Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

FINAL DEGREE PROJECT

TITLE: Design and implementation of advanced location systems for quadrotor drones controlled by STM32

TITULATION: Degree in Aerospace System Engineering

AUTOR: Boadella Aguado, Alejandro

DIRECTOR: Casanella Alonso, Ramon

CODIRECTOR: Royo Chic, Pablo

DATE: September 1, 2024

Títol: Disseny i implementació de sistemes de localització avançats per a drons quadrirotors controlats mitjançant STM32

Autor: Boadella Aguado, Alejandro

Director: Casanella Alonso, Ramon

Codirector: Royo Chic, Pablo

Data: 1 de setembre 2024

Resum

La família de microtroladors STM32 de 32 bits és la més àmpliament utilitzada per a disseny de controladores de drons en l'actualitat. En aquest treball es proposa el disseny i implementació de sistemes de localització avançats que puguin ser incorporats a un dron quadrirotor controlat mitjançant un microcontrolador de la família STM32.

D'una banda, s'estudia la viabilitat de la incorporació i programació dins del codi base del dron d'un sistema de geolocalització i la seva integració en el sistema de control de vol del dron. D'altra banda, s'estudia la incorporació d'un sistema de càmera dual estereoscòpica per a la localització d'objectes, considerant la seva identificació mitjançant intel·ligència artificial.

Opcionalment, es busca dissenyar i implementar l'enviament posterior de les dades de localització a una estació terrestre en la que es puguin implementar funcions addicionals avançades.

Es desenvolupen diferents sistemes de localització i la seva integració en el sistema del quadrirotor per tal d'aconseguir un prototipus final plenament funcional que permet la seva reconfiguració i millora el funció de necessitats futures.

Addicionalment s'estudien teòricament: diversos sistemes de localització, identificació de punts clau en imatges, estimació de profunditat en múltiples imatges, com realitzar una intel·ligència artificial entre d'altres. A més a més s'han realitzat altres estudis teòrics i pràctics com pot ser: vSLAM (Localització i mapeig visual simultani), sistemes de posicionament amb visió en primera persona i test estructurals per al hardware. Així com l'estudi, la comparació i possibles millores d'aquest sistemes. Tot aquest treball s'ha desenvolupat amb diferents softwares com: STMCubeProgrammer (STM32 codificador), Matlab, Arduino (en el STM32), Linux (a la raspberry pi), U-center (GNSS evaluator), etc.

Title: Design and implementation of advanced location systems for quadrotor drones controlled by STM32

Author: Boadella Aguado, Alejandro

Director: Casanella Alonso, Ramon

Codirector: Royo Chic, Pablo

Date: September 1, 2024

Overview

The 32-bit STM32 microcontroller family is the most widely used for drone controller design today. This work proposes the design and implementation of advanced localization systems that can be incorporated into a quadrotor drone controlled by means of a microcontroller of the STM32 family.

On the one hand, the feasibility of incorporating and programming a geolocation system into the drone's base code and its integration into the drone's flight control system is studied. On the other hand, the incorporation of a dual stereoscopic camera system is being studied for the location of objects, considering their identification through artificial intelligence.

Optionally, it seeks to design and implement the subsequent sending of the location data to a ground station where additional advanced functions can be implemented.

Different localization systems are developed and their integration in the quadrirotor system in order to achieve a fully functional final prototype that allows its reconfiguration and improves the function of future needs.

Additionally, the following are theoretically studied: various localization systems, identification of key points in images, depth estimation in multiple images, how to perform artificial intelligence, among others. In addition, other theoretical and practical studies have been carried out such as: vSLAM (Localization and simultaneous visual mapping), positioning systems with first-person vision and structural tests for the hardware. As well as the study, comparison and possible improvements of these systems. All this work has been developed with different software such as: STMCubeProgrammer (STM32 coder), Matlab, Arduino (on the STM32), Linux (on the raspberry pi), U-center (GNSS evaluator), etc.

INDEX

KEYWORDS / NOMENCLATURE

AI	Artificial Intelligence
AP	Access Point
ACF	Aggregated Channel Features
BRIEF	Binary Robust Independent Elementary Features
BVLOS	Beyond Visual Line of Sight
CAN	Controller Area Network
CW	Continuous wave
DNN	Deep Neural Network
FNN	Feedforward Neural Network
FAST	Features From Accelerated Segment Test
GNSS	Global Navigation Satellite System
GRU	Gated Recurrent Unit
GPT	Generative Pre-trained Transformer
IP	Internet Protocol address
KNN	K-nearest Neighbor Network
LAN	Local Area network
LSTM	Long Short-Term Memory
LIDAR	Laser Imaging Detection And Ranging
MLP	Multiple Layer Perceptron
MTI	Moving Target in Indicator
NP	Nondeterministic Polynomial-time
OS	Operating System
ORB	Oriented FAST and Rotated BRIEF
RPAS	Remotely Piloted Aircraft System
RPAV	Remotely Piloted Aircraft Vehicle
RASP	Raspberry Pi
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
RoHS	Restriction of Hazardous Substances
SD	Secure Digital
SSH	Secure Shell
SLAM	Simultaneous Localization And Mapping
SPI	Serial Peripheral Interface
STM32	32-bit STMicroelectronics microcontroller
ToF	Time Of Flight
UAS	Unmanned Aircraft System
UAV	Unmanned Aircraft Vehicle
UART	Universal Asynchronous Receiver / Transmitter
VLOS	Visual Line of Sight
YOLO	You Only Look Once

INTRODUCTION

Since the mid 1800s, drones have been one of the most unknown aerial vehicles. After WWII and especially these days, these vehicles have revolutionized how we live nowadays. Starting from making films to even fertilizing the fields. These systems have been converted into a multitool option for those who can afford it.

With this growth in the current market and the use of relatively modern technologies, these systems are positioned in a favorable position for study.

One of the most common problems to fly a drone is to position the vehicle into an unknown environment. This can lead into a loss of tracking, making the vehicle sometimes fail or not reach its purpose, navigating.

So, this mere research consists in a simple form to track an objective or even in this case the drone. Basically with the use and investigation of new technologies such as RPAV, AI and others.

The control of the drone inflight with localization information using a microcontroller of the STM32 family.

The experimentation on detecting an objective with artificial intelligence, measuring the approximate distance from an UAV using on board stereo cameras, making and approximating a 3D model of the environment. Once located and with GPS UAV position, then sending all this information to a ground station. Where, all this information will be processed and analyzed to localize an objective or the drone in a special screen, a FOV/POV screen (field/point of view).

What’s its main advantage? Instead of seeing a normal screen to view the UAV camera, seeing the location of the objective or the drone through obstacles or walls but from the person's view. An advantage for military (tanks and artillery), rescue on high ground/dense zones and large architecture applications.

Motivation

What makes a rescue or a military operation succeed in part? Time.
Waste of time when the rescuers or the police are trying to find in what direction or position the objective is, shall be reduced at any cost. In order to take actions as fast as possible, X-ray vision can be useful in these situations. However, X-ray vision like in the cartoons does not exist as far as i'm concerned, but 3D modeling and positioning, yes.

Objectives

This project have several objectives:

- Study the feasibility of incorporation and the programming of a geolocalization system into a drone's base code and the integration into the drone's flight control system (STM32 / 32-bit STMicroelectronics microcontroller).
- Develop and integrate different localization systems.
- Identification and positioning of an object using AI and stereo camera system.
- Make a fully functional prototype that allows its reconfiguration and improvement for future needs.
- Optionally: Develop a ground station to manage all this data.

Structure

The structure of this project consists of a set of chapters, the first chapter is about the frame or drone creation for the localization system. And the latter chapters are about how the system has been developed with traces of theory if they are needed. Furthermore annexes are incorporated !! I highly encourage the reader to watch them.

Programing language

The main programming language used in this project has been Matlab, basically due available resources, back-ground learning and simple multi-dispositive connection and data extraction. Other languages like Python can be used to do this project even with better performance with less delay; however, specially the use of multiple-dispositive and data extraction from the sensors of these dispositives (i.e: mobile phone) can make the whole project too extensive and more complex. To programme the stm32, it has used arduino code.

Schedule

The project has been carried out following a schedule shown below to meet all requirements. It started on January 15, 2024 and ended on September 1, 2024. The image doesn't want to give data, it only shows a mere planning structure detailed by week.

	Weeks			Objective	State	Description/Others
1	Exams week	2024-01-15	2024-01-21	Start the TFG, find resources, etc.	Achieved	
2	No school	2024-01-22	2024-01-28	Take a look at Simulink and think how to proceed	Achieved	1 day of delay in Simulink course
3	No school	2024-01-29	2024-02-04	Start theory about IA/stereo camera	Achieved	I made a rough summary of length measurement methods before the start of the project.
4	No school	2024-02-05	2024-02-11	Have a basic stereo camera model deployed in the Rasp	Not achieved	Due a problem of hardware (i need a cable), I cant do the objective. So I will do theory of stereo/AI. The following objectives will be redistributed.
5	School	2024-02-12	2024-02-18	Have a basic model and do theory	Achieved	I already have connection between computer and rasp, receiving camera data via Wifi connection.
6	School	2024-02-19	2024-02-25	Have a basic model and do theory	Achieved	Bus can connection tested on arduino uno, tested GPS sensor, tested different methods in order to receive image data. Problem: STM32 pins and connection and glue on stereo camera.
7	School	2024-02-26	2024-03-03	Think about bus connection and finish basic implementation stereo	Half-achieved	Problems with Rasp-Matlab comunication due versions and lack of support by Matlab, impossible do bus can, so SPI bus has been done. UDP/TCP via Python communication has been done, Matlab has been chosen forwarding the mobile support. Calibration and disparity maps.
8	School	2024-03-04	2024-03-10	Bus connection and theory	Achieved	Basic 1 frame 3D scanner with Matlab. The main part of theory has been done, another calibration from 6 meters distance 120 images. The following objectives will be changed by multiple reasons, feasible objectives.
9	School	2024-03-11	2024-03-17	Connect GPS and transmit data	Half-achieved	Problem with GPS cable, trying to find a feasible one. I found useful GPS code to use in practise. Mobile station connection and data extraction achieved. Angle vision of all sets has been measured.
10	School	2024-03-18	2024-03-24	Start thinking about AI and mobile connection	Half-achieved	Problems with different kinds of cables for the GPS, finally i found one. Implementation code and theory has been done about ORB-SLAM in a brief way. Test and improvement of SLAM code needs to be done in the future. Angle vision measured of one camera has to be measured again due the lack of rectified images.
11	No school	2024-03-25	2024-03-31	Start thinking about AI and mobile connection	Achieved	Preparation IA dataset for pretrained YOLOv4 (rower images) +2000 images, Compass -->"fake" (ant the honeywell version, is the chinese version), achieved a basic SLAM version --> needs more frames per second or move slowly the uav (tracking loss) or record in the rasp that can be pretty useful, preparation of a general arduino code to pass data (how to convert data for SPI bus).
12	Exams week	2024-04-01	2024-04-07	Start thinking about AI and mobile connection	Achieved	GPS baud rate is double than others. 2860 images are labeled for AI, Problems with YOLO (YOLO doesnt work very well, however ACF goes perfect), both explained and tested (AI completed).
13	Exams week	2024-04-08	2024-04-14	Do AI and FOV system	Achieved	3D structure for the uas, passing data to bytes and preparation of the overall/general code. Problem: height not well defined using GPS. Need to use barometric data, study in a latter part of the project.
14	School	2024-04-15	2024-04-21	Do AI and FOV system	Half-achieved	New structure and structural analysis (problem with integrity --> new design proposed/stronger but heavier), making the general FOV system (system of equations) and bus testing of the overall system. Altitude data achieved; however, with delay (not too much, less than 1ms).
15	School	2024-04-22	2024-04-28	Ground station model requested	Half-achieved	Bus works with multiple data in the general code combining Matlab and Arduino (high data frequencies doesnt works, high rate of errors >4Mhz). General code needs to be tested outside, in an open field. Multiple small works on the whole TFG are carried out. New structure will be received next week.
16	School	2024-04-29	2024-05-05	Support sesion	Half-achieved	New structure implemented. Problems with bus data type, resolved. Bus sending and receiving completed, overall operable, needs more accuracy on its data. Error on GPS extensor structure, new one for better accuracy for compass.
17	School	2024-05-06	2024-05-12	FOV system and STM32 Implementation	Half-achieved	Presentation quite finished. I2C resilence bus tested with GY-521. Test bus and gps with arduino uno at 3,3V, works well. Study how to implement STM32. BIG problem: SPI not compatible with STM32. Problems with STM32, seeking how to resolve them.
18	School	2024-05-13	2024-05-19	FOV system STM32 Implementation	Not achieved	Problems with STM32, partially resolved, a test will be done with a Nucleo-64. Other connections with STM32 will be tested. Tested code has been comented.
19	School	2024-05-20	2024-05-26	FOV system STM32 Implementation	Achieved	Connexion with STM32 resolved, in order to upload code. Creation of wifi long range area network. Bus with stm32 not achieved, even with matlab or python (SPI or UART), seeking to implement via arduino inside raspberry pi. Bus achieveed using arduino and command line via USB, needs to be implemented correctly.
20	School	2024-05-27	2024-06-02	FOV system STM32 Implementation	Half-achieved	Implemented the final serial bus. Determining object distance succesfull in images with relative error; however, AI needs modifications in the dataset for improvement or use same background as the dataset. Creation of the code in order to implement STM32 ongoing.
21	Exams week	2024-06-03	2024-06-09	FOV system STM32 Implementation	Half-achieved	GPS STM32 and mobile precision error estimation done, and object distance detection outside done. Problem camera recalibration (probably heat could affect camera)--> delay. Only need to test FOV system, everything is fine and a serie of final test. LAN implemented.
22	Exams week	2024-06-10	2024-06-16	Final Test	Achieved	Basics test done, multiple slam test done. Study of the feasibility of the implementation partially done. Last test to do FOV, the following week all test will be completed. Start searchin ghow to make a drone with arduino.
23	Exams week	2024-06-17	2024-06-23	Final Test FOV + Construction of a Drone	Achieved	FOV test done, starting drone construction
24	No school	2024-06-24	2024-06-30	Contruction of a drone + Modification of the code.	Achieved	Seeking for materials and partially the basic drone code done. Finishing drone contruction and starting to test the code within the drone.
25	No school	2024-07-01	2024-07-07	Drone coding	Achieved	Same as last week.
26	No school	2024-07-08	2024-07-14	Drone coding	Achieved	Same as last week.
27	No school	2024-07-15	2024-07-21	Drone Test	Half-achieved	First tests, use of GPS modifing EEPROM (big problem but resolved).
28	No school	2024-07-22	2024-07-28	Drone Test	Achieved	Test mode stable perfect, GPS needs to be improved,
29	No school	2024-07-29	2024-08-04		none	
30	No school	2024-08-05	2024-08-11		none	
31	No school	2024-08-12	2024-08-18		none	
32	No school	2024-08-19	2024-08-25		none	
33	No school	2024-08-26	2024-09-01		none	
34	No school	2024-09-02	2024-09-05		none	

