School of Technology and Architecture

Department of Information Science and Technology

Software platform to control squads of unmanned vehicles in real-time

Nuno Miguel Amorim dos Santos

A Dissertation presented in partial fulfillment of the Requirements for the Degree of Master of Telecommunications and Computer Engineering

Supervisor:
Professor Pedro Joaquim Amaro Sebastião, Ph.D.
Assistant Professor, ISCTE-IUL

Co-supervisor:
Professor Nuno Manuel Branco Souto, Ph.D.
Assistant Professor, ISCTE-IUL

October, 2016

ABSTRACT

Unmanned Aerial Vehicles (UAVs) applications are becoming more and more researched. “Drones” (UAVs) were mainly used as a military technology but are now becoming a leisure and professional activity for many civilian users.

Nowadays UAVs are mostly controlled by the use of a controller that operates in Radio Control (RC), although this method of communication limits the vehicle’s distance to the line of sight of the operator. As a need to overcome the line of sight obstacle, cellular networks provide a mean of connection and as the coverage is increasing they’re a natural solution as Wi-Fi is not present everywhere.

In order to accomplish this communication between Drone and Operator, there needs to be a Ground Control Station that provides the user all the tools needed to operate the vehicle.

This project provides a software platform that is able to monitor a squad of drones whilst also being able to control one at a time. The platform maintains the communication with the vehicle at all times, and is also be able to receive live-video in order to overcome the beyond line of sight obstacle. Besides this, the application provides an admin user, with the capability of overriding a regular user’s control, assigning the user’s drone to itself for controlling purposes. A public server is used to make the exchanging of messages possible, and to have a centralized control over drones and their respective user.

***Keywords*:** Monitoring, Remote Control, Wireless Networks, Software platform, Drones

This page was intentionally left in blank

RESUMO

Os Veículos Aéreos Não Tripulados (UAVs) são cada vez mais utilizados e desenvolvidos. O que antes era utilizado principalmente como tecnologia militar, tem-se vindo a tornar uma profissão ou um *hobbie* para muitos civis.

Hoje em dia os UAVs são controlados geralmente através de um comando, que opera em Radio Controlo (RC) e, embora seja muito utilizado, este método de comunicação limita a distância do veículo à linha de visão do operador. Este é um obstáculo que se tem procurado ultrapassar e as redes móveis providenciam o meio necessário para tal. Desta forma e como a cobertura das redes móveis tem aumentado progressivamente é hoje em dia uma alternativa ao Wi-Fi que não tem o mesmo alcance nem a mesma cobertura.

Para que a comunicação entre *drone* e operador seja viável, tem que existir uma estação de controlo que forneça ao utilizador todas as ferramentas necessárias para operar o veículo.

Este projeto visa a criação de uma plataforma de *software* que seja capaz de monitorizar uma esquadra de UAVs e seja também capaz de controlar um aparelho de cada vez. A plataforma mantém a comunicação com o veículo em todos os momentos, e permite ainda a receção de vídeo ao vivo, superando assim o obstáculo da linha de vista. Também é disponibilizada a um administrador a capacidade de retirar o controlo dos utilizadores aos seus drones alterando assim o responsável pelo controlo. É também utilizado um servidor público de forma a tornar a troca de mensagens possível e também por outro lado, controlar de forma centralizada os drones e os seus respetivos utilizadores.

Palavras-chave: Monitorização, Controlo Remoto, Redes Sem fios, Aplicação, Drones

This page was intentionally left in blank

ACKNOWLEDGMENTS

I would like to thank my family for all the support given throughout this thesis. Without their help, guidance and advices it wouldn't have been possible to finish this thesis and complete my academic journey. To my father, mother and brother, in particular, my dearest thank you.

I also want to thank one of the most important people in my life, my girlfriend whom has supported me all along and has given me the strength to do my best and persevere on the making of the thesis.

Of course, none of this thesis could have been accomplished without the guidance and help of Professor Pedro Sebastião, who was always available to help in any way he could. His vision and ideas for the project were key to keep the thesis on track and according to the main goals. Also, Professor Pedro, together with the Institute of Telecommunications, provided us with the facilities so that we had the best place to concentrate and work.

I'd also like to address a big thank you to my colleagues who worked on this project with me, Diogo Peres and António Raimundo. Our team work was crucial and it was important that we could rely on each other to solve most of the problems. An important mention also to Manuel Oliveira, for all the advices and for helping us overcoming some difficult issues.

Also, I'd like to leave a remark for Tiago Saraiva, his collaboration and help on the beginning of the project was very important to get us familiarized with the project.

Last but not least, I'd like to thank all of my closest friend for keeping up with me and giving me the support I needed.

This page was intentionally left in blank

CONTENTS

FIGURES

TABLES

ACRONYMS & ABBREVIATIONS

3G	3rd Generation
4G	4th Generation
ACK	Acknowledgment
APM	Ardupilot Mega
CAN	Controller Area Network
CPU	Central Processing Unit
CS	Control Station
CSI	Camera Specification Interface
CT	Control Tower
DSI	Display Specification Interface
ESC	Electronic Speed Controllers
FFMPEG	Fast Forward Moving Picture Experts Group
FPS	Frame Per Second
GCS	Ground Control Station
GPIO	General Purpose Input/output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System for Mobile
GUI	Graphical User Interface
HD	High Definition
HSPA	High Speed Packet Access
HSDPA	High Speed Downlink Packet Access

HSUPA	High Speed Uplink Packet Access
IMU	Inertial Measurement Unit
IP	Internet Protocol
JRE	Java SE Runtime Environment
LTE	Long-Term Evolution
MAV	Micro Aerial Vehicle
NAT	Network Address Translation
OS	Operating System
P2P	Peer-to-Peer
PWM	Pulse Width Modulation
RAM	Random Access Memory
RC	Radio Control
RGS	Remote Ground Station
RPi	Raspberry PI
RTMP	Real-time Transport Messaging Protocol
RTSP	Real-time Transport Streaming Protocol
RTT	Round Trip Time
TCP	Transport Control Protocol
UART	Universal Asynchronous Receiver/Transmitter (UART)
UAV	Unmanned Aerial Vehicle
UI	User Interface
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
UV	Unmanned Vehicle
TUAV	Tactical Unmanned Air Vehicle

VTOL Vertical Take-off and Landing

This page was intentionally left in blank

Chapter 1
INTRODUCTION

This Chapter, introduces the thesis, the motivations behind the project, the contributions of this work, the objectives and the state of the art of the field.

1.1 Overview

Drone activities have been increasing in the last decade, what then was mainly used for military purposes is now a leisure activity for many aviation enthusiasts. Radio Control (RC) is the major communication method used between the operator and the drone, however with the most recent developments regarding cellular networks, there is a way of exploring the beyond line of sight limitation that exists in almost all RC controlled vehicles.

Drones or Unmanned Aerial Vehicles (UAVs) vary in size and endurance, being categorized by these characteristics. This thesis is focused on one of the smaller variety of UAVs, and how can they when used in squads, of more than one vehicle, be remotely controlled and monitored by the use of a software platform.

The communication is achieved through the use of cellular networks or Wi-Fi; the flight controller is capable of receiving messages from the platform, translating the received parameters into movement induced by the propellers. The flight controller is part of the system that composes the whole UAV, it features sensors, that take an important part on the balance and stability of the drone and it also includes a GPS to achieve more precision and autonomous flight.

The software platform is responsible for the monitoring and controlling of each UAV, allowing for the monitoring of more than one at a time. Aided by a server that will be responsible for handling the users, associated drones and the forwarding of messages the software platform/server system should provide a secure and reliable way of managing a squad of drones without the limitation of the traditional RC range of communication.

Besides monitoring and controlling, the application will serve the purpose for an admin to keep track of all the registered and related user’s and their respective drones. With such information the admin will be capable of overriding control, removing access from certain users to the main server, disallowing them and making them incapable of communicating with their previous drone. When controlling, with the use of the onboard Raspberry Pi (RPi) and its camera, the users will have access to a live video feed.

1.2 Motivation

An unmanned vehicle has the capability of travelling through ground, air and sea, whilst being remotely controlled through a platform like a computer or any other that enables the communication between it and an operator. At first this technology, regarding UAVs, was mainly available for the military, however with recent development it is now a common leisure activity and the technology is far more accessible and cheaper.

There are several reasons to why development and research around UAVs is very useful. For starters, unmanned vehicles allow the work to be done without the need of having a person present, this is especially useful when talking about war scenes, which may be dangerous environments and so the presence of humans should be avoided as to prevent casualties. Other uses for this type of vehicles could be of surveillance type, monitoring forests to prevent fires, patrolling borders and coastlines to control illegal activities. Another example could be the use of unmanned vehicles to provide emergency equipment for people in need of urgent medical care. It could also be useful to have a drone to perform management tasks on a farm or other work places, increasing productivity (Global Research News, 2014). Some private companies are also investing in unmanned vehicles, like Amazon who recently stated their intentions on building “delivery drones” using UAVs (D'Onfro, 2014).

On 2015, Rutgers University from New Jersey, United States announced and released a video showing that they managed to create an UAV that was capable of going underwater and moving. The United States Navy already invested in the project as they see a lot of potential in the use of hybrid drones that have the capacity of flying as well as moving underwater (Blesch, 2015).

Some smaller companies also make use of unmanned vehicles, especially UAVs, for leisure purposes providing applications and innovative designs for smaller drones to be used by everyone who’s interested in UAVs.

The vast majority of hobby UAVs are controlled through the use of RC whilst the military ones are controlled by satellite communications. The recent development of cellular networks and the introduction of 4G networks like LTE makes room for the possibility of controlling unmanned vehicles with low latency making it possible to receive live video for control behind line of sight.

It is then useful to work on a software platform that will improve the already existing studies whilst enabling the control and monitoring of squads of unmanned vehicles, with the use of cellular networks as a mean to effectively establish a communication with drones.

1.3 State of the art

The research on UAVs over the years has provided studies and projects, that include the development of software and hardware, that is capable of providing open-source or proprietary tools which enable the drone to fly remotely controlled or autonomously with the required flight stabilization. These projects are constantly being updated and contribute to the high availability of civilian UAV technology.

1.3.1 Flight controllers

Focusing only on open-source flight controllers, two types were considered: ArdupilotMega (APM) and the Pixhawk.

1.3.1.1 Ardupilot

The APM is as the Pixhawk a free and open-source project and an autopilot able to control fixed-wing aircraft, multi-rotor helicopters as well as traditional helicopters. It is capable of delivering autonomous stabilization to the vehicle as well as way-point based navigation. The use of the APM allows for the following features to be used:

- Firmware loading for different types of UAV configurations
- Two-way telemetry using Mavlink
- Autonomous take-off, landing and special action commands such as video and camera controls.

These features make the APM a good alternative to the more up to date and faster Pixhawk.

1.3.1.2 Pixhawk

The Pixhawk autopilot is a high performance autopilot that represents the combination of the PX4FMU and PX4IO modules. It manages different types different types of firmware depending on the type of vehicle used thus requiring different firmware for fixed-wing, multi-

rotors, cars or boats for example. It features sensors and is a more up-to-date and faster processing alternative to the APM (pixhawk.org, 2016).

Being targeted at high end research or amateur activities it provides all the mechanisms needed for this thesis research purposes.

1.3.2 Raspberry Pi

The Raspberry Pi (RPi) is a low-cost, credit card-sized computer that may be plugged in to a monitor and be used as a regular computer using a keyboard and a mouse.
The latest and most powerful RPi, the RPi 3 Model B has the following specs (raspberrypi.org, 2016):

- A 1.2GHz 64-bit quad-core ARMv8 CPU;
- 1GB of RAM;
- 4 USB Ports;
- 40 GPIO Pins;
- Full HDMI port;
- Ethernet port;
- Combined 3.5mm audio jack, camera specification interface (CSI), Display specification interface (DSI), Micro SD Card Slot and VideoCore IV 3D graphics core.

Besides the possibility to be used as a personal computer, the RPi is useful for the making of electronic systems and for programming amongst other things. It is going to be used on the UAV as a companion computer to make possible the exchange of data between the ground control station and the Pixhawk/APM, through the use of a bus cable connecting its General Purpose Input Output pins (GPIO) to their telemetry pins. The RPi has the specs needed to work effectively and together with its low price constitutes a good mean to achieve the communication between the drone and the ground control station.

1.3.3 Software

To build the application it is important to choose what is going to be the language to be used, and what is the software needed to build the application. Several languages could be used, however Java, is the choice taken for its versatility and usability.

1.3.3.1 Java and JavaFX

Java is a programming language launched in 1995 by Sun engineers. It has since been key for the development of all type of applications. It is now in its 8th version offering good performance, being transversal to all operating systems and also providing vast amounts of documentation. It also offers good User Interface (UI) designing tools like JavaFX and Swing. JavaFX is the chosen API for the development of the UI.

First released in 2007 it developed slowly and only in 2010 Oracle announced that JavaFX was to have a much bigger part in Java than what it then used to have. The latest Java version 8 was the first one to have ever included JavaFX in the distribution, ready to be implemented by developers. According to Oracle, it is the natural replacement for Swing, which will continue to be present in the JRE, however Oracle encourages all developers to start working and deploying their applications using JavaFX. The Scene Builder, a visual UI designing tool also aids and visually improves the developers work.

1.3.3.2 Ground control stations

Over the years with the use of more and more open-source flight controllers from the user's behalf, some software platforms, more commonly known as Ground Control Stations (GCS), started as projects to control and monitor drones. Through the exchange of MAVLink Messages, the protocol used in the exchange of messages between the flight controller and the GCS, it is possible to control and monitor vehicles. The current MAVLink version to the date of this thesis is 1.1.

1.3.3.3 MavProxy

MAVProxy is a fully functional GCS for UAV's. It is a command line, open-source based GCS, however, there are some plugins that provide it with a GUI for better usability. Its simplicity, and the fact that it is built in python make it transversal to all OS's and make it possible for it to be installed on the Linux RPi operative system. Figure 1.1 represents the MavProxy GCS application.

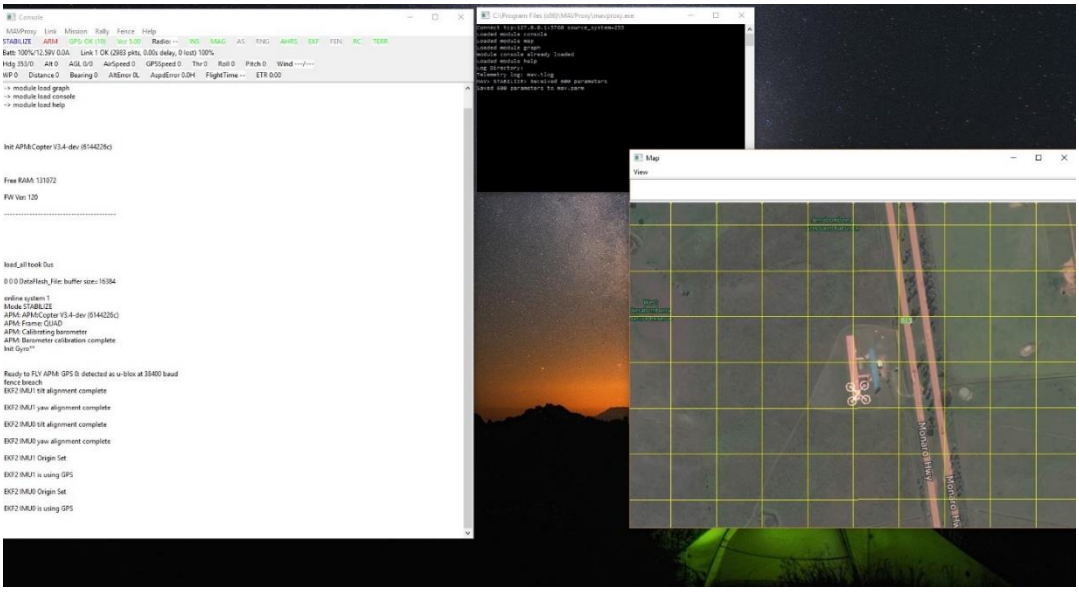


Figure 1.1 Example of the MAVProxy GCS. The console is on top, the visual UI on the right and the telemetry on the left.

1.3.3.4 Mission Planner

The Mission Planner GCS is one of the most complete open-source GCS's to be used on the control of UAV's. It is built to be used on the Ardupilot open-source projects. It gives the user the possibility of loading the corresponding firmware on to a flight controller, like the APM or the Pixhawk, to load configurations of three types of vehicles: plane, rover or copter.

Besides the firmware it also presents the user with tools to setup and configure the vehicle. If the user intends to plan autonomous missions for the vehicle it also provides that option as well as saving or loading previous planned missions. Regarding telemetry and the monitoring of the vehicle it also provides information about the UAV's statuses if the hardware is fit to do so.

Mission Planner is only available on Windows and this limits its use to only this operating system. The reliability and functionalities of Mission Planner make it a good standard of what tools a GCS should provide its users to make for a good user experience. Figure 1.2 represents the Mission Planner GCS and its main screen.

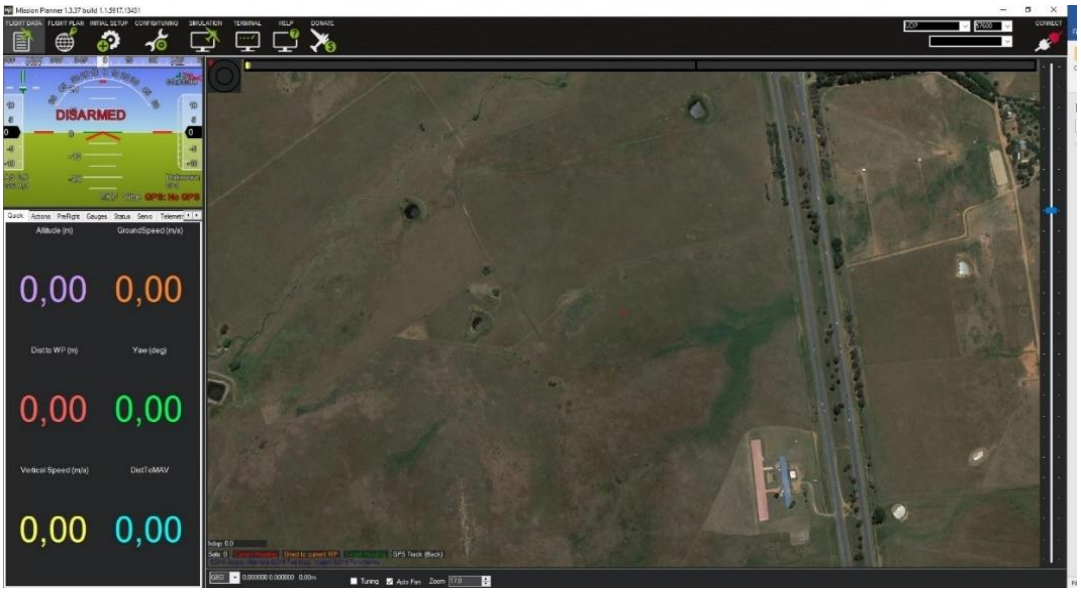


Figure 1.2 The Mission Planner GCS.

1.3.3.5 Previous work

This thesis follows on the work done by Tiago Saraiva during the last year. Tiago Saraiva's work focused on the making of an Android application capable of controlling a UAV through the use of a tablet and smartphone. Tiago's work was important as it involved some subjects worked on this thesis, such as the MAVLink messages protocol, a public server to overcome the Network Address Translation (NAT) and it is also contemplated the use of the RPi on the drone.

1.4 Objectives

The main goal of the thesis is to develop a software that allows the control and monitoring of squads of unmanned vehicles. As so, the objectives can be described in the following topics:

1. Communication with the vehicle: Development of a computer application to establish a connection to the vehicle and receive real time information such as video and telemetry.
2. Monitoring: After the initial phase and with a connection assured we have to ensure that we receive the coordinates for one or more vehicles for real-time monitoring purposes.
3. Control: As we monitor the vehicles, the user has to have the possibility of choosing a vehicle and take control of it. After this is done, the user should receive live-video from the vehicle and will be responsible for its control.
4. Override: If a drone is being controlled by another user, the controller behind the GCS should be able to override and take control of the vehicle previously controlled by the user.

1.5 Contributions

This dissertation proposes to develop a software platform that will allow for the monitoring and control of a squad of vehicles. It is intended that this thesis will achieve the goal of successfully monitoring more than one associated drones with the use of a remote server, to manage and guarantee the exchange of messages between drones. Although it can monitor several drones, it is intended that one user should only be able to control one associated drone at a time. The system should have an admin, or a user with special credentials, that has the possibility of monitoring the activities from all registered users and their drones, and, under special circumstances, the admin may overtake and remove the user from accessing the drone.

On the drone, a RPi is present to ensure the communication between the user and the flight controller, as well as a RPi Cam, to ensure the transmission of a live video feed. It is also proposed that this system should work using cellular networks for out of line of sight control.

This project has contributed to:

- Ciência Viva ‘16 Encontro com a Ciência e Tecnologia em Portugal: Demonstração Drones controlados por telemóveis July 2016;
- Lectures, drone workshops and activities during one week for Ciência Viva during July 2016.

Also, a scientific article about the work developed in this project is going to be written.

1.6 Dissertation structure

This chapter will serve to present the dissertation structure. It is divided in seven chapters in the following way:

Chapter 1 is the Introduction chapter, it will include a brief overview of this thesis, the motivation behind it, the state of the art and also the targets and goals to be achieved.

Chapter 2 presents the UAVs system, how UAVs are categorized, how GCSs are used to control the drones and the distinction between them and this thesis proposed application.

Chapter 3 describes the electronics behind the UAV. The RPi will also be introduced in this chapter as well as its camera module and the important MAVProxy GCS.

Chapter 4 is intended to explain how the UI of the software platform was built as well as what was the software used.

Chapter 5 will go into more detail on how the application works, the connection to the server, the server itself, the manual and autonomous control, the monitoring, the Raspberry Pi integration in the system and the MAVLink protocol.

Chapter 6 shows the results and an evaluation of the application system.

Chapter 7 presents the main conclusions and topics for future work.

