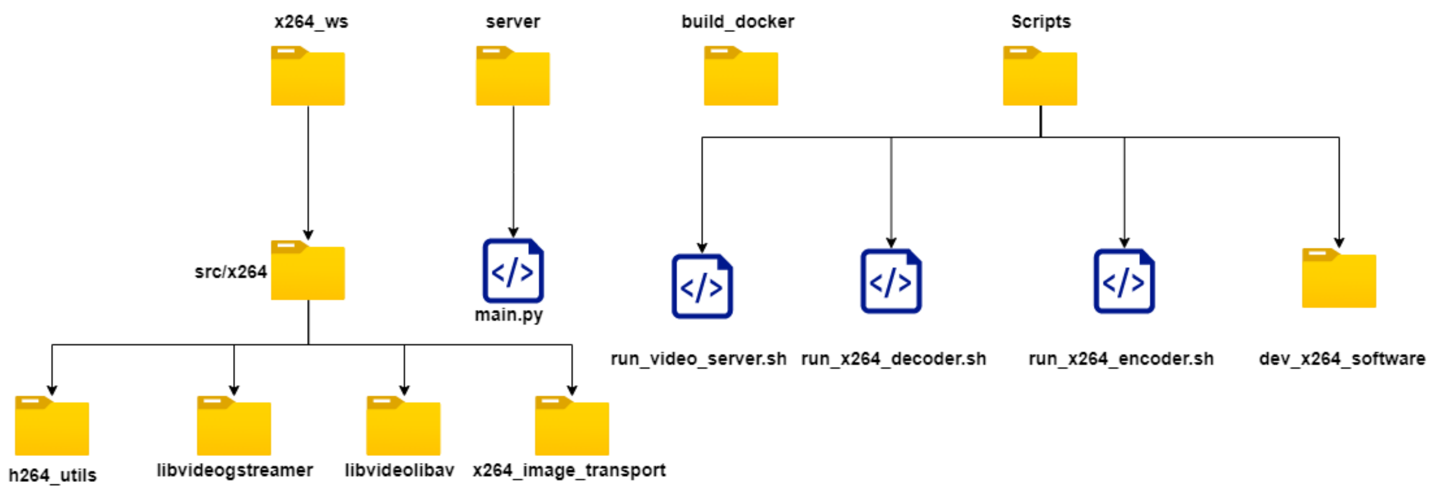


Figure 11: File Structure Video

- run_x264_encoder.sh - This executable builds and starts the x264 hardware accelerated encoder, it needs as argument the drone id (-n <droneid>).
- run_x264_decoder.sh - This executable builds and starts the x264 hardware accelerated decoder, it needs as argument the drone id (-n <droneid>).

- run_video_server.sh - This executable starts the x264 flask server, it needs as arguments the local ip of the non adhoc network (-a <ip>) and the port (-p <port>)
- h264_utils - This folder contains the code that starts the ROS2 node and instantiates the publishers/subscribers for the encoded image transmission.
- x264_image_transport - This folder contains the ROS2 plugin x264 used in the h264_utils publisher/subscriber, it uses an encoder/decoder instance that can either be gstreamer or avlib.
- libvideogstreamer / libvideoav - These folders contain the encoder/decoder instances, that can either be gstreamer for hardware accelerated encoding or AVLib for software encoding.
- dev_x264_software - WIP Folder to allow the software encoding / decoding
- server - Contains the flask server files.

6.2.3 Required components

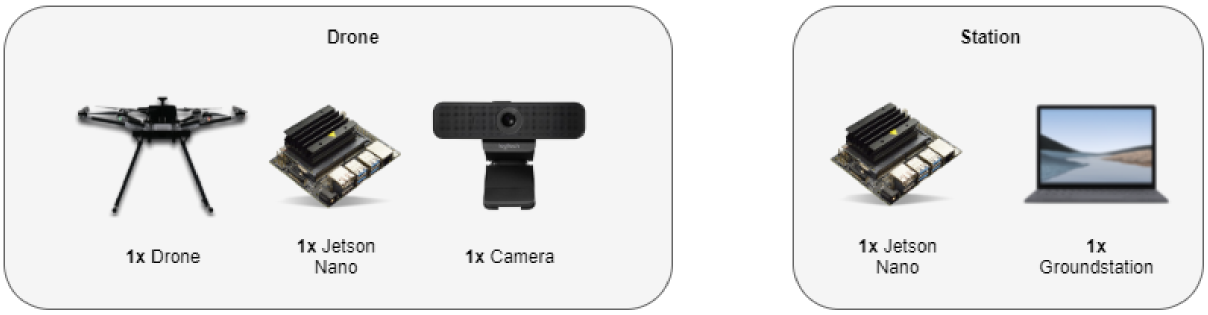


Figure 12: Required components for video

For the execution and testing of this module, the items required are presented in Figure 12.

The drone will be equipped with the Jetson Nano and the camera. The camera will be connected to the Jetson Nano that is responsible for the transmission and encoding of the video in this case.

The station that is located on the ground will also be composed of another Jetson Nano and a laptop running the grounstation and dashboard software. The Jetson Nano is responsible for the video decoding and transmission to the groundstation, which will show the video feed in the dashboard.

6.2.4 How to setup

Nvidia Jetson requirements:

- Gstreamer (pre-installed on Jetson Nano)
- FFmpeg (pre-installed on Jetson Nano)
- Python3 Flask and Opencv
- ROS2 Dashing and ROS2 Image Transport

Jetson Ubuntu Install:

ROS2: https://docs.ros.org/en/dashing/Installation/Ubuntu-Install-Debians.html (Use the dashing version)

ROS2 Image Transport:

```
sudo apt-get install ros-dashing-sensor-msgs ros-dashing-cv-bridge
ros-dashing-image-transport -y
```

Flask and Opencv:

```
pip3 install flask opencv-python
```

Setup:

1) Connect webcam to the drone Jetson Nano via usb cable.

2) Ensure that both the Jetson Nano's and the groundstation are connected to the drone's ad hoc network.

3) Run the following command on the drone's Jetson Nano.

```
./run_x264_encoder.sh
```

4) Run the following command on the station Jetson Nano.

```
./run_x264_decoder.sh
```

5) Setup the new network (different from the adhoc network) on the decoding Jetson. It's recommended to connect an ethernet cable between the decoding Jetson and the computer, manually setting ip's, as to use in the flask server.

6) Start the flask server, by running the following script, changing the ip and the port to the ones used.

```
./run_video_server.sh -a <ip> -p <port>
```

7) Open the groundstation dashboard, select the drone that contains the camera, and click on the camera tab.

6.2.5 Explanation

Overview:

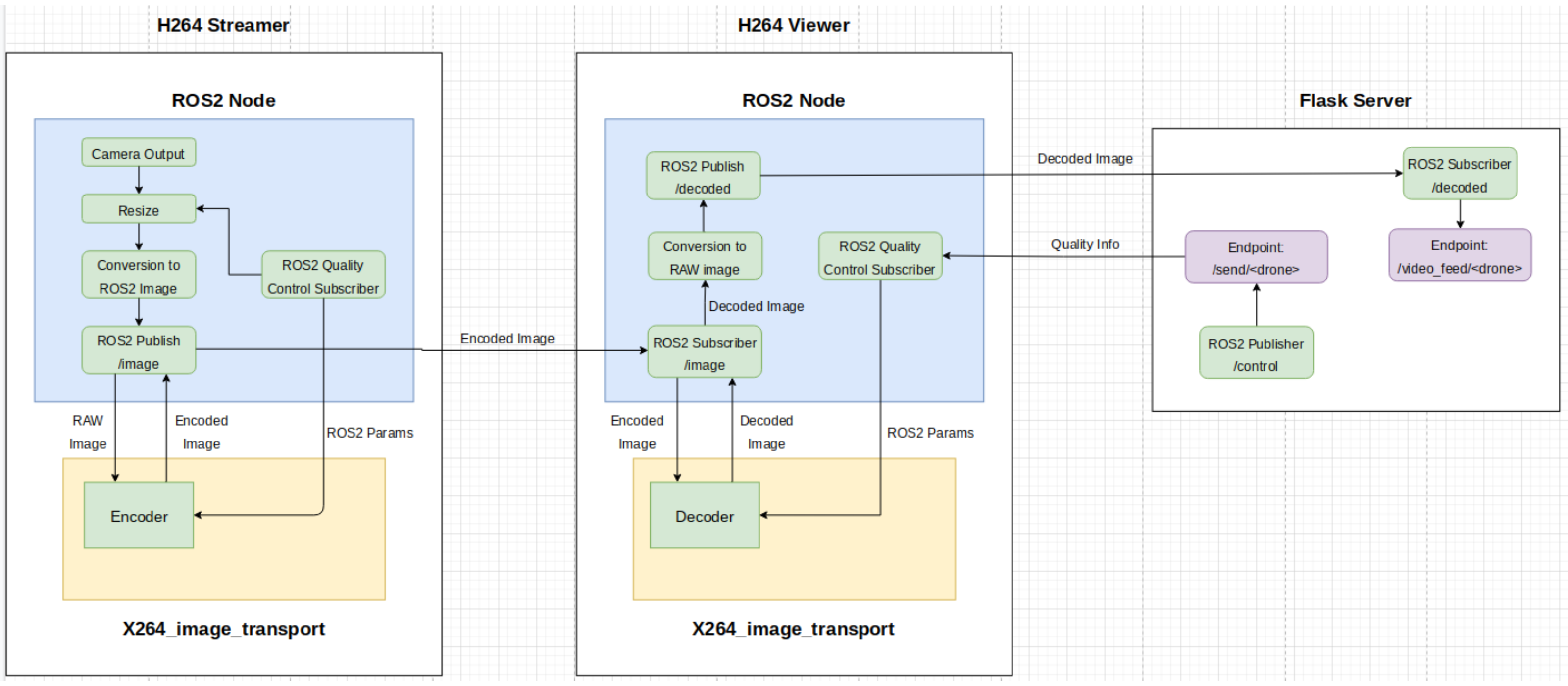


Figure 13: Video Transmission Architecture

The architecture of the video transmission, encoding and decoding can be seen in the figure 13. This architecture is divided in 3 main components:

- H264 Streamer - Responsible for capturing the raw camera feed from the camera, and performing the encoding for the video feed to be transmitted.

It is also responsible for changing the encoding video feed quality dynamically. This component of the software will run on the drones Jetson Nano.

- H264 Viewer - Responsible for receiving the feed from the h264 streamer and decoding the video feed, to be transmitted to a flask server.
  It is also responsible for changing the decoding video feed quality dynamically. This component of the software will run on the station Jetson Nano.

- Flask Server - Responsible for receiving the decoded video frames, and providing endpoints for the view of the video and quality control from the dashboard (or automatic video quality adjustments).