Fig. 0.1 Schedule

CHAPTER 1. DRONE CREATION

Before the creation of the localization systems it is necessary to develop a frame where it can be mounted, such as a drone controlled by a STM32.

1.1. Drone Materials

The Drone materials are divided into the main ones and the secondary ones, both are important.

1.1.1. Main Materials

- Adafruit Feather STM32F405:
  The main microcontroller of the drone, it receives, processes and delivers any type of information or signal in order to make the drone fly communicating with multiple dispositives and sensors.

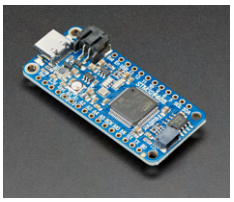


Fig. 1.1 STM32F405

- Frame DJI F450:
  Main structure of the drone.

Fig. 1.2 Frame DJI F450

- 4 Electronic Speed Controller E305 Lite DJI:
  Controls the speed and delivers the power to the motors.

Fig. 1.3 E305 Lite DJI

- 4 Brushless motors 2312E 960KV + 4 propellers:

Fig. 1.4 Brushless motors 2312E Kit

- MPU-6050:
  It is the IMU of the drone. It gives gyroscope and acceleration data.

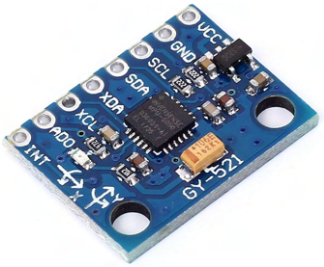


Fig. 1.5 MPU-6050

- Mini Ublox NEO-M8N GPS module kit with HMC5883:

Fig. 1.6 NEO-M8N GPS Module (GPS+Magnetometer)

- USB-C Cable:
  It is in order to power the STM32 or any other dispositive with this type of connection.

Fig. 1.7 USB-C + USB-A

- UBEC Cable:
  Used to power sensors with 5V as input.

Fig. 1.8 UBEC Cable

- Battery Lipo 3S 2200mA:
  It could be any type of 3-4 S Lipo battery, used to power the drone.

Fig. 1.9 Battery Lipo 3S 2200mA

- Landing Gear for drone:
  With the purpose of adding more space and better landing.

Fig. 1.10 Landing Gear for F450 DJI

- FS-I6X + receiver:
  To control the drone.

Fig. 1.11 FS-I6X

- Circuit ELEGOO Plate and connectors:
  It will enable us to make circuits and connect any sensor.

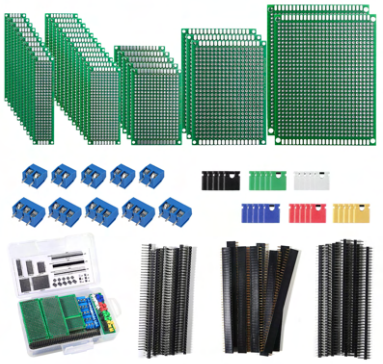


Fig. 1.12 ELEGOO Kit

1.1.2. Secondary Materials

These materials no photo will be shown as being too common.

- 1M Ohm resistance
- 100K Ohm resistance
- Multiple XT60 cable with connectors
- Multiple cables for connections (in my case i used ethernet cable leftovers)
- Tape
- Plastic Flanges
- 16 M3*8 Screws + 24 M2.5*6 Screws
- Heat shrinkable plastic for connections

1.2. Drone Creation

1.2.1. Drone circuit

The creation of the circuit has been customized in order to apply multiple sensors and dispositives. All the circuit connections are included in the annex, at the end of this document.

It has developed 2 circuits. One with the raspberry pi (see Fig 1.13) to use it with the location system, and the other one without the raspberry pi (see Fig 1.14), only the drone itself.

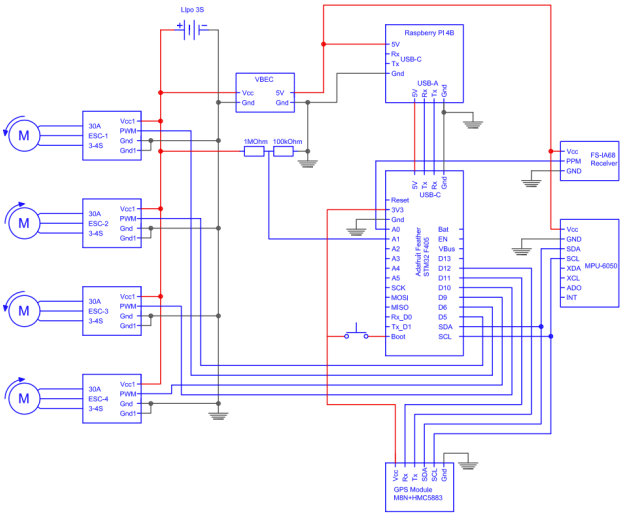


Fig. 1.13 Drone circuit + Raspberry Pi

As seen in the image the GPS is powered at 3,3V while other sensors are at 5V. The STM32 inside has a reset button. In order to implement the code, while uploading press the button outside the STM32.

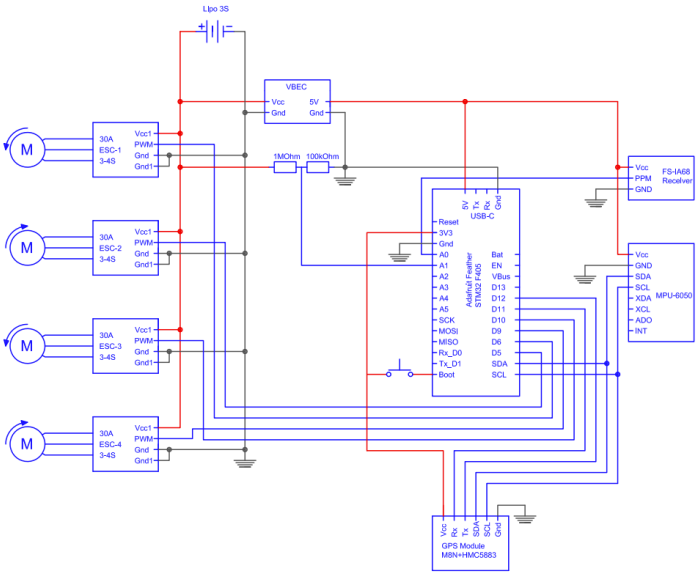


Fig. 1.14 Drone circuit

The 2 implemented resistances are a voltage divider in order to measure the battery voltage and consume the least power from them. If the led onboard is in red, the drone is operable.

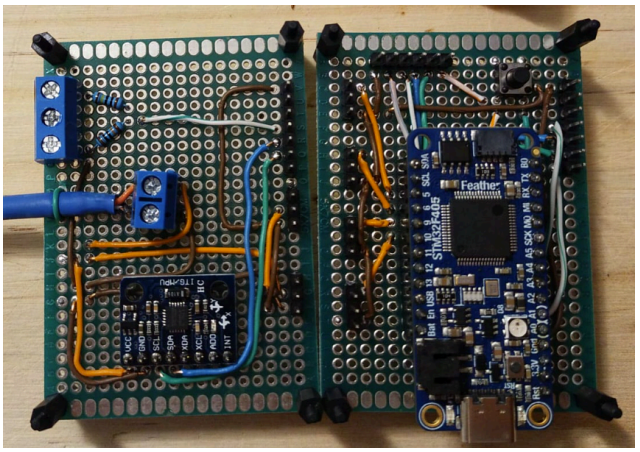


Fig. 1.15 Drone circuit board

1.2.2. Drone Software and others

The 2 main sources of code that have been used for this project are [1] and [2], especially Brokking. In this section not all concepts will be explained in detail; however, some key aspects shall be explained. If the reader wants a deeper knowledge I encourage the reader to look at the references.

Due to time factors the GPS Mode in the software has been applied but it doesn't actually work. It only needs a few changes in order to work. Other modes like Stabilize work properly. Stabilize battery mode depending on the battery, the user must read the data and map the values.

1.2.2.1. Code Setup

This section of the code is destined to prepare the drone to fly. In this set up there are 3 modes, each mode will be selected depending on the potentiometer state in the radio seen in Fig 1.16.

