University of Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Engenharia Informática

Regatta Monitoring with Drones

Author:
David Fernandes, 81882
Dimitri Silva, 80013
Fábio Barros, 80430
Manuel Roxo, 80308

CU Professor:
Prof. José Moreira
Supervisors:
Prof. André Zuquete
Prof. João Paulo Barraca
Prof. António Neves

July 10, 2018

Regatta monitoring with Drones

Project Report on Informatics Engineering at the University of Aveiro, conducted by Professor José Manuel Matos Moreira with the supervision of the Professors André Zúquete, João Paulo Barraca and António Neves at the Department of Electronics, Telecommunications and Information Technology.

Keyword

- Drone
- Regatta
- Autonomous flight
- Image detection
- MSP communication
- Video Transmission

Acknowledgements

First of all we would like to thank DETI, IEETA and IT for the funds to build a drone from scratch.

To our supervisors we would like to thank Professors André Zúquete, João Paulo Barraca and António Neves for the construction of the drone and their inputs in system architecture and video processing.

We would also like to thank Mário Antunes from IT for the help in 3d printing specific parts required for the project.

To professor José Moreira we would like to thank for overseeing and supporting this project over the course of the semester.

Abstract

A regatta is a nautical race between several boats with an important set of rules which exist to maintain it's integrity. However, due to the environment and adversal viewing conditions the referees have a hard time enforcing these rules, resulting in imprecise decisions and sometimes even having to restart the race.

This project focus on trying to solve these issues by building a drone capable of withstanding the environment of these races and creating a simple system capable of extending the referees reach and providing him with every piece of information needed to enforce the rules, while being intuitive and trustworthy.

The final system provided the referee with two camera modes, front or bottom of the drone. Video from the bottom camera was changed to include an identifier of each boat. A full map of every component and the ability to move the drone to a certain position was also implemented. Adjustments with image processing for better positioning was done with the objective of having the best point of view over a buoy.

Contents

List of Figures

Abbreviations

RPI - Raspberry PI;

GS - Ground Station;

GPS - Global Positioning System;

CPU - Central Processing Unit;

UDP - User Datagram Protocol;

TCP - Transmission Control Protocol;

OpenCV - Open Source Computer Vision Library;

ARM - Advanced RISC Machine;

AV - Analog Video;

USB - Universal Serial Bus;

FPV - First Person View;

PAL - Phase Alternating Line;

GPU - Graphics Processing Unit;

VHCI - Video Host Controller Interface;

MMAL - Multimedia Abstraction Layer;

TVL - Television Lines;

CSI - Camera Serial Interface;

DMA - Direct Memory Access;

SoC - System on a Chip;

UBEC - Universal Battery Elimination Circuit;

ESC - Electronic Speed Controller;

FC - Flight Controller;

UAV - Unmanned aerial vehicle;

Mbps - Megabits per second

Chapter 1

1 Introduction

A regatta is a nautical race between several boats - sailing, motor or oars - making a course marked by beacons (buoys). The starting line is delimited by a buoy and a boat with a signaling pole, from where the referee verifies if the line is being crossed. The boats only have to be behind the line at the moment that the race starts, being allowed to across it in the moments before, which difficults the referee's job. Additionally, it is also difficult for the referee to identify which boat is misplaced, which causes the race to be restarted when it is unknown. These boats have to cross the buoys and turn around them, usually at the same time, without touching each other or the buoy. However, the nature and context of regattas (on an aquatic environment on boats, usually with strong winds) difficults the referee's job of enforcing the rules, which compromises the integrity of the race. With this in mind this project hopes to provide the referees with better conditions in enforcing the rules in place and validating the calls made. Given the environment where they happen, the best approach is by using a drone, utilizing it's flight capability to provide a good view and tracking of each boat and buoy.

1.1 Context

This project is developed under the subject of Informatics Project, supervised by José Moreira and oriented by André Zúquete, João Paulo Barraca and António Neves. It focuses on the integration of a drone in a regata to aid the referees. It was developed in DETI/IT over the course of the second semester of the academic year of 2017/2018. It is part of the bachelor in Informatic Engineering.

1.2 Motivation

Drones are becoming a bigger part of society with its many possible uses and capabilities. As such we wanted to get in on this and have a shot at understanding what is behind the construction of one, the requirements of such and developing a system of communication capable of fulfilling what the project required. The chance to build one from scratch was something we felt would be very rewarding and we found that using a drone in this context was an elegant and innovative solution to the problem given.

Regattas are an important sport, but with space for improvement, especially in rule enforcment, as stated in the introduction. Given the context, using a drone

presents an innovative solution which can account for and correct every aspect of the problem.

1.3 Goals

The main goal is to build a drone capable of withstanding strong winds and a whole system capable of providing an aid for the referee at a regatta. This system includes a well designed interface of easy use for the referees, real time video transmission, GPS tracking by modules also developed by us and communication between each component. The drone and the system must be connected and as such, obviously, the system must control the drone and his actions must be appropriate to the context of its use.

1.4 Document Structure

This report will follow this structure:
Chapter 2 will be about the project hardware with special attention to the specification of said hardware.
Chapter 3 will the be state of art where we talk about what projects have been done in the past years related to this one and how they are a good point of reference. We will also specify what technologies are used in the development of this project.
Chapter 4 will be the specification of the system and how it is built and to what purpose. The architecture and technological model will be shown and explained.
Chapter 5 will be our implementation to solve this problem. We will talk about what was done to solve this and why. All technological developments and their implementation will be explained and shown.
Chapter 6 will talk about what was done and what should be done in a future implementation. We take a critical approach to the results we gathered and expose what failed what was accomplished.

Chapter 2

2 Project Hardware

For this project to be successful there were many requirements of hardware. This chapter we will talk about the hardware bought and how it was used.

2.1 Hardware Architecture

The present architecture describes all the modules integrated in the project, including its components.
They consist of the drone module, the external processing unit with its external hardware (Camera and WiFi pen), the GPS tracking modules and the web platform. To note that these modules communicate with each other mainly by the access point defined on the external processing unit.

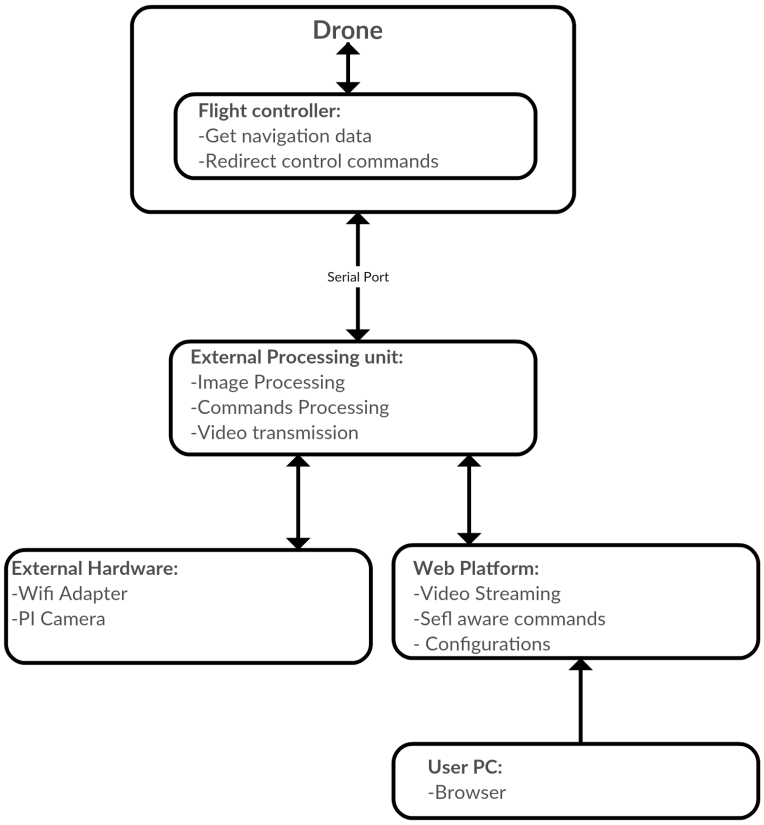


Figure 1: Hardware diagram

2.2 Hexacopter

The hexacopter was built with the following specifications:

- Propellers: APC 10x5 Thin Electric
- Motors: EMP N2836/11 750 KV Brushless
- ESCs: ZTW 30 A Pro
- UBEC: Hobbywing UBEC 10/20A
- Battery: SLS XTRON 5000mAh 4S1P 14,8V 20C/40C
- Flight Controller: OpenPilot CC3D EVO

- GPS and Compass module: Ublox Neo-M8N
- Frame: DJI F550
- Landing Gear: DJI F550 Landing Gear kit
- Radio Transmitter: Flysky FS-I6
- AV Transmitter: Eachine TX265
- AV Camera: Eachine 1000 TVL CCD
- AV Receiver: Eachine ROTG01 UVC OTG 5.8G 150CH

Since the drone was built from scratch there was some liberty in choosing the specific parts. The calculations and the reasoning for some of these parts can be found in the implementation section of this report.

Figure 2: Drone

2.3 External Processing Unit

As mentioned before, to have the drone behave like we intend him to we are attaching to the flight controller a Raspberry PI responsible for receiving and handling all the communication and processing it, giving the commands to the flight controller. It is also responsible for one of the cameras of the drone and the video transmission. The model used is a Raspberry PI 3B running Raspbian Stretch with the following specifications:

- SoC: Broadcom BCM2837
- CPU: 4x ARM Cortex-A53, 1.2GHz
- GPU: Broadcom VideoCore IV
- RAM: 1GB LPDDR2 (900 MHz)
- Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
- Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
- Storage: microSD
- Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ether- net, Camera Serial Interface (CSI), Display Serial Interface (DSI)

For this module it was also printed a 3d case capable of sticking to the drone, making it more secure and better looking than duct tape. This module is powered by the battery of the drone. To achieve the correct voltage for the RPI we used a Universal Battery eliminator circuit (UBEC).

Figure 3: Raspberry Pi

2.4 Wifi Adapter

Model: Tp-link 300Mbps - TL-WN821N USB
This network adapter is compatible with the IEEE 802.11n standard and can work as an access point. We picked 2.4 GHz Wifi for our communications with the Raspberry Pi due to its great compatibility with other devices and all the protocols implemented by it. The other alternatives available were Bluetooth, 5 Ghz Wifi and Cellular but all of these had drawbacks in range, cost and bandwidth. Theoretically the 802.11n standard supports up to a 288.8 Mbps data rate and a range of up to 250 meters. The adapter is configured in access point mode creating a network for all the needed devices to connect to. This access point is configured with 802.11n standard and uses WPA2 for security.

2.5 Raspberry Mounting System

To mount the Raspberry Pi on the drone we initially decided to make aluminium case and attempt to waterproof it. This would also provide passive cooling for the device due to aluminium's high conductivity. However, working with aluminium is not a simple task so we picked a simple alternative: 3D printing. The Raspberry Pi's case mounts directly to the drone's frame using rubber mounts to provide insulation from the drone's vibrations.
The camera's mount allows for it's angle to be adjusted so it can capture video in different positions to best suit each use case.
We searched on the internet for already existing cases and camera mounts and we adapted them to fit our needs.