This page was intentionally left in blank

Chapter 2
UNMANNED AERIAL VEHICLES

This chapter will describe the meaning of an unmanned aerial vehicles in the context of flying squads.

2.1 Unmanned aerial vehicles

It is understood that an UAV is a pilotless aerial vehicle that is capable of being guided through remote control by any person or autonomously by using pre-programmed software or navigation systems based routes (The Utilization of Unmanned Aerial Vehicles (UAV) for Military Action in Foreign Airspace, 2014). For these vehicles to properly fulfill their purpose they have to be able to take commands from a Ground Control Station (GCS), maintain a secure and reliable connection between the planes and controllers at all times, have failsafe mechanisms and have adequate navigation systems so that the vehicle can move to accurate and precise locations.

Figure 2.1 Example of a drone.

2.2 Unmanned aerial vehicles categories

UAVs can be categorized according to three main characteristics: range, size and altitude. Typically, the higher the drone goes, the heavier it is and the larger its range. According to this it is possible to categorize each UAV in the following way (Austin, 2010):

Table 2.1 UAVs categories description. Source: (Austin, Reg, 2010), (Unmanned Aerial Vehicles An Overview , 2008)

Category	Takeoff Weight [kg]	Range [km]	Endurance [hours]	Altitude [m]
HALE	2 500 – 12 500	Trans global	24 - 48	15 000 – 20 000
MALE	1 000 – 1 500	<500	24 – 48	5 000 – 8 000
TUAV	150 – 500	100 – 300	6 - 10	3 000 – 5 000
Close Range UAV	150	100	2 – 4	3 000
Mini UAV	< 30	30	< 2	150-300
Micro UAV	0.10	20	< 1	250

Table 2.1 describes how the different UAVs categories differ from each other, the description of each of these categories is:

- HALE: High Altitude Long Endurance;
- MALE: Medium Altitude Long Endurance;
- TUAV: Medium Range;
- Close range;
- Mini UAV;
- Micro UAV.

Each of these categories serve different purposes. The first four ones are only for military use whilst from the Mini UAV and below categories are adequate for civilian purposes.

The latter categories are dependent on three more important ones that are related to the drone's construction, these can be Fixed-Wing, Rotary-Wing, Fixed-Wing Hybrid and Multi-Rotor (Chapman).

Fixed-Wing aircraft have a fixed wing which surface will generate the major lift of the drone, giving the vehicle natural gliding capabilities and making these more suitable for carrying more equipment for longer distances with less power. These types of drones are not capable of vertical take-off and landing (VTOL). The HALE, MALE, TUAV and Close Range UAVs are usually fixed-wing aircraft.

Rotary-Wing aircraft is an aircraft with only one rotary wing, similar to a helicopter, that gives this vehicle the propulsion needed to fly and to vertically take-off and land.

Multi-Rotor drones consist of aircraft with several rotor-blades that revolve around a fixed mast. Many setups are possible and these vehicles have more stability than the fixed-wing and rotary-wing. Multi-rotors and rotary-wings are more indicated to be of the Mini or Micro UAV category.

For this thesis purpose it was used a Mini multi-rotor UAV, as it is more adequate for civilian activities mainly for the stability, the VTOL capability and the ease to pilot and configure.

2.3 Ground control station and control tower

As it is known a UAV needs to be remotely controlled, whether it is more or less autonomous it always needs human interference in order to make sure that the drone behaves appropriately. For this to happen the drone needs to communicate effectively with a Ground Control Station (GCS).

In this thesis it is going to be made a distinction between two types of controlling methods, the Control Tower and the Ground Control Station, that may sound virtually the same, however there are differences in their use and consequently they can have different purposes.

To effectively distinguish the two, it's important to take a look at the real life already existing examples and study their use.

2.3.1 Ground control station

UAVs are remotely controlled by the use of GCSs, this allow each user to pilot the vehicle in real-time, using video if provided from an onboard video-camera or by visual line of sight, and it is also possible to have a drone fly a pre-programmed path if it has the possibility of using navigation systems. The communication between the drone and the GCS also allows the user to retrieve additional information from the UAV like battery status, payload, flying information status, etc... In the case of civilian use these GCSs can be connected to a particular drone and are available as smartphone applications or as pc software. These civilian GCSs often need to be near the drone flying area in order to be able to communicate with it. Figure 2.2 is an example of a civilian made GCS.

Figure 2.2 A civilian type GCS Source: (diydrones.com, 2009).

If the drone is being used by the military, these GCSs are more complex, more reliable and also more secure, they often need to have a licensed pilot operating it and it offers the technology needed to control and monitor the UAV over large distances. The drones' onboard computers are also able to manipulate the vehicle according to the mission and the pilot instructions. These GCSs do not need to be near the drone as the budget allows for big distance communication (McHale, 2010). Figure 2.3 represents an example of a military grade GCS.

Figure 2.3 A military grade GCS Source: (Industry, 2011).

2.3.2 Control Tower

Control towers are more commonly known for being present in worldwide aviation airports. Stationed on the ground, they are responsible for a certain airspace and, within this, they provide air traffic control services that are required to guide airplanes in order to prevent collisions, organize and expedite the flow of air traffic, providing information and support for pilots.

Provided the above concept, used for general aviation, the same can be applied, although in a smaller scale and dependent on the user needs, to UAVs and GCSs. UAVs are already controlled through the use of a GCS, however if it's of our desire to monitor more than one drone, a typical one would not allow that to happen.

Combining the usability of a GCS and the concept of real-world Control Towers it is possible to build an application that monitors and controls more than one UAV adding extra-functionalities to a GCS, this will be further explained on chapter 4 and 5. From now on, this thesis software platform will be named as Control Tower (CT). In figure 2.4, a scheme is drawn to show the features.

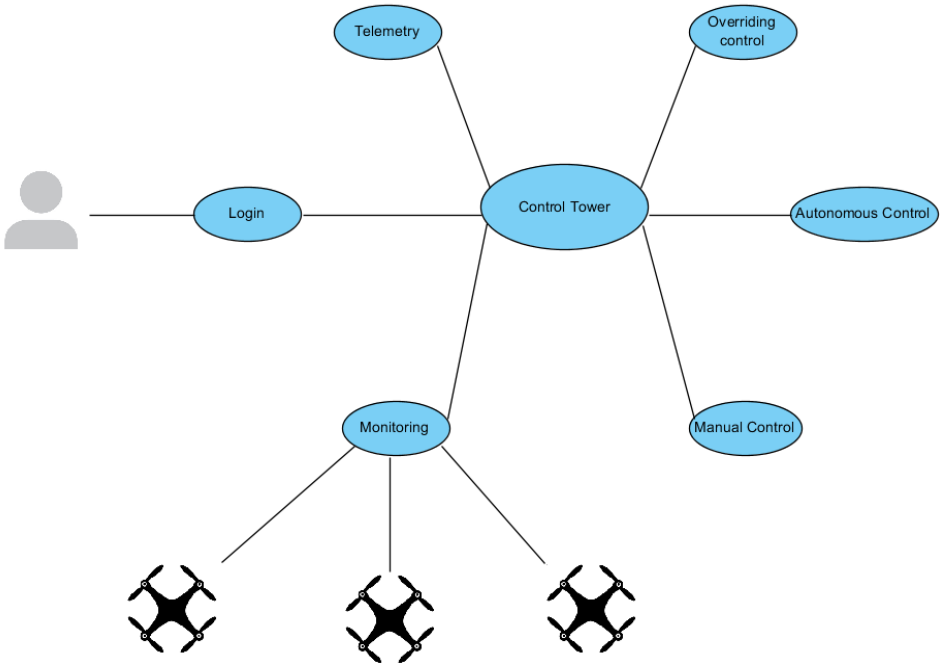


Figure 2.4 Features of the Control Tower software platform.

2.4 Communications

To make it possible for a user to operate a drone through the use of a GCS there has to be a mean of communication between the two. It's very important that the communications are assured all the time to make real-time flight possible.

Depending on the type of control there are several ways of communicating with the drone. If it is intended to control the UAV within line of sight radio control can be used, this transmits the information through radio frequency and it needs a transmitter on the GCS and receiver on the UAV. For control beyond line of sight, radio control is no longer an effective mean and as so the communication can be achieved through wireless communication links like the use of mobile networks (3G, 4G) or satellite communications.

In figure 2.5 the communications scheme pretended is explained with more detail. The drones, through the Raspberry Pi are connected either to a cellular network or a Wi-Fi access

point, connecting to the Internet and logging in to the server. The user, logins to the application, which in its turn connects to the server and then connects the user to its corresponding drone. The server is responsible for the delivery of MAVLink messages from the user to the corresponding UAV and vice-versa.

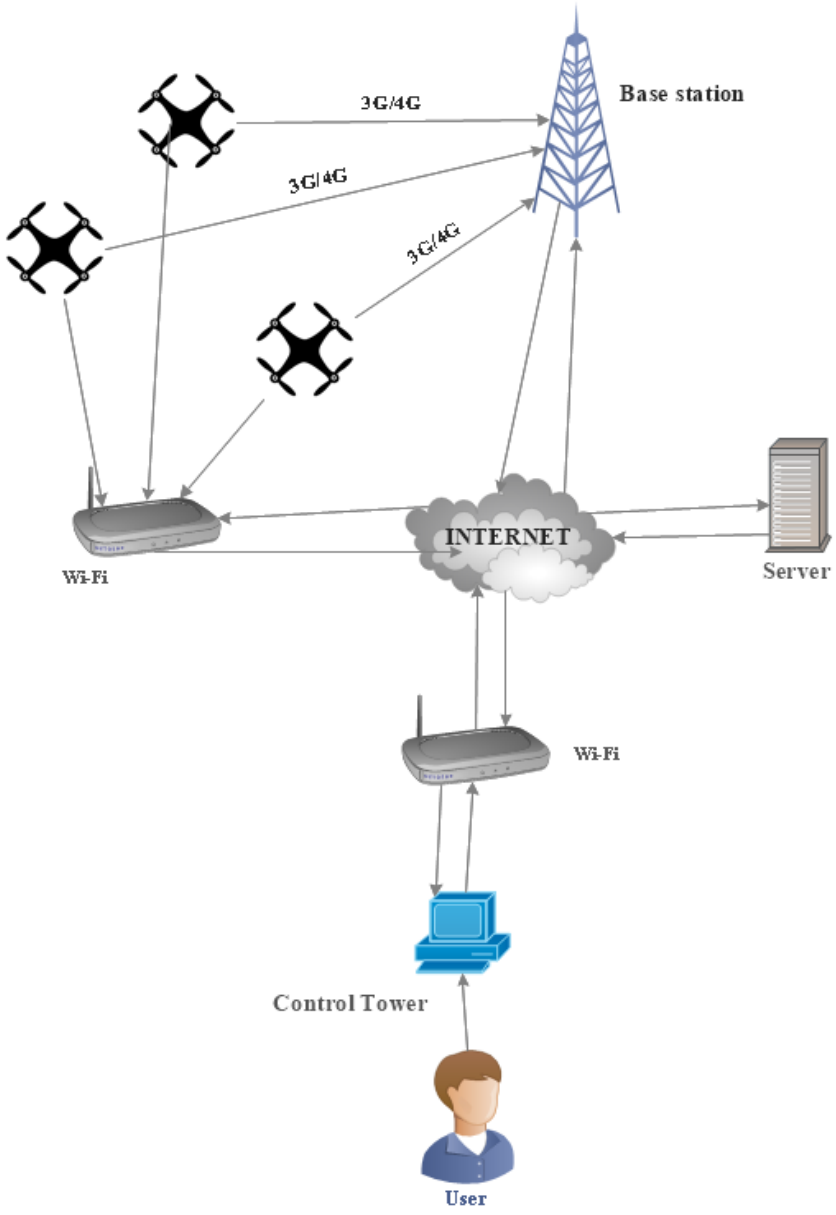


Figure 2.5 The communications scheme.

2.5 Squads of UAVs

This thesis aims to build an application focused on squads of UAVs. A squad of UAVs can be considered every time someone needs to have a task being performed by more than one vehicle, being autonomously or manually controlled. When having such a system it is useful that an admin or a more privileged user be given the possibility to monitor how all the drones are performing or being maneuvered.

According to whether the drone has an adequate behavior or not, the admin of the system has the possibility of intervening in the action and deciding to overtake its control and start to control it itself. This system has applications on a big range of markets, from the military to civilian.

This page was intentionally left in blank

Chapter 3
ELECTRONIC COMPONENTS AND SOFTWARE

This chapter presents a description of the electronics and components used in UAVs.

3.1 Electronics

A UAV has some basic components that are required for it to be able to fly, these include: the main frame, the Electronic Speed Controllers (ESC), the flight controller, the motors or actuators, the battery and the navigation systems (GPS for this thesis). Besides these main components other sensors may be present as well as additional payload, like a gimbal holding a camera. To make the communication between the UAV and the CT possible it also needs a Raspberry Pi to be present, as well as a Wi-Fi dongle.

3.1.1 Main frame

As explained on the last Chapter there are 3 main types of main frame configurations and this thesis is focused on a multi-rotor type UAV. Several configurations are possible with multi-rotors, the most usual are quadcopters, hexacopters and octocopters and they are distinguished by having 4, 6 and 8 motors respectively. The general rule, regarding theses configurations is that the more motors there are the more lifting power and stability the drone will have, however energy consumption will have to be taken in account and as more motors are present the more energy will be consumed by the UAV (Harris, 2014).

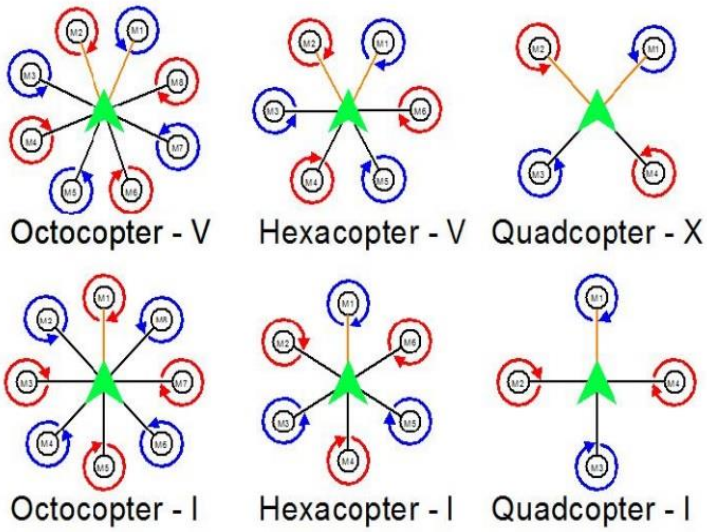


Figure 3.1 Multi-Rotor possible configurations. Source: (Harris, 2014).

For this thesis purpose, the main frame configuration chosen is an hexacopter. After having chosen the main frame configuration the frame will also have to be able to hold all of the other main components explained next.

3.1.2 Electronic speed controller

The ESC is responsible for varying the speed of the electric motor, they are connected to the power distribution board which is connected to the batteries.

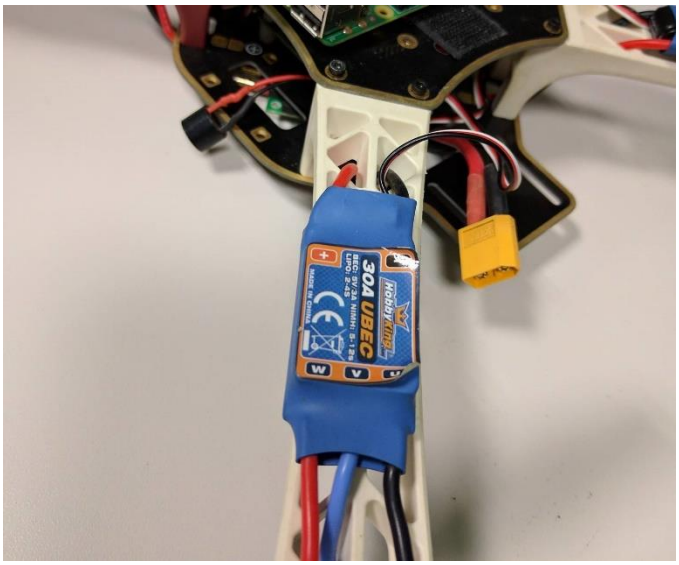


Figure 3.2 An example of a 30A ESC.

3.1.3 Flight controller

The flight controller is the unit responsible for controlling the UAV, it communicates with the ESCs to control the motors which enable the drone to fly according to the given commands. In this thesis the flight controller used was the Pixhawk.

The Pixhawk is an open-hardware project aimed at providing a high-end autopilot hardware at low cost, its main specifications are in figure 3.3.

Figure 3.3 The Pixhawk and its main specs. Source: (pixhawk.org).

As seen in figure 3.3 the Pixhawk includes four sensors, the 3-axis ST Micro LSM303D accelerometer/magnetometer, the ST Micro L3GD20 3-axis gyroscope, the Invensense MPU 6000 3-axis accelerometer/gyroscope and the MEAS MS5611 barometer. The accelerometer is used to measure the total acceleration present, whether it be the earth's gravity or 3D space movement, the gyroscope is able to measure spin and twist, the magnetometer is able to measure where the strongest magnetic force is coming from, generally the magnetic north, and the barometer is used to measure the atmospheric pressure.

The Pixhawk also features 14 PWM (Pulse Width Modulation)/servo outputs represented in figure 3.4. This means that out of these 14 outputs 8 can be used to connect to the ESCs and control 8 motors meaning the other 6 can be used to control servos.

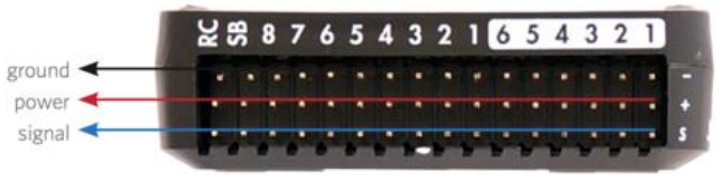


Figure 3.4 Pixhawk PWM Outputs. Source: (pixhawk.org).

There are also several input ports which are important for this thesis purposes. These are the GPS, which is connected to the GPS port allowing for the use of several flight modes during autonomous or manual flight, the Raspberry Pi is connect to the telemetry port to assure the RPi is able to receive and send data.

The radio input signal which goes on the Spektrum receiver port and enables the controller to receive radio communication, the power module connected to the power port as to provide power to the flight controller and the buzzer to the indicated port as a mean to let the user know if everything is going according to the expected. Regarding the buzzer four different sounds are distinguishable, one for errors, another for calibration, one for arm/disarm and one to inform that the GPS signal has been locked.

The firmware running on the Pixhawk is the APM Copter as the UAV is a multi-rotor. This firmware allows for the communication between the flight controller and the CT through the use of MAVLink messages. This offers the users several flight modes to be used when convenient and it gives the possibility of using autonomous waypoint based flight as well as real-time control. A list of the most used flight modes is exemplified on table 3.1.

Table 3.1 Most used APM Copter flight modes (Ardupilot Dev Team, 2016)

Flight Mode	Description
Stabilize	Allows the operator to fly the vehicle manually, regulating automatically the roll and pitch axis
Altitude Hold	Using this mode, the operator maintains a constant altitude
Loiter	The loiter mode maintains the current location heading and altitude. If the control sticks are released the vehicle will hold its position
Return-to-Launch (RTL)	Sends the Copter from its current position to the pre-defined home position
Auto	The Auto mode will instruct the copter to follow a pre-programmed mission stored in the auto-pilot
Guided	Guided Mode will instruct the copter to go to a designated target location

3.1.4 Motors

Motors along with the propellers are responsible for generating the thrust needed to lift the UAV. When building a UAV, it's important to choose the right motors as it is needed to have in consideration the thrust to weight ratio. As is the case with multi-rotors it is important that the motors can produce around 50% more thrust than the total weight of the drone (Alex, et al., 2015).

3.1.5 Battery

The batteries are the UAVs power supply. They provide the power to all the components present in the UAV. When choosing batteries to deploy on the drone it is important to consider their weight and the power they are able to output.

3.1.6 Navigation systems

Navigation systems are crucial for the piloting of the UAV. The user needs to know the drones position accurately at all times in order to accurately monitor from the CT and this is possible through the use of a GPS connected to the flight controller. Besides monitoring, the CT is also able to control the UAV, and with the aid of navigation systems it becomes

possible to set waypoints on a map, extract its coordinates, and send it to the flight controller that will enable autonomous flight and guide the drone to the desired waypoints. Besides this, as seen on table 3.1, the RTL mode is only available when navigation systems are present as it needs to know the “home” coordinates in order to guide the drone safely to that position.

3.2 Raspberry Pi

The Raspberry Pi (RPi), represented in figure 3.5, is a credit card sized computer used with the main purpose of connecting the UAV to the CT. Its preferable operating system is the Raspbian, a Linux distribution containing more than 35 000 packages and precompiled software. The RPi is able to connect to the internet using a Wi-Fi dongle or through a standard ethernet cable connection, and thus is able to connect to a remote server to start the communication process. It is connected to the telemetry port on the Pixhawk and with the use of Mavproxy and a Java program is able to transmit the MAVLink messages to the server and the CT. A RPi camera module for live video feed purposes.

Figure 3.5 Raspberry Pi 3.

3.2.1 Pixhawk Raspberry Pi connection

For the RPi to communicate the MAVLink Messages to the CT it has to be connected to the Pixhawk. This connection is made through the use of UART cables, that make the link between the telemetry port on the Pixhawk and the RPi I/O pins. This link is visually explained on figure 3.6.