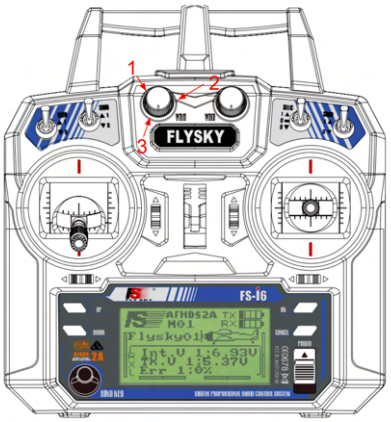


Fig. 1.16 Radio Setup channels

When the user powers up the drone it will initiate the led, the esc output and the radio. After that if the radio is in position 1, the ESC calibration mode will actuate giving the output the same as the throttle position is. If the radio is position 3, the compass calibration will be active, in this mode the user shall move the drone in all axes to obtain the maximum and minimum values from the compass to calibrate it. To stop the calibration move down the pitch joystick (in case of inverted pitch), otherwise move up the joystick. After that GPS mode will be enabled to use. In case of having the radio in position 2, no calibration will happen and the setup will continue. After the calibration or not, finally the imu will be calibrated automatically. So lay the drone onto the ground to make the imu calibration possible.

1.2.2.2. Code Loop

The code is dedicated to obtain, calculate and process everything within a 4 ms loop.

Basically the loop works in this way: Read the radio, then select the mode. Read all the sensors and then process them. With all the data modified from the radio and from the sensors, pass it to the margin section and PID section. Finally the output of the PID will be modified depending on the mode selected and send the opportune signal to the esc. And then repeat the same process.

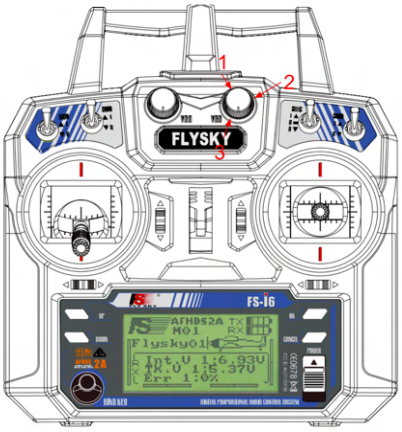


Fig. 1.17 Radio Fly Mode channels

There are different modes, the only mode that doesn't work properly is the GPS mode (position 3) due time factors I wasn't able to test it correctly. The Stabilize mode is in position 1 and Stabilize with battery correction is in position 2, Fig 1.17.

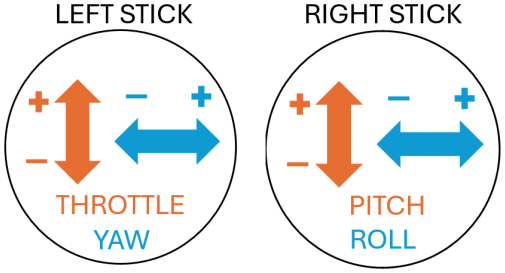


Fig. 1.18 Radio joystick (mode 2)

To control the drone in Fig 1.18 the reader can see how it moves the drone. The pitch is inverted, so I recommend inverting it on the radio as well. To start the drone move the left stick to the lower left part, and to disable the drone move that stick to the lower right part.

1.2.2.3. Receiver, MPU-6050 and signals

- Flysky receiver:
  The receiver of the radio will deliver a PPM signal. An interruption attachment will take rolethat every time the pulse changes a measure is done. Every time the pulses fall it will start a counter, the time between falls will be the channel signal and of course mapped into values from 1000 to 2000 microseconds. The signal is consecutive, delivering a minimum of 8 channels (6 in real + 2 phantom channels); if the user wants to expand the channels, these extra channels will disappear.

- MPU-6050:
  This sensor delivers gyro and acceleration data. To not have errors the X-arrow shall point the same direction as the heading of the drone, and the Y-arrow the left side of the drone, like in Fig 1.19. Mainly the gyro data will be used, though acceleration data will support the gyro to not have big noise/drift with a filter.

- ESC and motors:
  The motors shall be positioned as in Fig 1.19. To move these brushless motors, a PWM will be continuously delivered from the STM32, changing it every time the drone requires it to make a movement. These PWM signals are from 1000 to 2000 microseconds.

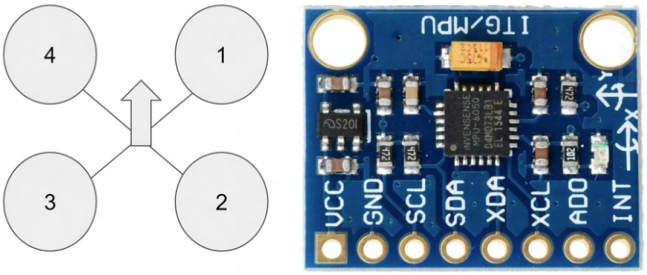


Fig. 1.19 ESC connections and MPU-6050 position

1.2.2.4. PID

The proportional–integral–derivative controller is one of the most complex parts in the creation of a drone (basic structure Eq. 1.1). Its objective is to control the drone as the user wants.

$$u(t) = K_e e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \quad \textbf{(1.1)}$$

To develop this control loop, a few things need to be configured. The outputs of the gyro angular rate and the radio shall be the same. Once with that data, if we subtract the receiver from the gyro data, we will obtain the error. With this error it is possible to generate an output, there are 3 parts of the PID. The proportional (Eq. 1.2), the output is proportional to the error. The integral part will sum all instantaneous errors over time in order to correct the output (Eq. 1.3). And finally the derivative part will correct the output with the rate of change as it varies over time (Eq. 1.4). The sum of all 3 parts is the PID.

$$P_{output} = (gyro - receiver) \cdot P_{gain} \quad \textbf{(1.2)}$$

$$I_{output} = I_{output} + (gyro - receiver) \cdot I_{gain} \quad \textbf{(1.3)}$$

$$D_{output} = (gyro - receiver - gyro_{previous} - receiver_{previous}) \cdot D_{gain} \quad \textbf{(1.4)}$$

The constants of the PID can be obtained easily following Brokking explanations [1], no further explanation will be given.

The output delivered to the esc will be the following, table 1.1.

Table 1.1. ESCs output

```
esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw;
esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw;
esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw;
esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw;
```

The battery correction mode increments the throttle of the drone if it's necessary in a proportional way.

Furthermore in the code presented there is a first attitude part, this is why the radio reception values need to have some dead bands and modification in its signal with respect to the drone angle.

1.2.2.5. Future Improvements

The drone is fully operational; however, there are a few things to improve:

- Enable more channels modifying the radio.
- Make the GPS mode work.
- Enable an interruption attachment to a special channel to disable the drone or do an emergency landing (needs more channels).
- Create a special structure to make the drone more impact resistance.
- Try to implement a return to home automatic procedure.

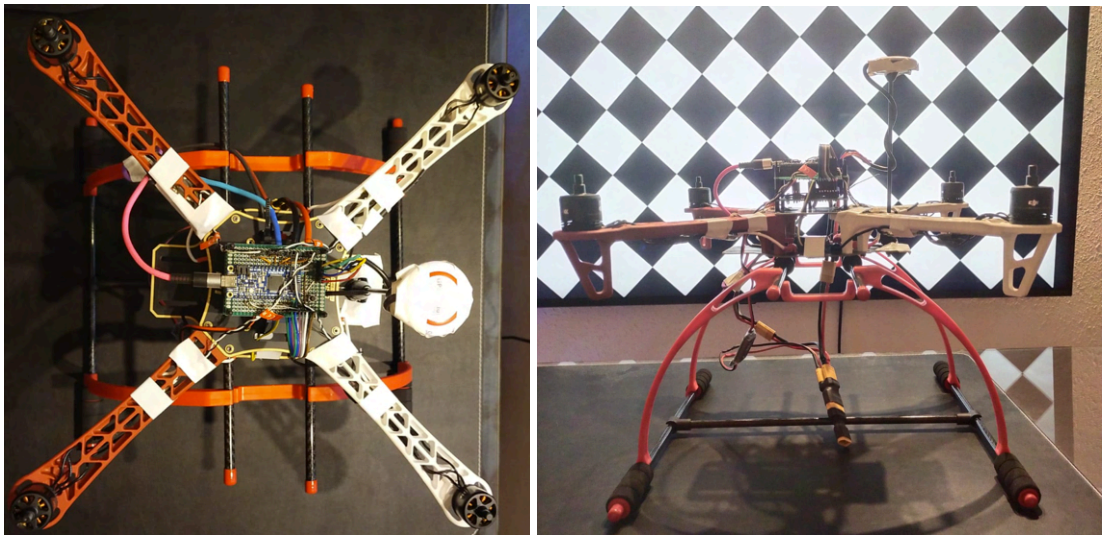


Fig. 1.20 Drone structure

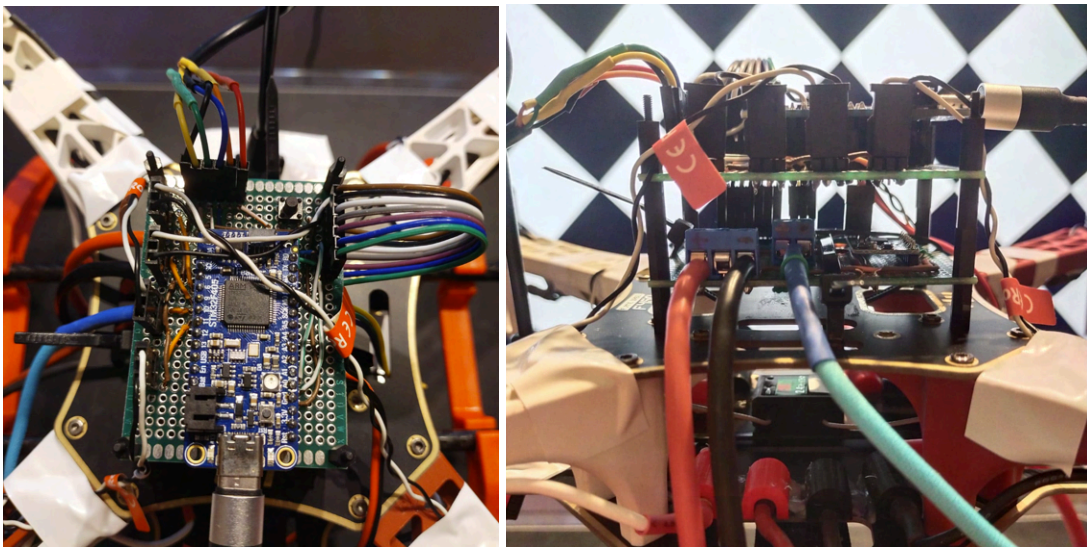


Fig. 1.21 Internal connections

CHAPTER 2. STEREO DEPLOYMENT ON A RASP AND SLAM

Once with the drone created, the localization system starts from now on.

2.1. Physical connection

The objective is to set up a raspberry pi and receive data from other sensors and the stereo camera. And then transmit all this data to the ground station.