For both the H264 Streamer and Viewer, the ROS2 Nodes are responsible for the transmission and receiving of information via the publish or subscription of topics. The x264_image_transport section is responsible for linking the image publisher to the encoder that encodes the image first for publishing.

Encoder / Decoder:

In the previous diagram (Figure 13), in the x264_image_transport section, it is described as the encoder / decoder block. In this section we will provide a deeper dive into both of these blocks.

For the current implementation two types of encoders / decoders can be used:

- Gstreamer - implementation that uses OpenMAX plugins (omxh264) to leverage hardware acceleration provided by some NVIDIA boards such as the Jetson Nano

- LibAv - implementation that is used to encode and decode H.264 via software. This library has the widest compatibility range, but is significantly slower compared to the hardware accelerated options.

Both of these solutions are used in similar ways, first the pipeline must be instantiated and setup with the desired encoding parameters. After this, the information from each RAW image is transported through the pipeline yielding the encoded image. A similar process occurs in the decoding of an image, we create a pipeline for the image decoding, feed the encoded image to the pipeline yielding back the RAW image prepared to be viewed.

At the moment, the software decoding module only encodes/decodes one quality preset, as the encoder that runs on the jetson is hardware accelerated, the software decoding component of the pipeline must be configured properly as to not create artifacts in the decoded image.

Gstreamer Pipeline:

For the image encoding/decoding, as the main focus is to encode/decode with hardware acceleration provided by the NVIDIA Jetson Nano, the software used for this processing was Gstreamer.

Gstreamer uses multiple components that connect with each other, creating a pipeline to manipulate a video feed.

In this project, two main pipelines were created to encode and decode the video feed from the drone.

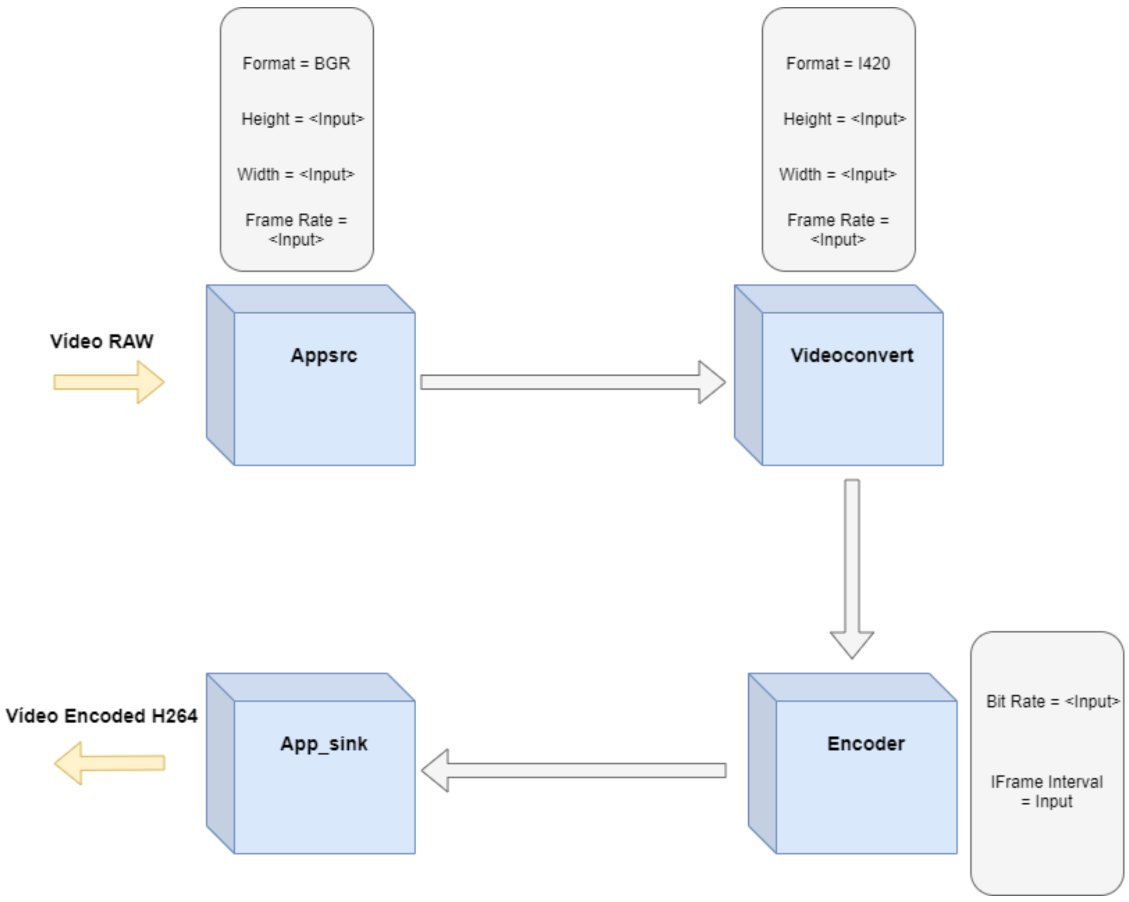


Figure 14: Video Encoder

The first pipeline as shown in figure 14 is used for encoding, the gstreamer components used are the following:

- Appsrc - The appsource is used to initiate the gstreamer pipeline and manually input the each frame data directly from the camera (via OpenCv).

- Videoconvert - A gstreamer component responsible for converting between different frame (images) formats. This is used in the pipeline to convert the pixel format to I420, that is one the available inputs for the next component (encoder)

- Omxh264enc - Gstreamer component responsible for the hardware encoding of the video, only available in the NVIDIA Jetson Nano, or equivalent hardware. This component needs to be configured with the iframeinterval and bitrate desired for the encoded stream.

- App_sink - Gstreamer component used to retrieve the encoded frames.

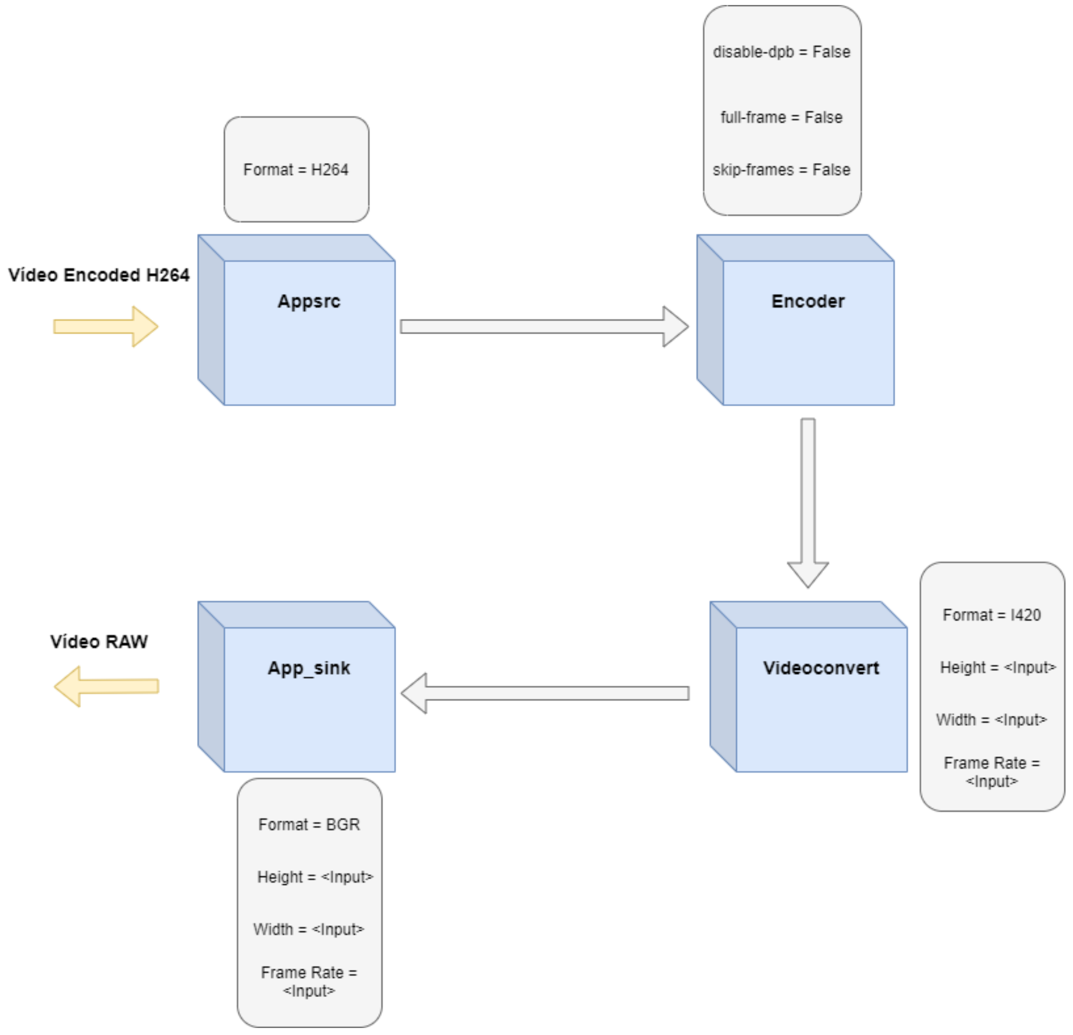


Figure 15: Video Decoder

The second pipeline as shown in figure 15 is used for decoding, the gstreamer components used are the following:

- Appsrc - The appsource is used to initiate the gstreamer pipeline and manually input the each encoded frame data from the ros2 message.

- Omxh264dec - Gstreamer component responsible for the hardware decoding of the video, only available in the NVIDIA Jetson Nano, or equivalent hardware. This component needs to be configured with the disable-dpb, full-frame and skip-frames flags.

- Videoconvert - A gstreamer component responsible for converting between different frame (images) formats. This is used in the pipeline to convert the pixel format from I420 to BGR, to be displayed.

- App_sink - Gstreamer component used to retrieve the decoded frames.