3.2.2 MAVProxy

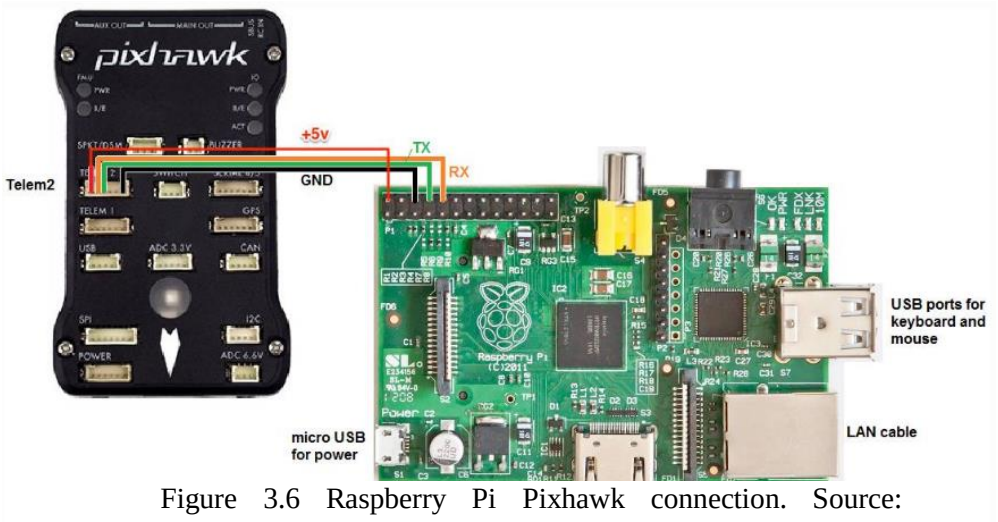


Figure 3.6 Raspberry Pi Pixhawk connection. Source: *(ardupilot.org, 2016).*

MAVProxy is a minimalistic GCS for UAVs. It supports the APM Copter software which is the firmware running on the flight controller.

MAVProxy is installed on the RPi, through the use of the connection with the Pixhawk it gets the telemetry data from the UAV which is then transmitted through the form of MAVLink messages to the Java program that is able forward these messages to the public server to be then redirected to the CT. The Java program will be later explained in more detail on chapter 5.

3.2.3 Camera module

The RPi Camera Module is used to transmit live video feed to the CT. This camera is attached to the RPi, and using the internet connection provided, live streams the video. To access the video camera, terminal commands can be used, such as the raspivid which records video with a certain amount of time or the raspistill which captures still images. The camera quality can also be changed according to the user needs being the camera module capable of recording

High Definition (HD) 1080p video. Streaming HD video requires a large bandwidth and a very good and stable internet connection, it is difficult to achieve good framerates and low latency when streaming HD video.

There are also several existing libraries to access the camera module like video4linux2 (V4L2) and the python Picamera.

FFMPEG was also installed on the RPi to compress the raw data and send the compressed video over the internet as it is the most complete multimedia framework available.

Figure 3.7 Raspberry Pi Camera Module. Source: (sparkfun.com).

3.3 SITL Software

Software in the Loop (SITL) allows for users to run Ardupilot software on the PC directly, without any hardware needed to be present. It can be installed on any platform or operating system meaning that it is possible to simulate a UAV within our computer.

By launching SITL and the MAVProxy GCS it is possible to test how a drone should behave in the outside world. The GCS receives the drone telemetry, and is able to send commands seeing that same vehicle following the received commands.

With the use of SITL, it is possible to test whether the application performs correctly without the need of having a real drone.

This page was intentionally left in blank

Chapter 4
USER INTERFACE

The chapter 4 describes the user interface, JavaFX and its main importance to the Graphical User Interface (GUI)

4.1 Application

The main thesis focus is the building of an application that allows for the control and monitoring of drones using a computer. This application is built using the Java programming language, being JavaFX the API used for the making of the GUI. The whole system is comprised with the Java application, the public server which is critical in the redirecting of messages and the Raspberry Pi, present on the drone to establish the communication with the flight controller. Figure related to subsections 4.2.2 and 4.3 are displayed on annex A for better visualization.

4.2 JavaFX

JavaFX consists in a set of graphics and media packages that enable developers to design, create, teste, debug and deploy rich client applications that operate consistently across diverse platforms (Pawlan, 2013).

JavaFX is being regarded as the substitute of Java Swing, which is still widely used but JavaFX is preferred as it comprises a set of utilities that are more useful to the building of the application.

4.2.1 FXML

FXML is an XML-based language and it provides a structure for building a user-interface without having to include this code within our main application (Fedortsova, 2014). This makes it easier for the developer to separate the GUI from the application logic.

Within JavaFX each FXML file is attributed a controller class that will perform all the actions regarding the interaction of the user with the scene. FXML has an active interaction with the Scene Builder, as the two are “connected”, any change performed on each one will influence the other one. Three FXML files were created in order to fulfill the applications needs, each representing a different scene.

```
<Button fx:id="button" layoutX="512.0" layoutY="288.0" mnemonicParsing="false"
onAction="#goToScreen2" text="Login" />
<TextField fx:id="username" layoutX="326.0" layoutY="275.0" promptText="username" />
<PasswordField fx:id="password" layoutX="326.0" layoutY="300.0" promptText="password" />
```

Figure 4.1 Code snippet from the Login.fxml.

In figure 4.1 is an example of a code snippet from the FXML file *login.fxml* which represents the Login screen and it shows the placement of a button with the id “button”, with a call to the controller class function named goToScreen2 through the onAction=”#goToScreen2” and with a text “Login” so that the user knows what the purpose of the button is. Two other text fields are present, so that the user is able to input a password and a username.

4.2.2 Scene builder

Scene Builder is a tool provided by Java to be used with JavaFX. As mentioned in 4.2.1 it is connected with a FXML file and gives the developer an easy way of designing the GUI saving the changes directly to the file. The version used is the JavaFX Scene Builder 2.0.

With Scene Builder it is possible for the user to have an easy access to all the UI Controls and graphic tools designed by JavaFX without having the need to code them directly in the application. It is possible to select Containers, Controls, Menus, Shapes, Charts and 3D objects. It also gives the user the possibility of using a Canvas for drawings as well as SubScenes and SwingNodes.

Besides the graphical elements, it also provides an easy way of modifying the layout, whether changing the size manually or by changing the background color of the element by using CSS elements.

Each element created has the possibility of having assigned code on the controller class. For this to happen the Scene Builder allows for the controller to be manually assigned to the FXML file and through the use of the Code option it is possible to make calls to existing functions on the controller class. An id can be given to the element so that it can be easily accessed with a call using @FXML on the Java class. Other calls can be made for example, the onAction, used as a response to when a button is pressed, or Mouse Clicked as a response to when a mouse is clicked over that element, and so on... In figure 4.2, the example of the MonitorinScreen.fxml as seen on the scene builder, with the all the customization options on the left of the screen, and the layout, properties and code logic on the right.

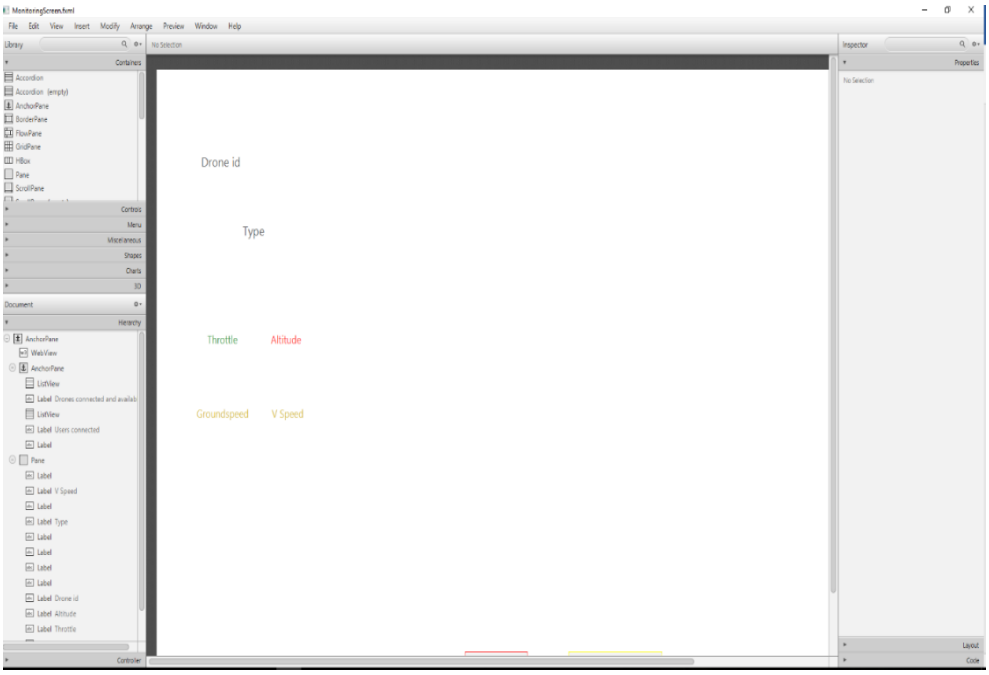


Figure 4.2 MonitoringScreen.fxml as seen on Scene Builder.

4.2.3 UI controls and CSS

The UI Controls are the heart of JavaFX. They are comprised of all the elements needed to create a visually rich GUI. UI Controls used in the application are for example:

- Buttons;
- Labels;
- Text Field;
- Password Field;
- List View.

Buttons are important because they represent “action points”, this means that the user is able to press a button to do a certain action, usually described in the button itself. Labels can be used to store graphics, images or just plain text, being mostly used in the application to show text information. Text Fields and Password Fields are similar, they both allow the user to input plain text that can later be accessed to perform a certain action, the only difference is the password field “hides” the text characters so that they cannot be read on the screen, this is useful, for example, during the login menu of the application. List Views are useful to show

information through the form of a list, making it possible for the user to scroll through a list of items.

Cascading Style Sheets or CSS is a language used to describe the style of an HTML document, and it is applied the same way it is to HTML to JavaFX. CSS can be applied to nodes in JavaFX making possible for the user to change the background color, text color or even add background images. This makes possible to change and diversify the text and background images with relative ease compared to other tools.

4.2.4 WebView

JavaFX also comes with an embedded web browser. This web browser interface is called a WebView and gives the user a component that is capable of full browser functionality through its API.

Based on WebKit, an open-source web browser engine, it has support for the afore mentioned CSS, Document Object Model (Dom) and HTML5 (Redko, 2014). This web view made possible for the rendering of HTML5 contents from local and remote URLs, the execution of JavaScript commands and the possibility of making up calls to be made from JavaScript to JavaFX.

Making use of the mentioned capabilities this tool, along with Web Engine, used to provide the basic web page functionality to support user interaction, allowed for the use of HTML and JavaScript files to render a web page. WebView and Web Engine are part of what is known as the JavaFX web component.

During the streaming of live-video some problems were found. The embedded JavaFX browser has some limitations regarding the rendering of scripts as it does not support WebGL and Adobe Flash. The solution to this was to use JxBrowser, a licensed API, which integrates a Chromium-based browser to JavaFX and thus providing the specifications needed to render and display the received video stream. The live-video stream issues and solutions will be further detailed on chapter 5.

4.2.4.1. HTML5 and JavaScript

HTML5 and JavaScript are related technologies, as they go hand-to-hand in web pages. HTML5 defines the latest standard of HTML and it is a markup language used to design web pages. JavaScript is a programming language that is mostly used to make web pages interactive and it can be directly integrated into the HTML file.

As mentioned, the JavaFX web component supports the latest features of HTML5 some of these being the Canvas and SVG, the Interactive Element Tags, web sockets, and so on...

4.2.4.2. HTML files

Two HTML files were created, one *monitoring_script.html* and one *control_script.html*, their purpose is to design a web page that embeds google maps, to be included on the corresponding web view. Since the UAVs have coordinates, it is a requirement of the CT that the user is able "see" the position of the drone in map, and as such the Google Maps JavaScript API, proved to be the most suitable JavaScript library available to customize and deliver the maps.

Starting with the *monitoring_script.html* its purpose is to create a map to allow the user to monitor the position of the drone. This is achieved by the interaction between the JavaScript and the JavaFX web components, in figure 4.3, is a scheme that helps to better understand the interaction between these two components.

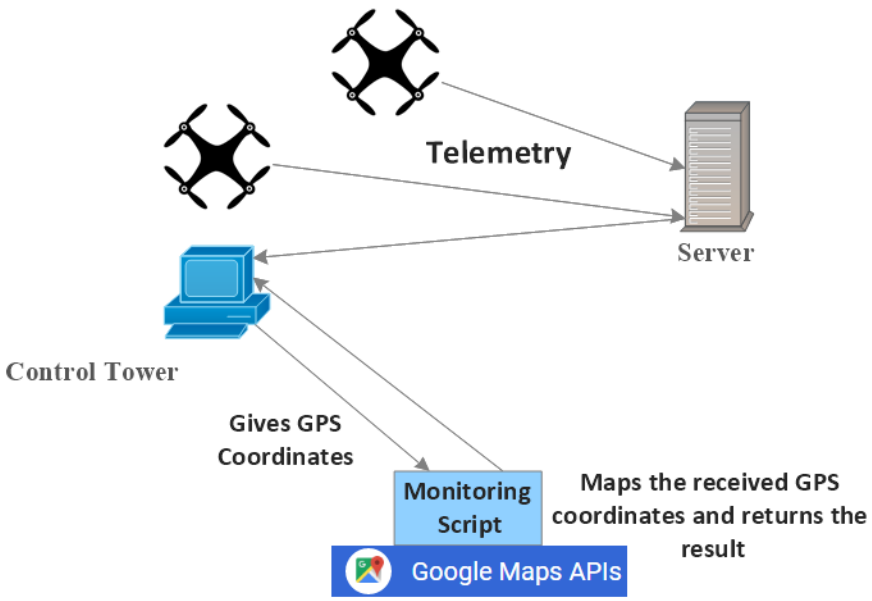


Figure 4.3 MonitoringScript and Java Interaction.

From the scheme in figure 4.3, it can be understood that the JavaFX loads the HTML file, creating the google maps then, when a drone is connected and it is intended to be monitored, the Java communicates the coordinates to the JavaScript by passing and executing the script within the file. Finally, when the coordinates are received the script creates a marker by using the Google Maps API and updates the marker constantly to keep the user up to date regarding the drones position.

The *control_script.html* file has the same logic than the file mentioned above, biggest difference is that the map is now focused on only one drone and the selected one is the only that matters. To make it easier for the user to follow this drone, the map is fixed on the drones given coordinates, and whenever the drone changes position or moves a polyline is drawn on the map so that the user knows the path taken.

4.2.5 Concurrency

To help with bigger processing demands, it is required that the application be thread safe, and this means taking advantage of multitasking to avoid making the application UI unresponsive. JavaFX is aware of this and as such provides a package, *javafx.concurrent*, that gives the developer the tools needed to create multi-threaded applications. (Fedortsova, 2012)

One of the main requirements of the CT is that it should be able to monitor more than one drone, and this would be very difficult to achieve without concurrency. By using the Task class, from the *javafx.concurrent* package, it is possible to have each drone be assigned to each task, thus enabling the user to monitor more than one drone. Each task is a thread, and runs independent from the JavaFX main thread and any other, allowing for the UI maintain its responsiveness without compromising any received data.

4.3 UI screens

As explained JavaFX is the software platform used to build the GUI. As so, the GUI is composed with all of the assets mentioned in 4.2, and for the needs of the application three main FXML files were created, the *Login.fxml*, the *MonitoringScreen.fxml* and the *ControlScreen.fxml*. Each one of these are meant for a different task and are independent of each other.

The *Login.fxml* is always the first screen to be loaded, its purpose is to show the login page which is the app loading screen. Through this, the user has access to a text field and a password field to enter the required credentials and then use the login button to verify them. It's a simple screen, that fulfills only the purposes of authentication. In figure 4.4, it is shown a screenshot of this scene.

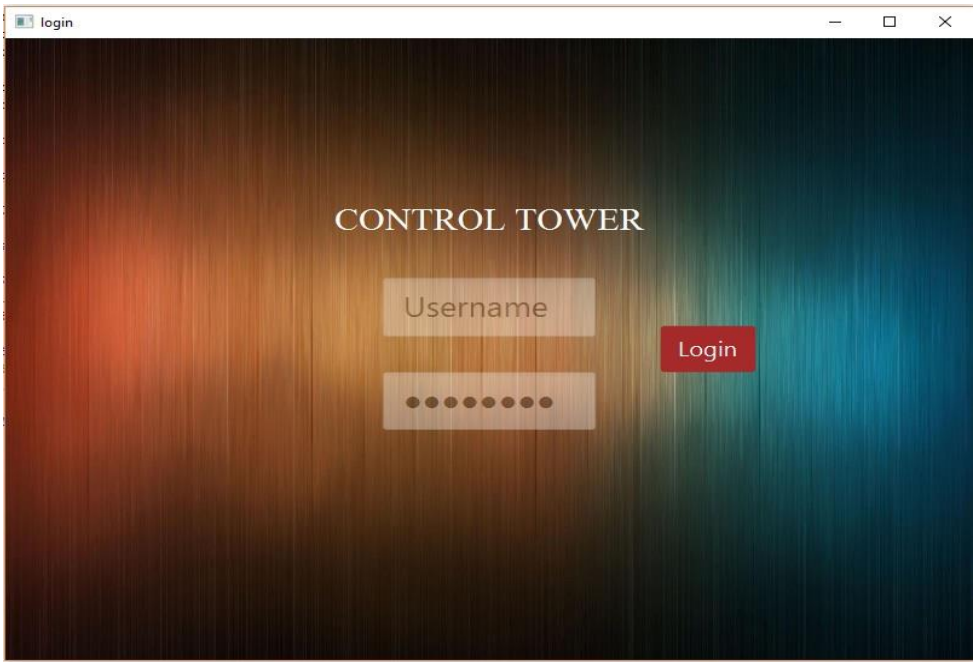


Figure 4.4 The login view.

Up next, following the authentication from the login screen is shown the *MonitoringScreen.fxml*, this screen represents the home screen, where a user is able to monitor the connected drones as well as the respective user connected to them. The background initializes the google maps and allows the user to situate where in the world each drone is connected. Two main buttons are present; one connects or disconnects from the server depending on the state of the connection and the other one allows for the control of a specific drone and advances to the next screen. On the left side of this screen, the drone status is represented; the UAV id, its type, vertical speed, groundspeed, altitude, throttle and the ping are present to show the user all the information requested. On the right side two lists are shown; the top one represent the users' connected and available to control drones, whilst the bottom list shows all the drones and respective users connected to the server at that moment. On figure 4.5, it is possible to see all the mentioned parts of the screen.

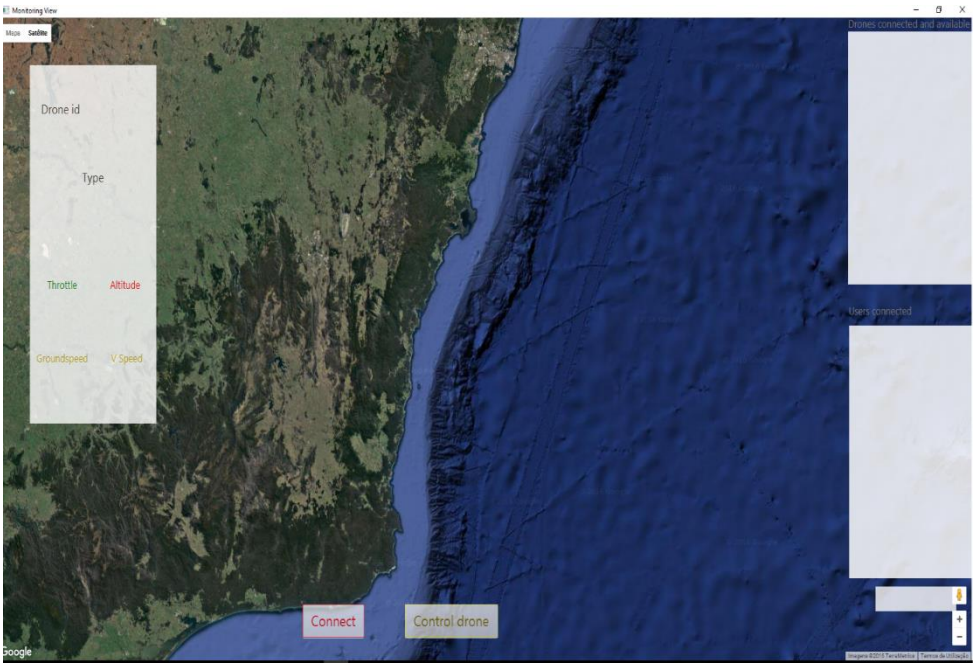


Figure 4.5 The Monitoring View.

Finally, the last screen is the *ControlScreen.fxml* and it represents the control part of the application. As this screens purpose is to control the drone it is also the one with the most features. Just like the *MonitoringScreen.fxml* the background also initializes google maps and it also provides the user with all the information regarding the drone status, on the right bottom half of this scene it is possible to see that the same information is shown. On the right top, however, more information from the UAV status is present; the battery percentage, the armed/disarmed lock and the current flight mode are present to give the user more insight about the drone. On the left bottom screen four buttons are present; the Manual Control, Autonomous Control, Start video and Go back. These buttons allow for the user the choose the type of control they want the drone to execute, manual or autonomous (mission planning). Regarding the manual control, three buttons and a combo box turn visible when prompted. The Arm/Disarm button will arm or disarm the drone, the change mode button together with the combo box allow for the user to change the drones flying mode and the Start Manual Control button will initialize the manual control of the drone. For the autonomous button, mission planning through waypoints can be executed. Six buttons will then appear:

- Add Waypoints, so that the user can add waypoints to the mission;
- Clear Waypoints, to clear any pending mission waypoints;
- Fly, to start the mission, after adding the waypoints;

- Show waypoints, this will show all the loaded waypoints on the screen;
- Send Takeoff, will send a takeoff message to the drone;
- Go to, will send the drone to a specific point in the map chosen by the user.

Some drones will also provide live-video streams and these can be accessed by the start video button. If the user however wants to return to the monitoring screen, the go back recedes to that. A screenshot of this screen can be seen on figure 4.6.

Figure 4.6 Controlling View.

This page was intentionally left in blank

Chapter 5
CONTROL TOWER APPLICATION

This chapter describes the full system behind the GUI, the application logic, public server, Raspberry Pi and the live video-stream.