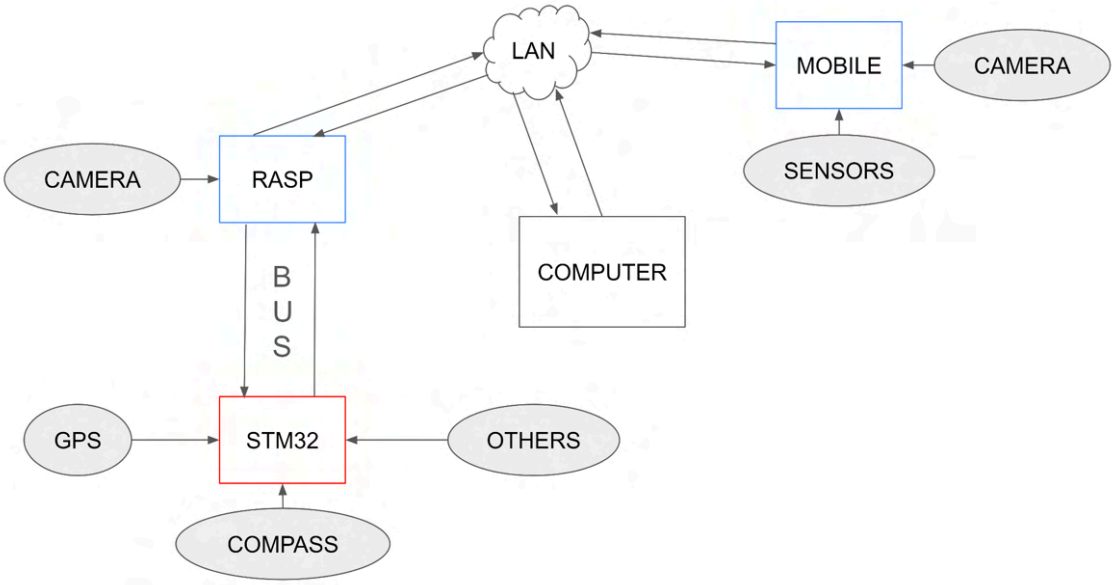


Fig. 2.1 Objective structure of connections

In this part as a first approach, it is needed 3 objects: a Raspberry Pi 4B or any other version, usb antenna to connect the raspberry pi to the LAN (Local Area network) and the hat with the stereo camera. For more information about this stereo camera hat, please see the annex 1.

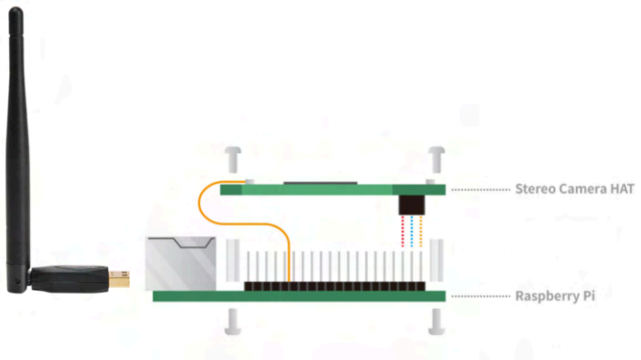


Fig. 2.2 Objective structure of connections

2.2. Software connection

Most of the problems come from here. The two basic software programmes tested in the development of this project are Python and Matlab. Although Python can be more useful and superior in the vast majority of its uses, Matlab has been chosen as the main programming language due to its simplicity, learning background, mobile connection and others. Furthermore Matlab can use python code as an alternative. Any programme used will be annexed at the end of this project.

Once everything is connected, it is necessary to set up the raspberry pi. With Raspberry Imager download in a SD card any 32-bit OS Raspbian, if it is a Bookworm OS it won't work. Then connect this SD card to the raspberry pi and connect to any LAN network. Any connection pins and camera shall be enabled.

To set up the stereo camera there is a guide in the Arducam webpage, though i tried many times, the stereo camera model is discontinued in the web page and the code available sometimes makes errors. Probably using Python or any other program can make it possible to configure the camera.

To set up Matlab inside the raspberry pi, it is necessary to download Matlab and the Raspberry package on the computer. Then inside the rasp connected to the LAN, with SSH connection enabled, make the following steps:

1. Enter command window
2. Put: sudo nano /etc/sudoers
3. Include: <user name> ALL=(ALL:ALL) NOPASSWD:ALL
4. Save (Ctrl-x)

These commands are done in order to activate simulink and have no problems with all the data. But it can be skippable.

In the computer, open the raspberry pi matlab package, and follow the instructions, it can last 30 min depending on the specifications. Moreover, to connect the Raspberry it will request the ip address, user and password. To get an ip address, put in the command line in the raspberry pi "ip address" and look for IPv4 results.

Somehow this procedure can get errors, like not detecting the can bus, not detecting the raspberry pi version and others.

2.3. Camera calibration results

With the raspberry pi configured, we can access the raspberry from matlab directly. The camera of the raspberry pi in matlab configuration, its max resolution is 1080p, though we can execute from matlab a python script at the raspberry to get a better resolution image and then get the file from matlab.

So in order to calibrate the camera image, in other words, get the intrinsic, rotational and translation matrices, we need to take different snapshots from the 2 cameras with a black-white/chessboard pattern. Then open Matlab stereo calibration app, preferable on a desktop file, insert all these images. Automatically it will discard those unuseful, if any image has a high error, eliminate the pair to have a better pixel error. Then export parameters to the main program.

Fig. 2.3 Calibration snapshots (rectified left, reprojection right)

A few pieces of advice, the pattern shall be big, it shall cover at least 20% of the image and it shouldn't be too much inclined, less than 45 degrees from the camera. However, it is not obligatory like in Fig 2.3.

For long distance stereo resolutions some people used extra big patterns, UAV, satellite points for calibration with special code. For many reasons I can't assume that type of calibration, so the best approach is to find a big paper like a din A1.

The set should be at least 20 pairs of images, in this project 100 pairs have been analyzed (200 images in total). Once calculated, the reprojection mean error shall be the lowest as possible with a minimum set of images. Eliminating the pairs of images with higher errors and recalculation, a good mean error will be less 0.25 for example.

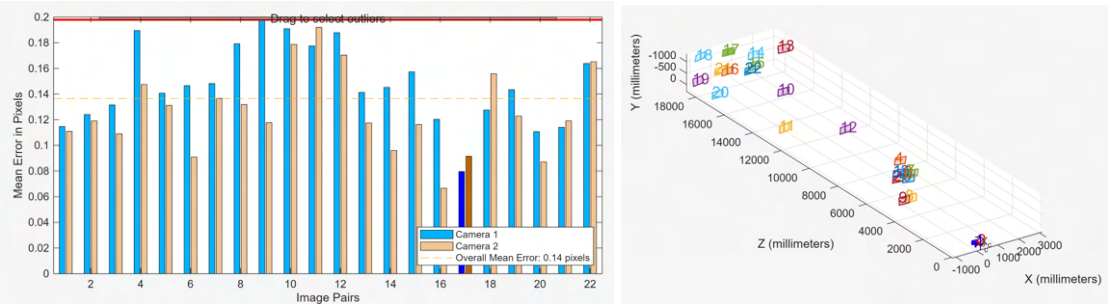


Fig. 2.4 Reprojection error of the set left, Pattern position from the camera right

Though the vast majority of the measures taken with the pattern were very accurate, some others, specially at larger distances, had errors in measurement. Patterns located at 12 m were around 11.5-12 m, but patterns at 14-16 m were 17-18 m in distance. These errors can be due to baseline, focal length and pixel definition.

2.4. Camera distance results

Once calibrated, it is as easy as having these 2 images rectified by the stereo parameters of the calibration. With rectified images take a disparity plot. Then with that plot it is possible to measure and create 3D scenarios.

As seen in Fig 2.5, it is possible to observe disparity changes between chairs, disparity changes at closer distances are more noticeable due baseline. However, these chairs are around +7 m, +12 m, +16 m from the camera, so the results are promising despite having relatively cheap hardware for these purposes. Furthermore the sensitivity (≈patch size) of the disparity can be changed.

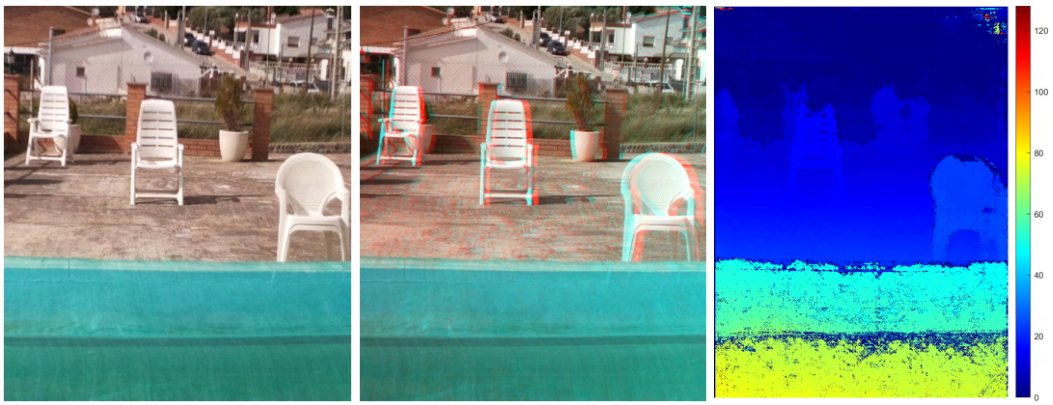


Fig. 2.5 Rectified image (Left), Both rectified images (Center), Disparity (Right)

2.5. SLAM

Simultaneous localization and mapping (SLAM), is a computational algorithm that can build a map of the environment while tracking the agent or vehicle. Used in many applications such as unmanned vehicles, exploration probes and any other activity that needs the mapping of the environment. In these systems several sensors are used, the most popular being the lidar, and many others like ToF cameras, normal cameras, microphones, acoustic sensors, etc.

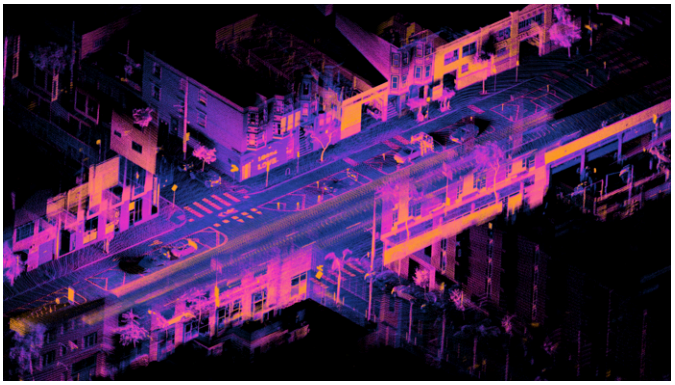


Fig. 2.6 Lidar SLAM point cloud [3]

For mere curiosity, the mathematical probabilistic expression using Bayes rule can be seen in (Eq. 2.1), considering: x_t agent state, m_t environment, u_t control, t time in discrete system and o_t that means observation. [3]

$$P(m_t|x_t, o_{1:t}, u_{1:t}) = \sum_{x_t}\sum_{m_t} P(m_t|x_t, m_{t-1}, o_t, u_{1:t})P(m_{t-1}, x_t|o_{1:t-1}, m_{t-1}, u_{1:t}) \quad \textbf{(2.1)}$$

2.5.1. Types of algorithms

SLAM is not a unique technique, there's extensive research in these techniques. I will only explain a sum of a few of these algorithms. For more information, see [4].

2.5.1.1. Filter based

- Kalman Filter:
  In 2 steps, prediction→ update. Assuming linearity with Gaussian noise, it can predict the state of a dynamic system weighed on its uncertainty.

- Extended Kalman Filter (EKF):
  An improvement of the Kalman Filter, using first-order Taylor expansion. Used on nonlinear systems, but also it can produce errors in its predicting.

- Unscented Kalman Filter (UKF):
  Despite being highly computationally costly, it is used to estimate nonlinear systems using unscented transformation and nonlinear transformation.

- Particle Filter:
  This filter is composed of a set of particles being each one as possible state, the more weight of the particle, the more accurate prediction. Undertaking nonlinear complex problems, the dimensionality, the divergence and the complexity can cause high computational cost.

- etc

2.5.1.2. Graph based

These algorithms are based on a graph problem where nodes represent poses while landmarks and edges are constraints.

- Square Root SAM:
  The optimization of a matrix (row nodes, columns connections) of the data extracted from landmarks and poses. Then this matrix will be factored using the square root SAM algorithm to estimate. It optimizes the data at the end of the data recording.

- Incremental Smoothing and Mapping (ISAM):
  Similar as the Square Root SAM, though it can have incremental updates each time it receives new data, reducing computational complexity and memory.

- General Framework for Graph Optimization (g2o):
  Basically an open source in C++ of all this kind of stuff, using Square Root SAM, ISAM and others.