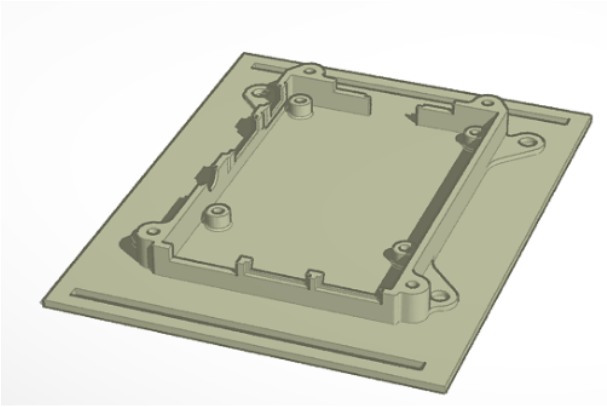


Figure 4: Bottom part of the case

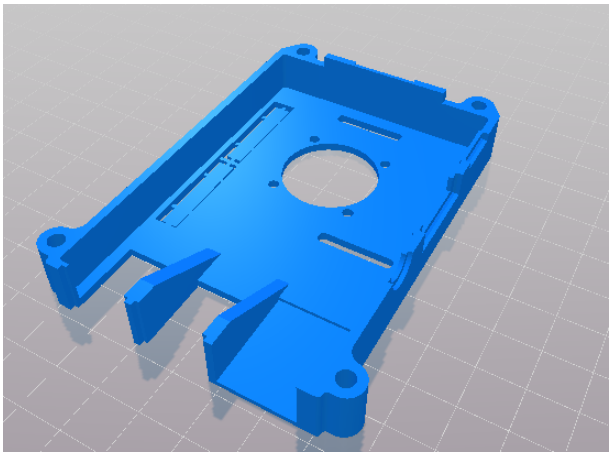


Figure 5: Top part of the case

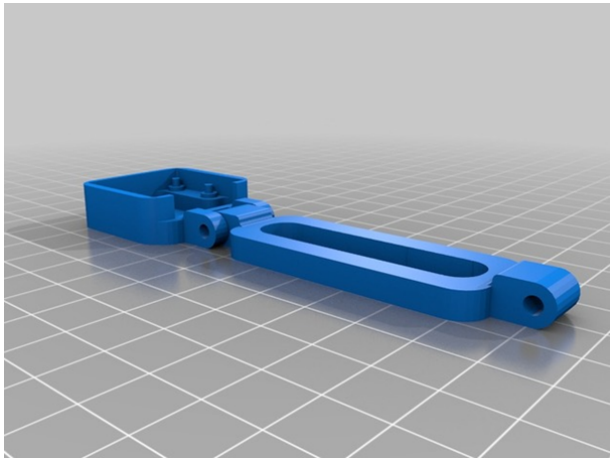


Figure 6: Camera Mount

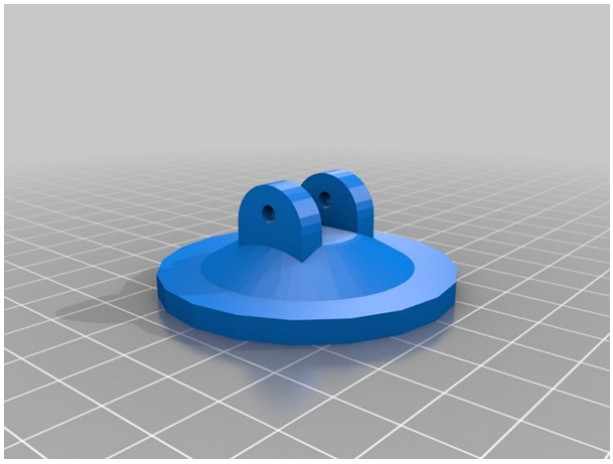


Figure 7: Mount support

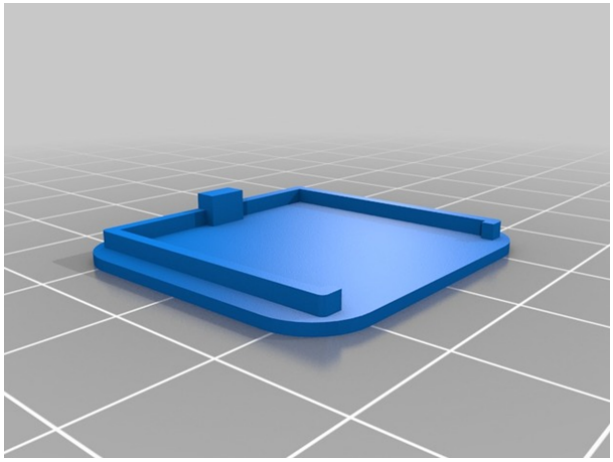


Figure 8: Camera cover

2.6 Camera Module

2.6.1 Front Camera

Model: Eachine 1000TVL CCD

The front mounted camera was designed for FPV (First Person View) applications and it is connected directly to an AV transmitter that broadcasts the video feed in real time. It has a resolution of 976x582 in PAL mode and can be used in any device able to connect to an AV receiver.
We display the feed provided by this camera using the AV receiver that connects to the GS using a USB cable. The receiver works in a similar way to a webcam (It uses the UVC Standard - USB Video Device Class) and can be used by an OpenCV capture to continuously generate jpeg stills. These stills allow us to display the stream on a web interface.

Figure 9: Analog camera

2.6.2 Bottom Camera

Raspberry Pi Camera Module V2

For the bottom mounted camera we opted for the usage of a Raspberry Pi Camera which is a small and compact camera module designed for the Raspberry Pi. There are a number of reasons why we picked this camera:
It has a narrow field of view due to its small lens, which is actually beneficial in our use case. A smaller lens means less image distortion and it allows us to capture a clear representation of the area desired. A wider lens would capture more area at once but the image distortion could cause some discrepancies when using the shots to calculate object positioning.
Because it shares hardware with our control module, the Raspberry Pi can handle both the video capture and the drone control so it does not require any other hardware that we weren't going to use already. It uses the Raspberry Pi's GPU for encoding which allows us to use heavy codecs like H.264 to compress the video to a low bitrate so that we can have a live video feed with decent quality over our Wifi network.

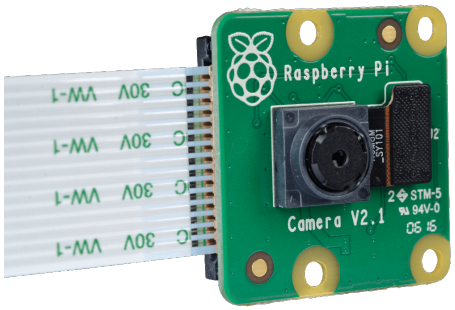


Figure 10: Pi Camera

2.7 GPS Trackers

For the tracking of all the components of a regatta we used two different kind of trackers.
For boats we used a android phone with its location services turned on. The reason behind using a phone in this case was that the orientation of the boat was data that we needed to have for our calculations.
For buoys we used a Nodemcu with a UBLOX - NEO6M. This is a small device easily powered with a powerbank capable of providing a really precise location. Since GPS was all the data that we required this simple module was all that was required.

Figure 11: Android

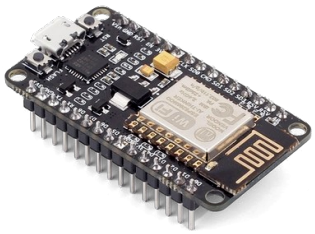


Figure 12: Nodemcu

Figure 13: UBLOX NEO-6M

Chapter 3

3 State of the art

As a growing topic among tech people, the amount of projects with drones has been growing. As such we took inspiration in some done in the past years.

3.1 Related projects

3.1.1 Monitoring system of recreational boats in the Aveiro coastal inlet

This project consisted in doing a pre-set flight capturing film and trying to recognize boats, counting them. This was done by image processing within a certain margin of error. This was accomplished with opencv, something we took into account and also applied similarly.

3.1.2 SmartDrone

This project consisted on implementing into a commercial drone a control unit capable of processing the images from its camera to make flight decisions. It was a good point of reference as to how you could control a drone and what components were used.

3.2 Technology

Here we present a sum up of the technologies used across the project

3.2.1 HTML5

HTML5 is a markup language used to display content over the world wide web. It is used to create our interface.

3.2.2 CSS

Cascading style sheets is a style sheet language used to describe the presentation of documents written in markup languages.

3.2.3 JavaScript

A interpreted programming language made so that scripts could run on client-side when required by a web-page.

3.2.4 INAV

Inav is a fork of CleanFlight, a Open-Source flight controller software for modern flight boards. This fork focus specially on GPS features and supports a wide variety of flight controllers. [16]

3.2.5 Flight Controller

A flight controller is an integrated circuit normally made up of a microprocessor, sensors and input/output pins.

3.2.6 MSP

MultiWii serial protocol is a communication standard to interact with a flight controller. Its implementation contains a list of the most common operations one would expect from a telemetry point of view. Custom functionalities are possible and the main reason for it's use.[11][12][16]

3.2.7 Ubuntu Desktop 16.04 LTS

A free and open source linux operating system based on Debian.

3.2.8 Raspbian

A Debian-based computer operating system for Raspberry PI.

3.2.9 Mosquitto

A lightweight open source message broker that implements MQTT, suitable for messaging across multiple devices.

3.2.10 Memcached

A general purpose memory caching system. Allows data to be stored and retrieved with ease during some procedures.

3.2.11 Python

A interpreted high level programming language capable of supporting multiple programming paradigms. Supported in many operating systems, its readability, simplicity and extensibility make it one the most widely used programming languages. Its support for what was required and simplicity made it the key language in this project.

3.2.12 C++

A compiled programming language supported by the Arduino IDE used to program the NodeMcu modules.

3.2.13 OpenCV

Open Source Computer Vision consists of a library of programming functions designed for real-time video processing. It's multi-platform and free for use. While developed in C/C++ it also supports several interfaces for other languages, like Python.

3.2.14 FFMPEG

A free software project capable of recording, converting and creating audio and video streams in different formats. [1][6]

3.2.15 ELK

ELk is the acronym for three source projects: Elastic Search, Logstash and Kibana. Elastic search is used for search and analytics, logstash is a server-side data processing pipeline and kibana for data visualization. Combined they are used for logs.

Chapter 4

4 System requirements and architecture

4.1 System requirements

This chapter will be about clarifying the system requirements and its architecture. We will talk about the use cases, its respective contexts and functional and non-functional requirements. Diagrams for better comprehension of the system will be shown.

4.1.1 Functional Requirements

Functional requirements were gathered during the initial reunions. As such, the defined requirements were these:

- Drone must be able to autonomously monitor buoys
- Drone must be able to follow a boat autonomously.
- Drone must be able to autonomously monitor the start of a regatta.

By autonomously it is intended that the drone must be able to move to a GPS position, making the best decision considering each use case.

4.1.2 User context description

This system is to be used by a referee to aid in his decision making. It is expected that he checks the real time video stream, the boats and buoys identifiers and make recording of video parts.

4.1.3 Actors

Even though there is some versatility, this system is for referees in regattas.

This referee is a user with knowledge to make the right decision calls in a regatta. He has at his disposal a web platform capable of choosing what actions is the drone to take, alter color configuration for buoy detection, check the information about the drone and track the race with both cameras from the drone and the GPS identifiers. Specific parts of the video stream can be recorded and it his possible to add real time video identifiers for better tracking.