Dynamic Quality:

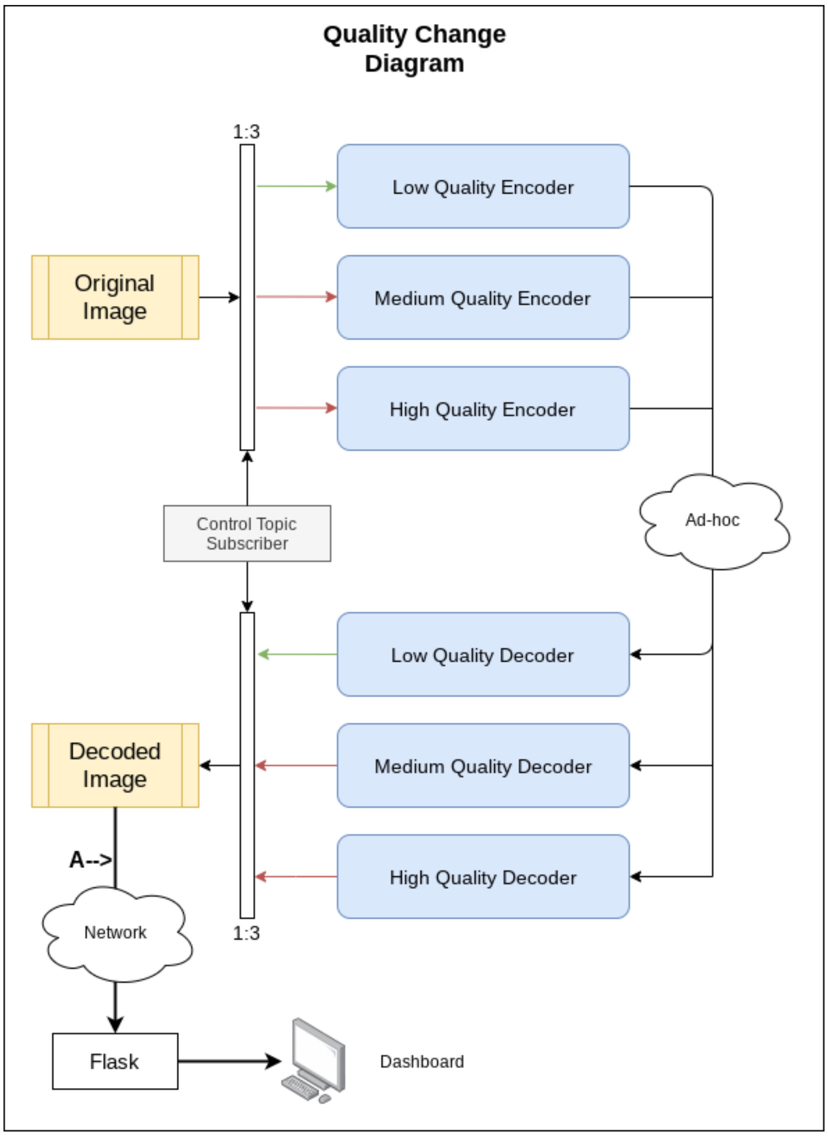


Figure 16: Dynamic Quality

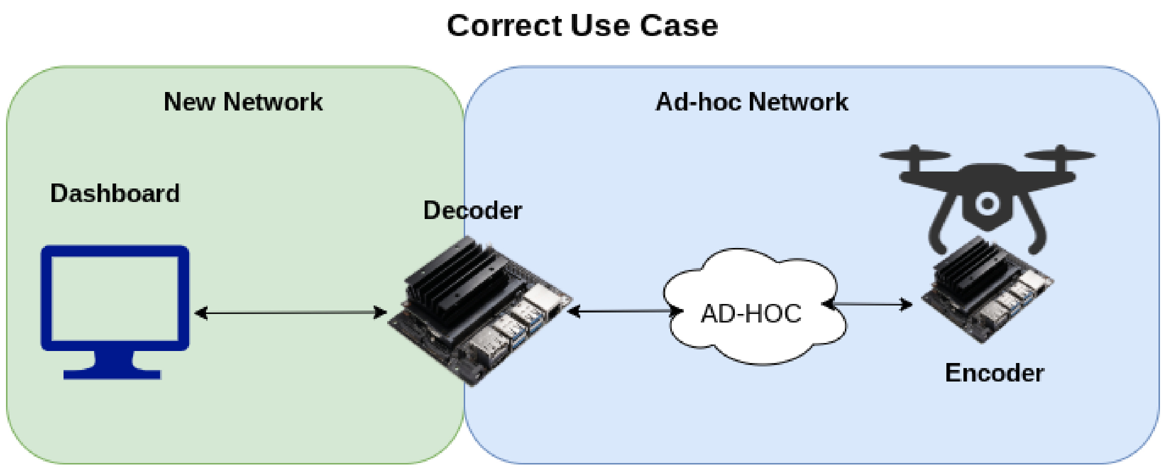
The dynamic quality change provided by this software works with multiple instances of encoders and decoders running on encoder/decoder NVIDIA Jetson Nano. This approach was chosen for the high speed when the quality changes, as the Gstreamer pipelines never have to restart, only the frames change in which pipeline they are being encoded / decoded. As only one pipeline is used at a time, the performance hit on the resources of the Jetson by multiple being instantiated is manageable.

As Figure 16 shows, the original image is processed only by one pipeline at a time, the one responsible for the quality that is being encoded. The encoded image is then transmitted via a ros2 image through the ad-hoc network. The Jetson responsible for the decoding receives the encoded frame, and processes the encoded image by the corresponding quality decoding pipeline, yielding the original image, which is published on a new ros2 topic (/image/decoded).

The mechanism that controls the quality Gstreamer pipeline that is used to encode/decode a ros2 control topic, is a ros2 control topic that is used to change the quality preset.

For the integration with the network sensor, the encoder is responsible for listening to the ROS2 messages from the corresponding drone network sensor and publish on the control topic the best quality (according to the sensor).

(Flask) Server and Dashboard:



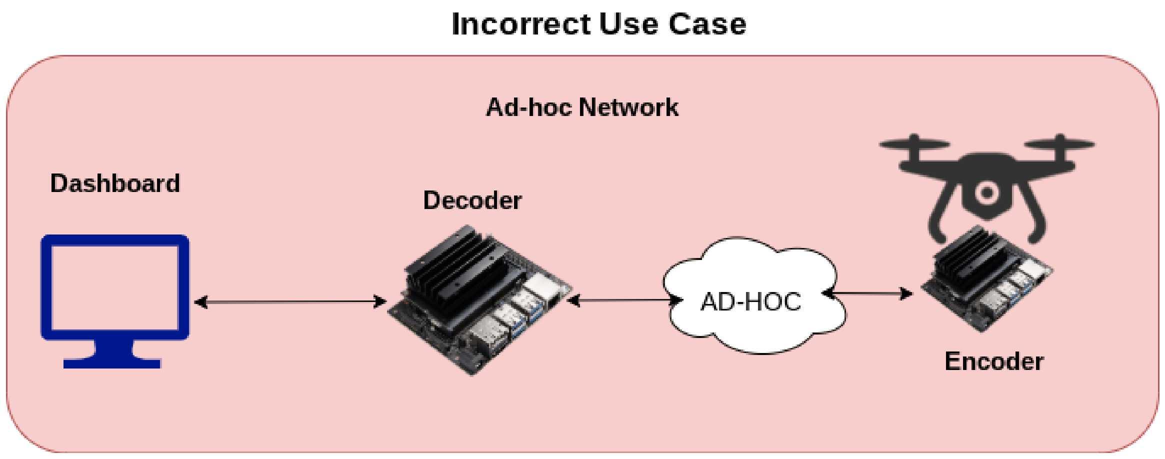


Figure 17: Correct/Incorrect Use

The flask server is responsible for receiving the decoded images from the ROS2 message, and creating an endpoint responsible for the video streaming to the dashboard. It algo controls when a user is visualizing a video, turning on the video stream while it's being used. This can be achieved using websockets to detect which drone camera the user is viewing, and turning on/off the stream, so as to not overload the network.

The server runs on a different network that is required besides the ad-hoc network, as displayed in figure 17, if the adhoc network has used to transfer the decoded packets from the flask server to the dashboard, this network would be easily overloaded, as it is not only transferring the encoded frames, but also the decoded ones, which makes the video stream not viable.

This server must run on a Jetson that is directly connected to the adhoc network and to the new network that is going to communicate with the dashboard via the flask server. A recommended setup is for the decoding Jetson to be connected to the adhoc network via a wireless adapter, and connect an ethernet cable directly from the decoding Jetson to a computer that will run the dashboard, setting manually ip’s to allow the communication. The run_video_server -a <ip> -p <port>, corresponds to the ip of the new network and the port in which the flask server will run in the Jetson.

6.2.6 Future improvements

There are still a few improvements that can be done in this video quality module.

For starters, this current version of the software decoder only allows for the decoding of either one quality encoded by the hardware encoder or multiple qualities by the software encoder. This problem occurs due to the hardware encoder used (Omxh264enc) allows for the configuration of the i-frame-interval and bitrate, and the software decoder (x264dec) was not designed to support the adjustment of these parameters. A possible solution is to research a couple of quality settings that allow a proper decoding via the software decoder. When using the software encoder in conjunction with the software decoder this problem does not occur.

Still, the best solution for these scenarios of usage is always the hardware encoder, and a possible solution to the problem of using a Jetson Nano for the hardware decoding, is to install and configure the NVIDIA Video Codec SDK that includes the NVENC encoder and decoder. Using these in the Gstreamer pipeline might lead with a few tweaks in the quality parameters that can lead to the decoding being performed on a computer/server as long as it contains an supported NVIDIA graphics card.

This version of the module only works on the ROS2 dashing version that is only compatible with Ubuntu based distros under the 18.04 version, updating to the latest ROS2 is advisable.

6.3 Wireless network sensor

6.3.1 Introduction

The wireless network sensor is the module that evaluates the overall connection quality between two nodes in the ad-hoc network, which can be used by other modules of this project such as the camera or the relay module to adapt the sending quality and adjust the relay drones position respectively.

The quality is measured with some parameters such as the latency, the received signal strength indicator, the transmission and reception bit rate and others. In this subsection we aim at explaining how the wireless network sensor and its simulated version work and can be used, as well as how it can be improved in a later version.