- etc

2.5.1.3. Deep learning based

Pretty new field of study, not too much information about it.

- RatSLAM:
  The use of a neural network with an hippocampal model based on a rat hippocampus, using odometry data. Useful under certain conditions.

- LIFT-SLAM:
  Using CNN networks. Illuminations, blurry images and low texture can affect estimation.

- EnvSLAM:
  Lightweight algorithm having a trade-off of accuracy and efficiency. Not all situations are feasible for this type of algorithm.

- etc

2.5.2. ORB-SLAM

Oriented FAST and Rotated BRIEF (ORB) SLAM. This type of visual algorithm (vSLAM) is one of the most popular in this world due its efficiency and easy implementation. It can be implemented in monocular, stereo and RGB cameras set ups. This algorithm has been chosen in this project due its properties. For more information see GPS-SLAM [5], ORB-SLAM [6] and ORB-SLAM2 [7].

In the case of this project, the reliable estimations using this stereo system and algorithm are around 40 times the baseline, in other words, around 3.2 m from the used camera.

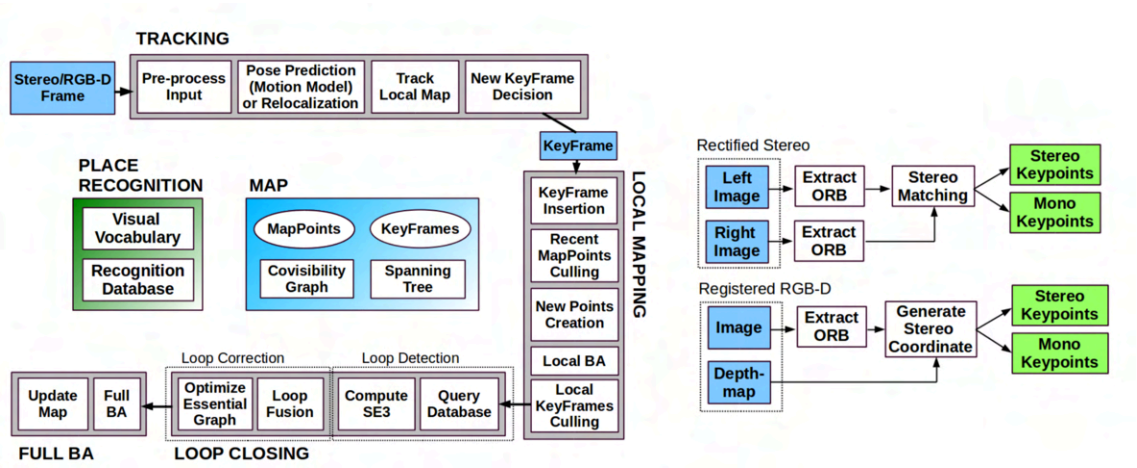


Fig. 2.7 ORB-SLAM analysis scheme [7]

As a clarification, BA means bundle adjustment of the point cloud and ICP means iterative closest point (i.e. [8], [9]). Both algorithms are used for map reconstruction.

This SLAM algorithm has many points to study and it can be too extensive to explain in this project but at least the most important key points are the ORB feature detection and BA (bundle adjustment).

About the idea of SLAM, the start of the local mapping is pretty close to the explanation of the General Stereo Analysis (see annexes), the Homography and other things, but of course with changes and a step forward in complexity.

ORB way better than SURF (Speeded-up Robust Features), its main purpose is to detect coincidences between images supported by FAST and BRIEF algorithms. [10]

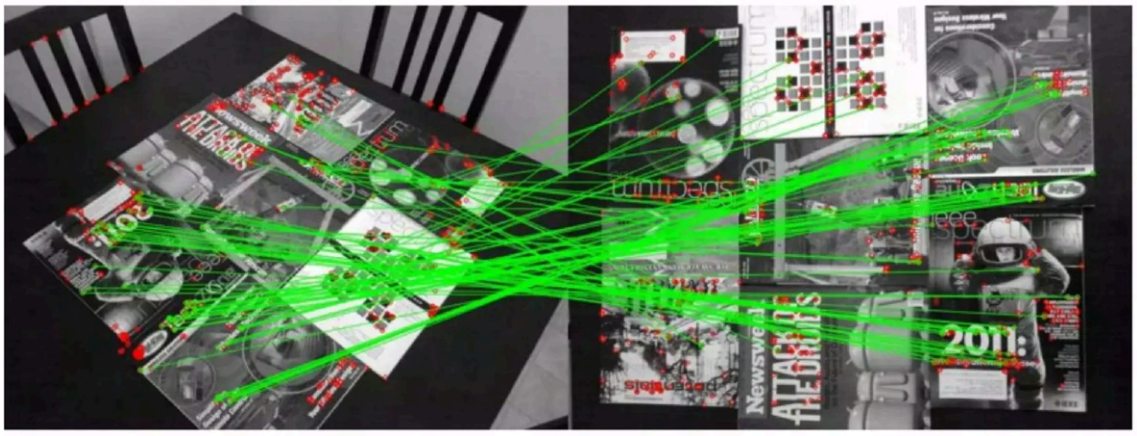


Fig. 2.8 ORB algorithm matches [10]

- oFAST:
  FAST algorithm is done in a multiscale (1:1.2) image, downsampling (using bilinear interpolation) the original image 8 levels. The objective of this algorithm is to try to find key feature points and a rotation orientation, Fig 2.9.

The algorithm starts choosing a pixel p and evaluating, comparing with the surrounding pixels, if 9 pixels are brighter or darker, that it is a keypoint. Comparing 1, 9, 5 and 13 could be a faster way to discard a keypoint.

Also it is applied Non-Maximal Suppression, that removes all adjacent corners with lower score. Consider a rounded patch of pixels and Gaussian Kernel filtering to eliminate noise in the image.

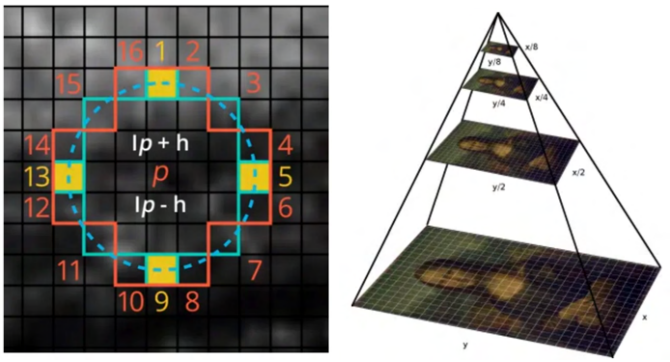


Fig. 2.9 Pixel patch left, Multiscale left [11]

In order to calculate the moment of the patch (Eq. 4.1), where considering a radius r, the intensity $I(x,y)$. To compute the intensity centroid we have (Eq. 4.2) and the orientation of the keypoint (Eq. 4.3).

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \qquad \text{i.e } (r = 15):\ m_{10} = \sum_{x=-15}^{15} \sum_{y=-15}^{15} xI(x,y) \qquad \textbf{(2.1)}$$

$$c = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \qquad \textbf{(2.2)}$$

$$\theta = atan2(m_{01}, m_{10}) \qquad \textbf{(2.3)}$$

Once with the centroid and the orientation, it could be possible to match some features between 2 images, the problem comes on how fast we want this comparison, specially in rotated images and here it comes rBRIEF.

- rBRIEF:
  The descriptor is a binary feature vector of the keypoint, it leverages the computer capacity to compare 2 keypoints vectors in a correlation. These vectors are 256 bits for each keypoint. Again filtering can reduce noise.

  To generate these descriptors, there are many ways, the typical one, the Gaussian. Pick a pixel in a Gaussian distribution around the keypoint and pick a second one in a Gaussian distribution around the first one. Compare these two pixels by their intensity and generate 1 or 0. The picking sequence of the pixels shall be the same for all keypoints.

Considering a patch I, the pixels p_i and q_i, the comparison can be described as in (Eq. 2.4).

$$\tau(\mathbf{I};\mathbf{p}_i,\mathbf{q}_i) := \begin{cases} 1 & : \mathbf{I}(\mathbf{p}_i) < \mathbf{I}(\mathbf{q}_i) \\ 0 & : \mathbf{I}(\mathbf{p}_i) \geq \mathbf{I}(\mathbf{q}_i) \end{cases} \quad \textbf{(2.4)}$$

$$f(n) = \sum_{i=1}^{n} 2^{i-1}\tau(\mathbf{I};\mathbf{p}_i,\mathbf{q}_i), \quad (n = 256) \quad \textbf{(2.5)}$$

$$\mathbf{S} = \begin{pmatrix} \mathbf{p}_1,\cdots,\mathbf{p}_n \\ \mathbf{q}_1,\cdots,\mathbf{q}_n \end{pmatrix} \in \mathbb{R}^{(2\times 2)\times 256} \quad \textbf{(2.6)}$$

$$S_\theta = R_\theta S \quad \textbf{(2.7)}$$

The descriptor can be computed as in (Eq. 2.5). Another method is proposed in ORB, steer BRIEF composes an S matrix (Eq. 2.6), using θ rotation angle a rotation matrix can be computed, having a rotated version S_θ, in order to have an improved descriptor.

- Bundle Adjustment:
  BA is also a key part of the SLAM algorithm, is an estimation technique for estimating joints of 3D coordinates and 6D position and orientation of the camera (extrinsics), minimizing the error. Considering j as a point in real coordinates, i as a camera view, X as a 3D points matrix, x as the coordinates of the points viewed from the camera and $\wedge$ as an estimation of that value. From the (Eq. 2.8), there would be a c camera matrix multiplied by the real coordinate points that can give us the camera image coordinates, of course, all of this without noise.

$$c^i X_j = x_j^i \quad \textbf{(2.8)}$$

From the (Eq. 2.9), the objective is minimizing the reprojection error, by minimizing the geometric distance of the estimated c and X per x camera view. This minimization is what is called BA.

$$min \sum_{i,j} d(\widehat{c^i}\,\widehat{X_j}, x_j^i)^2 \quad \textbf{(2.9)}$$

2.6. vSLAM 3D reconstruction results

With the rectified images, it is necessary to create in Matlab a vslam object. The first part of this reconstruction is going to take the positions (in red) and the feature points or landmarks. In these first tests the stereo camera was moving only a few feet away from the very starting point. In other tests the moving of

the camera is more visual in red, as seen in Fig 2.10. If the camera moves in a certain path, dense reconstruction will follow the path, reconstructing all what has been seen.

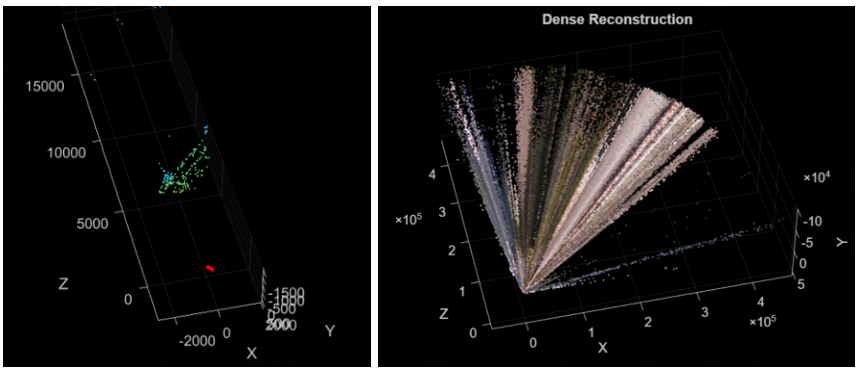


Fig. 2.10 Left→ Feature and position map, Right→ Reconstruction cloud point [milimeters]

To obtain all these features and position it is necessary to have the reprojection matrix from the stereo parameters from the calibration. Passing all this data to the object vslam, the object will make all necessary adjustments, to determine the scenario. In other words, make the loop.