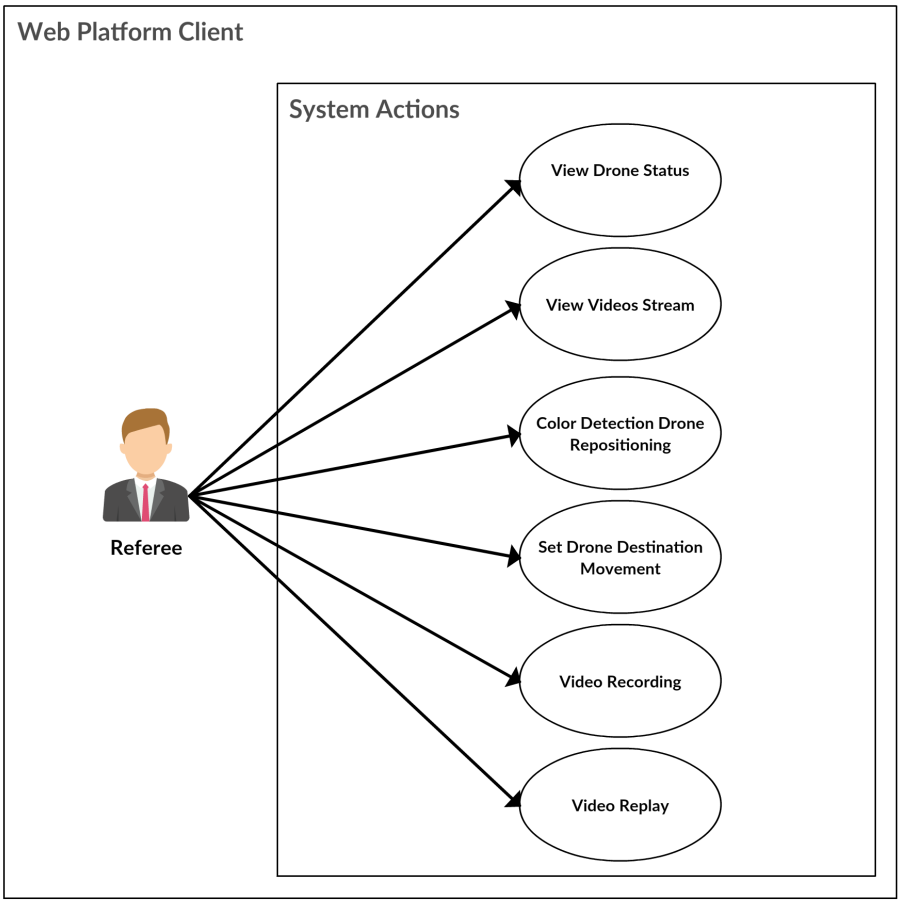


Figure 14: Possible system actions

4.1.4 Use cases

In the diagram presented it is given the model of the web platform as it is the only direct interaction module with a user.

- **View Drone Status** - The referee is capable of seeing all the information about the drone, like it's location, battery level, orientation and status.

  Priority - High

- **View Video Stream** - The referee is able to see live video transmission, either from the bottom camera of the drone or the front one.

  Priority - High

- **Color detection configuration** - The referee can set up a new color configuration for buoy recognition for better positioning.

  Priority - Low

- **Set Drone Destination** - The referee chooses a destination on the map and the drone must move to said destination.

  Priority - High

- **Video Recording** - The referee can choose to record a part or a full video from the regatta.

  Priority - Medium

- **Video Replay** - The referee can replay videos previously recorded.

  Priority - Medium

4.1.5 Non Functional requirements

- **Performance** - All communication happens in real time and everything depends on that. The system must be handle video streaming, communication over the several modules without problems.

  Priority - High

- **Usability** - The web platform must be user friendly for anyone new to the system to operate without a hitch.

  Priority - Medium

- **Portability** - The developed modules must work for every compatible drone and the communication system must be supported in many machines.

  Priority - Medium

4.1.6 Assumptions and dependencies

The system uses the raspberry pi as an access point for the network. It requires no Internet connection. The drone control module must be always turned on. The client module was developed in Ubuntu but can be used in every platform.

4.2 System architecture

In this section we will talk about what was used and how it was implemented, showing the domain system model and corresponding physical and technological models.

4.2.1 Domain model

The client connects to the drone control module through our web platform.
The web platform is responsible for the interaction with the control module. This web platform gives access to all the video received, from both cameras, all the real time data from the drone and GPS trackers, etc.
The control module is responsible for receiving commands from this web platform and processing them, adjusting the drone to the commands received or processing video for recording.
The drone control module receives instructions to be given to the drone and transforms them into the specific commands the flight controller takes.

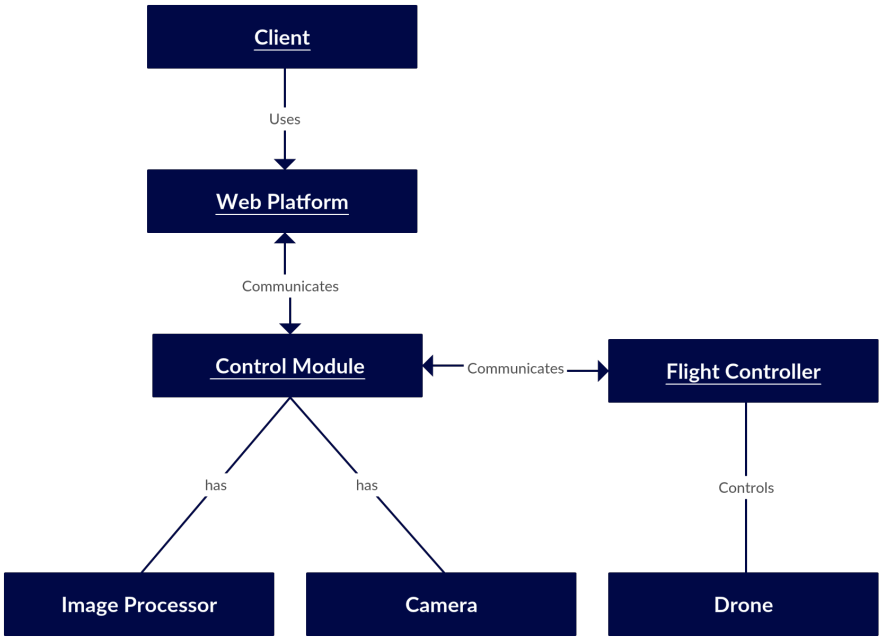


Figure 15: Domain model

4.2.2 Physical Model

The next diagram shows the physical model adapted to our system.

- **Wave Drone Ground Station**
  Represents the virtual machine in which the web platform is being hosted. The operating system is currently Ubuntu 16.04 LTS.

- **Wave Drone External GPS Trackers**
  Represents the GPS trackers across the regatta, either in boats or buoys. They communicate with the Raspberry PI.

- **Wave Drone External Processing Unit**
  Represents the module where most of the system processing is happening. It's responsible for handling most of the communication and video transmission. It's executed in a Raspberry PI 3B with Raspbian Strech.

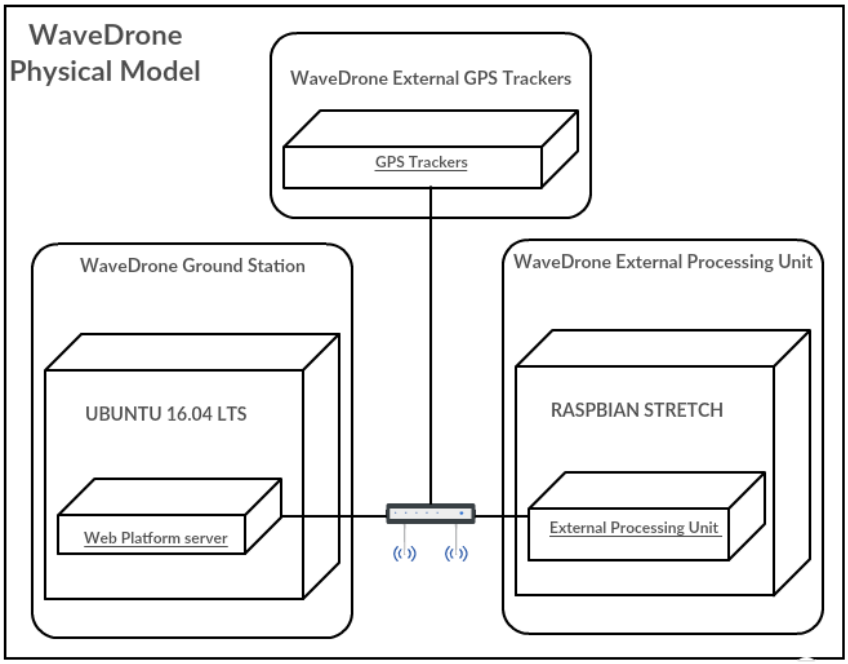


Figure 16: Physical model

4.3 Technological Model

The next diagram shows a high level vision of all the technologies adopted across the project. While the web platform is executed in Ubuntu, the processing unit is done in a Raspbian Strech environment.
This modules communicate through MQTT and UDP, depending on the kind of data being transmitted. While UDP sockets are used for most of the video transmission, MQTT is used more for telemetry data and commands, making sure that data received is correct. The external GPS trackers transfer their data to the processing unit through UDP Sockets because of it's lightweight value.

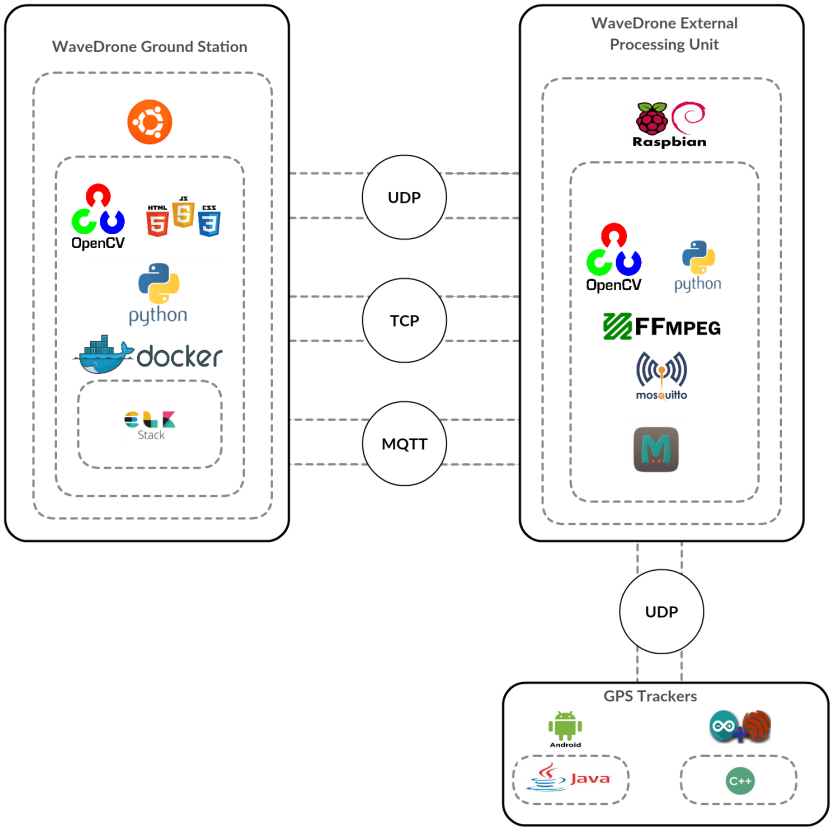


Figure 17: Technological Model

Chapter 5

5 Implementation

5.1 Drone specification

5.1.1 Payload needs

The drone was assembled for our project and the components were picked taking into account our necessities and the budget available. During the early phases of our project we decided what components we needed to fit on the drone to fulfill our needs. We needed a network interface to establish communications with the drone, a video capturing device and a device capable of processing video for object detection and calculating the positions where the drone needed needed to be sent to.