5.1 Control tower

Behind the GUI the application has a logic to make all the elements connect and work. The Control Tower application as explained on the earlier chapter is divided into three screens. One allows for the user to login to the server, being the other two responsible for the control and monitoring of UAVs. The CT integrates with a public server, for message redirecting purposes and with the Raspberry Pi connected to the flight controller. The messaging protocol between the CT and the UAV is the MAVLink protocol.

5.2 MAVLink

Micro Aerial Vehicle Link (MAVLink) is the protocol used for the exchanging of messages between the CT and the Pixhawk. It is a header only message library, first released in 2009 by Lorenz Meier, its current version is 1.1 as of the date of this thesis.

Each MAVLink packet can have a length of up to 263 bytes, the header occupies 6 bytes, payload can have any size up to 255 bytes, being the last 2 bytes reserved for checksum bytes. The header's 6 bytes are: the message header 0xFE, the message length, the sequence number of the message which can go up from 0 to 255, the System ID by default set to 1, the Component ID also set by default to 1 and the MessageID which has the purpose of identifying the payload. The payload has a variable size represents the data contained in each message and is what the CT and flight controller are interested in retrieving from each MAVLink packet. The last two bytes are used for checksum, using the CRC method.

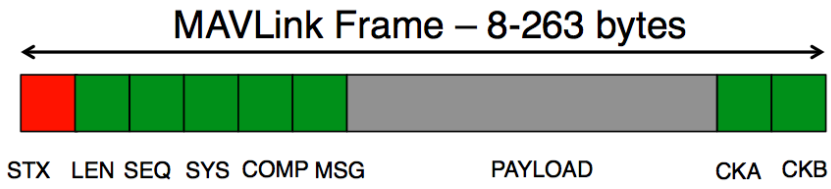


Figure 5.1 MAVLink packet structure. Source: (qgroundcontrol.org/mavlink).

5.3 Login

The login process represents the authentication of any user on the server. The process can be described in the following scheme, present on figure 5.2.

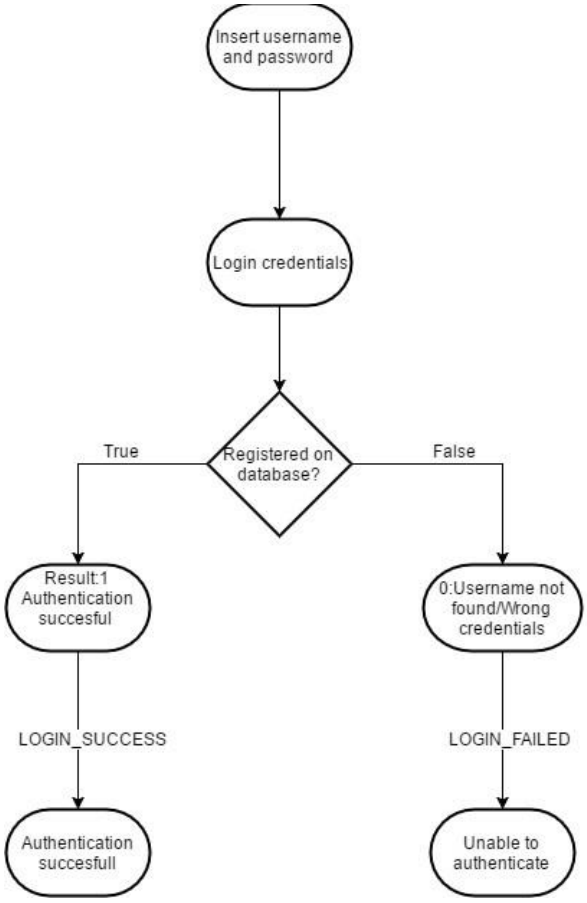


Figure 5.2 Login flowchart.

As it can be seen through figure 5.2, the user is prompted to fill the username and password fields. Next and when the user presses the login button it starts the process of authentication. The authentication has two “points” of credentials comparison to increase security. First it requires PHP authentication, this sends a JSON Post request to the server containing both the credentials and checking through the database if the user exist and the passwords corresponds, if they exist a JSON Response is issued and it can have three types:

- 0, username not found;
- 0, wrong credentials;
- 1, authentication successful.

According to each type of response an information is passed to the user so that he knows the result of the login, and in the case it failed, a warning window is shown and the user can try to login again.

The second "point" of security is through the Java code running on the server, when connecting, the application opens a TCP connection between the server and the app, and then a login message is sent to the Java code including both of the credentials, this will allow the server to check the database and respond either with a LOGIN_SUCCESS message, or a LOGIN_FAILED.

These two authentication methods are independent of each other, creating two layers of security, adding redundancy to the system, which makes it useful in case one of them is bugged or unable to work.

Another addition to the authentication system is the password encryption. To add an extra layer of security, using the *java.security* package, the cryptographic Secure Hash Algorithm SHA-512 is used to hash the password creating a safer method of authentication.

5.4 Monitoring

The monitoring process consists on the monitoring of all the information regarding connected drones. By switching MAVLink messages with the server and the RPi on the UAV it is possible to get the drones location, its information status, the user to which it is connected at that moment and if the user is logged in as the Admin it is possible to override the control of other users.

5.4.1 Drone positioning

The monitoring screen allows the user to locate the drone by using google maps. This is achieved by the exchanging of messages between the flight controller and the CT. As explained on 5.2, MAVLink messages are constantly being exchanged by the two parts, and each message provides a different type of information.

For the drone positioning the message used is the MSG_GPS_RAW_INT, that carries the values of latitude and longitude based on WGS84 frame reference and expressed in degrees $1 * 10^7$. To have the actual value of longitude and latitude we then have to divide the given value by $1 * 10^7$. WGS84 is an Earth-centered and Earth-fixed reference system.

By constantly receiving the updated position, it is possible to communicate these values to the html and JavaScript file and then update the drones position on the map.

5.4.2 Information status

Besides the location of the drone, to effectively monitor a drone it's important to receive and show all the data regarding the UAV state. This includes receiving its altitude, groundspeed, vertical speed, throttle, the type of drone and its ID. All of these information statuses are again contained on a specific type of MAVLink message, MSG_VFR_HUD. By analyzing these messages, it is possible to immediately retrieve these parameters and display them to the user.

5.4.3 Control overriding

The primary feat of the CT is the control overriding. This feature, allows the admin to take or give control to determined users according to its preference. This is mostly useful for security purposes, for example, having a squad of drones controlled by different users and performing different tasks the admin can monitor if all the users have an adequate behavior when piloting.

The control overriding is achieved by interacting with the server, as all the information is stored in it. The flowchart depicted in figure 5.3 better explains this interaction.

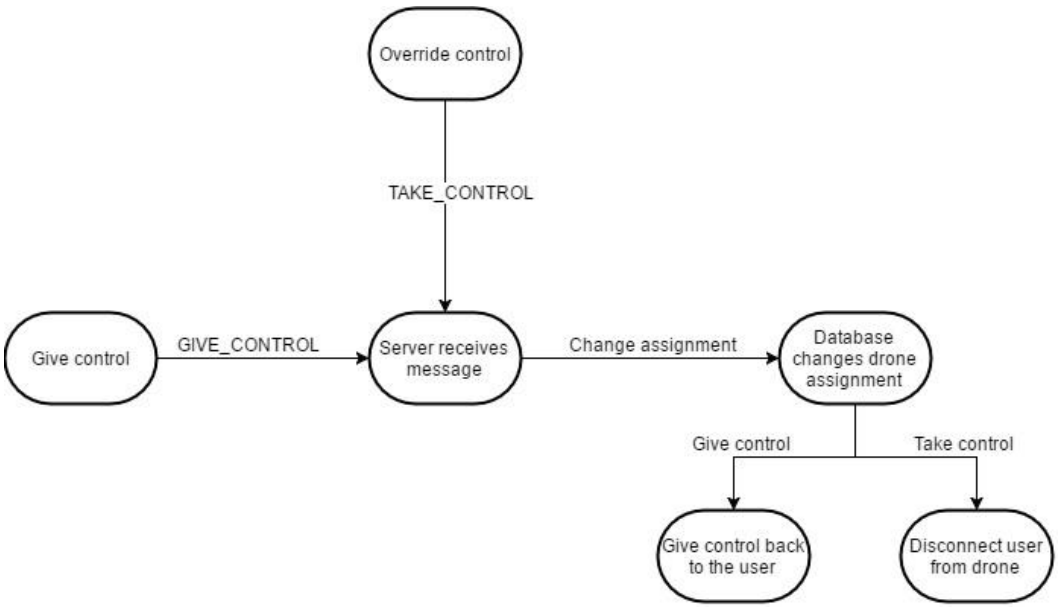


Figure 5.3 Overriding and giving control flowchart.

As it can be seen on figure 5.3, the interaction starts with the login. Only admin credentials have the privilege of being able to know which drones are online and connected to a certain user. Then, when connected, the admin will be able to identify all the drones on the map, monitoring their location and getting their information statuses as well. If, however, user and drone are not connected to each other at that moment, neither the drone nor the user are visually identifiable.

The overriding process takes place when the admin clicks on a connection between drone and user and chooses the take control option. This will trigger a TAKE_CONTROL message containing the operator username and the drone ID, sending it to the server. When the message is received on the server end, it retrieves the data, updates the database and associates the drone with the admin's IP redirecting all the upcoming MAVLink messages to the admin instead of that user.

When the user loses control of a drone to the admin it won't be able to connect back to the drone unless permission is given to do so. With this approach, some problems may arise, for example, if the operator is manually flying the drone, the sudden loss of control may cause the drone to crash. To deal and try to avoid this outcome there is a failsafe solution presented, however, good judgement from the admin is important to avoid accidents. The proposed solution to this problem is to change the drone mode to ALTITUDE_HOLD, before losing the control so that drone maintains its altitude and stays in the same position.

As the admin is capable of overriding control, it also has the possibility of giving the control back to the previous user. During the overriding process the CT knows which user has lost control, so when the user wants to give the control a GIVE_CONTROL message is again issued to the server that will follow the same principle as earlier. The database will be updated, the drone then becomes associated with the previous IP again, and communication between the user and the drone is now assured.

5.5 Control

The control part of the CT allows for the user to remotely command the drone, and it features two main points, the manual and autonomous control. As explained on chapter 4, there is a dedicated screen for this part and it gives the user more detailed information regarding the drone's status.

5.5.1 Manual control

To describe how the manual control works it's important to first understand some basic principles on how multi-rotors are able to fly.

There are 3 axis involved when a drone is moving, yaw, pitch and roll. Yaw axis represents the rotation movement of the drone, roll axis represents the side movement making the drone go left or right and the pitch axis tilts the UAV forward or backward making it accelerate or decelerate. Figure 5.4 better represents these axes.

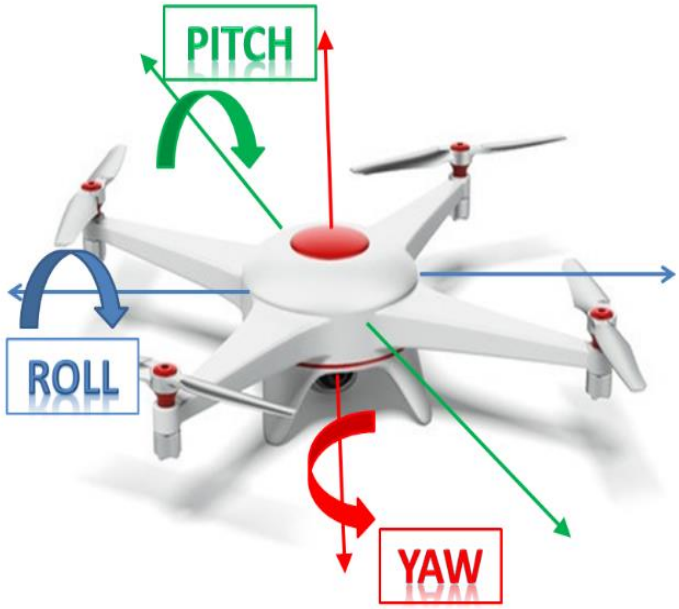


Figure 5.4 Yaw pitch and roll axes. Source: (norunway.com/wp).

The control of each of these axis is achieved by changing the motors speed. To roll and pitch all the drone has to do is speed up half of the motors and slow down the other half. The yaw is achieved by speeding up diagonally across motors, slowing down the others.

When manually controlling a drone the flight controller is in charge of converting the received commands in order to make the motors speed increase or decrease to make the drone have movement. This is achieved by the CT communication with the drone through the use of the MAVLink message MSG_RC_CHANNELS_OVERRIDE.

The MSG_RC_CHANNELS_OVERRIDE MAVLink message has as data 8 communication channels, each receives an input value between 1000 and 2000, representing the minimum value accepted and the maximum respectively. If the value given for any channel is 65535

then this means that the value is not set and is thus ignored. The first 4 channels represent the roll, pitch, throttle and yaw respectively.

As the CT has the purpose of being used on a computer there were two possibilities to manually control the drone, one is through the use of the keyboard keys, the other is by using an external controller. The first possibility had a problem with a lack of sensitivity, the keys have only two states, they are either pressed or not pressed and as such it is not very intuitive for the user to know how much it needs to press to have the drone respond accordingly. As so, the choice fell over the second possibility, using an external controller more specifically, a game pad as depicted in figure 5.5, as it resembles the radio remote controls used by RC hobbyists worldwide.

Figure 5.5 Gamepad used for manual control.

To gather input from the controller, the LightWeight Java Game Library(LWJGL) is used. LWJGL is basically a Java library that enables cross-platform access to popular native API's and in the case of the CT application the Jinput library offered by it is key to gather input from the remote controller. Jinput is also a cross-platform API, but it allows the program to poll input devices like the keyboard, mouse, joystick and game pads (Davison, 2006).

When using the game pad, just like the traditional RC controllers, there are two control sticks that can be used for the same task, one for throttle control and another one for steering. These

control sticks are identifiable as left and right analog sticks and have two axis of movement, x and y. The gathered values are float numbers, ranging from -1.0 to 1.0 on both axis (Davison, 2006). Figure 5.6 better represents this correlation.

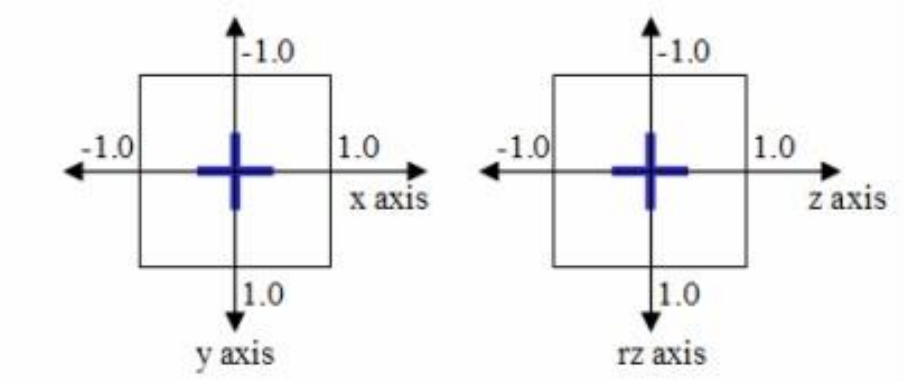


Figure 5.6 Values on the axis of the controller. Source: (Davison, 2006).

The MSG_RC_CHANNELS_OVERRIDE values oscillate between 1000 and 2000 so in order to get these values it is important to find the mathematic correlation formula between these values and the ones that are gathered from the analog sticks input. The following mathematic expression allowed for this conversion:

$$channelValue = slope * (x|y) + intercept \quad (1)$$

Explaining the equation above, the channelValue represents the value to be sent into the MSG_RC_CHANNELS_OVERRIDE, the slope has a fixed value of 500, the (x|y) represents the analog sticks axis chosen input received value, between -1.0 and 1.0, and the intercept has a value of 1500. This equation should return a value between 1000 and 2000, that will be used on the MSG_RC_CHANNELS_OVERRIDE to be sent to the flight controller.

To be able to manually control the drone the user will have to have a game pad connected to the pc, and the drone as to be set to armed. To set the drone to armed a MSG_COMMAND_LONG has to be sent to the flight controller with the command COMPONENT_ARM_DISARM and the first parameter set to 1, if it is set to 0 the drone will be disarmed.

Some problems arise from the use of a game pad instead of a RC controller, being the most important feature the lack of sensitivity. The analog stick is smaller than the ones used on RC control, and due to this the slightest changes will have a bigger value difference, this means that the user should operate the controller more carefully.

5.5.2 Autonomous control

The autonomous control is achieved through the use of waypoints navigation. This method enables the flight controller to receive GPS coordinates, and stores that information in the form of waypoints. Then when the user commands it to start, the flight controller knows the waypoints order and starts flying the drone to the correct location at the specified altitude through each one.

MAVLink has what it calls the Waypoint Protocol. This protocol describes how waypoints are sent and read from the UAV in order to ensure the consistency of information between the sender and the receiver (QGroundControl, 2016). Three different types of communication between UAV and the CT can be done regarding the waypoints protocol: read, write and clear.

5.5.2.1 Write waypoint

To write a waypoint, or to send a list of waypoints, there as to be a switch of messages between the CT and the UAV. The figure 5.7 better represents this message switching system.

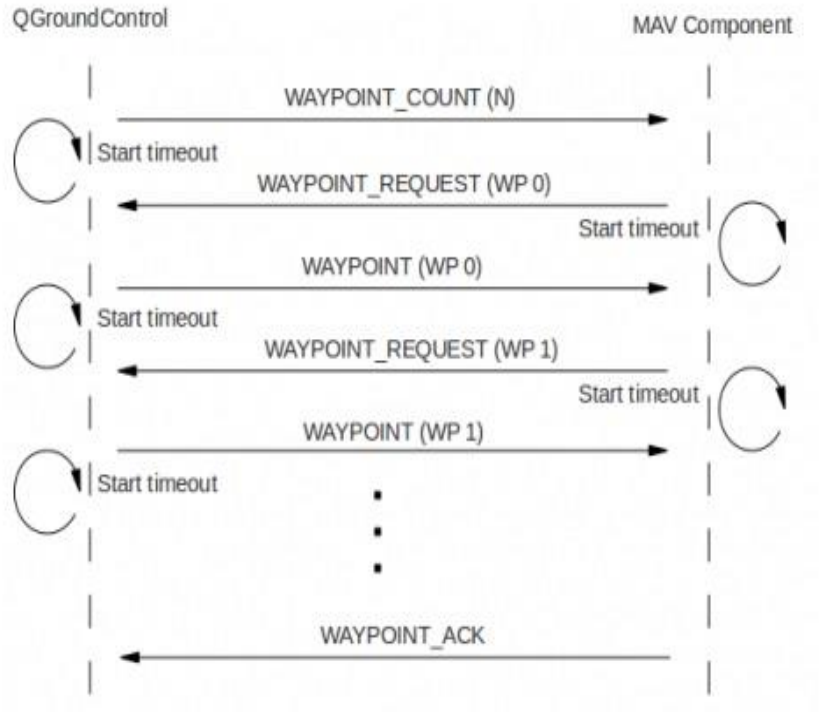


Figure 5.7 Write Waypoint diagram. Source: (qgroundcontrol.org/mavlink).

As it can be seen from figure 5.2, the process starts with the sending of a MAVLink message, WAYPOINT_COUNT, containing the number of waypoints the user wants to send to the drone. This generates a response message, WAYPOINT_REQUEST, from the flight controller asking for the first waypoint, attributing it the sequence number 0. The CT must then respond with the first WAYPOINT_MESSAGE. This procedure will repeat until the last waypoint is sent and then a WAYPOINT_ACK is sent from the flight controller confirming to the CT that all waypoints have been received. For this procedure to work, the waypoint message sent has to have the same sequence number has request from the flight controller.

5.5.2.2 Read waypoint

Within the waypoints protocol, it is also possible for the user to request the UAV about the loaded waypoints. To do this, the CT when requesting has to follow the procedures described on figure 5.8.

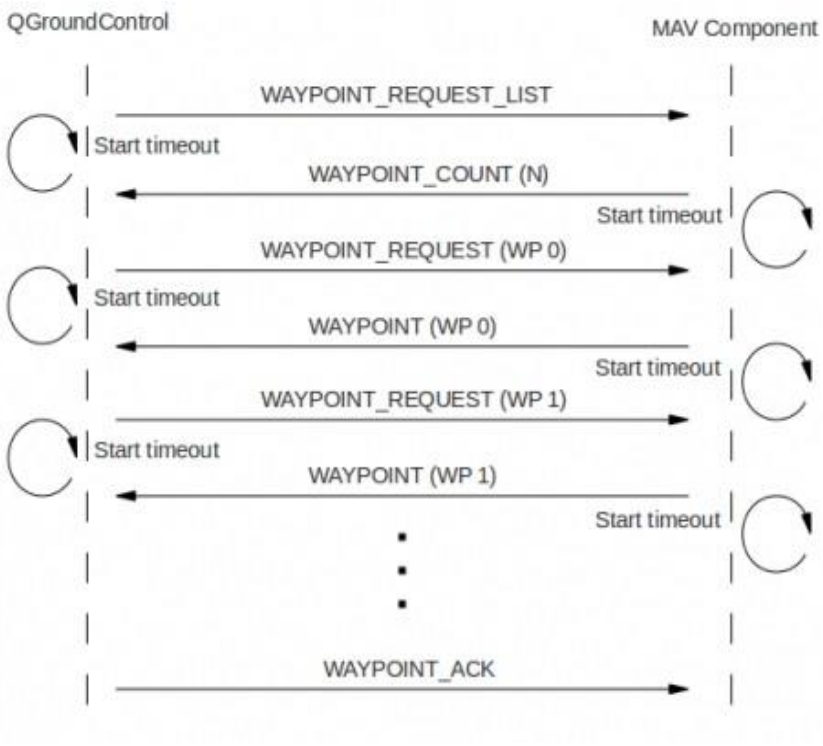


Figure 5.8 Read Waypoint diagram. Source: (qgroundcontrol.org/mavlink).