The second term of the reconstruction, taking all these feature points and key frames of the looping process, using the disparity images it is possible to recreate a 3D scenario as seen in Fig 2.10. Disparity patch size can make differences in the overall results.

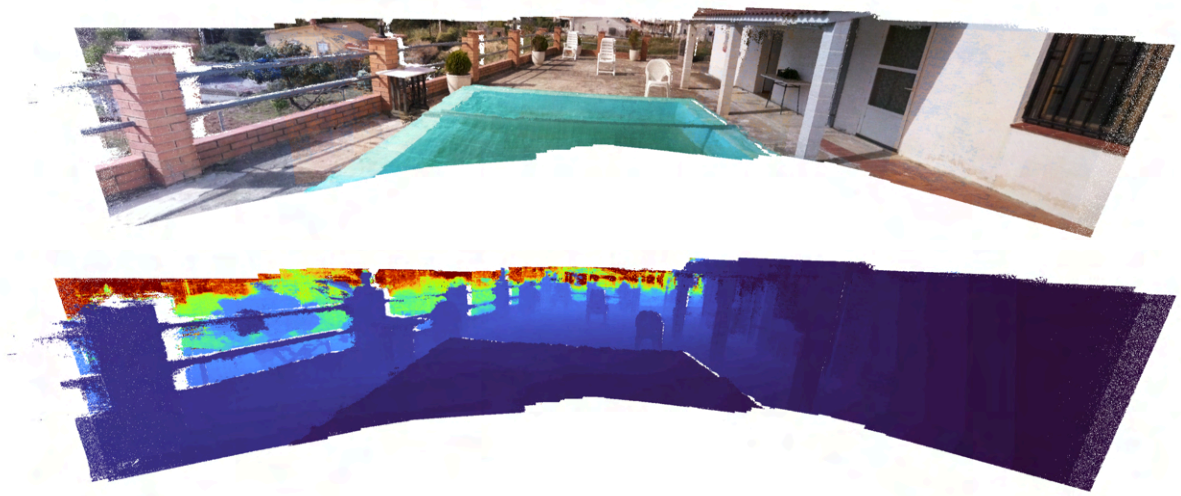


Fig. 2.11 Top→RGB reconstruction, Bottom→Disparity map/Depth

It can be seen that there are small errors of cloud points in the reconstruction, not to be confused with white gaps that are produced by the camera movement.

In these scenarios far away distance points are included, though they can be a source of errors. They can be extracted for any case of importance.

The main drawback seen in this system is the reliability of the data extraction, receiving a low image per second with the purpose of having higher resolution, or even rapidly rotating the camera can make the looping procedure lose the tracking. Because slam cannot make a tracking calculation without enough feature points of previous images, if they are any. When the loop breaks only the previous reconstruction can be saved till that point.

Furthermore a method to avoid this problem is to record a video locally in the raspberry pi and then pass it to the ground station. In this project it has been recorded a h264 video and then converted into MP4 using FFMPEG commands, being able to process. If the video is too long it would be better to convert the h264 with another programme. And importantly, all these functionalities come from the Matlab version 2024a and now onward. Natsortfiles Add-on can be useful to sort data if the datastore in Matlab is not well developed.

CHAPTER 3: STRUCTURE ANALYSIS AND FOV SYSTEM

3.1. Structure

To test the stereo system it is necessary to implement all these concepts in a structure.

The most important constraint in this development is structure integrity and impact resistance. So Z-HIPS (Impact Resistant Filament for Mechanical Prototyping) thread has been used to build this structure in 3D printing. Used on mechanical purposes like motorcycle 3D structures.

Supposing that the UAV mass is around <3 kg (in fact normally it would be 1.5 kg) with a height of <10 m in testing.

$$v^2 - v_0^2 = 2 \cdot g \cdot \Delta h \qquad v = v_0 + g \cdot \Delta t \tag{3.1}$$

It is possible to use Kinematic equations (Eq. 3.1) to determine what velocity and time has been spent in a free falling of the drone. With these parameters (v=14m/s and t=1.42s), the total applied force will be (Eq. 3.2).

$$F = \frac{dp}{dt} = \frac{m \cdot \Delta v}{\Delta t} = m \cdot g \simeq 29,6N \tag{3.2}$$

Nevertheless, the impact force takes into consideration the distance after the impact. The distance after impact can be stressful to calculate due multiple considerations, for high speed objects, Newton's Approximation can be used. For low speed objects, in the case of this project, the approximation of impact on soft dirt ground will be around <4 cm of depth as a rough measure, the best way to measure is to test it.

$$E = F \cdot d \quad \Leftrightarrow \quad m \cdot g \cdot h = F \cdot d \quad \Leftrightarrow \quad F_{imp} = m \cdot g \cdot \frac{h}{d} \simeq 7357N \tag{3.3}$$

The average impact force can be calculated like (Eq. 3.3), using potential or kinetic energy. What's more, not all the impact force will be supported by the system structure, also the drone frame structure will support part of the impact force.

3.1.1. Initial prototype

The program used for this design was Solidworks with standard HIPS material.

3.1.1.1. Structure

This prototype was discarded due brittleness of the material when thin walls are overstressed. The prototype is a first approach of what's searched. It is composed of different parts: main structure and GPS extensor for BN-880 GPS module (2 versions).

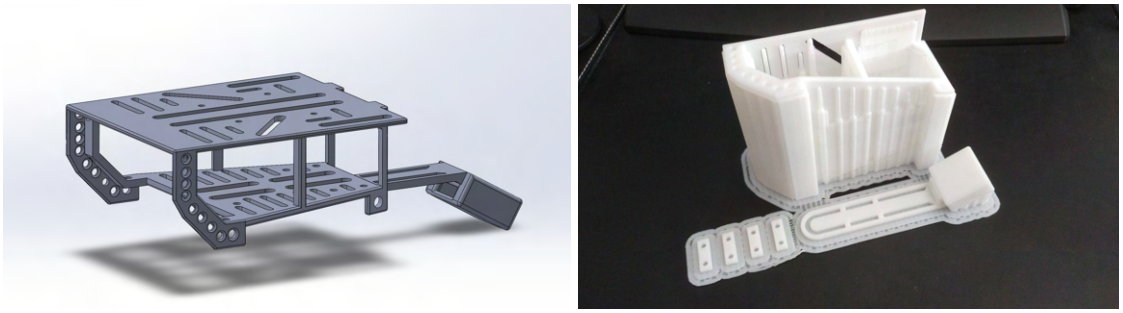


Fig. 3.1 Left→Design, Right→3D impression of the first version

3.1.1.2. Structural analysis

After printing, the structure was really lightened, indeed, too much. The structure was too thin even removing plastic leftovers, some cracks appeared. So it's not a good idea to use it. A static test using Solidworks can be a source of improvement, the force used for these tests will be lower, due not all the force will be supported by the system structure and also cause the force would break this simple structure. Furthermore, a frequency and fatigue study of the structure can be useful; however this project won't go to that realm. In case of curiosity everything can be done with the Simulation tool in Solidworks.

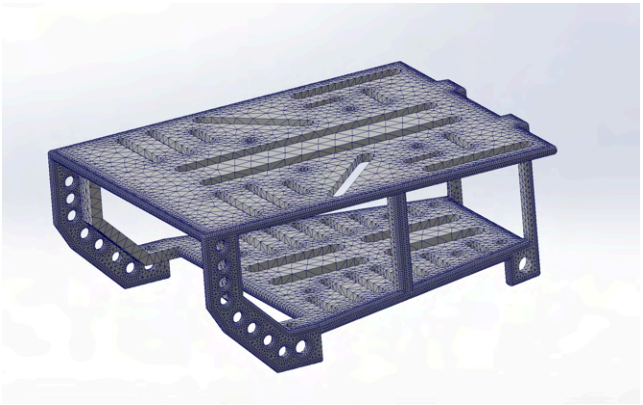


Fig. 3.2 Structure mesh

For these analyses the upper surface has been fixed and applied a force of 1500 N beneath and surrounding the structure. Higher forces than 1500 N hardly would continue without breaking.

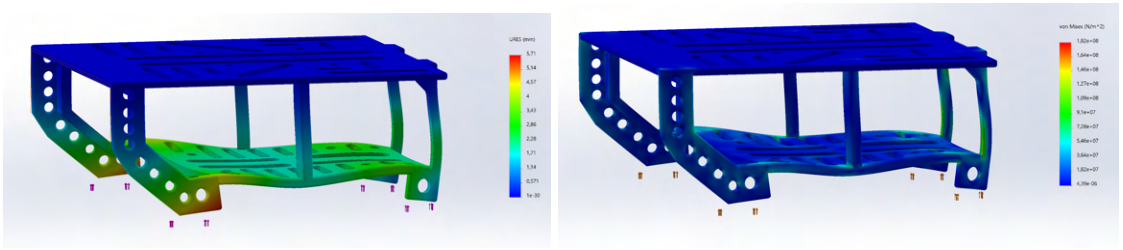


Fig. 3.3 Left→ displacement, Right→stress
(Force applied beneath)

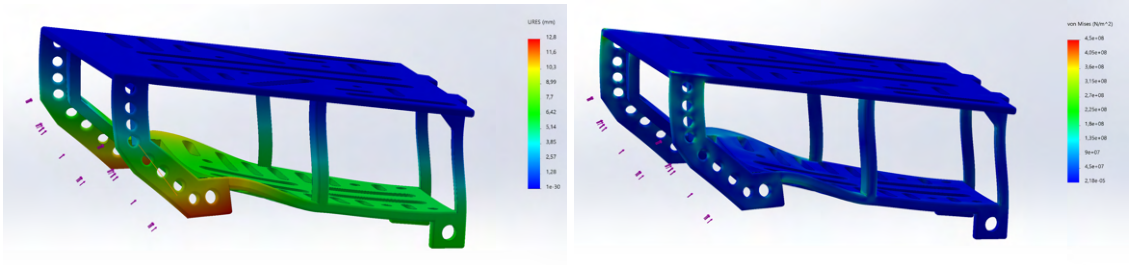


Fig. 3.4 Left→ displacement, Right→stress
(Force applied in front)

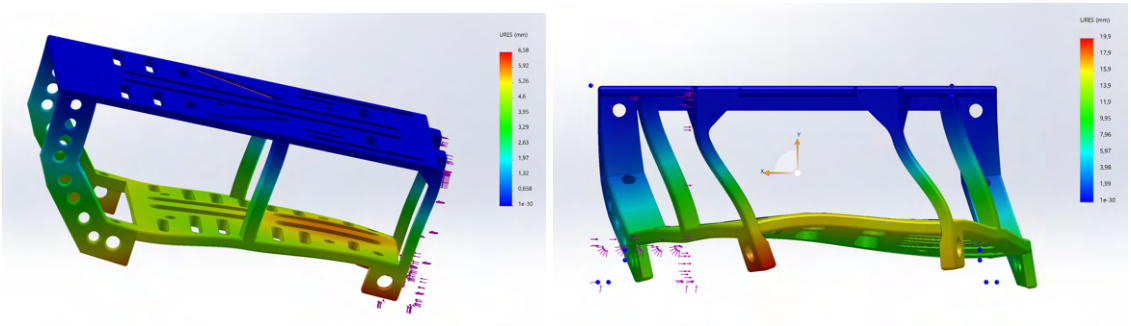


Fig. 3.5 Left→ Rear displacement, Right→Lateral displacement

The structural analysis shows, having poor lateral resistance, forces applied beneath are well supported and frontal structure also has to be reinforced.

3.1.2. Final structure

3.1.2.1. Structure

This new structure, though it has more weight, is more difficult to break, specially for its triangular shape with columns. Being more resilient in front of impacts. The basis basically is the same, with better and more columns, smaller holes and with the capacity of changing SD card without dismounting the raspberry pi of the structure.