5.1.2 Payload components

The payload was built around a Raspberry Pi which is a very versatile SoC. This device handles the connection of the drone's flight controller to the network interface as well as the digital video capture and streaming and gps data collection from the nearby markers. We estimated the total mass and power requirements of the payload to estimate the requirements of the drone. We ended up with the following table:

Component	**Mass (g)**
Raspberry Pi 3 B+	42
SD Card	0,5
Raspberry Pi Camera V2	3
USB Network Adapter	6
USB to USB Micro cable	10
Total	61,5

Figure 18: Payload table

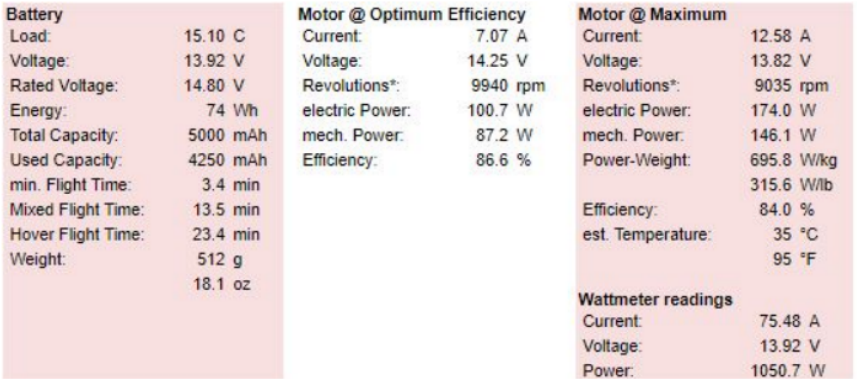
In regards to the power consumption of this payload, this model of the Raspberry Pi has a recommended power supply capacity of 2,5 amps at 5 volts. To supply the

power needs of the Raspberry Pi we draw current from the drone's UBEC which converts the battery's current from 14,8 to 5V. Batteries designed for drones have a small capacity but can output large amounts of current and compared to the drone's motors the current draw of the Raspberry Pi is almost irrelevant so the Raspberry Pi will always be powered during flight. This also saves weight since a dedicated 5 volt power bank for the Raspberry Pi would have less energy density (energy per unit of mass) and substantially increase the mass of the payload.
The USB network interface was needed to configure the drone as a wifi AP as it offered much better range than the onboard antenna and this network would be used for 3 channels for all the communications needed to have a GS attached to the drone. These feature. A Video channel, a Telemetry channel and a Control channel all on the same interface.

To fit the payload on the drone we combined a few 3d models available online to design a case to house the Raspberry Pi and its camera in the correct position. This case is mounted on the frame with rubber mounts to isolate the payload from the vibrations generated from the motors

5.1.3 Drone components and reasoning

After the payload was estimated our project advisors estimated that we would need a drone with a total mass of 1 to 1,5 kg. This would make a drone powerful enough to carry our payload while leaving some room for more components to fulfill the needs of future projects that reuse this drone. The extra weight also helps with drone stability during flight as it is less affected by the intense wind at higher altitudes. Stability is very important, keeping the drone in a stable position is crucial for video capture. Besides the drone components themselves we also added an AV camera and transmitter, for a backup live video feed.
According to an online calculator (https://www.ecalc.ch/xcoptercalc.php) here are some estimates of the drone's capabilities and other useful data about it.



Battery		
Load:	15.10	C
Voltage:	13.92	V
Rated Voltage:	14.80	V
Energy:	74	Wh
Total Capacity:	5000	mAh
Used Capacity:	4250	mAh
min. Flight Time:	3.4	min
Mixed Flight Time:	13.5	min
Hover Flight Time:	23.4	min
Weight:	512	g
	18.1	oz

Motor @ Optimum Efficiency		
Current:	7.07	A
Voltage:	14.25	V
Revolutions*:	9940	rpm
electric Power:	100.7	W
mech. Power:	87.2	W
Efficiency:	86.6	%

Motor @ Maximum		
Current:	12.58	A
Voltage:	13.82	V
Revolutions*:	9035	rpm
electric Power:	174.0	W
mech. Power:	146.1	W
Power-Weight:	695.8	W/kg
	315.6	W/lb
Efficiency:	84.0	%
est. Temperature:	35	°C
	95	°F

Wattmeter readings		
Current:	75.48	A
Voltage:	13.92	V
Power:	1050.7	W

Figure 19: Estimates of flight

Figure 20: Estimates of flight 2

This calculator gives us estimates on several aspects of the drone such as flight time, range and payload capacity.
The drone possesses 6 motors making it a hexacopter. Most drones available on the market today usually only have 4 (quadcopter). Having 6 motors allows for better control of the drone since there are 2 extra points where thrust can be adjusted. The motors on an hexacopter are displaced 60º apart from each other and this allows for the drone to continue its flight if one of the motor fails, using the 2 adjacent motors to compensate for the thrust that the failing motor should be providing.

5.1.4 Flight Controller

Description

To explain how the different components work together to create a drone we are going to take a brief look at the FC, what are its functions and how we use it in our project. The FC is, in simple terms, the brain of the drone. It possesses several sensors such as a Barometer, a Gyroscope, an Accelerometer and a Compass to collect data useful

to for the positioning of the drone. The flight controller combines the input of these sensors and the radio receiver to control the drone adjusting the thrust of each motor individually. This is done sending a speed reference signal to each ESC. The ESCs convert the DC current from the battery to a 3 phase AC and the speed of the motor is controlled by adjusting the timing of pulses of current delivered to the motor's windings.

Firmware

There are multiple open source firmware solutions for UAVs that are installed on the flight controller itself. This firmware usually allows the user to connect to the drone via a serial interface to do all the configuration tasks that need to be done. The one we used for our project is INAV. INAV is a fork of cleanflight and it's main focus are navigation and positioning features. It also implements several communication protocols that allow the configuration and control of the drone and also has multiple flight modes. Although it is a fairly recent project, the navigation features it provides simplifies a lot of the work needed to be done for automated flight. The only thing that needs to be supplied is the target GPS position. The multiple flight modes allow the drone to fly in different ways combining the different sensors on the drone.For example position hold forces the drone to continuously maintain a certain GPS position while Altitude hold forces it to maintain the barometer reading at a fixed level.

Radio controller

The radio receiver connected to the flight controller has multiple channels to receive commands for the multiple controls needed to pilot a drone. Each of these main controls has its own channel and they control pitch, yaw, throttle and roll. Besides these there are other channels available that can be mapped to several different features through the firmware, such has controlling the drone's mode and engaging a killswitch.
The controller itself has a radio transmitter and it needs to be paired with the receiver. The custom channels also had to be mapped to the switches on the controller. The radio control system broadcasts on 2,4 GHz frequencies and uses frequency hopping to provide some security in the connection.

AV Transmitter

The entire AV Video system is pretty light and simple. The camera is connected to the transmitter and both are connected directly to the drone's battery. Both of this components accept the voltage outputted by our battery so there is no need to run the power from the UBEC.

The transmitter broadcasts the video through a radio link in the 5.8 GHz frequency. This frequency has been increasingly popular for drones and is used for FPV applications. The main advantages are that hardware for this frequency is cheap and the frequency itself does not require a license to be used in most countries. The main drawbacks are that in our country the limited output power of an antenna for this frequency is 25 mW which is quite low considering that our transmitter can output 600 mW so that poses a bottleneck on the range and also the low penetration of the radio waves of this frequency have poor signal penetration which makes them unable to pass through obstacles leading to possible loss of signal when the drone loses line of sight.

Drone related issues

A lot of issues related to the drone itself affected us during the development of the project and to understand some of the choices we made we need to go over very briefly all the main setbacks we faced.

- Telemetry link:
  Our initial plans were to have a dedicated radio link for telemetry that would be broadcasted from the FC itself and received on the GS. INAV does not support this feature yet so we had to use the Wifi network for this purpose.

- Toilet Bowl Effect:
  The toilet bowl effect occurs when the drone starts flying in a circle that gets progressively bigger. There are various reasons why this could happen and in our case it occured when the drone was being tested in the POS_HOLD mode. The POS_HOLD mode is one of INAV's flight modes. In this mode the drone is continuously trying to maintain its current GPS position. The problem was probably due to some magnetic interference causing some inaccurate readings on the compass, which is a very important sensor needed for automated flight. The high amounts of electrical current running in the drone's frame generate a magnetic field and to avoid its negative effects on the sensors these have to be placed as further as possible from the areas where the wires run. We have extended the GPS module's antenna in hopes to fix this issue but no tests have been done yet to observe if the drone still suffers from the Toilet Bowl Effect. Without fixing this we couldn't test any navigation features on the drone, due to the fact that the drone would not be able to remain in a stable position.

5.2 Drone communication

All communication with the INAV controller has to be made via serial port. INAV uses MSPv2 as it's remote messaging protocol, and all communication with the controller has to be made using it.

Offset	Usage	CRC	Comment
0	$		Same lead-in as V1
1	X		'X' in place of v1 'M'
2	direction indictor		As V1, relative to the FC, <, > or ! (error response)
3	flag	✔	uint8, flag, usage to be defined (set to zero)
4	function	✔	uint16 (little endian). 0 - 255 is the same function as V1 for backwards compatibility
6	payload size	✔	uint16 (little endian)
8	payload	✔	n (up to 65535 bytes) payload
n+8	checksum		uint8, (n= payload size), crc8_dvb_s2 checksum

The fields marked with a ✔ are included in the checksum calculation.

Figure 21: MSPv2 Message Format [11]

The function parameter represents the type of message we are sending, and each type of function has it's own parameterized payload type, which must be used in conjunction with it's corresponding function number. There is a checksum parameter which is generated using a provided checksum operation and operates over the flag, function, payload size and payload parameters. There was no API or library for drone communication available, so there was the need to implement the drone communication logic ourselves.

We will now do a walk through of how we decided to implement it.

5.2.1 Structs class

All messages issued to the controller need to be serialized using the MSPv2 format. To facilitate the serialization process, the python construct library was used. This library automatizes the serialization, deserialization and parsing of data using custom built formats. The struct.py class holds the parameterized python construct elements corresponding to the MSPv2 encapsulation format and to the payload formats of each message. The use of these structures allows for the automatic conversion of a list or dictionary of parameters into a MSPv2 serialized message and its corresponding payload and vice-versa. Additionally, this class also contains the function numbers corresponding to each MSPv2 message. [13]

5.2.2 MSP class

This class is responsible for maintaining an open connection to the usb port that is connected to the controller and building and communicating MSPv2 messages using the structures implemented in Structs.py. Because all drone communication had to be done via usb, there was the need to implement a global connection accessible to all modules that require access to the drone, either to issue commands or obtain data.

5.2.2.1 Initialization Upon start, a new serial connecting is started. This connection is the only direct connection to the controller and all messages exchanged between the RPI and it are done via this channel.

5.2.2.2 sendCMDreceive This function is used for issuing messages to the drone and receiveing it's responses. It takes the function number, payload data and payload format structure as input.

Message construction and serialization

The payload is serialized beforehand. Since this parameter's size and structure are different depending on the message type, this parameter is serialized by itself and

then parsed as a whole array of bytes in the payload parameter in the building of the MSPv2 message.
Firstly, the payload parameter is serialized into it's correct format using the payload data and payload structures provided.

```
payload=payloadformat.build(data)
```

Then, the MSPv2 message excluding the checksum parameter are built using the function number provided, the serialized payload and the payload size.

```
parameters = checksumformat.build(dict(function=function,
payloadSize=data_length, payload=payload))
```

With the correctly serialized parameters, the checksum value for the message can now be calculated. The checksum operation used by INAV's MSPv2 messages is disclosed in their website in c:

```c
uint8_t crc8_dvb_s2(uint8_t crc, unsigned char a)
{
    crc ^= a;
    for (int ii = 0; ii < 8; ++ii) {
        if (crc & 0x80) {
            crc = (crc << 1) ^ 0xD5;
        } else {
            crc = crc << 1;
        }
    }
    return crc;
}
```

Figure 22: Checksum function C implementation

Since this module was being developed in python, this operation was re-implemented in python.
The generated checksum value is then serialized and added to the serialized message to complete it.