6.3.2 File structure

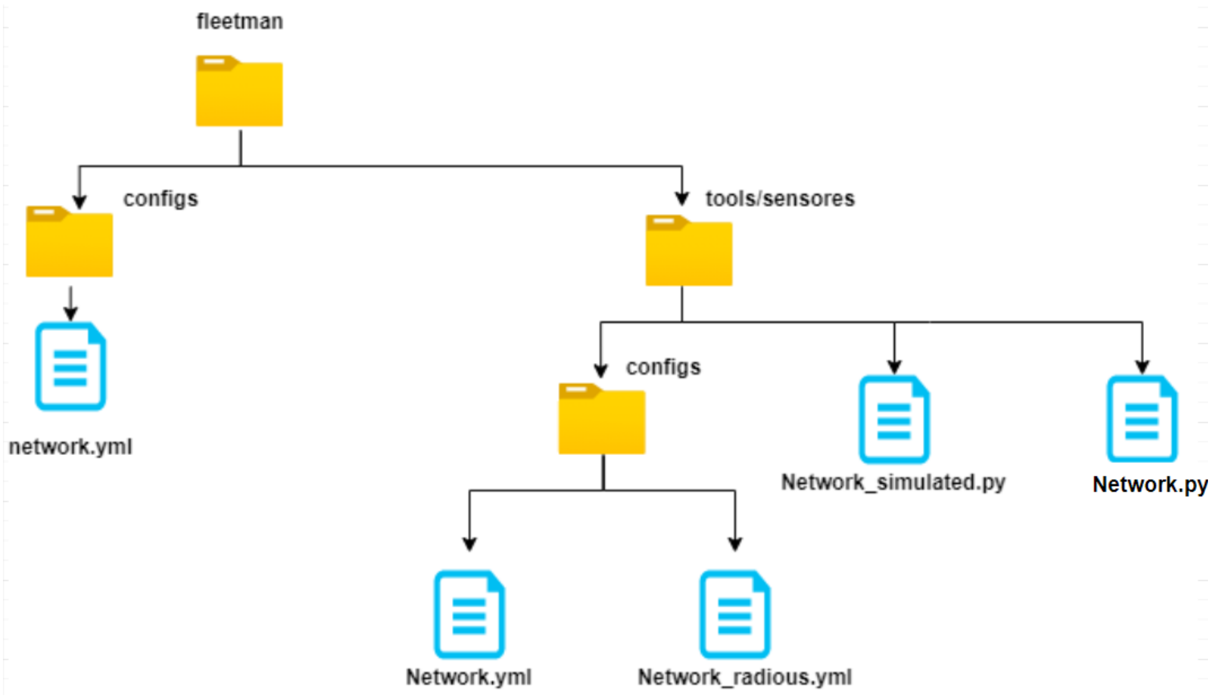


Figure 18: File Structure Wireless Network

The Figure above showcases all the different files used by the wireless network sensor and its simulated version. We now give a brief explanation on what these files are.

- network.yml: This is the configuration file used by the groundstation in order to know how the sensor is configured, for example in what format the information is going to be sent.
- Network.py: (wireless network sensor) This file contains the code that retrieves the network quality parameters and sends that information via ROS2.

- Network.yml: Configuration file used by the wireless network sensor. It contains essential information that must be configured prior to launching Network.py such as the network interface to be used, the drones name, the sending ratio depending on the network qualities and more.

- Network_simulated.py: (simulated wireless network sensor) This file contains the code that simulates network parameters.

- Network_radious.yml: Configuration file used by the simulated version of the wireless network sensor. It can be configured prior to launching Network_simulated.py in order to for example define the max range in which the nodes in the ad-hoc detect each other.

6.3.3 How to setup

The network sensor needs the following tools installed in the drones image: Iw, ping, haversine. This can be achieved either by downloading them inside the drone's running image or simply by putting the installation inside the drone image building file.

For the simulated version:

- Launch the docker image of the drone using the sim launcher already available in the project under the tools/sim_launcher folder.
  Here is a example on how to run the image for a single drone using the config file single_drone_aveiro.yml provided by the previous work:

```
./sim_launcher -c configs/single_drone_aveiro.yml
```

- Enter the docker workspace with the command:

```
docker exec -it drones bash
```

- Inside the docker workspace go to the folder that contains the sensor src/sensors and run the Network_simulated.py file:

```
python3 Network_simulated.py -c configs/Network_radious.yml
```

For the “real” version firstly we explain how to run it using the simulated version of the drone, and then how to run it inside the jetson that controls the drone.

Using a simulated drone:

- Launch the docker image of the drone using the sim launcher already available in the project under the tools/sim_launcher folder.
  Here is a example on how to run the image for a single drone:

```
./sim_launcher -c configs/single_drone_aveiro.yml
```

- Enter the docker workspace with the command:

```
docker exec -it drones bash
```

- Inside the docker workspace go to the folder that contains the sensor src/sensors and run the network.py file:

```
python3 Network.py -c configs/Network.yml
```

Inside the drone’s jetson:

1. Connect the drone to the jetson using its usb cable.

2. Use ssh to connect to the jetson and run the drone controller script in the scripts folder:

```
./launch_drone_container -c ../configs/drone_cfg_serial.yml 'droneId'
```

3. Using another ssh connection go inside the drone docker container workspace and run the sensor in the folder src/sensors:

```
python3 Network.py -c configs/Network.yml
```

6.3.4 Explanation

To develop this module we made a version of the sensor that doesn't need to run on the drone where we tested different different network parameters and different ways to get them. A version of this module is provided on the repository (Network_quality.py). In this version we used the mtr tool to get values such as the average Round Trip Time, the average jitter, the percentage of packet loss and then the iw command to get values like the signal, the transmission bit rate, the receiving bit rate.

The values that we got from the mtr command were not used on the sensor version that is running on the drone due to the fact that we needed to provide values faster than that that the command gives us.

Before going in depth on how the simulated version of the sensor and the real one that runs on the drones are, here are some examples of messages containing the connection quality in this case between "drone01" and the groundstation, sent by both the real and the simulated version of the sensor respectively.

```
[INFO] [drone01.sensor.network]:
{"droneId": "drone01",
"sensorId": "real",
"type": "network",
"timestamp": 1620918763143,
value": [{ "groundStation": [{"Station": "f8:d1:11:08:47:7c",
"IP": "10.1.1.100", "Latency": "6.52",
"Signal": "37", "TxByte": "21084", "RxBit": "54.0 MBit/s",
"TxBit": "54.0 MBit/s","Relay": "GroundStation"
"NetworkQuality": "HIGH"}], "Relay": "HIGH"}]}
```

```
[INFO] [drone01.sensor.network]:
{"droneId": "drone01",
"sensorId": "simulated",
"type": "network",
"timestamp": 1620918965309,
value": [{ "groundStation": [{"Station": "00:00:00:00:00:00",
"IP": "0.0.0.0", "Latency": "1", "Signal": "30", "TxByte": "13",
"RxBit": "49.9 MBit/s", "TxBit": "49.9 MBit/s","Relay":
"GroundStation"
"NetworkQuality": "HIGH"}] , "Relay": "HIGH"}]}
```

The sensor starts by subscribing to the ROS2 topics of /telem where it publishes the values for the groundstation to read, and also the /info topic where the groundstation publishes asking for the quality of one or more connections in the ad-hoc.

The groundstation publishes messages like the following were the droneId represents the target drone of this message, the connectionsId, connectionsIp and connectionsMac are the values of the entity that the drone will measure the network values with:

```
send droneId: “drone01” connectionsId: [“drone02”] connectionsIp:
[“10.1.1.4”] connectionsMac: [“00:00:00:00:00:00”]
```

```
send droneId: “drone01” connectionsId: [“groundStation”] connectionsIp:
[“10.1.1.100”] connectionsMac: [“b4:ff:cc:00:1f:00”]
```

After a drone receives a message like this one he will retrieve the network parameters between himself and the entities that the groundstation asked for using the ping command to retrieve the latency, the iw command to retrieve the signal, the transmission bit rate, the receiving bit rate and the number of packets make a estimate of the network quality with the previous values.

This estimate varies between high, medium and low and it depends on the parameters mentioned above , when they pass a certain threshold the quality changes. These thresholds were defined during the tests using the video module to verify which ones were best to make the video change quality when the stream becomes unstable.

It is also important to mention that the groundstation or any ROS2 Node can also send a message like this one to specify who the relay drone is.

```
send droneId: “drone01” RelayId: [“drone02”] RelayIp: [“10.1.1.4”]
RelayMac: [“00:00:00:00:00:00”]
```

In this case the sensor will calculate the network parameters, make an estimate and send it on the messages with the tag “Relay” where the telemetry module and the video quality adaptation will listen to and adapt according to this parameter.

Bellow (figures 19 to 21) is a example on how the groundstation gets the quality of the network
between two nodes in the ad hoc and an image (figure 22) that explains how the modules in the project use the different ROS2 topic, the groundstation to retrieve and ask information, the video module to retrieve which quality presset it should use, and the drone which retrieves the requests, publishes the network probes and updates the telemetry sending ratio.

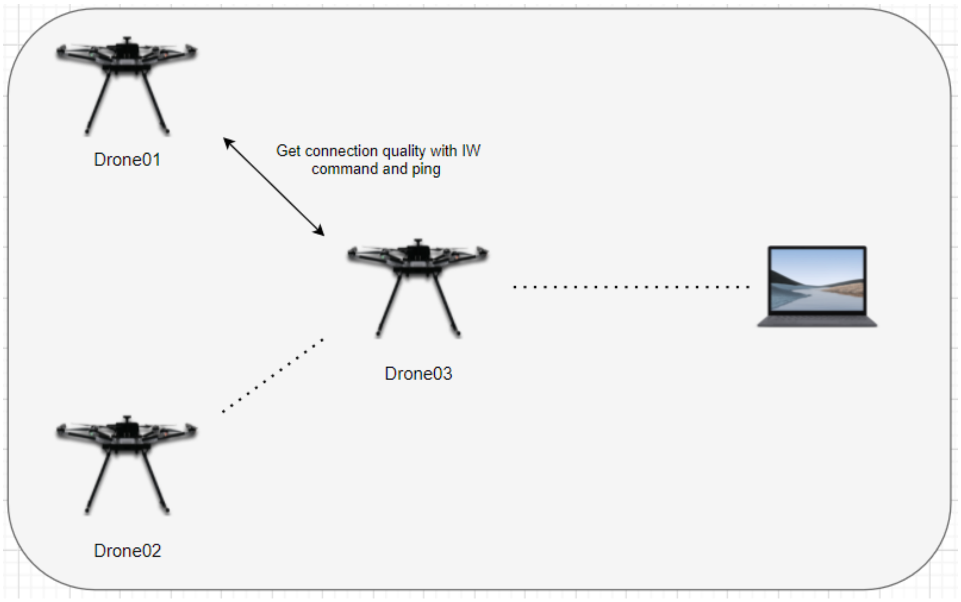


Figure 19: Network Sensor_1

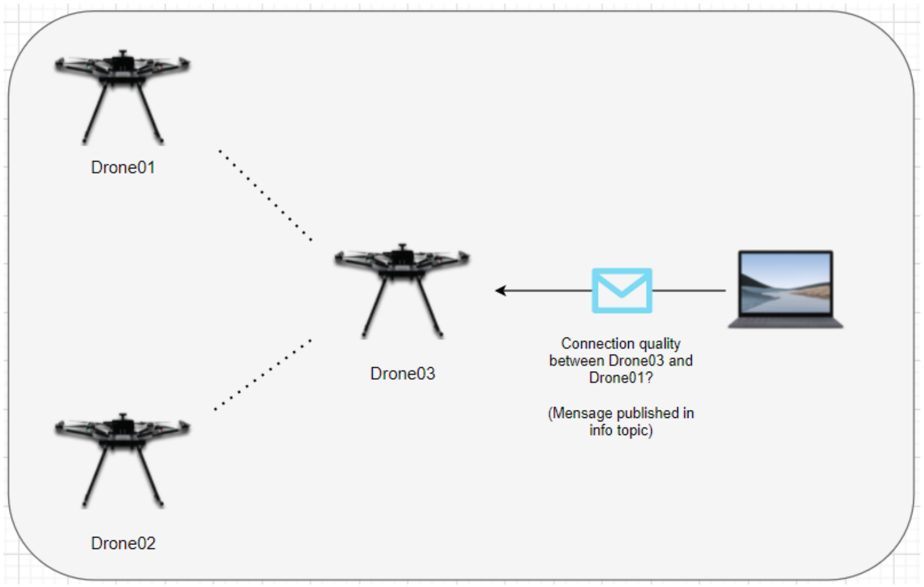

Drone01
Drone03
Connection quality between Drone03 and Drone01?
(Mensage published in info topic)
Drone02