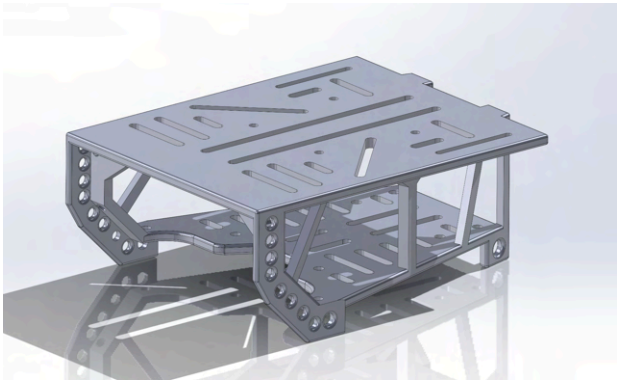


Fig. 3.6 Main final structure

3.1.2.2. Structural analysis

The structural analysis is the same as before, applying 1500 N (153kg). Shown in Fig 3.8, 3.9 and 3.10. Indeed, the real structure would be different, the material is similar but not exactly the same, thermo, stress factors, and more can affect the final structure.

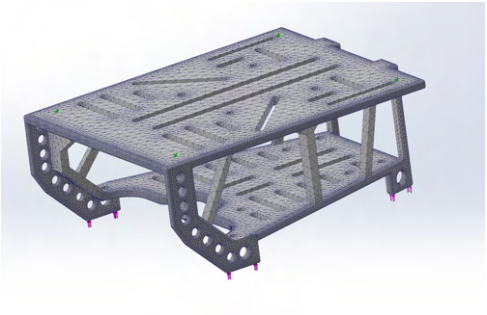


Fig. 3.7 Structural mesh

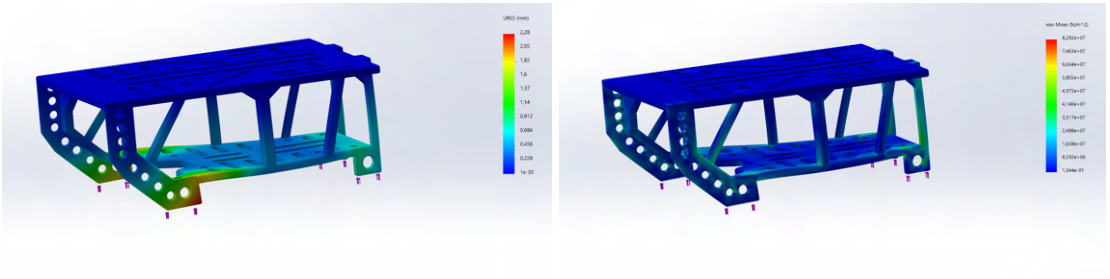


Fig. 3.8 Left→ displacement, Right→stress
(Force applied beneath)

Applying a force beneath, the basic structure has a maximum displacement of 5.71 mm, while the new one has a maximum displacement of 2.28 mm. Ratio of 2.5.

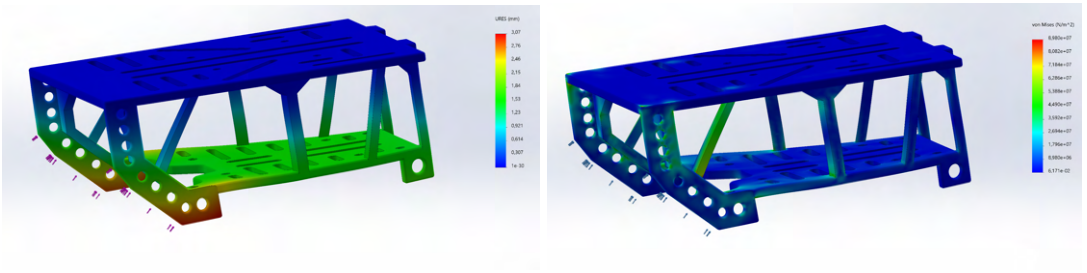


Fig. 3.9 Left→ displacement, Right→stress
(Force applied in front)

Applying a force in front, the basic structure has a maximum displacement of 12.8 mm, while the new one has a maximum displacement of 3.07 mm. Ratio of 4.16.

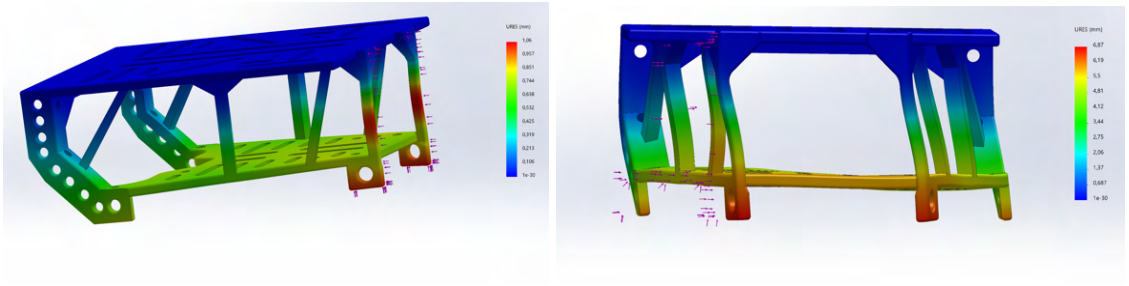


Fig. 3.10 Left→ Rear displacement, Right→Lateral displacement

Applying a force in the back, the basic structure has a maximum displacement of 6.58 mm, while the new one has a maximum displacement of 1.06 mm. Ratio of the displacement of 6.2.

Furthermore applying a force in front, the basic structure has a maximum displacement of 19.9 mm, while the new one has a maximum displacement of 6.87 mm. Ratio of the displacement of 2.89.

In overall the improvement of the resistance is quite noticeable with a minimum ratio of 2.5; though the weight has augmented, not too much.

3.2. FOV System

This last geolocalization system is the most complex in terms of accuracy. I called the FOV system; however some people call it in a different way. Sensor calibration, lens distortion, variation and form of the structure can turn down the purpose of this system easily. Due to a sequence of problems and restrictions in quality, altitude of the objective won't be shown and the horizontal position itself will be a big approximation. Also, earth curvature, camera distortion (from the perspective of the external agent) and any other parameter that can perform the system with a better accuracy, these parameters are not taken into account.

The objective of this system is to estimate the position of the objective in a first perspective of the external agent like it shows in Fig 3.11.

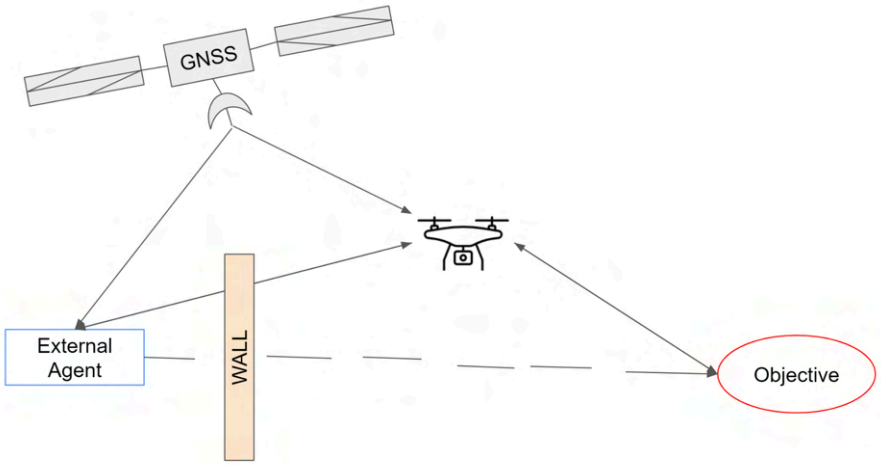


Fig. 3.11 FOV scheme system (ground view)

The development of this system has been done taking only pictures and data and processing it step by step. Continuous video and data streaming and processing can be done easily though it hasn't been applied.

A whole system can be observed in Fig 3.12. From this schematic the formulae can be obtained easily using trigonometric equations.

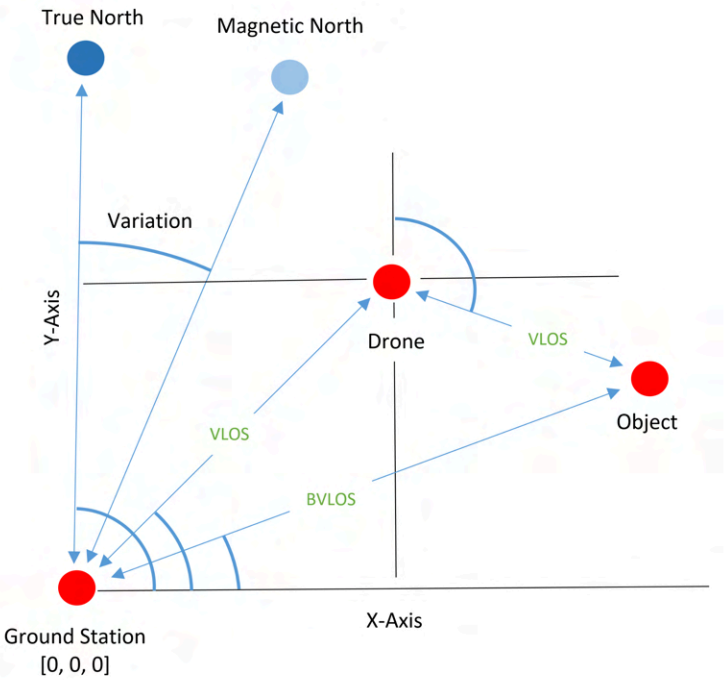


Fig. 3.12 Schematic of the FOV system (top view)

But a few considerations should be taken into account, the magnetometers (compass) will give the magnetic north. To use the geographic coordinates system everything will be referenced to the true north, applying the magnetic deviation to the compass. Many webpages on the internet can give these coordinates (in my case +2 degrees of deviation approx).

To have local variables in meters and not in latitudes and longitudes special functions like latlon2local are used. Also this function can be done manually easily using GNSS resolution depending on the latitude (i.e: 0.00001 degrees is like 1.11 m).

For the following equations, the reference point as in Fig 3.12 will be the ground station, the x-axis is parallel to the equator line being the main angle axis, the y-axis is parallel to the latitude lines and z-axis is the height. In order to project the final angles into the camera; the angles from these equations shall be corrected depending on the sign/position of the object, in programming atan2 will be used.

The compass angle should be modified as the reference to the x-axis to apply these equations.

$$\alpha_{x-Axis} = 90º - x_{North\ compass} \quad \textbf{(3.4)}$$

- VLOS Drone tracking:

$$\alpha_{Horitzontal\ angle} = atan(\frac{\Delta y}{\Delta x}) = atan(\frac{y_{drone} - y_{station}}{x_{drone} - x_{station}}) \quad \textbf{(3.5)}$$

$$d_{ground\ distance} = \sqrt{\Delta x^2 + \Delta y^2} \quad \textbf{(3.6)}$$

$$\beta_{Vertical\ angle} = atan(\frac{\Delta z}{d}) = atan(\frac{z_{drone} - z_{station}}{d_{ground\ distance}}) \quad \textbf{(3.7)}$$

- BVLOS Object tracking:

Using the drone stereo camera and compass it is possible to know the distance and the angles of the object respectively from the drone.