Communication and parsing the input data

After the message is correctly serialized, it is sent to the controller via the usb port connection established earlier, and the drone will respond with another MSPv2 Message. Upon receiving the response from the controller, the message is parsed using the MSPv2 encapsulation format, from where the payload will be extracted as an array of bytes.

```
format = structs.Message.structRecv

parsed_data = dict(format.parse(blist))

blist = parsed_data["payload"]
```

The payload data is then parsed as an array of bytes using the payload format structure provided. After this step, a dictionary data structure containing all parameters of the payload and their values will be generated and returned.

```
payloadBytes=construct.Array(parsed_data["payloadSize"],Int8ub)
.build(blist)

parsed_data = dict(payloadformat.parse(payloadBytes))
```

5.2.2.3 DroneCommunication.py Ensures that drone communication is available to all the modules residing within the RPI. This class is implemented as a singleton, responsible for providing global access to the msp.py object which holds the serial connection, and also implements the concurrency that is necessary to integrate the access of the different threads that are running. This class together with the defined communication functions defined in MSP_Thread.py work as a communication API to the rest of the components, creating an abstraction from the MSPv2 communication.

5.2.2.4 MSP_Thread This class implements all drone communication sequences. There are different sequences of commands that need to be issued by each module for different circumstances.

Drone information

The function getDroneData retrieves all the relevant drone data and returns it. This function aggregates drone data obtained by sending three different messages corresponding to it's current gps location, orientation and altitude. This method is used by all the modules that require access to the drone's information.

Following a boat

In order to assure that the use case of following and recording a boat does not depend on ground station connectivity, the boat following loop is executed on the RPI itself. To start following a boat, the startFollowingFunction executes based on selected altitude, heading and boat id. The altitude and heading define the altitude and orientation of the drone while following the chosen boat, and the boat's id represents the id of the boat we want the drone to follow. The gps locations and id of all boats are available on memory in the RPI, and because the network's point of access is in it too, this allows for an isolated following of boats, even when the ground-station is unreachable. This method constantly commands the drone to move to the selected boat's latest GPS location . This loop executes continuously until another method, stopFollowing, is called.

Dispatching to a buoy

To send the drone to a target buoy, this class provides a method which sends a single set waypoint message to the controller, which updates the drone's target destination to the most recent position of the selected buoy.

Dispatching to a custom position

Another method is also implemented, which represents the issue of a single set way point message to the drone's controller, but with a custom gps location. This method is used to send the drone to custom gps locations directly from the web interface.

5.3 Calculus

There was a lot of calculus involved in processing images and GPS information together to output, for example, a position on an image or to recreate a GPS position.

As such this topic will talk about all the math involved and explain how it was done. It's integration will then be specified.

5.3.1 GPS to image position

This module calculates the expected image position for an array of boats using their given gps position and the drone's data. The parameters of drone data necessary are it's gps position, compass orientation, x and y inclination angles and height.

Generate the x and y axis rotation matrices

Firstly, the appropriate rotation matrices, that equate the translation in the camera's orientation made by the drone's inclination, are generated.
These matrixes will be used further ahead to rotate the boats' three dimensional space representations around the drone's position, the origin, to account for the drone's inclination.

Calculate bearing and distance between the drone's gps position and the gps positions of different boats

Using each boat's gps position we can calculate the distance and bearing between each boat and the drone. This data is necessary in order to place the boats in a third dimensional space correctly. We then calculate the boat's x and y position using the distance and bearing, and place them at an z axis value of – height, while the drone representation is placed at the origin.

Rotate points around the X and Y axises according to the drone's x and y rotation angles, respectively, over the drone position

After we have generated the three dimensional points representations, by rotating them using the previously generated rotation matrices, which rotate the point around the origin (drone position), the drone's inclination is correctly accounted for.

Rotate point around Z axis to account for the drone's compass orientation

The last thing to do in order to obtain the correct and final three dimensional space representation of a boat's position, is to rotate the plane around the z axis based on the drone's compass orientation, which will correctly place the boat's position in reference to the drone's camera, and allow us to calculate the angles between the object-drone line and the camera's x and y planes.

Calculate the angle between the Object-Drone line and the y=0 and x=0 planes

Using the final position and the imaginary camera planes (x=0, y=0), calculate the angle between the drone-object line and said planes.

Calculate the object's expected position on camera

After obtaining the correct camera angle for each boat, the image frame position of the object is calculated using the camera angle and the canvas size:

$PosX = angleX/(cameraAngleX/2) * canvasSizeX/2 + canvasSizeX/2$
$PosX = angleY/(cameraAngleY/2) * canvasSizeY/2 + canvasSizeY/2$

5.3.2 Image position to GPS conversion

This module calulates the gps position of a given buoy using it's position on the camera image and the drone data. The drone data parameters necessary are the same that are used by the gps to image position converter: it's gps position, compass orientation, x and y inclination angles and height. The buoy's image position is calculated beforehand using the image detection module. This module is, essentially, an inverse of the gps to image position coverter, as it does the opposite calculus, using the same data.

Firstly, we calculate the angle between the buoy and the the x and y planes, using the tile size image position and camera angle:

$angleX = (imagePosX/(tyleSizeX/2) - 1) * (cameraAngleX/2)$
$angleY = (imagePosY/(tyleSizeY/2) - 1) * (cameraAngleY/2)$

Having access to the angles between the object and the x and y planes and the drone's height, it is possible to recreate the object's position as three dimensional point.

$hypotenuseX = height/cos(radians(angleX))$
$hypotenuseY = height/cos(radians(angleY))$

$posObject = Point3D((hypotenuseX * sin(radians(angleX))), (hypotenuseY * sin(radians(angleY))$

This three dimensional point represents the objects position as if the drone were turned north and without inclination. The drone's inclination still needs to be accounted for. To achieve this, two lines representing the drone inclination, are created. One line along the x plane, and another on the y plane, with inclination corresponding to the drone's inclination angles. The line that is perpendicular to both these lines

represents the camera's center point. The rotation in position between the original center position of capture and the new one represents the shift in the capture space made by the inclination of the drone. By applying this rotation to the position point, the point's placement is adjusted for the drone's inclination and consequent rotation in the capture area. The resulting imaginary point now represents the objects position as if the drone were turned north, taking into account it's inclination.

Lastly the drone's orientation also represents a shift in the capture space so the three dimensional position of the object is rotated along the z axis with an angle corresponding to the drone's heading, placing the point's relative position to the drone equal to the object's relative position to the drone.
After this step, the objects imaginary position is corrected, and placed the way the object is placed in the real world, according to the geographic coordinate system orientation . The object's gps position is then calculated based on the drone's gps position using the distance between the imaginary point and the origin (drone's imaginary position), and the point's bearing.

5.4 Web Interface

The web interface offers a simple graphic user interface so that the system can be easily used. It was implemented using HTML5, CSS, Javascript, Jquery and BootStrap. All the communication with the interface is done by MQTT and REST.
In the platform it is possible to do:

- Check the map and each component's info
- Watch the real time video with the option to choose which camera
- Check all the information relative to the drone
- Record video so that certain parts can be seen afterwards
- Make adjustments for the buoy recognition
- Re-watch and transfer all available recorded videos
- Select and dispatch drone to follow boat/stay at buoy
- Re-watch and transfer all available recorded videos

Let's do a little walk through of the full interface:

5.4.1 Home Page

The platform has a homepage, giving context about its purpose and it can do.

5.4.2 Map

As mentioned before we have at the users disposal a map filled with all the components of the regatta. Every module developed(GPS Trackers and Drone) will show on the map in its current GPS position. As this is supposed to be in a environment with no Internet connection Google Maps API couldn't be used. As such we used Open Street Maps that could be used with no network. Since we didn't want to cause a huge space problem we only used the Portugal map. For no dependencies we have it deployable over docker using the library "tileserver-gl".
All the GPS information is retrieved through REST.
In this page the user can visualize the markers and click on them for more details. This are distinguished with different colored markers, identified on the bottom of the page.
In this markers it is also possible to execute actions like send a drone to a certain position and make it return home. It is also possible to check if it is connected and its battery level.

5.4.3 Real time information

This web page is one of tha main pages of the platform. It allows the user to not only watch the video feeds but to also see all the drone data, like it's orientation, battery, status, etc.
The drone data is received through MQTT.
The video feed is received by consuming REST. As our system has two video cameras, in different positions, it is important to have access to both of them. With this in mind we have a button so a user can easily switch what video feed he wants to see.
This page also has three different commands. It allows for a full recording of a regatta, specific parts of the race and the return home command for the drone.
About the recording, this is a key feature of the project. Since the need for revision is necessary, having the feature of replaying certain parts of the video is crucial. After starting and ending a recording, the video is processed in the Raspberry PI and made available on the web interface for replaying with several options on how to replay it (slower, faster, where to start seeing, etc). These commands are done through MQTT.

5.4.4 Adjustments page

Since the drone adjustments by video are only done sometimes and given that buoys can have multiple colors this page allows for manual calibration of the HSV parameters of the detection. This page has two video feeds(object detection) and 3 sliders for the calibration. This allows to have the 3 sliders being altered simultaneously with the

video received, giving a better perception of what is happening. Only after saving the settings is the command sent to start using this adjustment. This communication is done through MQTT.

5.4.5 Storage page

This page shows all the available content previously recorded. They are identified and sorted by data. This page also shows information about the storage unit on the Raspberry PI like how much is being used and how much is used. All this information is consumed on the REST.