Figure 20: Network Sensor_2

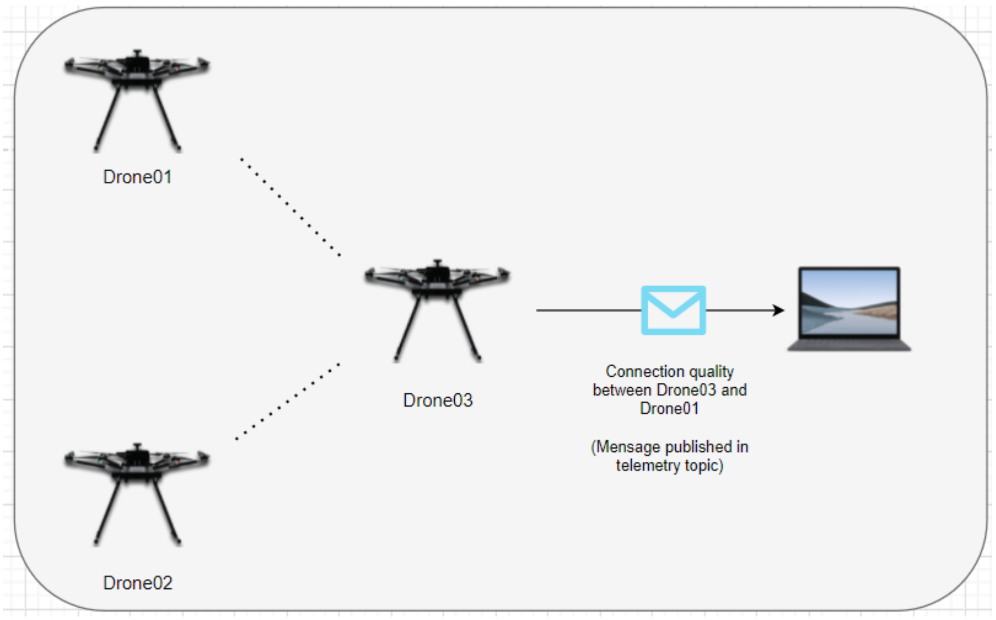

Drone01
Drone03
Drone02
Connection quality between Drone03 and Drone01
(Mensage published in telemetry topic)

Figure 21: Network Sensor_3

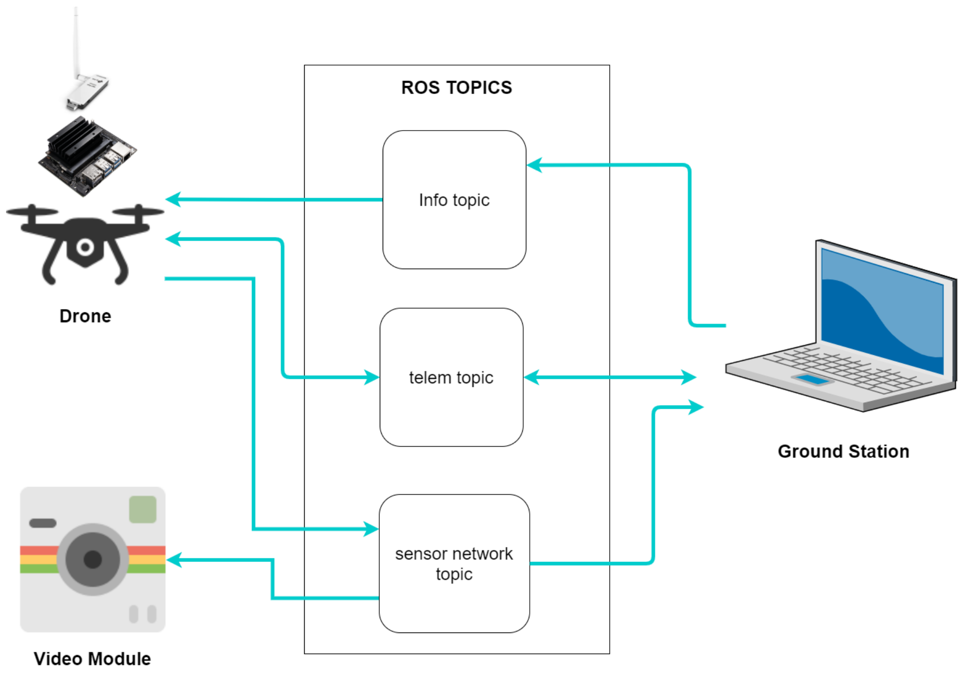


Figure 22: Overview of how the components interact with the ROS2 topics

6.3.5 Future improvements

In a future version this module can be improved in some aspects. One such improvement is in the way this module calculates the latency between two nodes, as it makes use of pings which inject traffic in the network. Although generating a relatively small amount of traffic compared to the amount already passing through the ad-hoc, there are certainly other ways to do it such as having synchronized clocks between the nodes and measuring the already available traffic or even checking the time between a tcp SYN and SYN ACK packet.

In the simulated version it might be needed to include more scenarios, what we mean by that is that the way it is now doesn't take into account factors like buildings or other objects that obstruct the signal between the nodes.

One can also find a way to retrieve other parameters such as the jitter or the loss of the packets and take them into account when deciding between the 3 network pressets.

6.4 Telemetry Module

6.4.1 Introduction

In this chapter we will explain how the adaptation of the telemetry/sensor sending ration was developed.

Notice that this adaptation works under the simulated and the real version of the wireless network sensor.

6.4.2 Explanation

As we mentioned in a previous section the wireless network sensor indicates three network qualities: High, Medium or Low. As our video only had three presets we decided that we were gonna also use these presets to change the telemetry sending ratio between three different values that can be configured in the Network.yml file prior to running the wireless network sensor.

To make this possible when the network alternates from any of this three presets to another we change the publishing ratio of the sensor by updating the timer object that triggers the publishing event and we also set the parameter telemetryRateMs via ROS2 to the value that is indicated in the configure file.

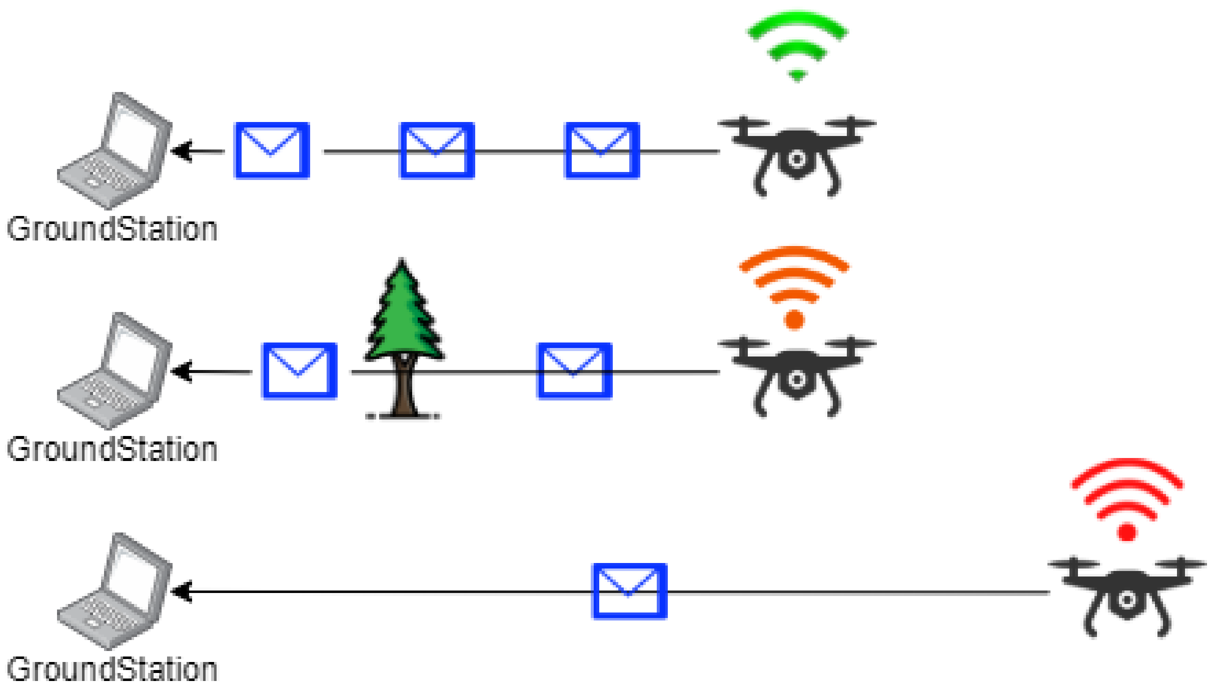


Figure 23: Example of how the adaptation occurs

```
droneId: drone01
sensorTopic: /sensors
telemTopic: /telem
rateHigh: 200
rateMedium: 600
rateLow: 1000
groundStation_IP: "10.1.1.4"
groundStation_MAC: "b4:6b:fc:48:80:19"
interface: "wlan0"
```

Suppose that we have a configuration like the one in figure 23, in the case that the drone has a good connection with the groundstation it will publish the telemetry messages and the sensor messages at a rate of 200 milliseconds, as it gets further or gets blocked by an object then the telemetry sending rate will change to 600 milliseconds and if the network gets even worse it will change this sending rate to 1000 milliseconds, this correlates to the previous 3 quality presets.

6.5 Groundstation Module

6.5.1 Introduction

The groundstation is the module where the user interacts with the drone modules. It is capable of sending commands to specific drones, monitoring drone data and sending mission scripts.