To start requesting the saved waypoints the CT first sends a WAYPOINT_REQUEST_LIST message to the flight controller. This message will trigger a WAYPOINT_COUNT message has a response containing the number of waypoints present. The CT then becomes aware of how many waypoints are presents and will send a WAYPOINT_REQUEST for each one. Each one of these messages received on the flight controller will trigger a waypoint message response containing each waypoint according to the sequence number requested. At the end of this procedure the CT will send a WAYPOINT_ACK message to inform the flight controller that all waypoints have been received.

As soon as the described procedure ends, the waypoints are then displayed to the user on the google maps display.

5.5.2.3 Clear waypoint

It is also possible for the user to clear the existing waypoints list from the flight controller if it intends to do so. The procedure is much simpler than the latter two, and it is only required that the CT sends a WAYPOINT_CLEAR_ALL message to the flight controller that when received will clear the stored waypoints list, sending as a response a WAYPOINT_ACK message confirming to the CT that it is now clear. Figure 5.9 graphically shows a better representation of the transaction. (Davison, 2006)

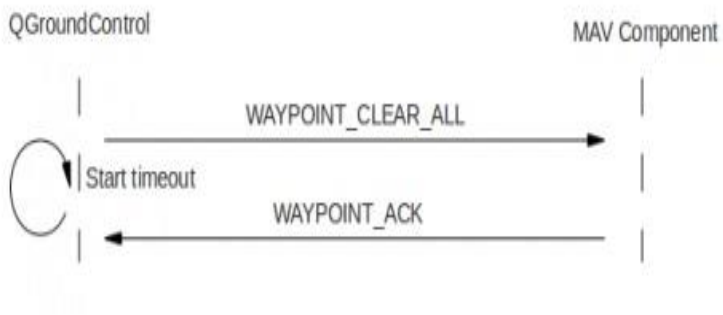


Figure 5.9 Clear Waypoints diagram. Source: (qgroundcontrol.org/mavlink).

5.6 Server

To have the CT communicate with the RPi on the UAV over the internet they both need to know each other's IP addresses. Due to both these applications being behind NATs there is no way they can effectively communicate directly. The server manages to solve the NAT

problem, using a public IP address both the CT and RPi can have access to the server, log in and switch MAVLink messages between each other.

As explained on 5.2, 5.4 and 5.5 the CT application is divided between the Login, Monitoring and Controlling, this means that the server has to fulfil certain requirements for each one of these. There are two protocols of communication available for the communication between the CT, UAV and the server, these are the UDP and TCP protocols. Each have their own advantages and disadvantages and are both used to fulfil different types of requirements.

For login purposes, it is important to have an extra layer of security to make sure that the messages exchanged are not lost, this way the TCP protocol is the best alternative as it is a more reliable protocol meant to provide error-free data transmission, retransmitting lost packets and confirming that all of them are received. When logging in, the speed of the message exchange is not important, reliability is the main requirement and TCP is the more suitable choice. For monitoring and controlling purposes, the message exchange has to be fast, otherwise the delay won't allow for real-time control. This way, the UDP protocol is the more suitable one, it offers less reliability and won't control whether the message reached their destination nor retransmit lost packets, however it will keep on delivering messages constantly as needed for real-time control. The majority of the MAVLink messages don't require any type of confirmation, however when they do the MAVLink protocol is able to provide responses in the form of ACK that will enable the CT to interpret on determined situations whether the message reached or not its destination.

The server is also supported by an online database that stores all the users and their respective passwords for login purposes, as well as all the drones that are capable of logging in, allowing for several users and drones to be online at the same time without interfering with each other. The database also has default assignments for each of the users and drones, this means that each user can only have access to its assigned drones being the admin the only one who has the capability of visualizing all of the online drones and users.

When the server has more than one user online, each assigned to a different drone, it faces a redirecting problem on how will it know which message belongs to whom. To address this issue, the server assigns the IP address of each user to the IP address of each assigned drone using a handler, this way, every time it has to dispatch a message coming from either of them it checks each handler for the assigned IP address and then sends. Another issue arises when a user has multiple drones connected to it and it needs to distinguish them from each other. The

MAVLink messages on their own don't have any identifiers, and to the CT when they're analyzed they don't differ from UAV to UAV, so in order to address this problem, the server encapsulates each message with the IP address of the drone it is being sent from before dispatching. This way, when the CT receives an encapsulated message all it needs to do is unpack the incoming message and extract the IP address of the drone it comes from and the user is now able to distinguish multiple UAVs from each other.

5.7 Raspberry Pi

Each drone is equipped with a RPi, to allow for a connection to the internet and a connection to the flight controller. By connecting to the internet, and running the inside scripts the RPi starts the Mavproxy which identifies the connection to the Pixhawk and thus captures all the MAVLink messages coming from and to the flight controller. The scripts also start a Java program that is responsible for logging the drone to the server, dispatching the MAVLink messages to the server and receiving the incoming ones. The RPi is also responsible for the handling of the streaming, and together with the server will provide a live video stream to the CT.

5.8 Streaming

In order to have live video streaming from the drone two approaches were studied and tested, one used Wowza Streaming Engine as a streaming server to redirect the stream from the RPi Camera to the Control Tower and the other was through html5 video stream through web sockets. Both of these methods used the RPi Cam, and required a stable internet connection to have the required delay times.

5.8.1 Wowza

Wowza Streaming Engine came as a solution to find a streaming server that was capable of delivering the video live stream to the CT. To do this, Wowza was installed on the public server to be accessed at all times.

Wowza incorporates streaming protocols such as the Adobe Flash RTMP, the Adobe Flash Dynamic Streaming, Apple HTTP Live Streaming, MPEG-DASH, RTSP/RTP, among others. This made possible the encoding of the video-stream in the RPi using FFMPEG, to an RTMP link redirected to the public server's IP address. When the stream connected all it is needed is

a player on the Control Tower application to open the link to the server, to render the video stream.

With this approach some problems were encountered, the first of which is the lack of compatibility between the JavaFX incorporated media assets, and, as was previously explained on chapter 4, there is no flash plugin supported on the incorporated web engine.

The JavaFX Media Assets only support four types of protocols, FILE, HTTP, JAR and HTTP Live Streaming (HLS) (docs.oracle.com, 2016). Of these, only HLS is supported by Wowza, and it is not a protocol that fulfills the requirements needed for live video streaming. HLS works by sending video as a series of small files, with a typical duration of 10 seconds, although this value can be reduced to smaller values (developer.apple.com, 2016). By segmenting files with a minimum duration, the HLS ends up introducing a delay of around 10 seconds on average which makes it impossible to control a live drone. Using the incorporated media assets was no longer an option, due to its limitations.

The other studied possibility was the use of an HTML player, such as JW Player to render the RTMP stream provided by Wowza. However, another limitation of JavaFX was found, as RTMP streams require the Adobe Flash plugin to work. Flash is not supported on the web engine, and thus the player won't render the incoming live video stream.

As explained on 4.2.4, the solution to render the incoming live stream is a licensed API, JxBrowser. This API, when integrated into JavaFX, made possible for the inclusion of a more complete web browser that included the latest Flash plugin, needed for the rendering of the stream. The results, using Wowza, FFMPEG as an encoder on the RPi and the browser on the CT, averaged around 0.5 seconds.

5.8.2 HTML5 via web Sockets

To complement the last alternative, another solution was found based on the work of Phoboslab where the author provided a tutorial and all the necessary html and javascript files needed to make the project work (Szablewski, 2013). This project was based on HTML5 and web sockets, and an encoded stream on the RPi behalf to MPEG-1.

To make this project work, two things were mainly important. The first was the use of the public server to store the stream-server.js javascript file, that, through web sockets receives the stream from the RPi via HTTP. Then all the receiving end needs to do is to run the

stream.html file inside the browser, which, together with the jsmpg.js javascript file, will render the webpage and the live video stream with surprisingly low delays. On average, the delays achieved using this project were 0.2 and 0.3 seconds which are enough to enable live control. These results will be further explained on chapter 6.

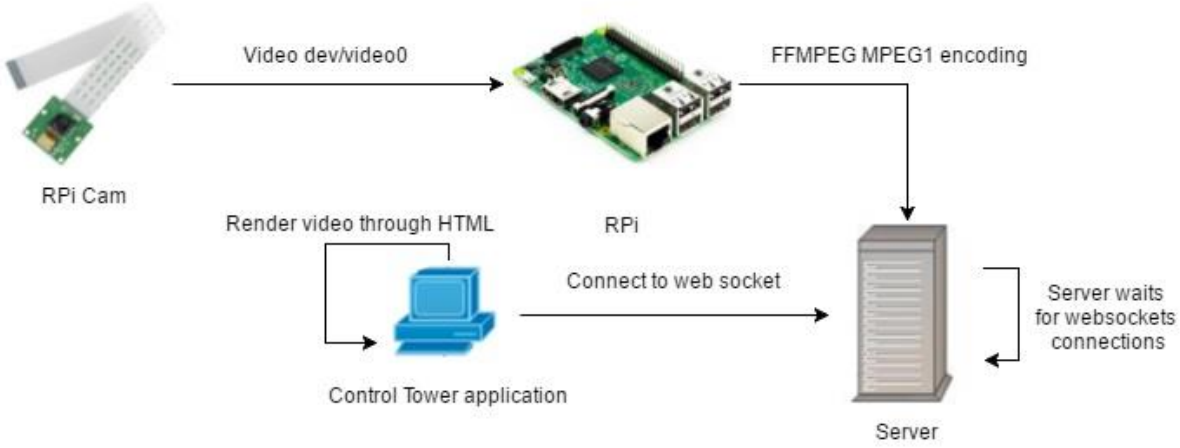


Figure 5.10 Video Stream flowchart.

This page was intentionally left in blank

Chapter 6
RESULTS

The chapter 6 presents results and performance evaluation to the application performance on experimental conditions.

6.1 Tests

To obtain the application results, some features were defined as milestones that the application had to be able to reach in order to fulfill its main purpose. These results can be defined in four different areas: monitoring, controlling, overriding and video streaming. In the monitoring phase two scenarios are distinguished if the user is the admin and if the user is not and admin. Table 6.1 details how each test should be conducted and the expected result.

Table 6.1 Tests, objectives and expected results.

Tests	Objective	Expected result
1.Monitoring	Test whether the user is able to monitor its assigned drone. If admin should be able to monitor more than one	The application should behave accordingly and allow for a regular user to monitor its assigned drone. An admin should be able to monitor more than one drone
2.Control	Test whether a user, or admin, is able to control its assigned drone, manually or autonomously	The user or admin should be able to control the vehicle live through the internet. The vehicle's movement should be visible and all the waypoints must be completed during autonomous/mission control
3.Overriding	Test whether the admin is able to override a regular user control	The admin should be able to remove the user's control. The user should lose access to the database and won't be able to control the drone unless the admin returns the control
4.Live video-stream	Test whether the application receives live video feed using Wi-Fi, 3G and 4G	The application should receive the live video feed and the time on the video should be compared to the one showed on the chronometer. The latency time shouldn't be over 500 ms and ideally should be less than 300ms

The first three tests are made under simulation conditions, using the SITL software, previously explained. The last test, will be made using a RPi and a RPi Cam, as this is what would go onto the drone to transmit the live video feed. All the following subsections figures are present in the indicated annexes, for better visualization.

6.2 Monitoring Tests

The monitoring tests serve the purpose of evaluating whether the application successfully monitors one or more vehicles in real-time. Two scenarios were evaluated, the admin monitoring and user monitoring. As explained on the previous chapters, the admin has different permissions when comparing with regular users. Admin should be able to monitor online drones, and see to which users they are connected to. Regular users are only able to monitor their assigned drones, according to the server's database. Figures from the following subsections are present in annex B.

6.2.1 Admin monitoring

This scenario tests whether the admin is capable or not of monitoring more than one vehicle at a time, the admin is able to see all the connected users on a list, and from that list is able to start the monitoring of the select drones.

Figure 6.1, represents the admin view when two users are connected and both their drones are being monitored. The two drones are represented on the first list, identified by their ID. Below the top list, it is visible to which user each drone is assigned.

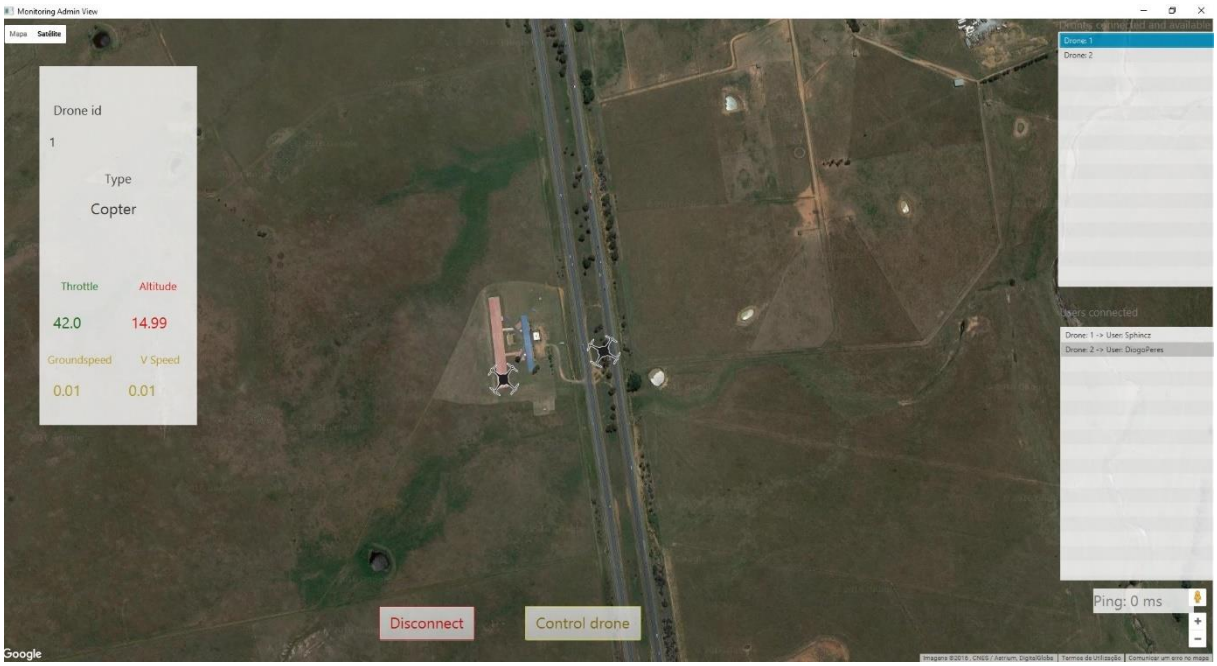


Figure 6.1 Admin monitoring.

6.2.2 User monitoring

The application provides users with the possibility of monitoring their assigned drones. When registered on the server's database, a user is assigned with one or more drones. These, when online, will be the ones the user is capable of monitoring and controlling. While monitoring, the application will provide all the information regarding the position of the drone, its ID and its information statuses such as the altitude, groundspeed, vertical speed and throttle. When the UAV symbol is pressed it shows its ID in order to distinguish between them if the user has more than one drone online in the nearby space.

The first test happens with a user connected to one drone. In this test it will be shown that the application is able to monitor one drone and receive its information.

Figure 6.2 User logged in, monitoring one drone.

In figure 6.2 it is then visible the drones positioning on the map, it is identified as drone 1. As the user doesn't have any more of its drones connected only this one is available. On the bottom half it is also indicated the ping times. It is achieved using the equation (2), Round Trip Time (RTT) which returns the result in ms.

$$RTT[ms] = t_{CTower/Server} + t_{Server/FlightController} + t_{FlightController/Server} + t_{CTower/Server} \quad (2)$$

The ping result presented in figure 6.2 is around 119 ms, and this represents the average time the messages take from one end to the other. It was calculated according to (2), sending 5

ping messages every second and then calculating the arithmetic average based on those 5 times received.

6.2.3 Evaluation

The monitoring tests showed that the application successfully manages to distinguish between two types of users, the admin and the normal user. Different screens are shown for each type of user, and they both allow for different types of information to be seen. With the normal user connected it can monitor all their assigned drones, and when the admin is connected it manages to monitor all the logged in and connected vehicles.

6.3 Control tests

In order to perform the control test two scenarios had to be evaluated, the autonomous and manual control. The application provides two ways of controlling and they are useful in different types of situations, has such it will be tested whether the drone is able or not of completing these tasks. Manual control subsection figures can be found in annex C.

6.3.1 Manual control

The application features the manual control of the drone, this means that it is possible for the user to connect a controller on to the pc and use as mean to get the UAV moving. Three different scenarios will be tested. The analog stick on the left side of the remote represents the throttle, and will make the UAV rotors rotate faster or slower. The right side analog has the purpose of changing the roll and pitch, changing the direction of movement from the drone. As such the scenarios involve showing that when using the analog sticks, the drone reacts and its movement is visible on the application.

Table 6.2, shows the different scenarios tested and the figure indication that shows each test result in the annexes.

Table 6.2 Manual control test scenarios.

Scenarios	Action	Result	Figure
Throttle up	The left analog stick is pointing forward, the application sends a command to the drone increasing the throttle value	The drone gains altitude and goes up as a result of increased throttle values. The altitude values are high, and the positive vertical speed indicate the drone is rising	Annex C Figure C. 1
Throttle down	The left analog stick is pointing backwards, towards the user, sending commands for the throttle values to get lower	The drone goes down and loses altitude when throttle values are decreased. The altitude is getting lower and the vertical speed values are negative indicating the drone is lowering its altitude.	Annex C Figure C.2
Moving and turning	The right analog stick goes to any side, sending commands changing the pitch and roll values	By changing the yaw pitch and roll the drone gains movement, being capable of going forwards, backwards and turning to any side or over itself.	Annex C Figure C.3

6.3.2 Autonomous control

In this scenario the drone is stationed on the ground and receives an order from the operator to go to a certain location. This is achieved by sending waypoints to the flight controller so that it stores them and then when in ‘Auto’ mode it takes off and heads to the correct location. The operator, using the Control Screen of the application, is able to locate the drone on the map and selects the ‘Add Waypoints’ button to start registering waypoints on the drone, and when ready, presses ‘Fly’ so that the drone takes off and heads to the destination.

In figure 6.3 the user selects how many waypoints wants to add to the flight controller, the number chosen will be the number registered and sent.

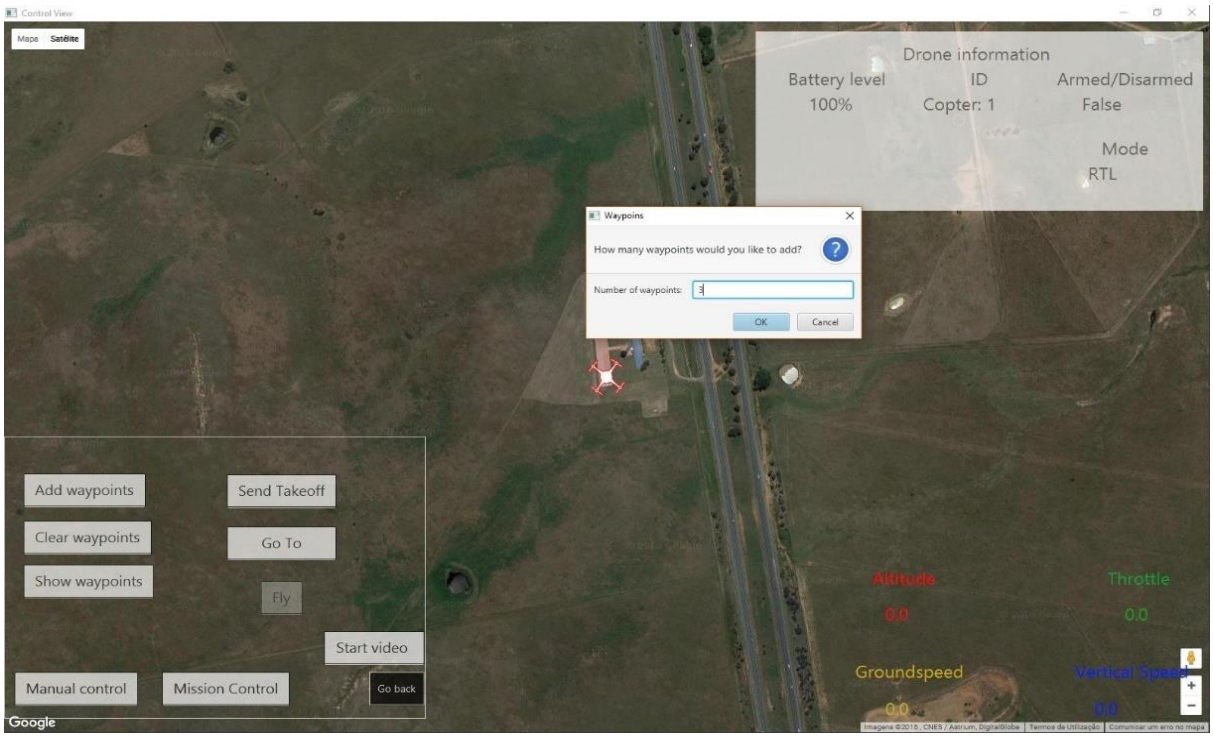


Figure 6.3 Adding waypoints.

In figure 6.4 the drone has already registered the waypoints and has received the ‘Fly’ command, which switches its mode to ‘Auto’, arms the drone and makes it take off to the first waypoint. The location is now changing and, the red trace shows the path taken by the drone and the ‘Altitude’ shows a value of 2.47 m, showing that it is ascending. The next step is

getting to each waypoint until it finally reaches the last destination.

Figure 6.4 Waypoints registered and "Fly" button already pressed.

Figure 6.5, represents the drone in the last waypoint, where it will now wait for further instructions. In the map it can be seen the path taken by the drone, reaching all three waypoints and advancing until the last one is reached. The height is now 14.99 m, as the user input was around 15 m for this last waypoint.