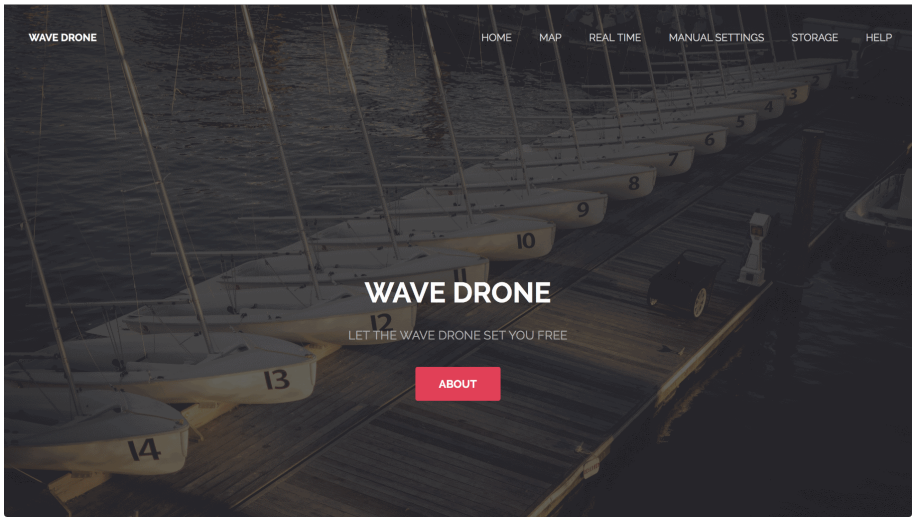


Figure 23: Home Page

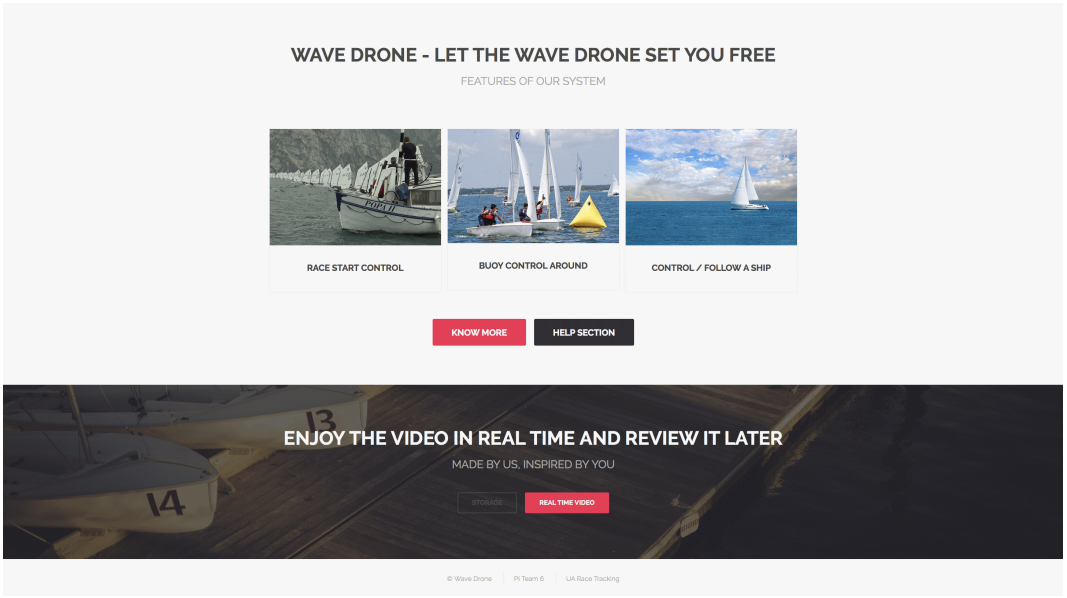


Figure 24: Home page 2

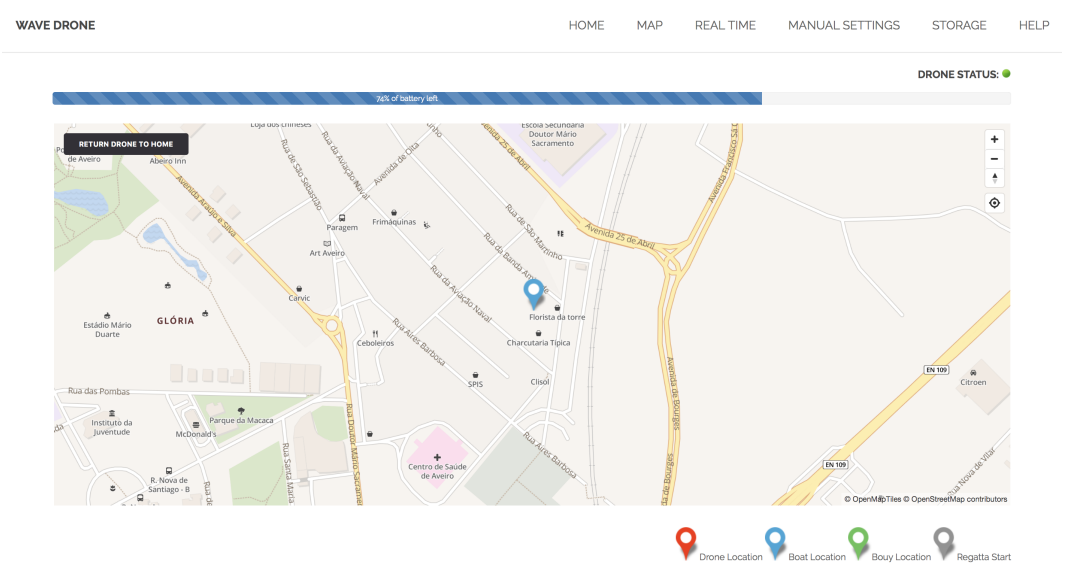


Figure 25: Map

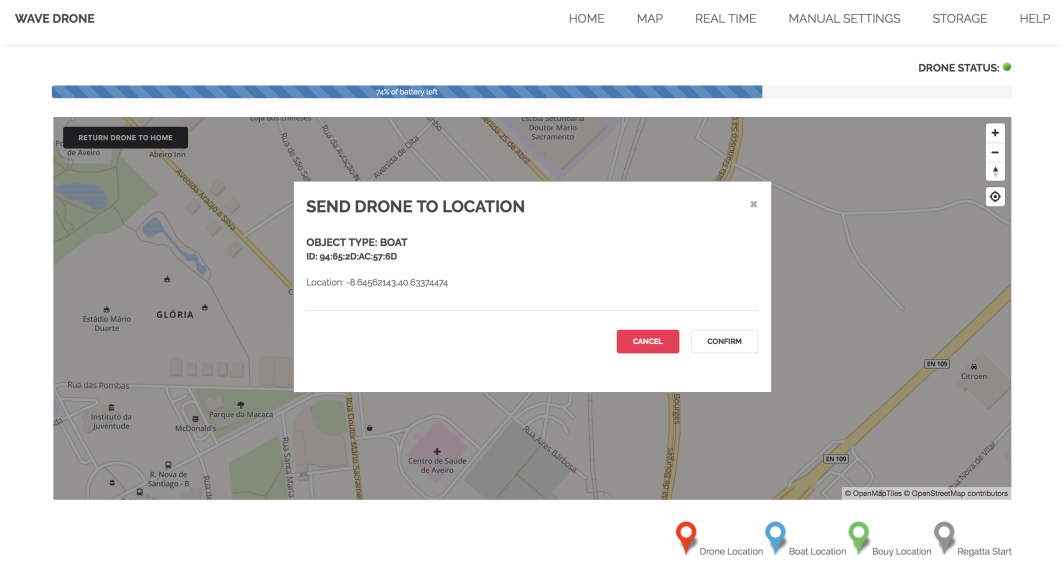


Figure 26: Map Info

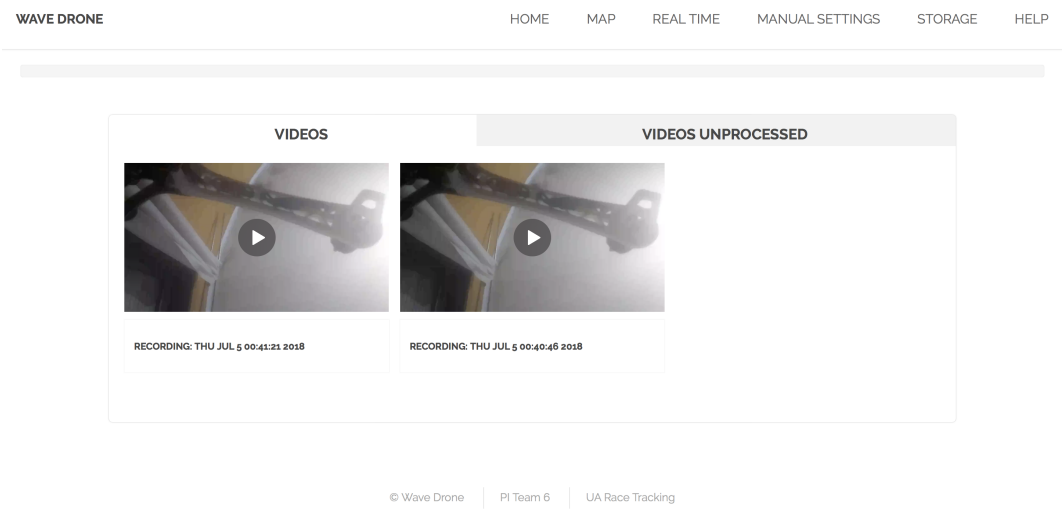


Figure 27: Storage

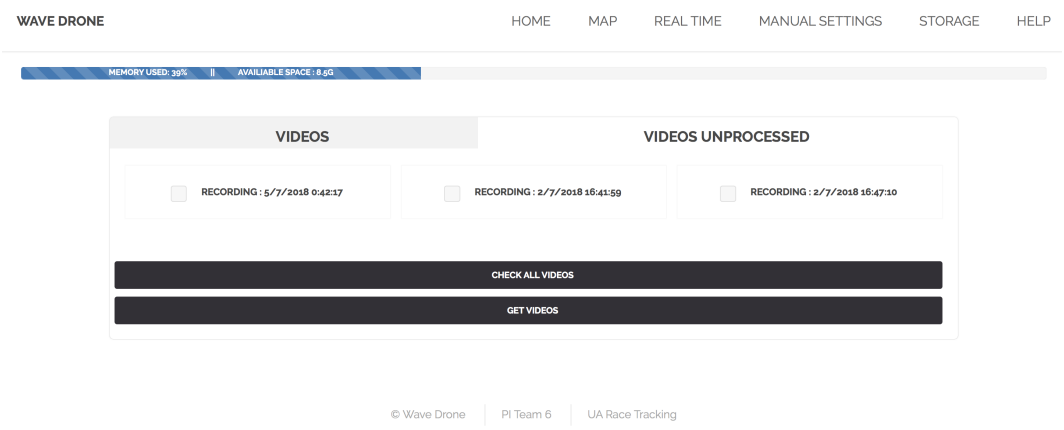


Figure 28: Storage 2

5.5 External Modules

5.5.1 GPS Trackers

We implemented two kinds of GPS trackers: an android device and a nodemcu with a Ublox NEO-6M.
This was done because each module would serve a different purpose.
Both serve data through a UDP Socket in the network created by the raspberry PI. The data however is different.
The android application allows a user to set it's ID and type of component it is representing(boat, buoy, etc). It has also the option to allow to send data to a different ip and port, but this is for different tests mainly. After the press of a button data starts to be sent every two seconds with all of its set information and its GPS and orientation in degrees. The purpose of this is to be used by a boat, where orientation information is relevant. This orientation information is important to set a drone in a good place for filming.
The application itself was developed in Android Studio and uses android device location services and phone sensors to determine it's data. GPS data and orientation are checked every second to determine the most reliable data.

The Nodemcu was programmed with the Arduino IDE and flashed. It connects to the network automatically and after catching enough satellites for GPS data it starts sending it's data do the Raspberry PI.The code was done in a manner so When there is no correct data about it's position it sends the information saying it is available but with no correct data. This is done to make sure it is there and it is connected but just waiting for correct data. This information is also sent every two seconds.

5.6 Video

5.6.1 Pi Camera

To use this camera we used the python module picamera. This module allows us to capture and encode video and images in various formats and resolutions using the MMAL API to communicate via messages with the GPU via the VHCI - Video Host Control Interface. The Raspberry Pi contains a BCM2835 which is comprised of an ARM Cortex CPU running Linux, and a VideoCore IV GPU, the physical memory is divided between them and the camera module is connected directly to the GPU using a CSI-2 cable. To output the video or images requested to the GPU, the GPU transfers the latest frame to the CPU's allocated memory using Direct Memory Access (DMA).

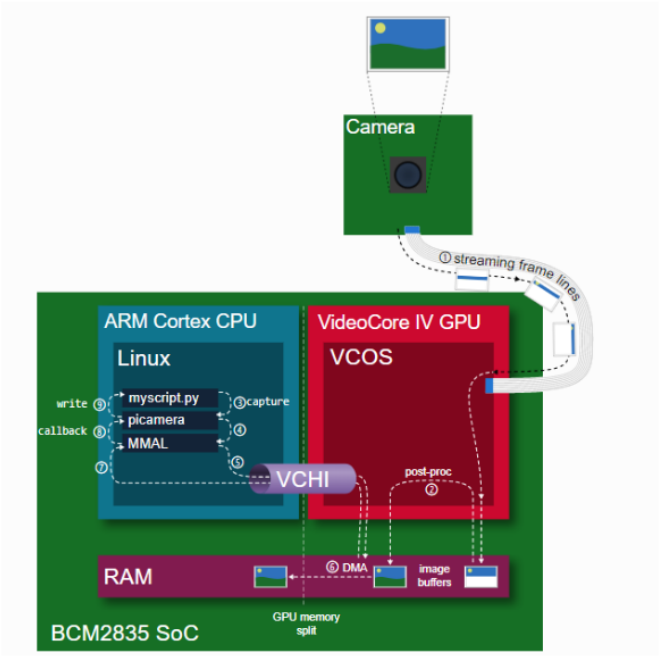


Figure 29: RPi with camera architecture

The MMAL API provides a simple interface for the camera firmware running on the GPU. It presents the camera in 3 different ports: the video port, the still port and the preview port. The still port is used to capture images and a strong noise reduction algorithm is used to improve the quality of these pictures. The video port allows for continuous video capture without as much noise reduction, allowing the camera to capture more images per second. The preview port works just like the video port but it is always connected to some form of output to ensure that the auto-gain algorithm can run.
The picamera module uses these ports provided by MMAL to allow for video or image capture from a python script, using the Pi Camera. It also has multiple Splitter, Resizer and Encoder components that can be used to obtain multiple video feeds with different sizes and formats.