6.5.2 File structure

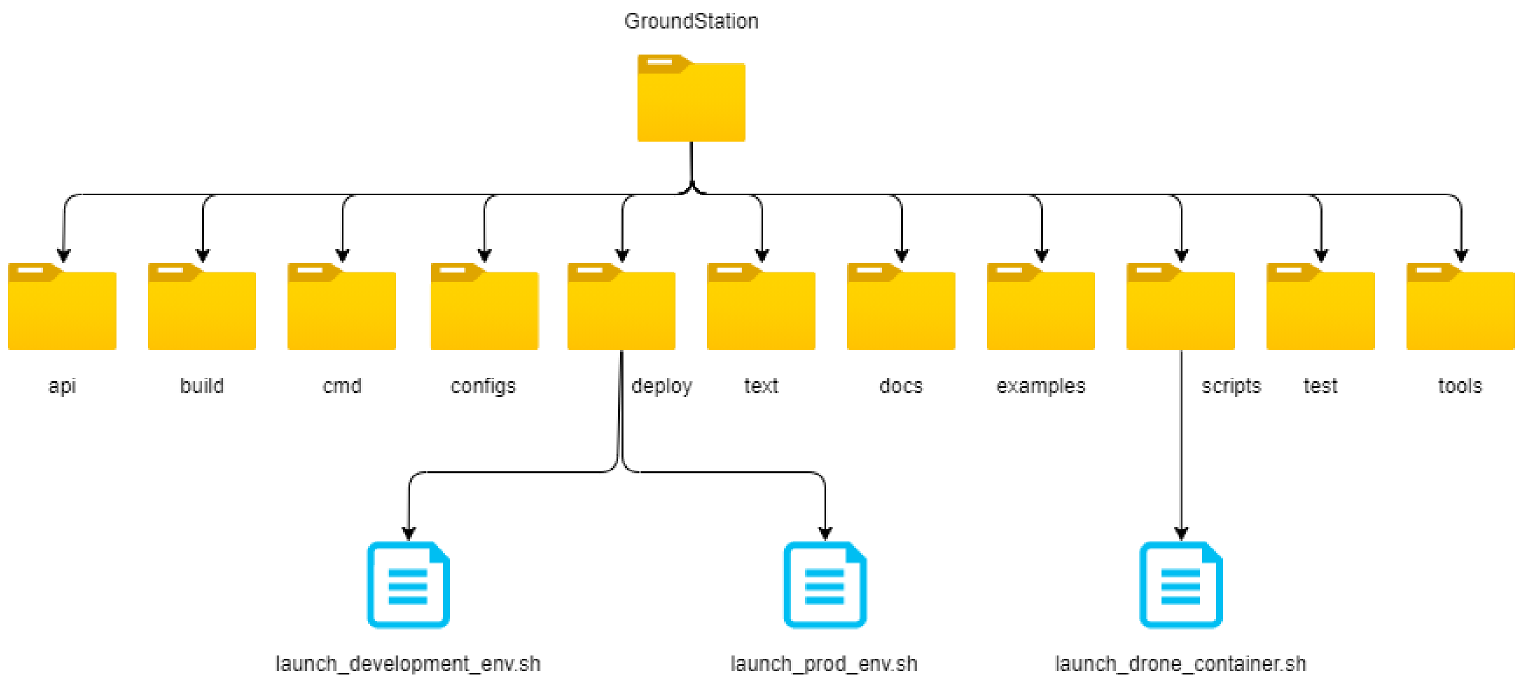


Figure 24:File Structure GroundStation

6.5.3 How to setup

Server / Computer Setup:

- Build the grounstation docker image.
```
cd build && build_docker_images.sh ground
```

- Open the deploy folder

- Run the launch_development_env.sh script for normal execution, or launch_prod_env.sh for development.
```
./launch_development_env.sh
./launch_prod_env.sh
```

jMAVSim simulator :

Requirements:

- GoLang

Setup:

- Build the jMAVSim docker image.
```
cd build && build_docker_images.sh simulator
```

- Open the tools/sim_launcher folder

- Build with go
```
go build
```

- Run the sim_launcher file with -c and select the configuration.
```
./sim_launcher -c configs/single_drone_aveiro.yml
```

Drone Container :

This container runs on the Jetson Nano connected to the drone via the usb cable.

Setup:

- Build the drone docker image.
```
cd build && build_docker_images.sh drone
```

- Open the scripts folder
- Run the launch_drone_container.sh file with -c and select the serial configuration when connected to the drona via usb cable.

```
./launch_drone_container.sh -c ../configs/drone_cfg_serial.yml
```

6.6 Dashboard

6.6.1 Introduction

The dashboard is the module that has the objective to create an abstraction layer between the code behind the functionalities and the final user. So it gives a lot of information to the user, values of the sensor, location, commands to control the drone, battery information, live video, and much more.

For the frontend development we used nuxt which is a javascript framework to create SPAs using also html and css for the structure and visuals of the site. For the backend the Django framework was used and for the database mongoDB.

That module was almost completely developed, we just added some new features.

6.6.2 File structure

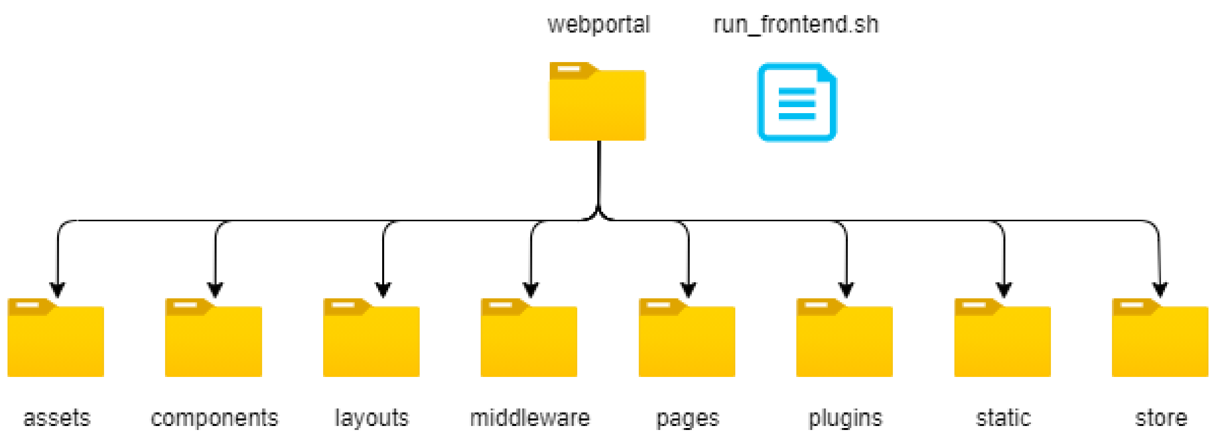


Figure 25:File Structure Dashboard

6.6.3 How to setup

1. Launch drone-backend

   Requirements:
   - docker (https://docs.docker.com/engine/install/)
   - docker compose (https://docs.docker.com/compose/install/)
   - mongodb (https://docs.mongodb.com/manual/installation/)

   Setup:
   On folder drone-backend:
   (available in https://code.nap.av.it.pt/uavs/drone-backend)

```
sudo systemctl start mongodb
cd Django/GpsTracker
sudo docker-compose up --build
```

2. Launch dashboard

   Requirements:
   - docker (https://docs.docker.com/engine/install/)
   - docker compose (https://docs.docker.com/compose/install/)

   Setup:
   On folder dashboard:
   (Disponível em https://code.nap.av.it.pt/uavs/dashboard)

```
cd webportal/
sudo docker-compose up --build
```

   Access dashboard in: http://localhost:3000

3. Launch drone

   On folder fleet-manager:
   (Available in https://code.nap.av.it.pt/uavs/fleet-manager)

```
cd tools
cd sim_launcher
sudo ./sim_launcher -c configs/single_drone_aveiro.yml
```

On folder configs there are many different modes, in case of the number of drones and the positions of it.

4. Launch groundstation

   On folder fleet-manager:

```
cd scripts
sudo ./launch_ground_container.sh
```

5. Configure station

   Go to Stations, click “+”, put the name and IP as http://localhost:8001and lastly click submit.

   Obtain information about the drones on drones.

6.6.4 Future improvements

In future work we could improve some details about past missions. After doing a mission and measuring many sensors and paths, we could record and save that detailed information, with some description and comments for later analise.

7. Results

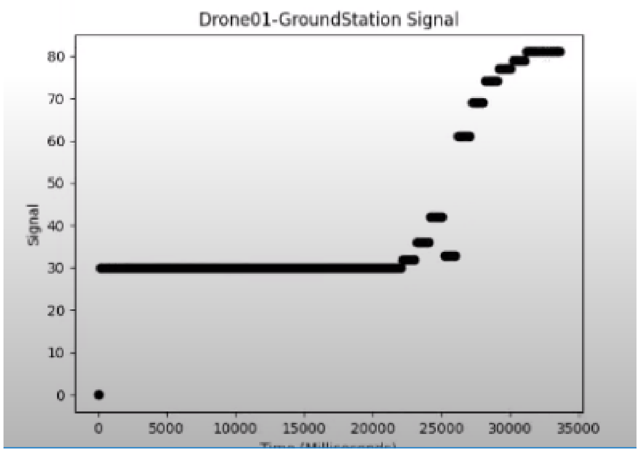
7.1 Introduction

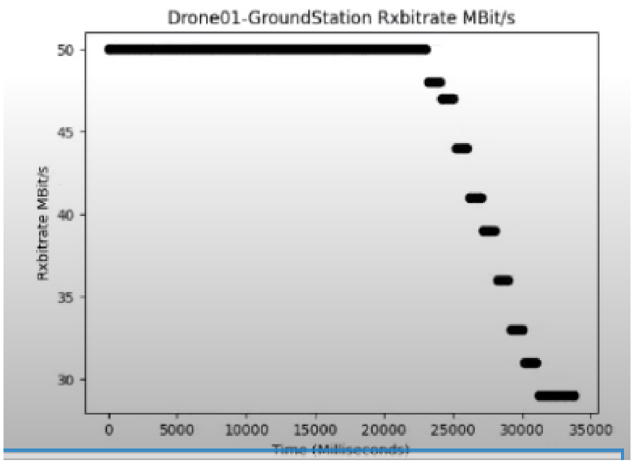
In this section we will demonstrate some of the results that we obtained as to better demonstrate the functionalities we have developed along this project

7.2 Results for the wireless network sensor

As we developed the wireless network sensor we also made sure to test it to understand not only if it was working but to also make some adjustments as to make sure, for example, that estimated network quality value we mentioned previously didn't make the video (that uses this value to change the quality) go unstable.

7.2.1 Simulated version of the sensor





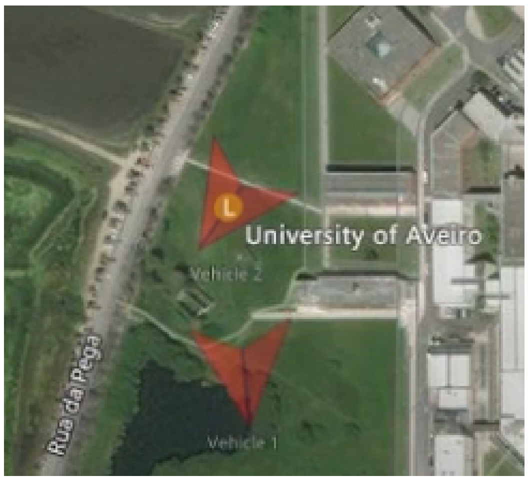


Figure 26: Results simulated version_1

Figure 26 is an example of the signal (that varies from 0 to 90 being 0 the best value) and the receiving bit rate,as we can see the signal gets higher as the vehicle 1 moves away from the groundstation that is marked with a L in the figure and the bit rate also decreases.