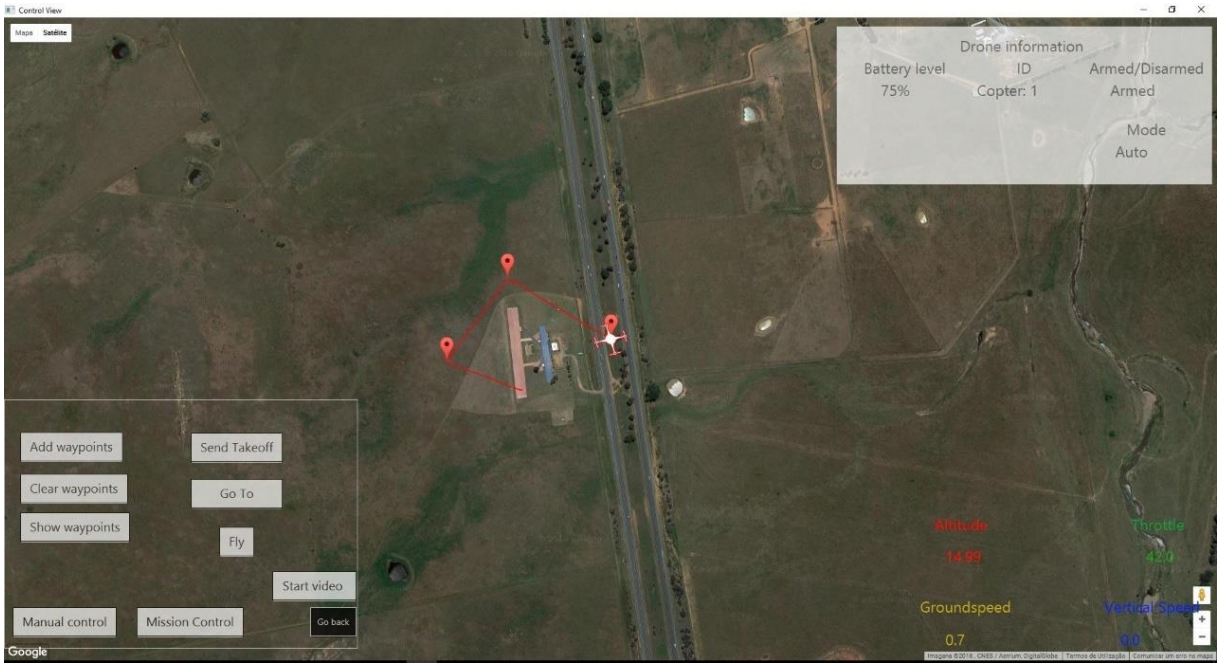


Figure 6.5 Drone reached last waypoints.

As part of the autonomous control, the flight controller features the ‘RTL’ mode, explained earlier, that autonomously leads the UAV to a pre-defined waypoint marked as ‘home’. This process is illustrated in figure 6.6 and 6.7, as the user only needs to select the RTL mode and the drone will return to the initial location.

Figure 6.6 Choosing RTL mode.

As a result of the ‘RTL’ mode, in figure 6.6, it can be seen that the UAV is already above the initial location, and the altitude is decreasing, meaning it is descending and will land on the ground, as this is visible in figure 6.7.

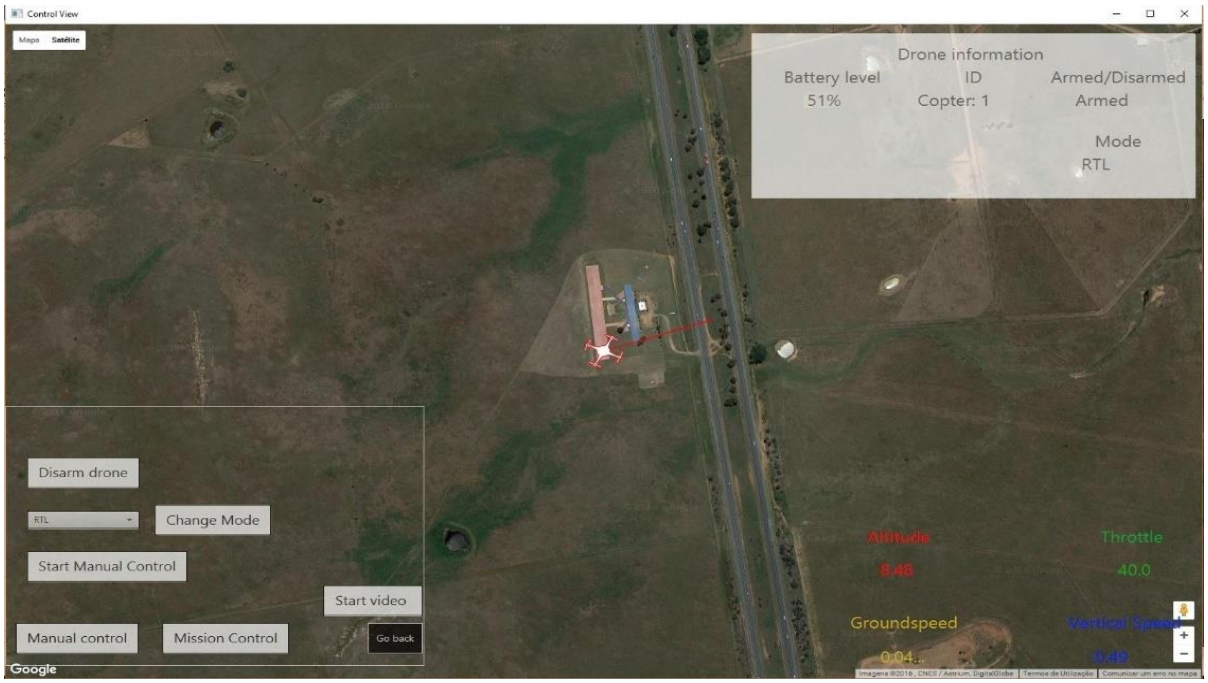


Figure 6.7 RTL mode selected and UAV returned to "home".

Besides the waypoints there is another type of autonomous control, and that is through the use of the ‘Guided’ mode. By using this mode, the user is able to send a single waypoint to the UAV and it will then go to the sent coordinates. However, for this mode to work and the drone start flying it has to already be on the air, and that can be achieved by sending a TakeOff message with a chosen altitude. The application allows for the user to use this option, through the use of the ‘Send Takeoff’ and the ‘Go To’ buttons. This is visible in figure 6.8 and 6.9.

Figure 6.8 Takeoff sent.

In figure 6.8 the ‘Takeoff’ was sent and the drone is now rising, the throttle is at 41.0 and the vertical speed is around 0.14 m per second. The UAV is at an altitude of roughly 5 meters. It is important to notice that the drone did not change its position from the original one and is only going up, the groundspeed is 0.0 m/s. It is now possible to send a ‘Go To’ and choose a location on the map to send those coordinates.

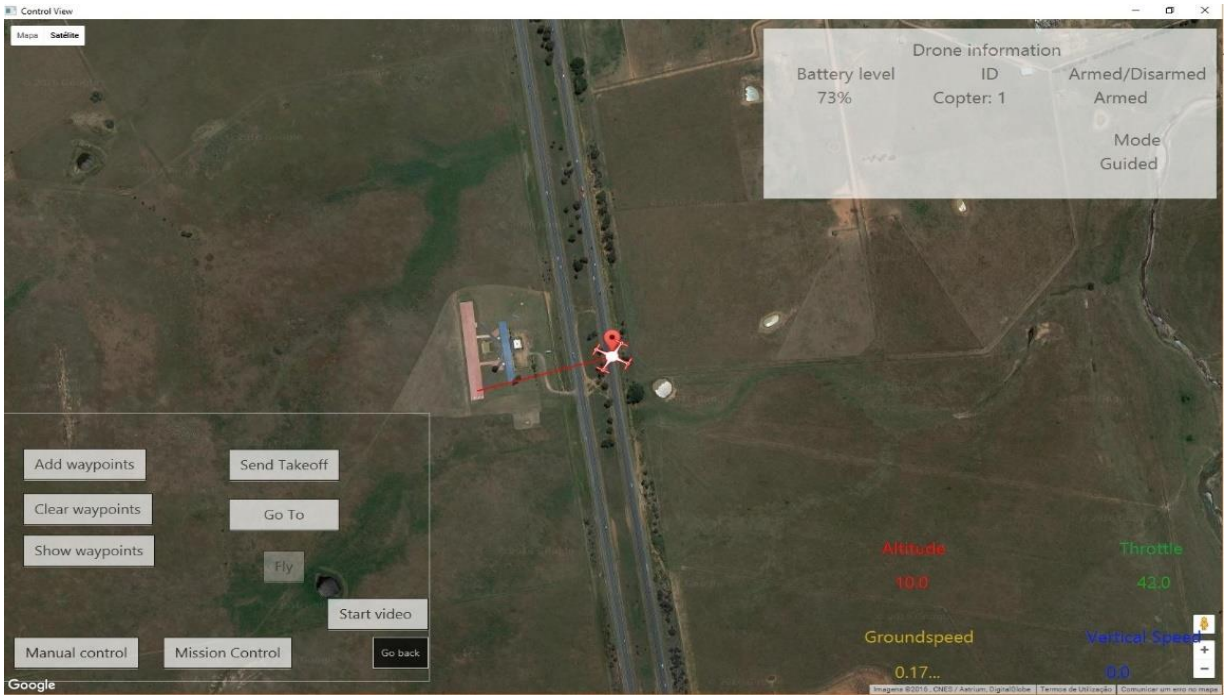


Figure 6.9 Go to sent and finished.

Figure 6.9 represents the Go To and the drone is now reaching its destination waypoint. It went from the 5 m of the takeoff to 10 m, as defined by the user.

6.3.3 Evaluation

Analyzing the results obtained for both manual and autonomous Control it is possible to retain that the applications reacts well to the commands given by the user. Manual control relies on the use of an external controller connected to the application, this through the use of both analog sticks is able to control the drones throttle and movement. Each scenario tested showed that the simulated UAV reacted accordingly to the sent commands.

During autonomous control, the user is presented with three forms of control: mission (multiple) waypoints, guided (single) waypoint and RTL. All of these are well integrated into the application and work the way it is intended. When sending multiple waypoints receives all of them, stores them in the flight controller and waits for the user's signal to start flying. It then manages to get each of them and go to the next one, and when reaching the last one, waits for the next command. The second mode, guided waypoint allows the user to send the UAV to a specific location, however, it does need the drone to be armed and off the ground. Sending a takeoff allows the user to lift the drone to a specific altitude, and after the user can now send a 'Go To'. The 'RTL' mode, successfully manages to send the drone to the home position, with the user only having to select the correct mode.

6.4 Overriding test

The overriding test has the purpose of testing whether the application is able of taking and giving control back to regular users. All the subsection figures can be found in annex D for better visualization and comprehension.

6.4.1 Taking control

This test was made using the Admin account, and figures 6.10 and 6.11 are representative of this action.

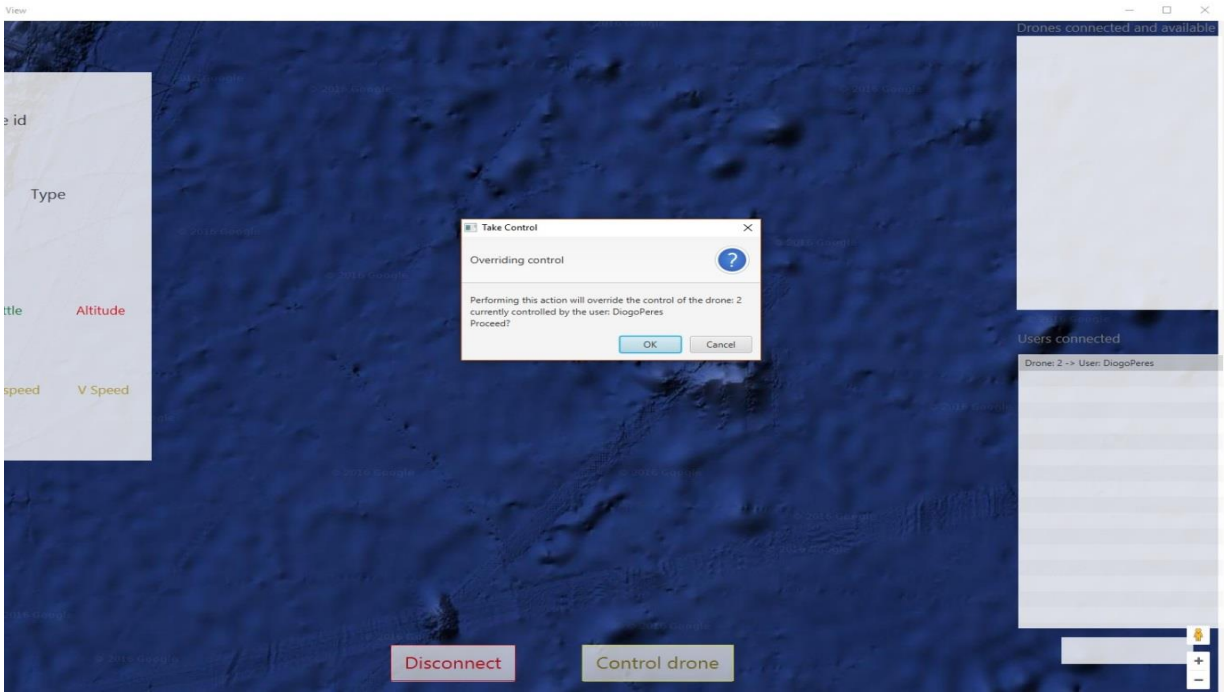


Figure 6.10 Confirming the overriding.

In both figures 6.10 and 6.11, it is shown the transaction in the taking control action. At first the image was centered on the ocean, no drones are connected to the Admin as it is supposed, then when taking control the map relocates to the location of the drone that was previously controlled by DiogoPeres. The user DiogoPeres was then removed from the database and can no longer enter the application again to control this drone until the admin decides to give the control back. The drone is now visible on the upper list, and is now available to be controlled and monitored by the admin.

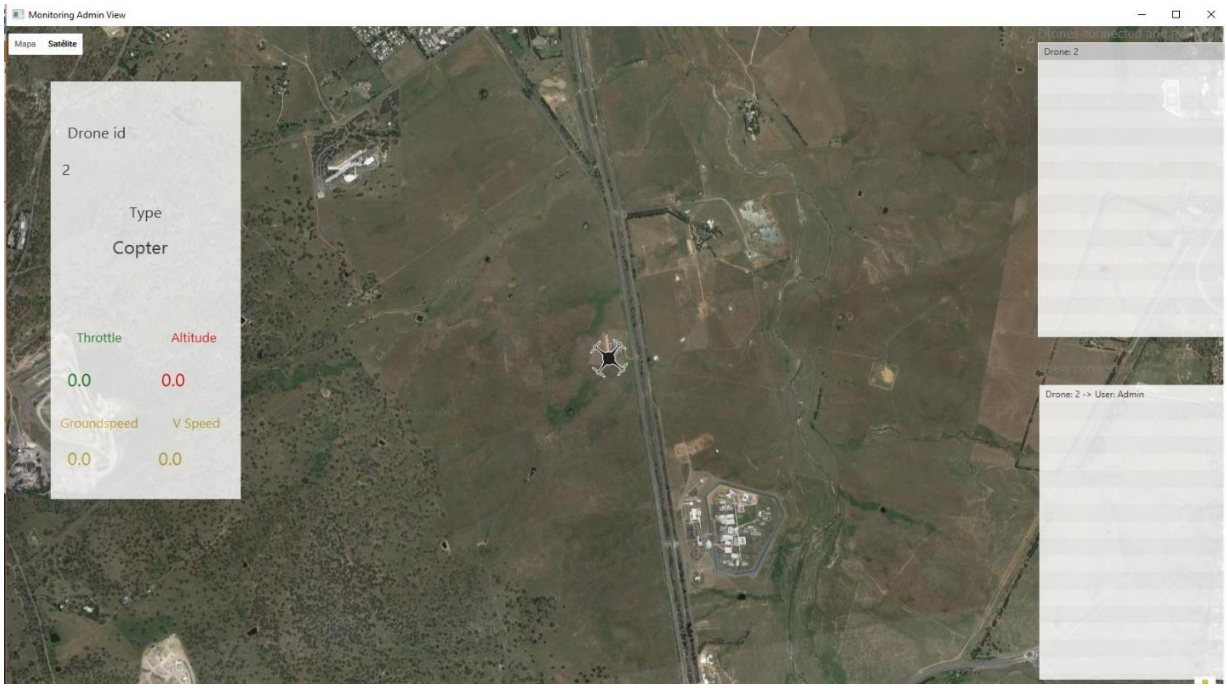


Figure 6.11 Drone overridden.

6.4.2 Giving control

In this scenario it will be tested whether the admin is capable or not of giving the control back to the user.

Figures 6.12 and 6.13 are representative of this action and shows how the giving control back is processed.

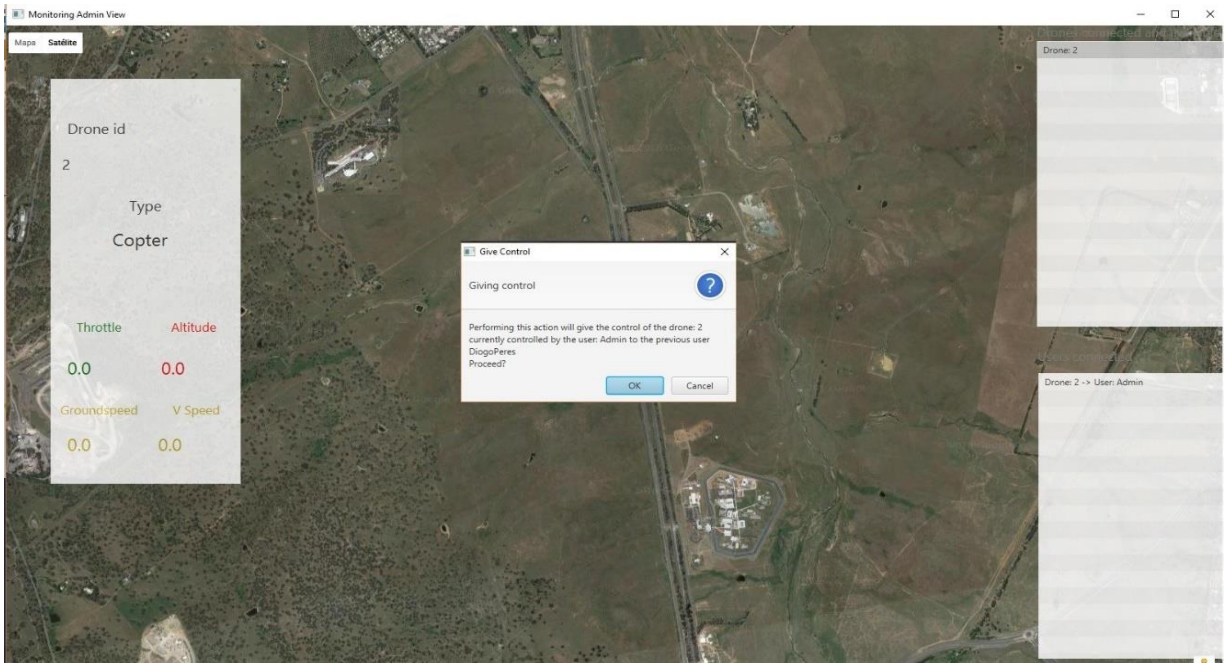


Figure 6.12 Confirming the giving of control.

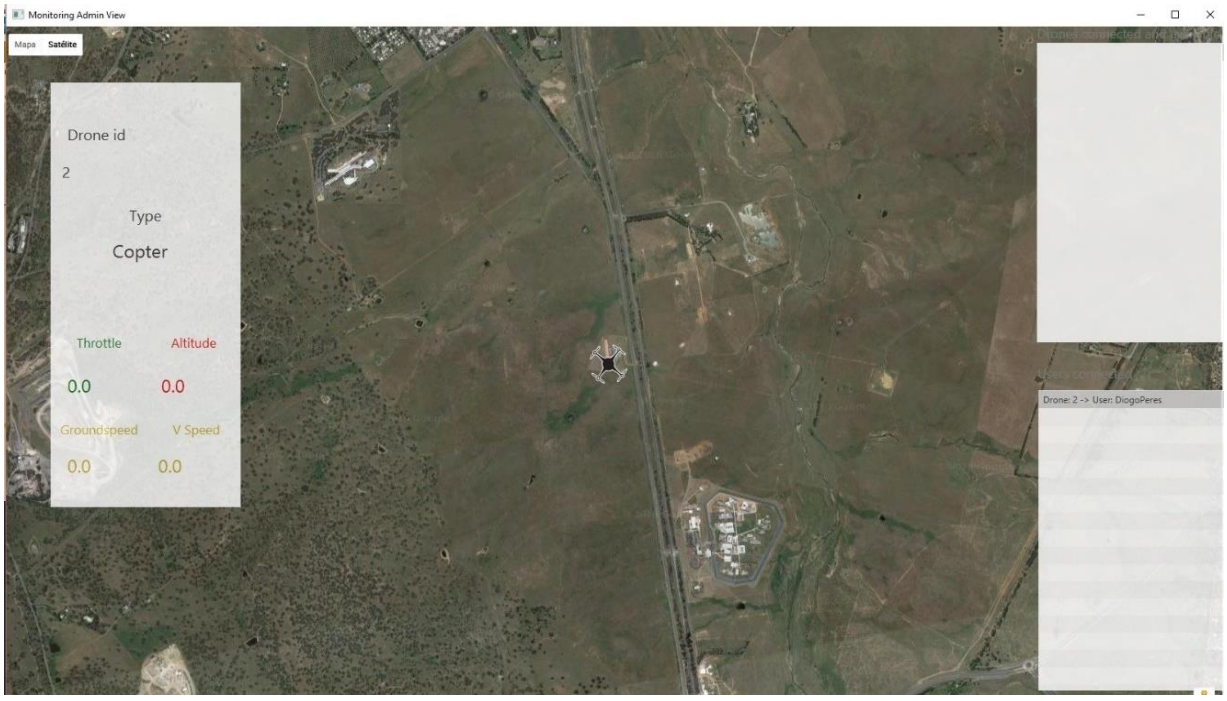


Figure 6.13 Control given back.

In figures 6.12 and 6.13 the flow of giving control action is represented. At first, a confirmation window is shown, then when the user accepts it, the database assigns the drone to the previous user and it can now log in and control this user.