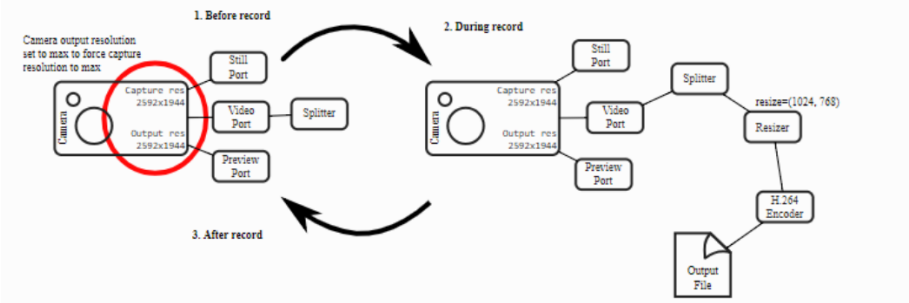


Figure 30: Recording procedure

We used these components to create a module that allows for 4 different captures can be made simultaneously, for different purposes in our project:

- The Stream port: It continuously writes the content from the output buffer to a udp socket, this port is used to transmit a real time video feed encoded in h264 to the GS, used to feed the interface. The video feed has a resolution of 854x480 at 15 fps with a max bitrate of 0,6 Mbps.

- The Recording port: This port records raw H264 video to the Raspberry's SD card. These recordings can be later multiplexed into mp4 containers using ffmpeg and transfer them to the GS. This allows the videos to be viewed in any media player on the GS and also on the Web interface. Video is captured at a resolution of 1280x720 at 15 frames per second from the video port with a max bitrate of 3 Mbps.

- The Recording port with data: This port is similar to the Recording port, but it also sends data to the GS to process and great a dataset to enable the user to view a replay of the GPS module positions during that video segment. The video settings are also the same as the recording port, but with a higher bitrate of 5 Mbps. This is port is intended to be used when it is crucial to have as much image quality as possible.

- The Frame port: This port uses the still port to capture frames from the camera which we use to process in OpenCV for color based object detection. The frames are captured in BGR at a resolution of 1280x720.

The resolutions, framerates and bitrates picked for each of these ports were selected taking into account the necessities of each use case and the capacity of the hardware. The network bandwidth and the Raspberry's SD card's write speed are limited so these resources had to be carefully divided between the different components. The settings for the live video stream were picked after testing the bandwidth available in our Wifi network and the main goal was to obtain the best image quality possible while maintaining a decent range. For the video recordings the settings were very dependent on the Raspberry Pi's SD card. The card we used has a write speed 10 Mbps so we split that between the recording functions we needed leaving also some room for other tasks that need some to read or write files on the SD card.

5.6.2 Video Encoding

For this project we attempted two different types of encoding MJPEG and H264. We started with MJPEG because it is a lighter codec performance wise. In this codec the frames are encoded individually as a JPEG image and played in sequence to obtain

the video stream. This encoding is very fast but at the cost of image quality, MJPEG requires a much higher bitrate (amount data per second) than H264 to represent an image with the same amount of quality. We ended up switching to H264 because it allowed us to reduce the bitrate of the video stream quite substantially, reducing the amount of data needed to be transmitted. This allowed us to improve the quality of the image and improving the maximum range at which the stream is presentable due to the lower amount of network bandwidth needed.
H264' s main advantage is the usage of movement prediction to reduce the amount of space required to represent a video. It can generate 3 different types of frames I,P and B. I (Intra-coded) frames represent an entire frame without needing information from other frames and they are usually bigger than the other types. P (Predicted) frames contain image data and motion data, with this motion data and previous frames the decoder can recreate the full frame reducing the amount of replicated data. B (Bidirectional) frames are similar to P frames, but can also reference segments from future frames to combine with the motion data. In our project we are using the default encoder settings that picamera provides for H265. It uses the High profile which allows it to generate both I and P frames. For our live video stream this may cause some issues because we are using a UDP socket to transmit each frame in an individual packet. This does not guarantee that the frames arrive in the correct order leading to some errors when packets are lost or received out of order. If an I frame is lost the P frames that use the data contained in the lost frame will not be correctly decoded leaving some errors in the image. A solution to this would be to only generate Intra-coded frames but it would also increase the bitrate of the video unless we reduced the quality so we opted for the first option because our main goal was image quality. The errors related to out of position or missing frames are handled by the OpenCv videocapture object that receives the stream of video.
Even with the usage of heavy compression algorithms the bandwidth of the network link is the main bottleneck to the quality of the video stream. At longer distances the wireless signal is much lower, sometimes not even strong enough to transmit the stream. In these situations the usage of Predicted frames really shows its strength, since most of the footage we need to capture and show the user in real time are taken with the drone stationary. This footage's bitrate is reduced to as low as 0,2 Mbps when there isn't much movement involved improving stability of the stream over greater distances.

5.6.3 Video for the front-end

We have 2 types of video that needed to be displayed on our web interface: live videos and recorded videos.
The live video from our digital camera is received from our udp socket in a splitter

that replicates the same stream on 4 other sockets, this is to allow multiple OpenCV VideoCaptures at once so we can have multiple feeds at once. The OpenCV VideoCapture object that receives the stream outputs the frames it contains in BGR, these frames are then converted to JPEG and if needed some details are added to the image, such as GPS marker positions and objects selected by colour. The final JPEG frames are served to the web interface using REST. The same process is used for our analog camera but instead of connecting to a network stream the OpenCV VideoCapture object is connected to the AV Receiver. We have the following feeds of live video on our REST API:

- /video.mjpeg - The live feed from the Raspberry Pi's Camera. If the show identifiers option on the interface is set, also displays the boat identifiers on their correct image position.
- /mask.mjpeg - Shows the mask generated from the selected colour values for image detection and the feed from /video.mjpeg.
- /final.mjpeg - Shows the same feed as /video.mjpeg but with a green contour over the detected objects from the selected colour values.
- /analog.mjpeg - Shows the live feed from the analog camera

The recorded videos are recorded in a raw H264 format. This format does not contain any information related to the duration or the framerate of the actual video and it isn't playable in most devices. When the GS user needs to view a recorded video he first must send a command for the video to be processed. Processing the video means taking the raw H264 video and muxing it into an mp4 container and we are using ffmpeg on the Raspberry Pi to do this operation. After being muxed and transferred to the GS the videos can be played on any device capable of playing mp4 files as well as in our web interface with the usage of html5's video tag. For the transfer of these videos we used TCP sockets. The processing and transferring of these videos is intended to be done with the drone landed, due to the high processing and network requirements of these operations but it can also be done during flight for small segments of video if a replay must be available in a short amount of time.

5.7 Buoy recognition for better positioning

To center the drone better around a buoy we felt that GPS information was not enough. As such we decided to implement image recognition. This was done by processing a frame every few seconds from the bottom camera of the drone. This frame comes from the video splitter done and explained previously. A frame is requested and processed according to the parameters set on the interface, also explained previously.

With the HSV parameters set previously, a color mask is applied and the biggest object is selected as the buoy. The position on the image is then discovered and we retrieve the X and Y corresponding to the center of said object. With this discovered we apply the math previously explained to recreate a GPS position and feed it to the drone as the new location. This new position should be more centered than before as this new position is based on the drone's GPS position and the object's relative position to the drone.

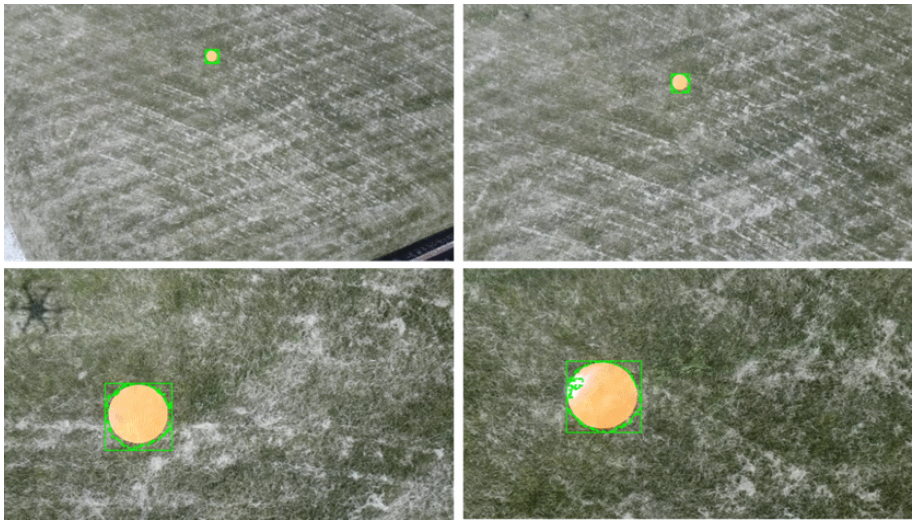


Figure 31: Buoy detection

5.8 Communication and work flow

With all the modules explained it is important to understand how they communicate and what data flows through the network. As such, let's take a deeper look at the technological model presented and the reasoning behind it.

5.8.1 External GPS trackers

GPS data is sent by the GPS trackers every two seconds to the rpi through an open UDP socket.

5.8.2 RPI

The rpi has a main class responsible for starting all the communication modules present. This class, DroneMain does the following:

- **Start the udp socket** - This socket receives all the data from the trackers (id, latitude, longitude, name) and stores it on the memcache server of the rpi. This data is needed for future use like following a boat or moving to a buoy. Every time there is new data it is sent to the ground station through mqtt, on the topic "id".
- **Start the drone data broker class** - This class is responsible for constantly requesting the drone data through the drone communication singleton and sending it through mqtt to the ground station under the topic "droneInfo".
- **Start the mqtt listener for video** - This mqtt listener, under the topic "videoRequest" receives several commands from the groundstation for video-related actions such as to start and stop recording a video, to process it and send it to the ground station or to get storage related data.
- **Start the mqtt listener for drone commands** - This mqtt listener, under the topic "droneCommand" receives drone movement commands from the groundstation such as moving to a buoy, follow a boat, adjust its positioning with the new calibration on the interface, move to a certain position or to return home.

5.8.3 Ground station

The ground station has several modules receiving data that they require and using them for the purpose set. Let's go through them:

- **Receiving drone info** - A mqtt client subscribes to the topic "droneInfo" and receives all the updates sent by the rpi of the drone data. This data is received and published on the REST for later retrieval by the interface. If the show identifiers option on the interface is enabled, the calculations that translate boat ids into image position are also performed with the data obtained and the boat gps positions in memory. These image positions are updated in the memcached server, to be placed over the video stream's frames that are received.
- **Receiving GPS trackers info** - Another mqtt client is subscribed to the topic "id" and is receiving all the updates from the rpi about the GPS trackers. This data is also published on REST to be used on the interface and stored in the memcache server of the ground station, in order to be acessible when calculating their image position.