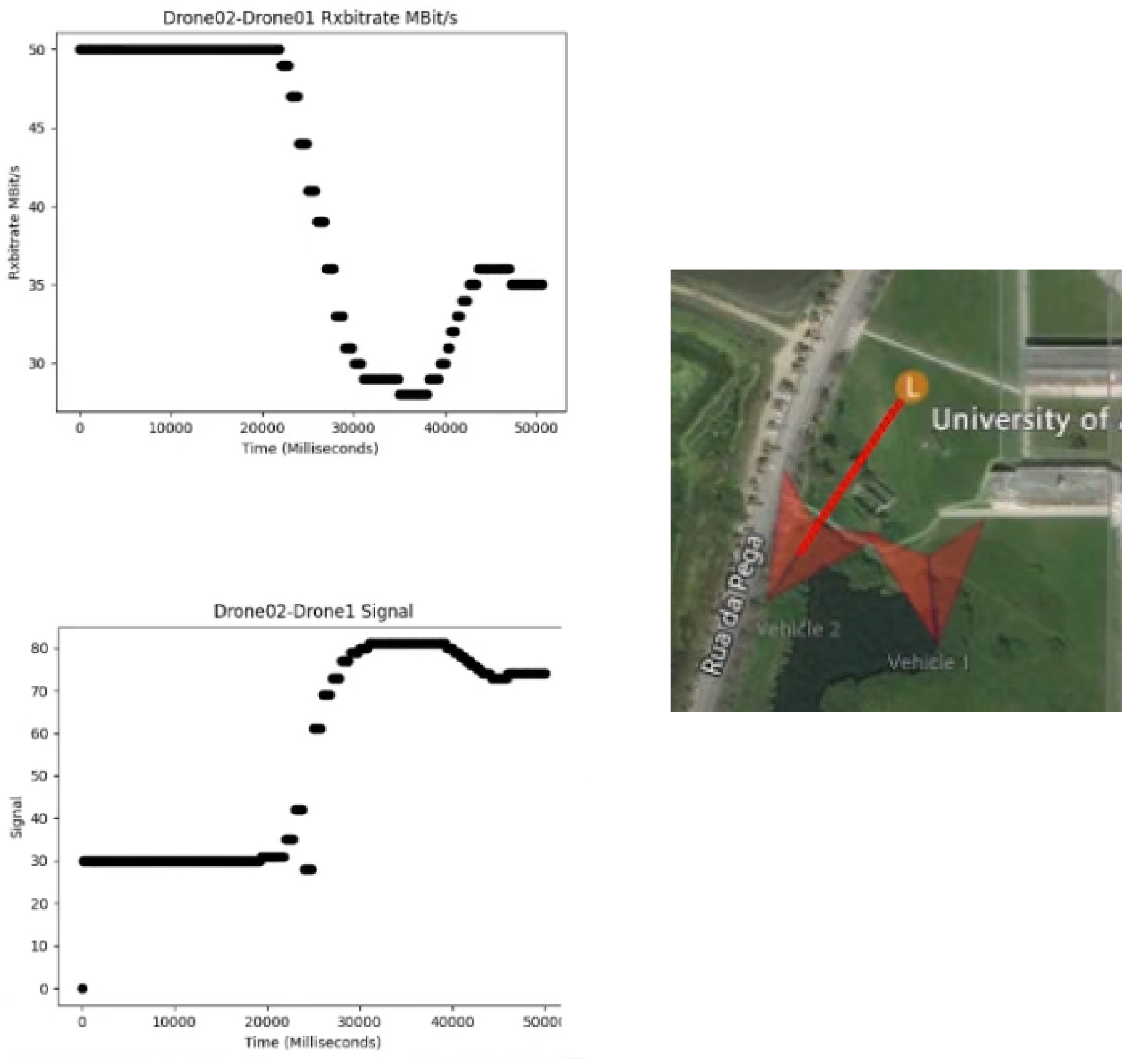


Figure 27: Results simulated version_2

Figure 27 shows the values of the signal representing the connection that the drone02 has with drone01 and as we can see, when the drone02 approaches drone01 that was already in the final position, the signal gets slightly better as the drones are closer (35000 - 50000 ms).

7.2.2 Real version of the sensor

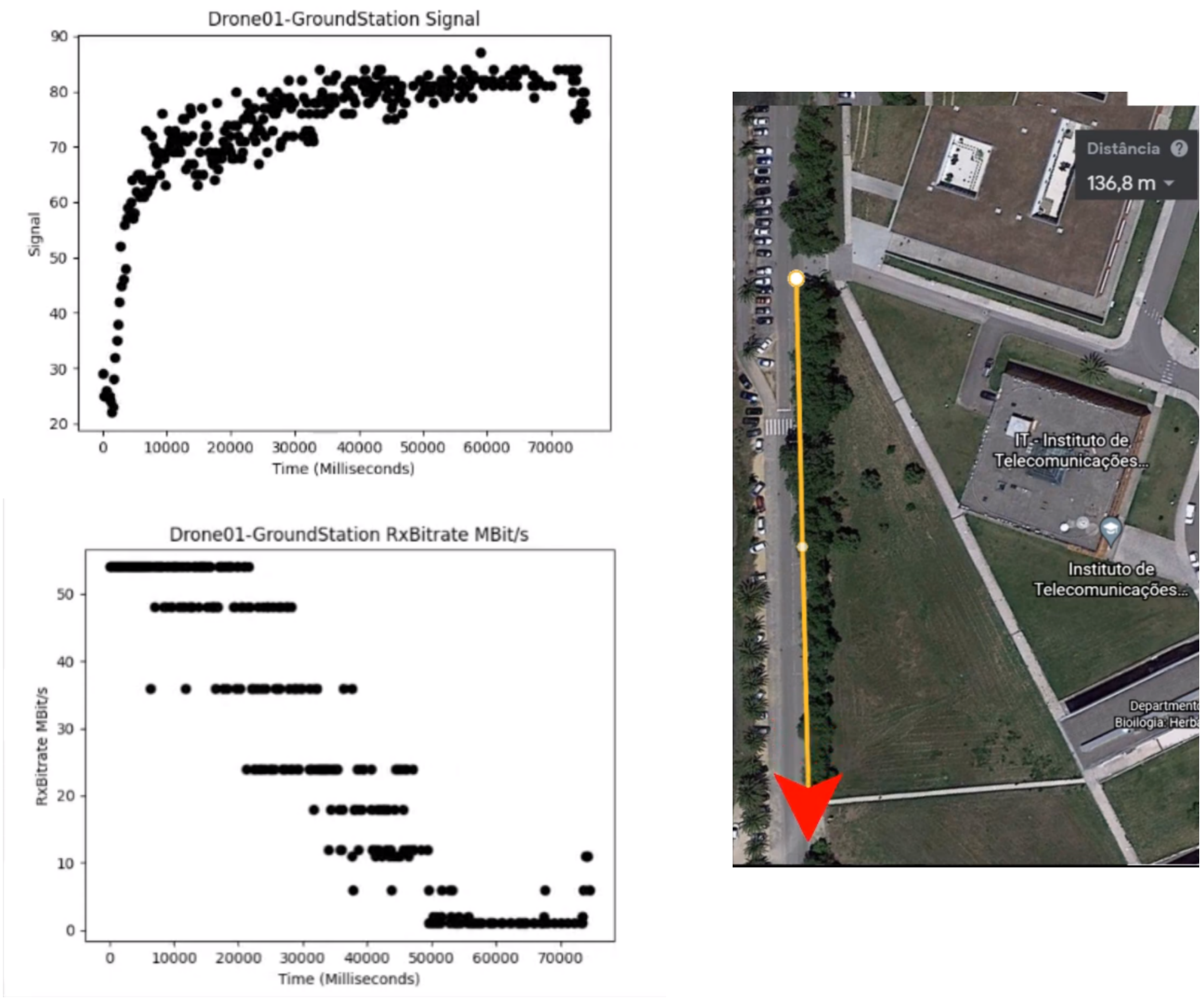


Figure 28: Results real version

Figure 28 is a test that was performed where the Jetsons were physically separated, running the network sensor (represented by the red arrow) from the groundstation (represented as a white dot in the figure) as expected the signal gets worse and the bit rate also declines.

7.3 Results for the Mission algorithm

7.3.1 Introduction

In this part of the results we present some tests we made for the relay mission algorithm with the simulated network sensor and the real one.

7.3.2 Algorithm tests in a simulated environment

Video demonstration of the relay algorithm within a simulated environment: Link to the video .

7.3.3 Algorithm tests in a real scenario

Videos demonstrative of the real scenario test of the mission relay algorithm: Example one, Example two, Example three .

7.4 Results for the camera

7.4.1 Introduction

In this part we can see several videos that demonstrate how the adaptive quality video stream performs under different conditions.

7.4.1 Videos demonstrative of the camera module

Video demonstration of the adaptive quality under a normal scenario: Link to the video.

Video demonstration of the video under a condition where the network was unstable with high latency: Link to the video.

7.5 Results for the adaptation of the telemetry sending ratio

7.5.1 Introduction

In this segment we can see the an example of the telemetry module working in a simulated environment for better understanding as in the real world its is more hard to demonstrate however it works the same way in both scenarios

7.4.1 Videos demonstrative of the adaptation of telemetry sending ratio

Video demonstration of the telemetry sending ratio decreasing as the drone network gets worse due to the fact that the drone is getting further from the groundstation: Link to the video

8. Difficulties

While developing this project we faced some unexpected problems and some updates on the project requirements that had a major influence in our overall time management and performance.

Most of them occurred frequently and we had no control over them, like the signal/latency that very significantly fluctuated the network quality, hardware problems (individual components problems, drone controller errors), etc…

We also had to learn how to setup, control, manipulate drones manually and configure new gadgets never used (cameras, gimbal).

One factor that also had an effect on our project, but we understand it is necessary due to the integration of our project with many others, the continuous integration with the previous and continuous work that is being developed with other researchers and students that led to many unexpected errors and much time on the debug part, related to the compatibility between modules.

Summarizing, the real-life testing component of this project that included a lot of fluctuation in variables that were controlled in the laboratory environment was a real challenge, considering that the setup of the drone was required every time some test was to be performed, which in itself, is time consuming.