6.4.3 Evaluation

Both the scenarios were successfully executed. The taking control was able to remove the user from the database and by doing this the user will be unable to connect again to the drone. With the control overridden, the admin is now in charge of that drone and it now waits for commands coming from the admin.

In order for the user to connect again to the drone, the admin has to give the control back, and that possibility was successfully executed during the tests.

6.5 Video stream testing

In order to test the live video stream three scenarios were studied. The first was through the use of Wi-Fi, the second and third were done using cellular networks 3G and 4G. As explained earlier, all the tests involving the application's interaction with the UAV were done under simulation conditions using the SITL software. As such, all the tests done on the performance of the live video-stream were made under simulation conditions, and not on a flying drone. The application is able to a video-stream through the use of the technology explained on 5.8.2, and then under each scenario a simple capture was taken. This capture is a photograph and it represents an instant in which two times can be compared, the real time, present on the phone and the application time, viewed on the computer screen. The difference in time represents the delay taken by the stream to reach from the Raspberry Pi to the application.

On table 6.2, a summary of the tests and results is shown, each of these will be explained with more detail on the following subsections. All the times on the test figures are in the format [minutes]:[seconds]:[tenths of seconds].

Table 6.3 Video stream scenarios and results

Scenarios	Resolution	Framerate [fps]	Delays [s]
Wi-Fi	320x240	~24	0.26
4G	320x240	~24	0.3
3G/ HSPA	320x240	~24	0.31

6.5.1 Wi-Fi scenario

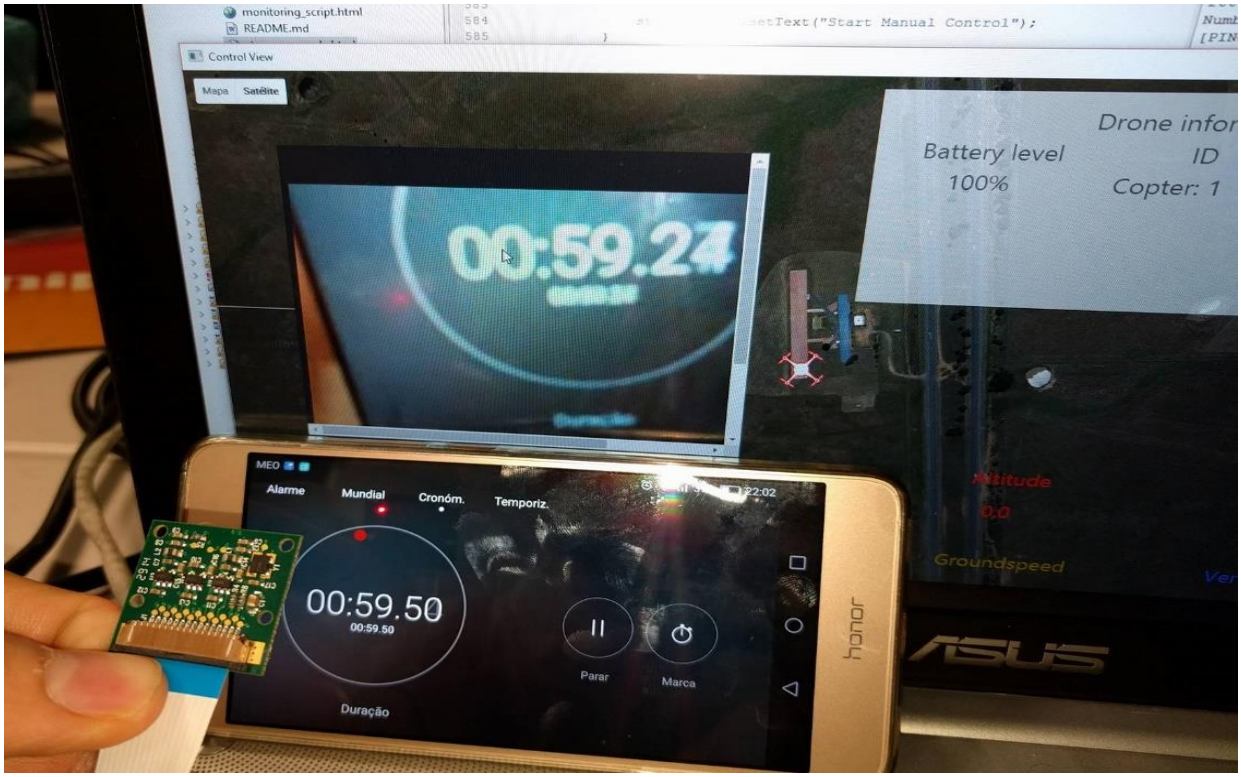


Figure 6.14 Wi-Fi Scenario result.

In figure 6.14 the Wi-Fi scenario capture is represented. The time on the phone is seen to be 00:59:50 seconds, and the time received on the application is around 00:59:24 seconds. This means that there is a latency of about 0.26 s, for a capture averaging around 24 fps and with a resolution of 320x240. This is a good result, anything under 0.3 s would allow for live video control, however latency can have some variance, and depending on the receiving conditions it can increase or decrease the tested time.

6.5.2 4G

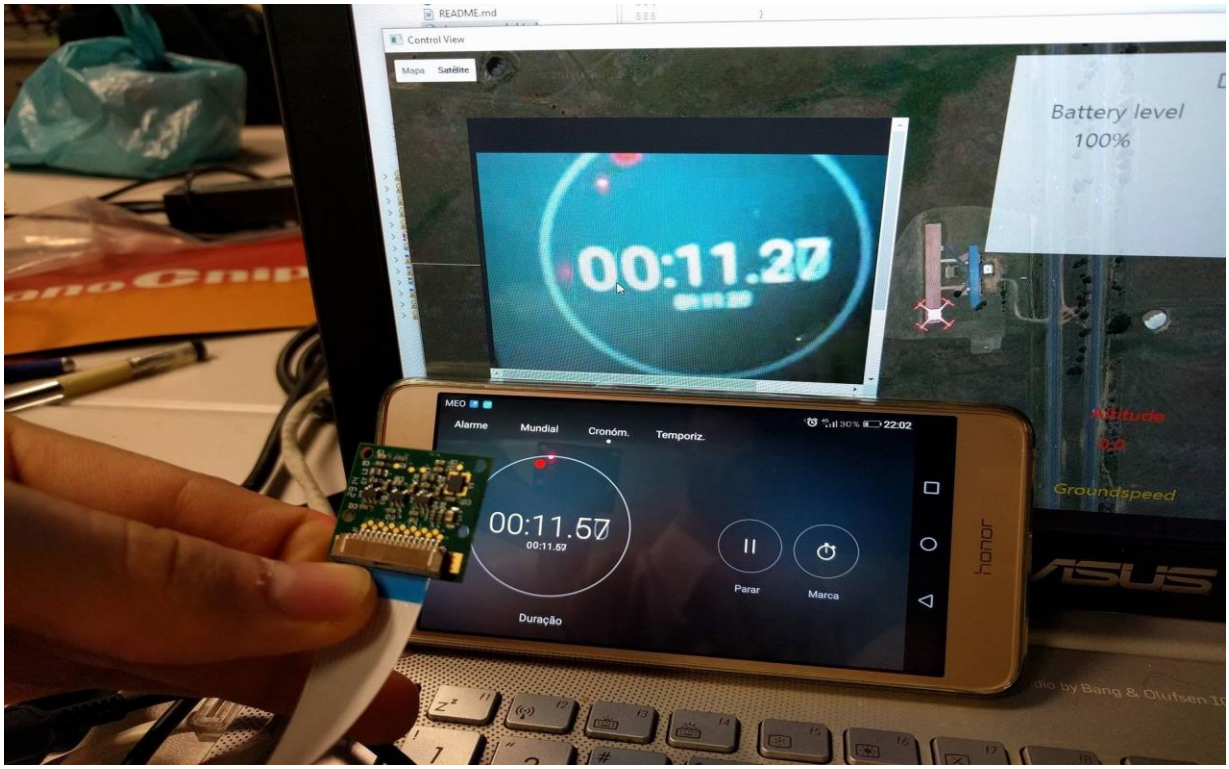


Figure 6.15 4G scenario result.

In the 4G scenario, delay times have risen to 0.3 s as visible in figure 6.15. The time on the phone is around 00:11:57 seconds and the time received on the application is 00:11:27 seconds, this is an expected result when comparing to Wi-Fi, as the download/upload speeds aren't as high in 4G. However, it is indeed a good delay time, as the video quality and framerate stayed constant from Wi-Fi to 4G, averaging 24 fps and the same resolution of 320x240.

6.5.3 3G

In this scenario it is tested the latency of the video using cellular networks, in this particular case, HSPA technology was used for the connection to the internet.

Figure 6.16 3G scenario result.

In figure 6.16 it is visualized the latency of the video when using cellular networks. Just like when using Wi-Fi the, the video is encoded with a size of 320x240 and the framerate is around 24 fps, which gives a good and fluid quality of the transmission. The delay is around 0.31 s, as the time on the phone is 3:51:65 and on the application is perceived to be around 3:51:34. The difference from Wi-Fi is around 0.05 s which doesn't make for a big difference, however when flying it may be noticeable by the operator.

6.5.4 Evaluation

All three scenarios showed promising results as the times obtained on the testing's was around 0.3 s. As referred earlier, it is intended that delay be as low as possible so that the user is capable of controlling the drone live and for the purpose of monitoring outside the delays values will be highly dependent on the connection to the network and on its type. During the tests and since they were done in an area where it is possible to get coverage of 3G and 4G networks both results were positive showing that under any of these the live delay times were mostly acceptable. For the results to get better and to try and improve the results, the quality of the transmission would have to be changed, like changing to a smaller resolution and reducing the framerate on the RPi.

This page was intentionally left in blank

Chapter 7
CONCLUSIONS AND FUTURE WORK

The chapter 7 presents the conclusions and some topics for future work.

7.1 Conclusions

This thesis primary goal was to develop a software platform that was capable of the monitoring and controlling of more than one UAV, through the use of wireless networks such as Wi-Fi or cellular networks.

Through the development of this thesis, the main topics studied and developed revolved around on how to build a Java application, named Control Tower, that was capable of transmitting and receiving messages over the internet to keep track of one or more drones. In order for this application to be built, several lines of work had to be crossed. The first was understanding how a drone works, how flight controllers integrate with drones and how can they be integrated and connected to a device that is capable of providing a connection to the internet to transmit the packets. This device happened to be the Raspberry Pi, a very small sized computer that fit the job of transmitting the messages, and connecting to the Pixhawk through its I/O pins minimizing the latency on the messages delivery.

The next line of work involved thinking on how could the application transmit to an unknown IP over the network, and on how could the application distinguish between the incoming messages of more than one drone when the messages don't have any identifier capable of being used. The solution to this problem was the use of a public server, avoiding thus the NAT problem that doesn't enable for direct communication between the application on different networks. As the application has to monitor not only the drones, but the users connected to them, the server also includes a database, that keeps the records of each registered user and their associated drones. The database makes a distinction between two types of users, the Admin and the regular users. The Admin has privileges, and is capable of overriding the control of one user so that it can be the Admin the one to control it. The Admin also has the capability of giving the control back, if the situation demands so. The server now knows all the registered drones, their identifier, and the users they're assigned to, this means that the server is now capable, through the use of the Mavlink protocol, of redirecting each incoming message to the Control Tower application.

After being capable of receiving Mavlink messages containing the information about each connected drone, the application is able to monitor and position each drone on the map, so that the user can identify their location. Besides monitoring, the application is also capable of controlling one drone at a time, either through manual or autonomous control. For the

application to be complete, it also features the live streaming of video, through the RPi Cam and the RPi.

The simulated tests result, showed that the Control Tower works well on receiving, monitoring and sending control commands, it performs well and the messages are received on the end half making the drone execute the right actions. The overriding accomplishes its goal, removing the user's control and their access to the database, this way making them enable to log in again until the vehicle's control is returned.

Regarding the video stream times, it is well shown that the results deteriorate from the use of Wi-Fi, 3G and 4G. Even though the delay times increase, they still are good times for the technology used and the transmissions framerate, this shows results could be further improved by testing on different encoding values.

7.2 Future work

For future work some improvements can be made, e.g.:

- Testing the application on the real-world, using the right vehicles on the right weather and safety conditions;
- Improving the applications UI, add more features, make the application more responsive;
- Improving the overriding system, not only can the Admin override but even without removing the other users control the drone should still accept commands from the admin as a priority;
- Reducing the average video stream times and ping times;
- Encoding transmission messages for increased security;
- P2P between one user and its drone without the need of a centralized server.

This page was intentionally left in blank

REFERENCES

Sparkfun. [Online] [Cited: 09 October 2016.] Available from: https://www.sparkfun.com/products/retired/11868.

Adafruit Sensor Overview. *Adafruit.* [Online] 04 August 2014. [Cited: 19 September 2016.] Available from: https://learn.adafruit.com/adafruit-lsm9ds0-accelerometer-gyro-magnetometer-9-dof-breakouts/overview.

Adding HTML content to JavaFX Applications: 3 Supported Features of HTML 5. *Oracle Java Documentation.* [Online] [Cited: 20 September 2016.] Available from: http://docs.oracle.com/javase/8/javafx/embedded-browser-tutorial/html-five.htm.

Alex and Sam. 2015. How to choose the right motor for your multicopter drone. *Drone Trest.* [Online] October 2015. [Cited: 19 September 2016.] Available from: http://www.dronetrest.com/t/how-to-choose-the-right-motor-for-your-multicopter-drone/568.

Ardupilot Mega [Online] [Cited: 29 September 2016]. Available from: http://www.ardupilot.co.uk/.

Austin, Reg. 2010. *Unmanned Aircraft Systems, UAVs Design, Development and Deployment.* s.l. : Wiley, 2010.

Blesch, Carl. 2015. Navy Funds Rutgers to Develop Drone Equally Adept at Flying and Swimming. *Rutgers The State University of New Jersey.* [Online] 23 October 2015. [Cited: 22 September 2016.] Available from: http://news.rutgers.edu/research-news/navy-funds-rutgers-develop-drone-equally-adept-flying-and-swimming/20151022#.V-RNAvArKUl.

Chapman, Andrew. Types of Drones: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL. *Australian UAV.* [Online] [Cited: 27 August 2016.] Available from: http://www.auav.com.au/articles/drone-types/.

Communicating with Raspberry Pi via MAVLink. *Ardupilot.* [Online] [Cited: 08 October 2016.] Available from: http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html.

Davison, Andrew. 2006. *Chapter 28.9. Building a Game Pad Controller with JInput.* 2006.

DIY Drones. *DIY Drones.* [Online] 17 December 2009. [Cited: 06 October 2016.] Available from: http://diydrones.com/photo/hornet-gcs-and-hornet-micro?xg_source=activity.

D'Onfro, Jillian. 2014. Why Amazon Needs Drones More Than People Realize. *Business Insider.* [Online] 30 July 2014. [Cited: 22 September 2016.] Available from: http://www.businessinsider.com/amazon-drones-2014-7.

Fedortsova, Irina. 2012. Concurrency in JavaFX. *Oracle.* [Online] June 2012. [Cited: 20 September 2016.] Available from: http://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm.

Why use FXML: Oracle. *Oracle.* [Online] January 2014. [Cited: 20 September 2016.] Available from: http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDCHIBE.

FFMPEG. *FFMPEG.* [Online] [Cited: 19 September 2016.] http://ffmpeg.org/.

Global Research News. 2014. Unmanned Aerial Vehicles (UAV): Drones for Military and Civilian Use. *Global Research.* [Online] 21 March 2014. [Cited: 22 September 2016.] Available from: http://www.globalresearch.ca/unmanned-aerial-vehicles-uav-drones-for-military-and-civilian-use/5374666.

Harris, Phil. 2014. Aerial Insight, Multirotor Configurations. *Aerial Insight.* [Online] 2014. [Cited: 05 September 2016.] Available from: http://static1.squarespace.com/static/52dabcbfe4b00bd4279a5cb3/t/545a1f0ce4b0dc3018ad0a10/1415192332960/Multirotor+Configurations.pdf.

Industry, Defense. 2011. It's Better to Share: Breaking Down UAV GCS Barriers. *Defense Industry Daily.* [Online] 03 October 2011. [Cited: 06 October 2016.] Available from: http://www.defenseindustrydaily.com/uav-ground-control-solutions-06175/.

JavaFX Tutorials: What is JavaFX. *JavaFX Tutorials.* [Online] [Cited: 20 September 2016.] http://www.javafxtutorials.com/whatisjavafx/.

Mavproxy. *QGroundControl.* [Online] [Cited: 18 September 2016.] http://qgroundcontrol.org/mavlink/mavproxy_startpage.

McHale, John. 2010. Ground control stations for unmanned aerial vehicles (UAVs). *Military Aerospace.* [Online] 18 June 2010. [Cited: 28 August 2016.] Available from: http://www.militaryaerospace.com/articles/2010/06/ground-control-stations.html.

Pawlan, Monica. 2013. What is JavaFX: Oracle. *Oracle.* [Online] April 2013. [Cited: 20 September 2016.] Available from: http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm.

Pixhawk Autopilot. *PX4 Autopilot.* [Online] [Cited: 03 September 2016.] Available from: https://pixhawk.org/modules/pixhawk.

QGroundControl MAVLink. *QGroundControl.* [Online] [Cited: 09 October 2016.] Available from: http://qgroundcontrol.org/mavlink/start.

QGroundControl**.** QGroundControl: Waypoint protocol. *QGroudControl.* [Online] [Cited: 16 October 2016.] Available from: http://qgroundcontrol.org/mavlink/waypoint_protocol.

Raspberry Pi Camera Module. *Raspberry Pi.* [Online] [Cited: 19 September 2016.] Available from: https://www.raspberrypi.org/documentation/raspbian/applications/camera.md.

RaspberryPi: Raspberry Pi 3 Model B. *RaspberryPi.* [Online] 2016. [Cited: 29 September 2016.] Available from: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

Redko, Alla. 2014. Adding HTML content to JavaFX: Oracle. *Oracle.* [Online] January 2014. [Cited: 20 September 2016.] Available from: http://docs.oracle.com/javafx/2/webview/jfxpub-webview.htm.

The Utilization of Unmanned Aerial Vehicles (UAV) for Military Action in Foreign Airspace. Ronconi, Giordano B. Antoniazzi, Batista, Thais Jessinski and Merola, Victor. 2014. 2014, UFRGSMUN, pp. 138-139.

Unmanned Aerial Vehicles An Overview . Bento, Maria de Fátima. 2008. 2008. InsideGNSS. pp. 54-56.

Welcome to Raspbian. *Raspbian.* [Online] [Cited: 19 September 2016.] Available from: https://www.raspbian.org/.