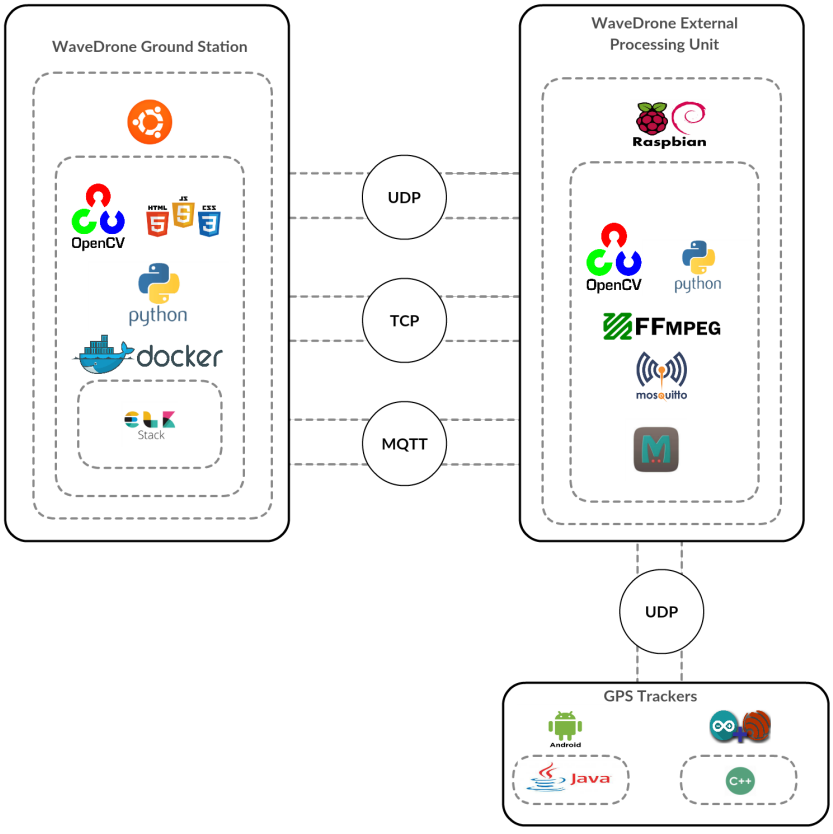


Figure 32: Workflow

5.9 ELK

We used ELK has a log repository. Data was sent through a UDP Socket which logstash processed and stored in InfluxDB.
All the data, from every GPS tracker or the drone after being processed in the ground station went through this cycle to be stored. The point was for future work to use kibana for a good log page, capable of providing a map with a history of drone or trackers movement and all information related.
Unfortunately, due to time restrictions this last part was not implemented. Data is stored and shown as text in the kibana page. Visualization models were not configured so currently only text is available.

Chapter 6

6 Conclusion and future work

6.1 Summary

This report starts by explaining the context of the project and its ambition. A full specification of the hardware used then follows, with attention to detail about the main components of this project and how they connect. A state of the art specification is done, specifically about projects from previous years. All the technologies used are specified and given a little context about. The architecture of the project and several models are shown, giving a better perspective about every component and their relations. In the implementation we talk about every different module developed, how and why it was done that way. Both hardware and software are discussed, and the logic behind it.

6.2 Main Results

Looking at the initial proposal, we have the notion it was really ambitious and that this project was a prototype. Many of the asked features could not be tested due to restraints in testing the drone. However, the software developed is ready for most of these tests and the controlled part (video, location tracking) is implemented correctly and working flawlessly. The drone part was developed and, while communication is established and most of the commands are developed, only information is tested. Flight commands are ready but couldn't yet be tested due to rotors problems on the drone.
The system is capable of providing the referee with a video of the race where every boat is identified, helping him this way make better decisions.
Buoys can also be seen closely through the video and checked if there is any foul play when there are boats crossing.

6.3 Future work

Currently we stand with two main issues that would be the main key for future work. Communication range is something that should be worked on, either by finding a new solution other than Wi-Fi or adapting the current solution with a more potent transmitter. Then tests, many tests. We were unfortunately stopped by some rotors problems that made us not be capable of doing autonomous tests with the drone. BY

fixing this rotor problems our system should be capable of performing. However it requires testing and adjustments to both the commands and calculations.

References

[1] PiCamera, http://picamera.readthedocs.io/en/release-1.13/fov.html#, Retrieved 05/07/2018.

[2] DMA, https://en.wikipedia.org/wiki/Direct_memory_access, Retrieved 05/07/2018.

[3] SoC, https://en.wikipedia.org/wiki/System_on_a_chip, Retrieved 05/07/2018.

[4] PAL, https://en.wikipedia.org/wiki/PAL, Retrieved 05/07/2018.

[5] PiCameraHW,https://www.raspberrypi.org/documentation/hardware/camera/, Retrieved 05/07/2018.

[6] FPVCamera,https://www.banggood.com/pt/Eachine-1000TVL-13-CCD-110-Degree-2_8mm-Lens-Mini-FPV-Camera-NTSC-PAL-Switchable-p-1053340.html?cur_warehouse=CN, Retrieved 05/07/2018.

[7] Calculator, https://www.ecalc.ch/xcoptercalc.php, Retrieved 05/07/2018.

[8] GPS Mount, https://www.thingiverse.com/thing:2343649, Retrieved 05/07/2018.

[9] RPI Case, https://www.thingiverse.com/thing:922740, Retrieved 05/07/2018.

[10] Geo Distance, https://github.com/ulope/geopy/blob/master/geopy/distance.py, Retrieved 05/07/2018.

[11] MSP-V2, https://github.com/iNavFlight/inav/wiki/MSP-V2, Retrieved 05/07/2018.

[12] MSP Navigation, https://github.com/iNavFlight/inav/wiki/MSP-Navigation-Messages, Retrieved 05/07/2018.

[13] Python Construct, http://construct.readthedocs.io/en/latest/, Retrieved 05/07/2018.

[14] Dronin, http://dronin.org/doxygen/flight/html/index.html, Retrieved 05/07/2018.

[15] MSP Java, https://gist.github.com/treymarc/988b25f12305682648c3, Retrieved 05/07/2018.

[16] iNav Missions, https://github.com/iNavFlight/inav/wiki/iNavFlight-Missions, Retrieved 05/07/2018.

[17] MSP NAV protocol, https://docs.google.com/document/d/16ZfS_qwc-rJeA7N5TxODA6wtgxl6HdGgaz-jE3lHBWs/edit, Retrieved 05/07/2018.

, Retrieved 05/07/2018.

[18] Sympy, http://www.sympy.org/pt/index.html, Retrieved 05/07/2018.

Appendices

Appendix A

Installing the ground station module in a linux machine

```
1       . Docker CE
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg
| sudo apt-key add

$ sudo apt-key fingerprint 0EBFCD88

$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce

2       ELK STACK

$ cd PROJECT/GroundStation/ELK
$ sudo docker-compose up

3       OFFLINE MAP

$ cd PROJECT/GroundStation/OfflineMap
$ ./ deploy_mbtiles_map.sh

4       GroundStation Servives

$ cd PROJECT/GroundStation/Services
$ ./gs_initialize.sh
```

5 Starting all

```
$ cd PROJECT/GroundStation
$ ./start_all.sh
```

Appendix B

Installing the external processing unit:

There is a configuration file to change the parameters like the IP of the ground-station or the video parameters:

```
#Main Drone configurations
[DEFAULT]
GroundStation_IP = 192.168.1.65

#Settings for the video capture, this is the
resolution and framerate that will be used
for the recordings and is the resolution at which video is recorded
#No other resolution can be higher than this one
because all the other video feeds are derived from this one
#The sensor mode corresponds to the mode which the
camera is forced, leave 0 for auto and beware of the
framerate and resolution limits of each mode:
http://picamera.readthedocs.io/en/release-1.13/fov.html#sensor-modes
[video-main]
Framerate = 15
Width = 1280
Height = 720
Bitrate = 3
Sensor_Mode = 2

#Settings for the live video feed
[video-live]
Width = 854
Height = 480
Bitrate = 0.5
Port = 10000

#Settings for the launch recording feed
[video-recording-launch]
Width = 1280
Height = 720
Bitrate = 5
DataReadingsPerSecond = 1

#Settings for the opencv processing
```

```
[image-processing]
Width = 1280
Height = 720

$ cd PiDrone
$ python3 DroneMain.py
```

Appendix C

MSP_V2 Messages exchanged with the FC

Obtaining the drone's inclination and heading data:

A message with function parameter 108 with empty payload is sent to the controller:

```
"function" / 10
"payloadSize" / 0
"payload" / [ ]
```

The message returned by the FC:

```
"function" / 108
"payloadSize" /  6
"payload" /          "angx" / Little Endian 2 bytes
                     "angy" / Little Endian 2 bytes
                     "heading" / Little Endian 2 bytes
```

angx : Drone's x inclination
angy : Drone's y inclination
heading : Drone's current heading

Obtaining the drone's altitude and vario data:

A message with function parameter 109 with empty payload is sent to the controller:

```
"function" / 109
"payloadSize" / 0
"payload" / [ ]
```

The message returned by the FC:

```
"function" / 109
"payloadSize" /  6
"payload" /          "alt" / Little Endian  4 bytes
                     "vario" / Little Endian 2 bytes
```

```
alt : Drone's current altitude

Obtaining the drone's gps data

A message with function parameter 106 with empty
payload is sent to the controller:

    "function" / 106
    "payloadSize" / 0
    "payload" / [ ]

The message returned by the FC:

    "function" / 106
    "payloadSize" /
    "payload" /          "fix" / 1 byte
                         "num_sat" / 1 byte
                         "coord_lat" / Little Endian  4 bytes
                         "coord_lon" / Little Endian  4 bytes
                         "altitude" / Little Endian  2 bytes
                         "speed" / 1 byte
                         "ground_course" / 1 byte
To update the drone's pos_hold defined waypoint
and move the drone to said destination:

A message with function parameter 209 and
parameterized payload is sent to the controller:

    "function" / 209
    "payloadSize" / 0
    "payload" /  "wp_no" / 1 byte
                 "action" / 1 byte
                 "lat" / Little Endian  4
                 "lon" / Little Endian  4
                 "alt_hold" / Little Endian 4
                 "heading" / Little Endian 2
                 "p2" / Little Endian 2
                 "p3" / Little Endian 2
```

```
                "nav_flag" / 1 byte

    wp_no : 255
    action : 1
    lat : target latitude
    lon : target longitude
    alt_hold : altitude
    heading : drone's target heading
    p2 : 0
    p3 : 0
    nav_flag : 0
```

The FC returns a mirror message, corresponding to the message sent.

Appendix D
Format of the messages:

Over UDP Socket from GPs trackers:

```
{"deviceId":"********","degree":"***",
"Long":"******", "Lat":"*******", "module_type":"****"}
```

deviceId is the mac address, module_type says if it is Android or MCU, degree is the orientation, Long and Lat are longitude and latitude respectively

The same format is used to send data over MQTT from the RPI to the ground station and to publish it on the REST. With this format the REST creates a dictionary with all the components.

Over MQTT the drone info:

```
{"type":"drone","degree":"***",
"long":"******", "lat":"*******", "module_type":"droneInfo",
"angx":"****", "angy":"*****","status":"*******" }
```

degree is the orientation, Long and Lat are longitude and latitude, angx and angy are the positioning of the drone and status is what function it is currently performing

Over MQTT the commands to the RPI for video under the topic videoRequest:

```
{"type":"********"}
```

type corresponds to the function to be executed

Over MQTT the commands to the RPI for video under the topic droneCommand:

```
{"type":"********", "module_type_name":"********",
"latitude":"*****", "longitude":"*******",
"module_id":"********"}
```

type corresponds to the function to be executed, module_type_name to the function to be executed, latitude and longitude to coordinates and module_id to the identifier of a component. Some are optional, depending on the command.

Appendix C

Repository:

https://code.ua.pt/git/pi2018_grupo06_monitorizacao-de-regatas-com-drones