9. Acquired skills

In this chapter we present some of the technologies, frameworks, tools and hardware that we learned to work with alongside this project.
Bellow is a list that contains the names of all of them:

- Ros2
- Gstreamer
- FFMPEG
- FLASK
- Groovy
- Fleetman framework
- SpringBoot
- Postman
- Vue
- Nuxt
- Django
- MongoDB
- Docker
- QGroundController configuration (drones)
- sysstat tool
- mtr tool
- IW tool
- Ifcong tool
- ping tool
- sysstat tool
- OpenCv
- Nvec encoder/decoder
- JmavSim (drone simulator)
- Drone setup and maintenance
- Manually flying drones
- Jetson Nano configuration

10. Conclusion

To conclude our project,t had as objectives, the creation of a wireless network sensor to measure the quality of the network which would be used in other modules (also developed), the mission algorithm, the video quality codec module and the telemetry module. All of that to be integrated into a modular infrastructure, enabling the autonomous control and monitoring of a fleet of aerial drones in a mission context to manage many different situations.

To sum up, the wireless network module that evaluates the overall connection quality between two nodes in the ad-hoc network, the mission algorithm that aims to create an algorithm to find the optimal position for relay drones in order to expose mission drones to the best network connection possible, the video quality codec module that is responsible to capture, change the quality dynamically and transmit from the camera to the dashboard, the telemetry module that adapts the telemetry transmission rate automatically and lastly the dashboard.

Relatively to the results, we tested in a simulated environment and in a real life scenario.

The requirements we have established in this project have been met for both the drone requirements, the groundstation requirements and the dashboard ones with a caveat, the modules main functionality is working, but all have the opportunity to be refined and tuned with more real-life scenarios in different conditions and environments.

11. References

Drone-Based Wireless Relay Using Online Tensor Update :
(https://ieeexplore.ieee.org/document/7823731)

Performance Improvement of Drone MIMO Relay Station Using Selection of Drone Placement:
(https://ieeexplore.ieee.org/document/8536637)

Linux iw documentation:
https://wireless.wiki.kernel.org/en/users/documentation/iw

Linux mtr documentation:
https://linux.die.net/man/8/mtr

Linux ping documentation:
https://linux.die.net/man/8/ping

Install Docker Engine:
https://docs.docker.com/engine/install/

Install Docker Compose:
https://docs.docker.com/compose/install/

Mongo DB:
https://docs.mongodb.com/manual/installation/

jMAVSim documentation:
https://docs.px4.io/master/en/simulation/jmavsim.html

Gstreamer documentation:
https://gstreamer.freedesktop.org/

ROS2 Foxy documentation:
https://docs.ros.org/en/foxy/index.html

12. Attachment

12.1 Weekly Work

Every week we meet with supervisors to understand the situation point of the project and to understand which are the next steps.

- **Week 1 - W2 March:**
  15/03/2021

  Meeting with several professors to choose our project.

  Choosing the topic of our project “Drone Relay: Aerial Drone Platform with relays for scenarios of limitation and emergency” and the members of the group.

  Meeting with professor Susana Sargento, professor Miguel Luís and Margarida Silva to clarify our main objectives.

  Obtain the bases of the work previously developed. Beginning of the analysis of the existing code.

  Research and planning.

  Beginning of the preparation of the milestone presentation 1.

  Define the objectives and schedule of our project.

- **Week 2 - W3 March:**
  22/03/2021

  Research and planning.

  Overview of existing architecture.

  Completion of the milestone presentation 1.

  Presentation in the \ class, Milestone 1 - 24/03/2021.

  More in-depth view of the architecture already made.

- **Week 3 - W4 March:**
  29/03/2021

  Code analysis “fleet-manager”.

  Code analysis “dashboard”.

  Code analysis “drone-backend”.

- **Week 4 - W1 April:**
  5/04/2021

  Continuing code analysis: Fleet-manager, dashboard and drone-backend.

  Requirements analysis and State Of Art.

  Production of mockups according to the new features of the dashboard.

  Initialization of the development of the mission algorithm, in python.

  Construction of a simple simulator for testing functionalities.

  Initialization of the development of the wireless network sensor.

  Initialization of the documentation writing.

  Initialization of the definition and planning of the architecture.

- **Week 5 - W2 April:**
  12/04/2021

  Completion of the architecture.

  Presentation in the PEI class, Milestone 2 - 14/04/2021.

  Completion of code analysis.

  Construction of the structure for the relay.

  Investigation of commands and ways to obtain information about the ad hoc network for the sensor.

Observe and understand which of the ad hoc nodes we are connected to (sensor network).

Beginning of the description of the various points developed each week.

- **Week 6 - W3 April:**
  19/04/2021

  Definition of the position for the drone relay in cases of optimal network.

  Obtaining several fields of the network: RSSI signal, TX bitrate and RX bitrate.

  Documentation of the general structure of the project architecture.

  Initialization of the camera integration.

- **Week 7 - W4 April:**
  26/04/2021

  Testing of drones on the field. Learning how to prepare and fly drones manually.

  Development of an optimized search (Search Tree), for the relay of drones.

  Obtaining latency for the wireless network sensor.

  wireless network sensor documentation.

  Research and development of the functionality to change the quality of the camera.

- **Week 8 - W1 May:**
  3/05/2021

  Convert the mission algorithm in python to the final language used, groovy.

  Completion of the wireless network sensor, small improvements and ways of passing the data to the relay.

  Integrate quality change functionality into the dashboard.

  Initialization of the dashboard extension with the new features.

Relay documentation.

Beginning of preparation for milestone 3.

- **Week 9 - W2 May:**
  10/05/2021

  Session with Margarida Silva and Nuno Ferreira to test the drones on the field, in automatic programmed mode.

  Continuation of the milestone 3 presentation.

  Debugging errors on mission algorithms.

  Add video live features.

  Initialization of the automatic adjustment of video parameters.

  Initialization of the automatic adjustment of telemetry sending ratio.

  Camera documentation.

- **Week 10 - W3 May:**
  17/05/2021

  Production of the video for the milestone 3 presentation.

  Integrate wireless network sensor on mission algorithm.

  Manual quality video changing on the dashboard.

  Trying to adapt quality video changing, manual to automatic.

  Analise how to change telemetric sending ratio.

  Initialization of the final technical report.

- **Week 11 - W4 May:**
  24/05/2021

  Development of the “predict algorithm” part of the relay. Final adjustment and bugs fix.

Using the endpoints provided, connect sensor information and display it on the dashboard.

Automatic video quality change done.

Testing automatic telemetry sending ratio.

Some improvements on the documentation of the modules separately.

- **Week 12 - W1 June:**
  31/05/2021

  Try to set up the drones to be ready for final tests.

  Create some charts on the dashboard using data collected by drones.

  Fix some bugs and adapt the way of measuring quality on the automatic video quality change. Final version done.

  Fix some bugs on the automatic telemetry sending ratio module. Final version done.

  Development of the overall structure of the documentation.

- **Week 13 - W2 June:**
  7/06/2021

  Testing in a real environment.

  Record and edit the final video.

  Final documentation.

  Start to prepare for the final presentation.

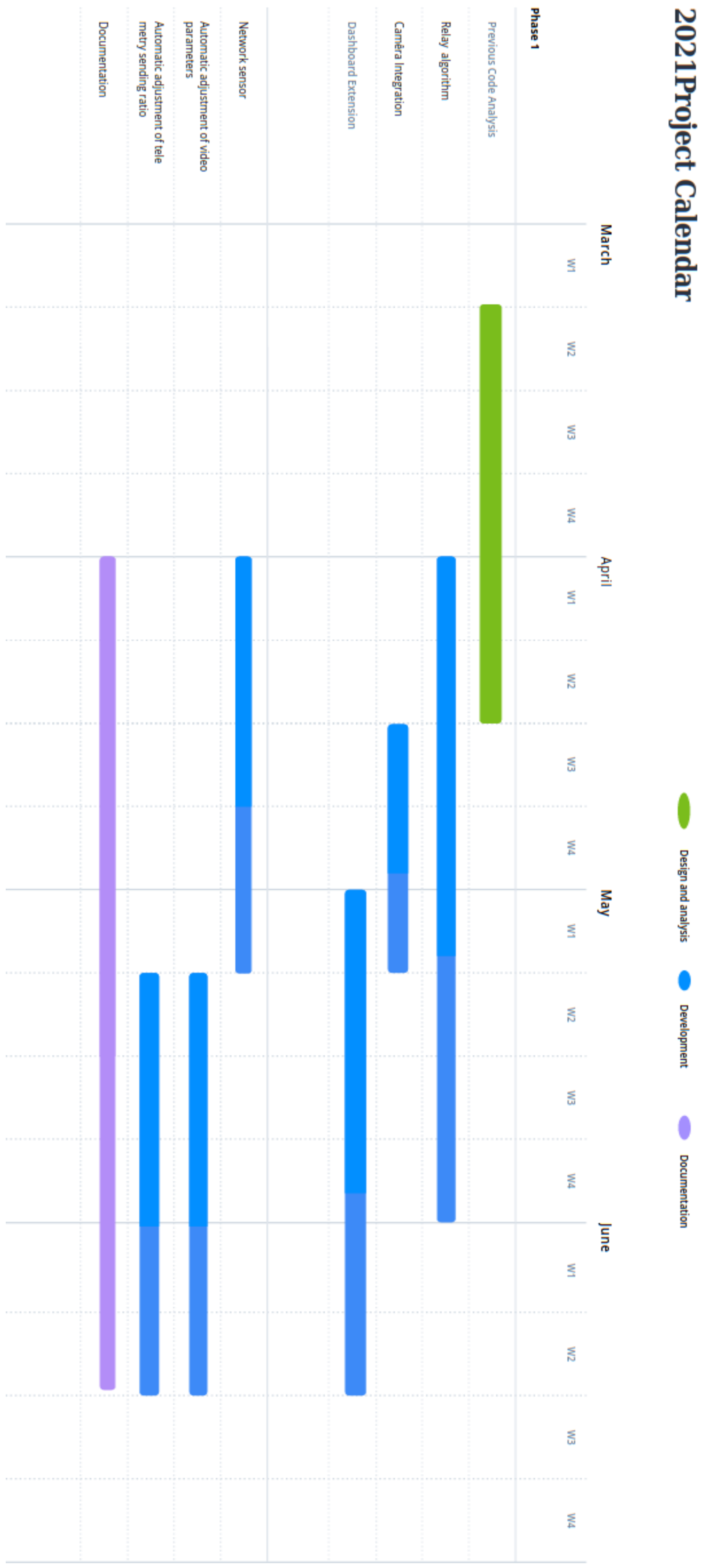
- **Week 14 - W3 June:**
  14/06/2021

  More tests in real life.

  Completion of the presentation.

  Presentation in the PEI class, Milestone 4 - 16/06/2021.

12.2 Project Calendar


2021Project Calendar
Design and analysis
Development
Documentation
Phase 1
Previous Code Analysis
Relay algorithm
Camêra Integration
Dashboard Extension
Network sensor
Automatic adjustment of video parameters
Automatic adjustment of telemetry sending ratio
Documentation
March
April
May
June
W1
W2
W3
W4