This page was intentionally left in blank

ANNEXES

Annex A – Scene builder and UI screens

Figure A. 1 Scene Builder

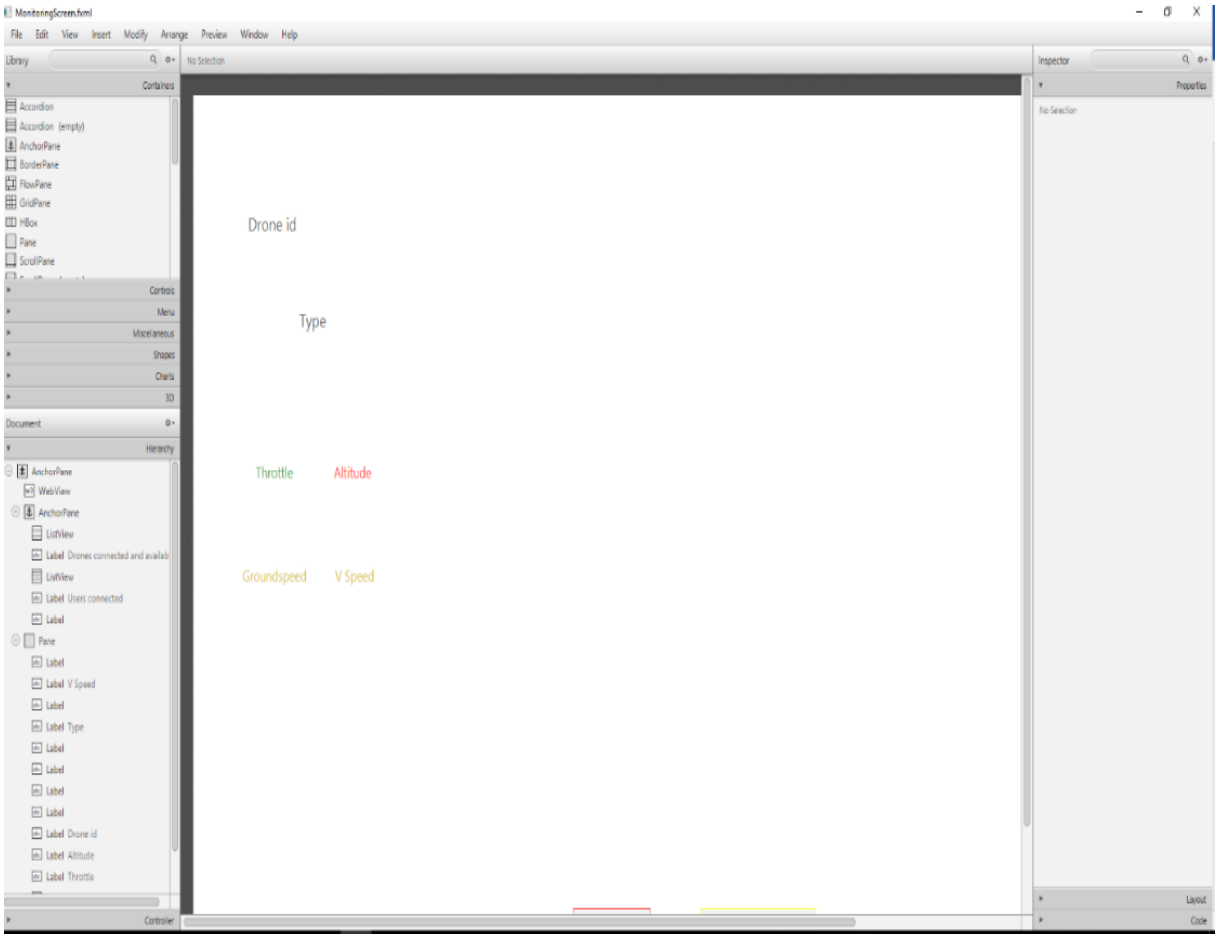


Figure A. 3 Monitoring View

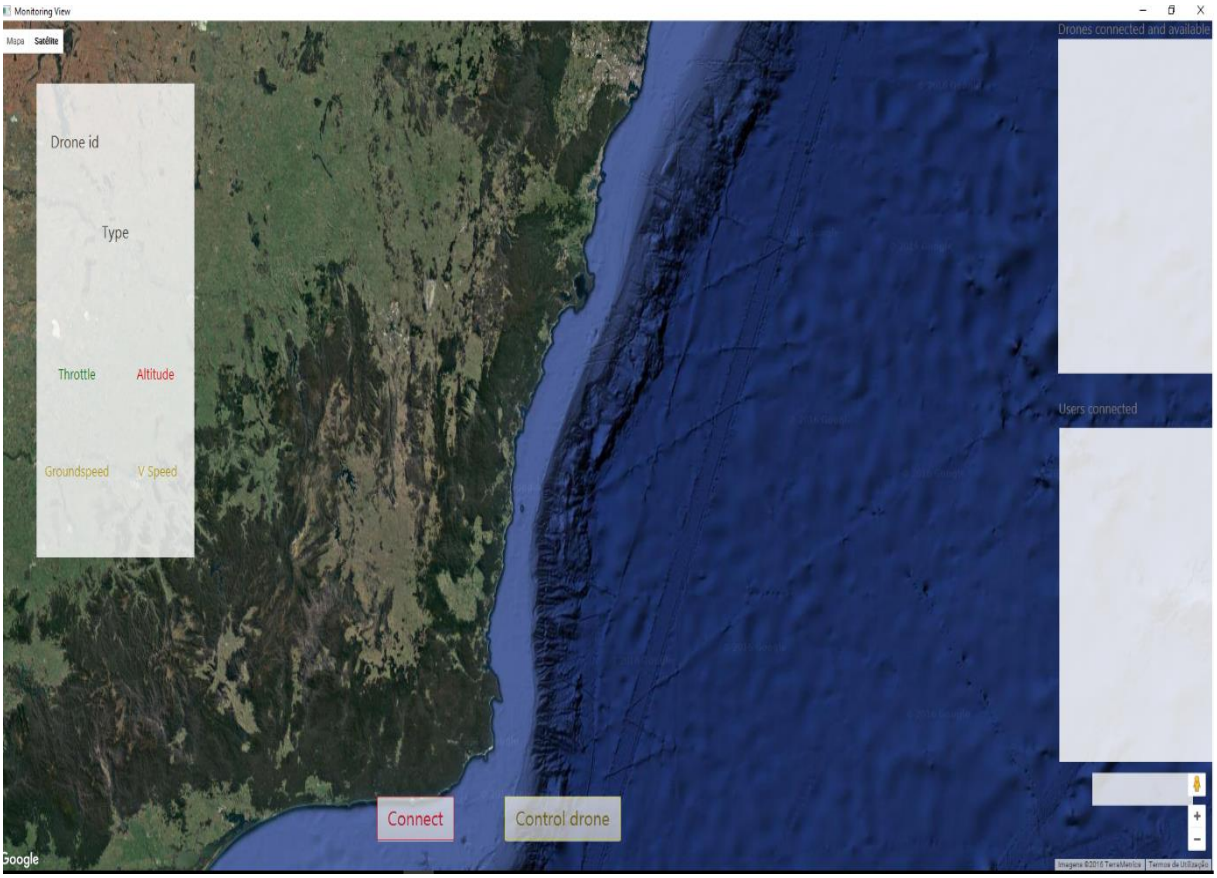


Figure A. 2 Controlling View

Annex B – Monitoring tests

Figure B. 1 Admin monitoring [two drones]

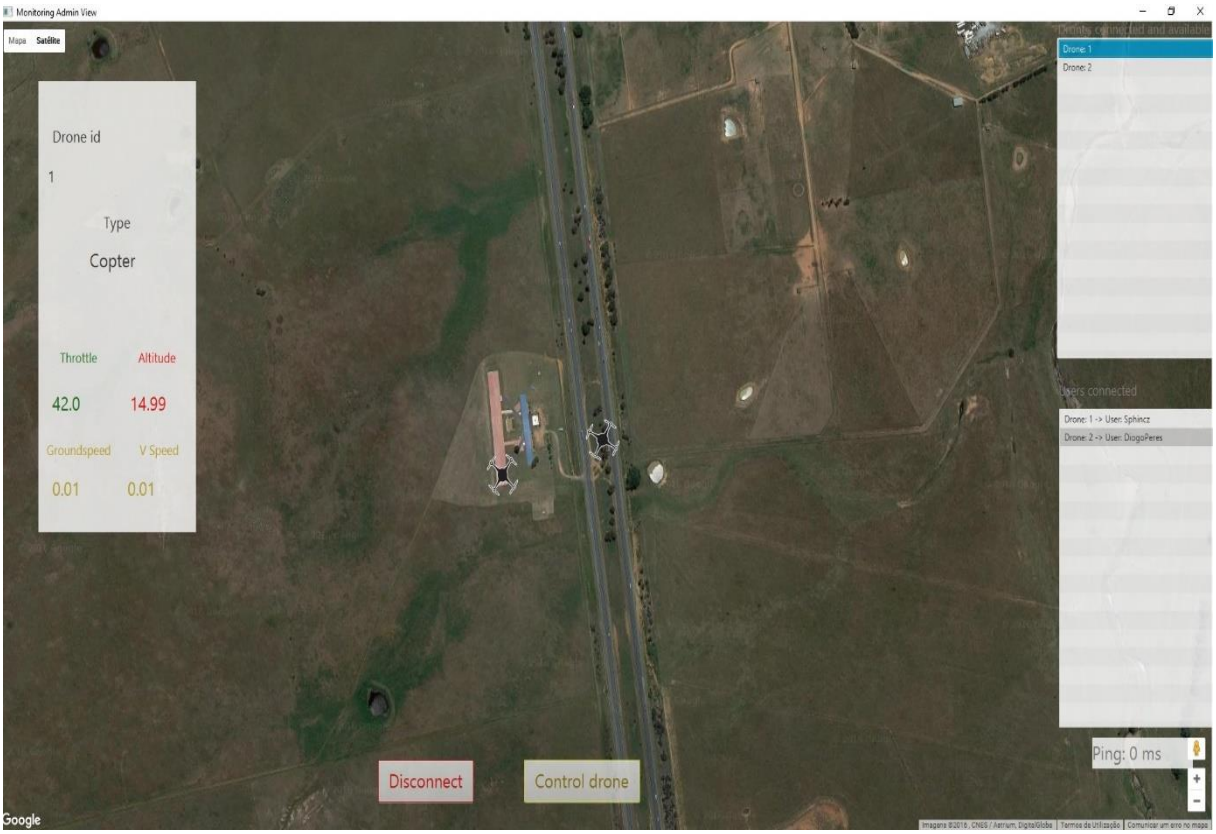
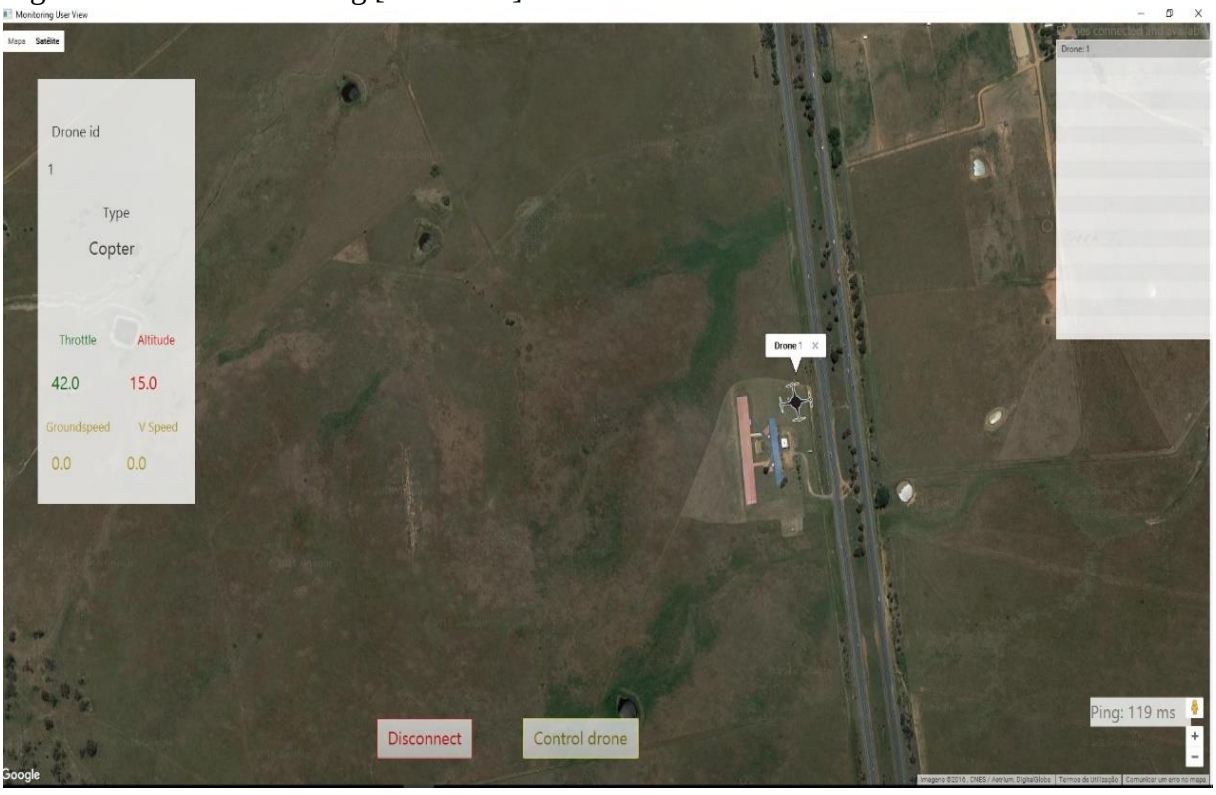


Figure B. 2 User monitoring [one drone]



Annex C – Manual control tests

Figure C. 1 Throttle up

Figure C. 2 Throttle down

Figure C. 3 Drone moving and turning

Annex D – Overriding test

Figure D. 1 Confirming the overriding

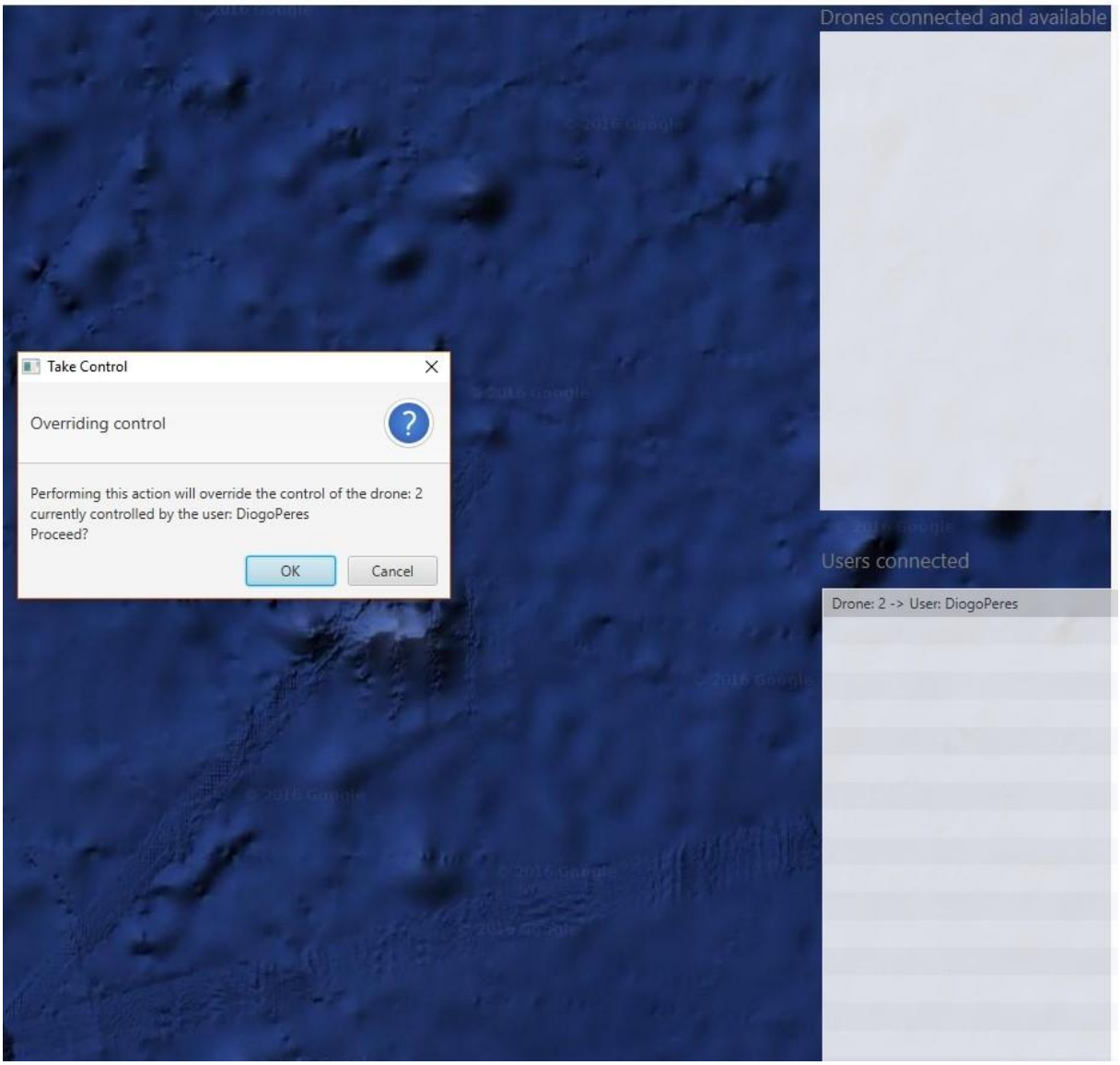


Figure D. 2 Drone overriden

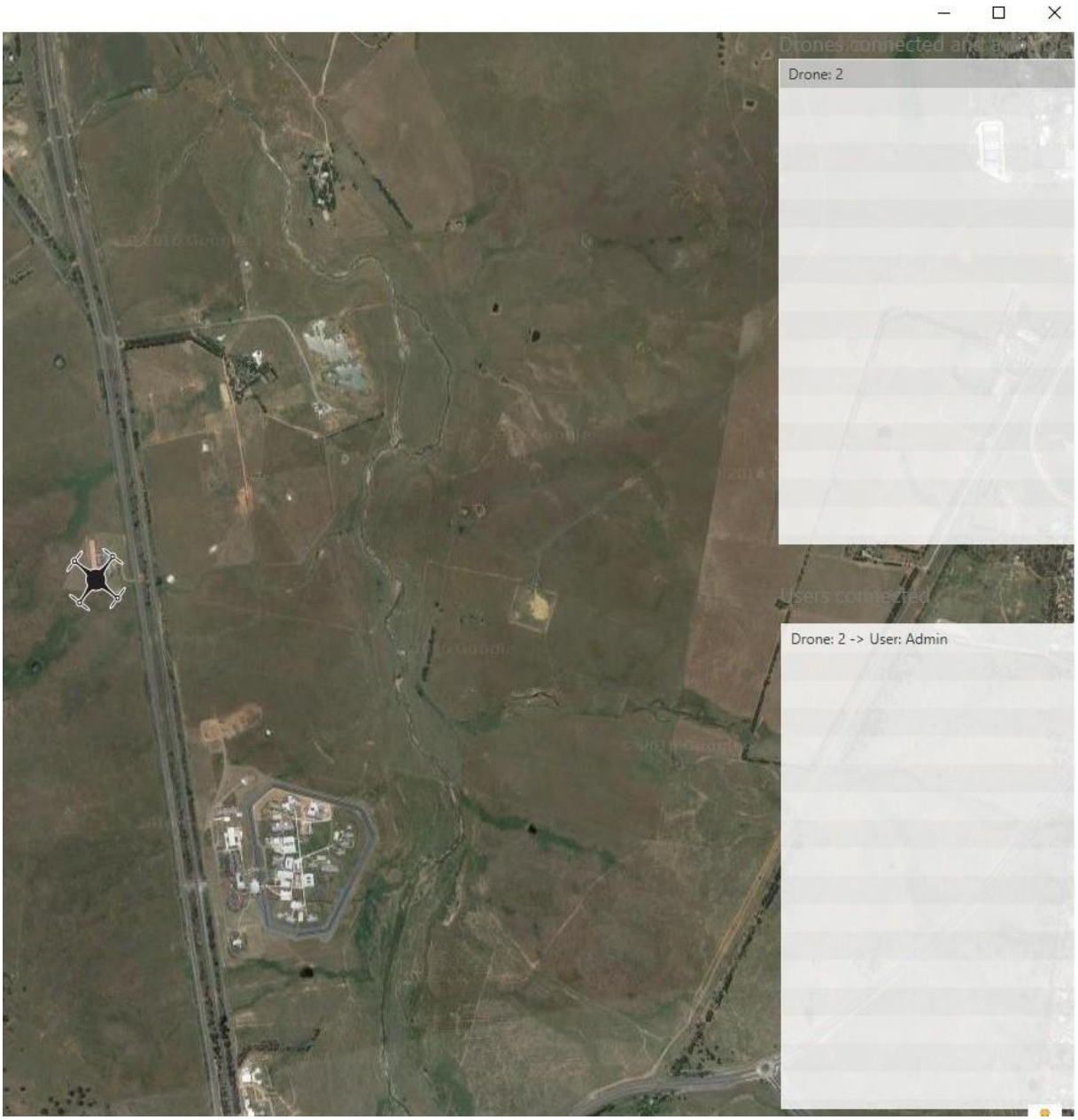


Figure D. 3 Giving control information window

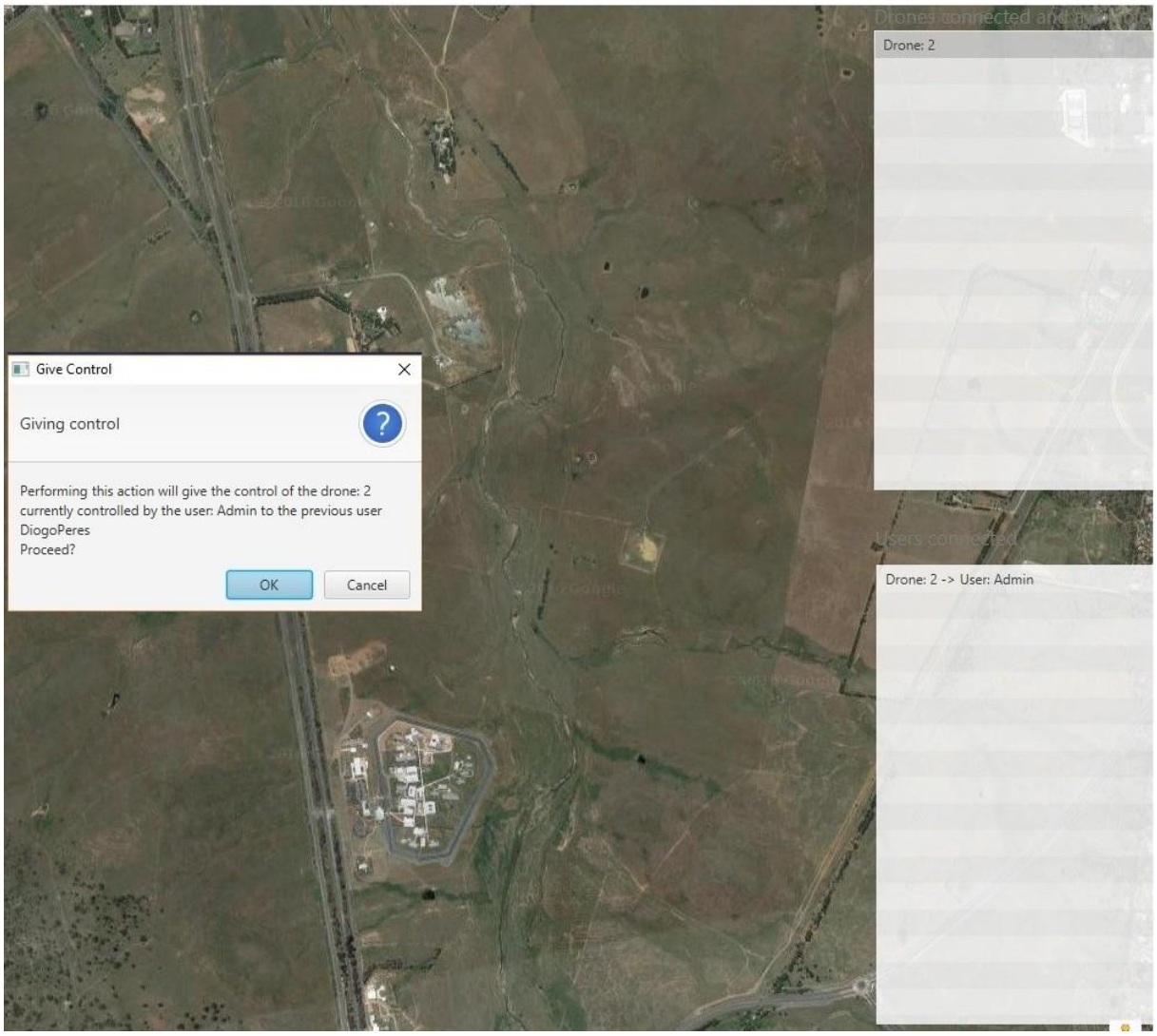


Figure D. 4 Control given back