$\alpha' = compass\ angle + angle\ object\ in\ camera \pm magnetic\ deviation$
$\beta' = vertical\ angle\ object\ in\ camera + bias$
$d' = distance\ to\ the\ object$

If the camera is located with an inclination a bias should be added to β'.

$$\alpha_{Horitzontal\ angle} = atan(\frac{\Delta y}{\Delta x}) = atan(\frac{(y_{drone} + d' \cdot cos(\beta') \cdot sin(\alpha')) - y_{station}}{(x_{drone} + d' \cdot cos(\beta') \cdot cos(\alpha')) - x_{station}}) \quad \textbf{(3.8)}$$

$$d_{ground\ distance} = \sqrt{\Delta x^2 + \Delta y^2} \quad \textbf{(3.9)}$$

$$\beta_{Vertical\ angle} = atan(\frac{\Delta z}{d}) = atan(\frac{(z_{drone} + d' \cdot sin(\beta')) - z_{station}}{d_{ground\ distance}}) \quad \textbf{(3.10)}$$

Once with all this data, especially with the horizontal angle and the vertical angle; it is as simple as pointing where the object/drone is in the camera of the ground station. Remember that the camera has an angle width and the compass also rules the positioning.

CHAPTER 4. OBJECT DETECTOR DEVELOPMENT

4.1. Objective

Nowadays AI is pretty used in the UAS world, it can bring many curious and useful applications, specially in sensor processing data. In this project an AI has been developed to provide object detection on a UAS. Many objects can be detected; however, in order to make things simple and easy, only 1 object has been used to train the overall system on a pre-structured AI. This object is a radio control car Fig 4.1.

Fig. 4.1 Traxxas Slayer radio control car

4.2. What AI to choose?

As far as I commented in the annexes (AI chapter), there are several types of AI. Matlab offers r-cnn, fast r-cnn, faster r-cnn, different kinds of YOLO versions, ACF, feature detectors, etc.

Seeking for the most appropriate AI to use in this project, YOLOv4 has been one of the most suitable AI due its fast and accurate results compared with others. Furthermore, to compare this network with another, an ACF network will be introduced due its time and single class performance detection. (YOLOv4 Add-on needs to be downloaded).

4.2.1. YOLO v4

This object detector (You only look once = YOLO), is composed of 3 parts. [12]

1. Backbone:
   In this stage, it computes features maps with different sizes from input images. In 5 main blocks using a CNN, this network can be pre trained such as VGG16 or CSPDarkNet53 on datasets.

2. Neck:
   It is composed of Spatial Pyramid Pooling (SPP) which concatenates max-pooling outputs with the objective of extracting the most important

features, specially from small objects. And the Path Aggregation Network (PAN), which upsamples and downsamples the features maps in order to make predictions combining with SPP to have low and high level features maps. The outputs are 19x19, 38x38 and 76x76 of feature maps.

3. Head:
   Here comes the typical object detector (a YOLOv3), where prediction of boxes and classification scores are revealed. For more information about YOLOv3 see [13].

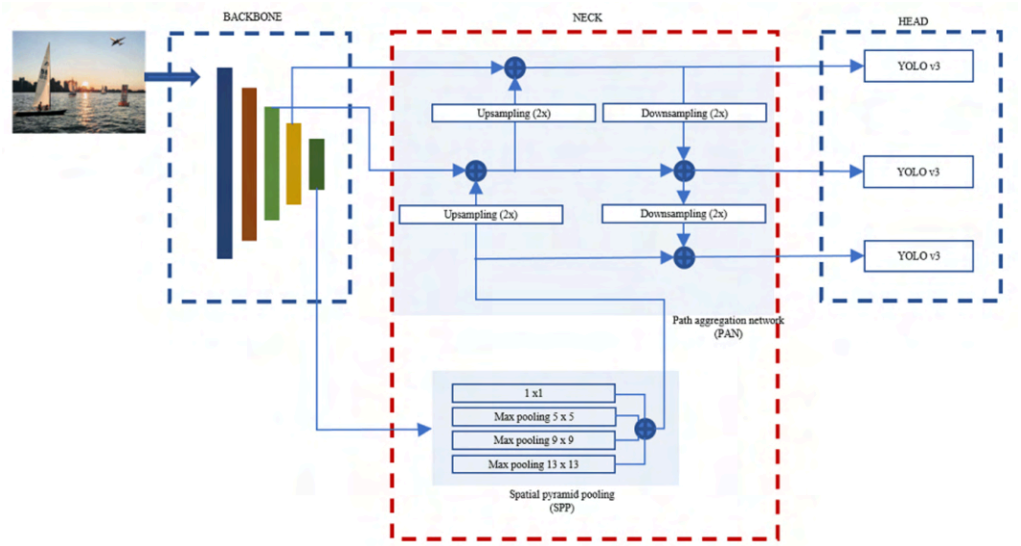


Fig. 4.2 YOLO V4 entrance structure

The tiny YOLOv4 uses 2 heads instead of 3 heads, with 13x13 and 26x26 in size. This project uses a custom "tiny-yolov4-coco" network. With a total 74 layers and 6 millions of learnables, this network can handle around 80 class names. The “layers” used in this network are convolution, normalization, ReLU, concatenation and resize. As seen in Fig 4.3. In Matlab App Network Designer it is able to change and customize this network in an easy way.

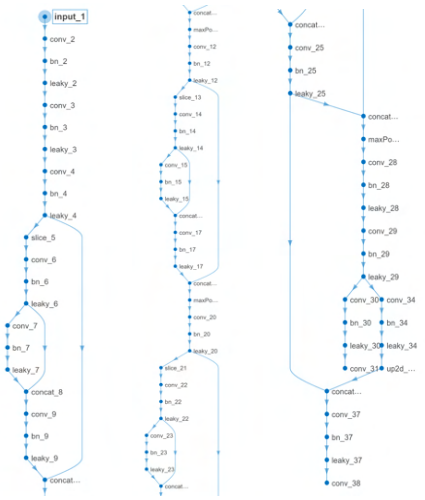


Fig. 4.3 Internal YOLO network
(starting from left to right images)

4.2.2. Aggregated Channel Features (ACF)

What makes this network differ from others is practically the input. As seen in Fig 4.4, from an image the image processing takes part making diversal channels of information about features, summing them and downsampling with filtering. Converging multiple information into a vector and then analyzing multiple pixels by this vector in a decision tree in a multiscale form.

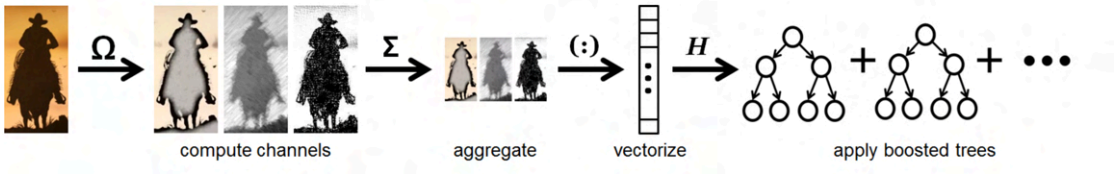


Fig. 4.4 Scheme ACF detector structure [14]

A precursor of this method is the Integral Channel Features (ICF). For more information, see [14].

4.3. Object Detector Creation

The most important and time consuming task in the whole process of the development of an object detector.

4.3.1. Data labeling

To complete this task I have been using the labeling app from Matlab, recording a video and then labeling at this app. It's pretty easy, I really recommend selecting the Temporal Interpolation Algorithm and automatizing everything. Around +2850 images were labeled in this task. However the app gets slower when too many images are uploaded, multiple sessions can be done and then merge all them with code, as I did.

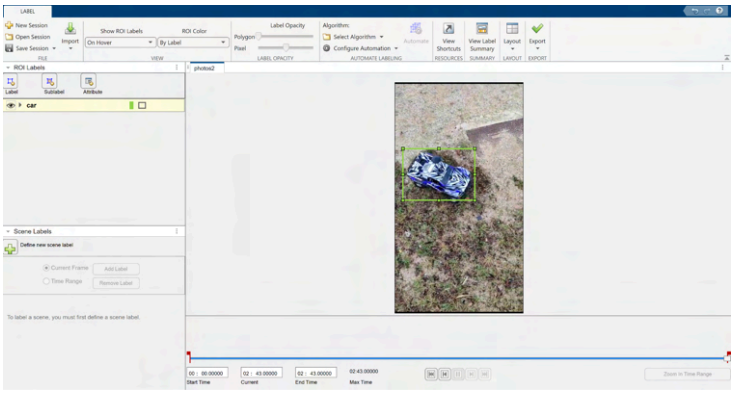


Fig. 4.4 Matlab App Video Labeler

From all this dataset, not all images will be to train the network, some images from this set will be for evaluating the network; around 200 images for the evaluation.

4.3.2. Training and results

To evaluate the networks a few features must be taken into account. The ideal precision plot would be a step function. Several networks have been developed of each type, only the most fruitful were selected.

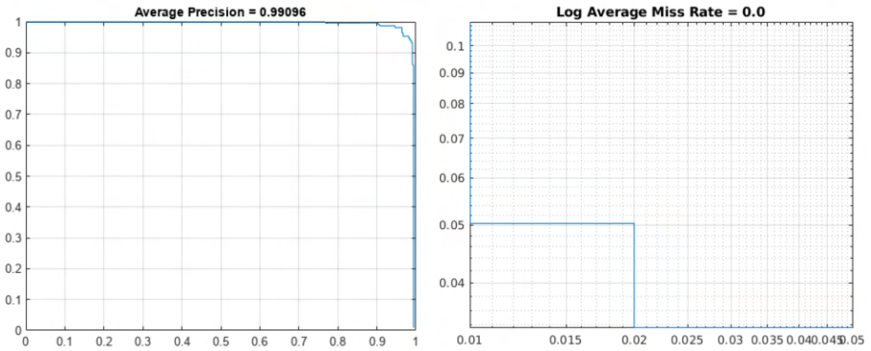


Fig. 4.5 Matlab results example of a pretty solid network

I considered that having an overlap of 35% between true and detection labels would be enough to consider where the object is.

- Precision: ratio between true detections and detections.
- Recall: ratio between true detection and sum of all true and false detections.
- Miss Rate: the same word says it.
- Fppi: false positives per image.

4.3.2.1. YOLO v4

Remember that this network is used to detect multiple objects in short periods of time with high complexity. The time taken to develop this network was less than 1h and 30 min. With an initial learning rate of 0.0001, <200 epochs, 16 of mini-batch size, normalization of the data and final training loss around 0.05. Using stochastic gradient descent method “sdgm”, see (Eq. A-3.15). Furthermore there are other methods like “adam” that can be used to train.

- Results with only new images:

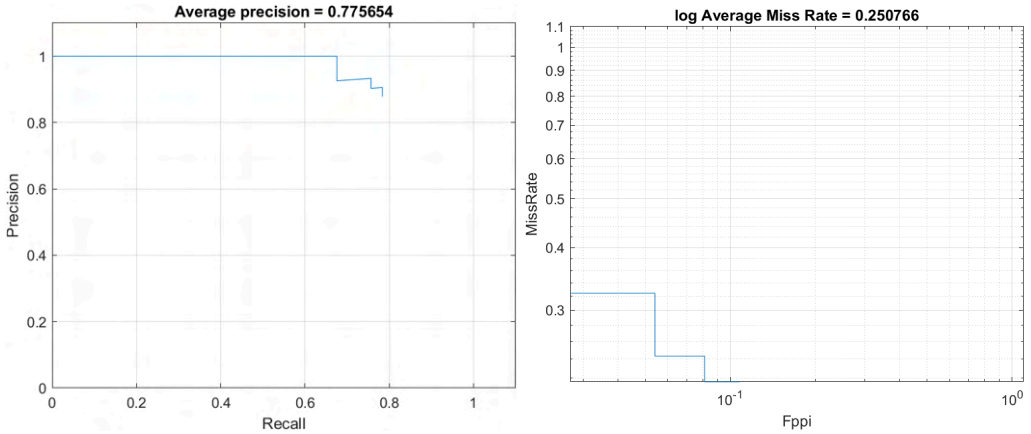


Fig. 4.6 Results with new images

- Results with all the dataset including new images:

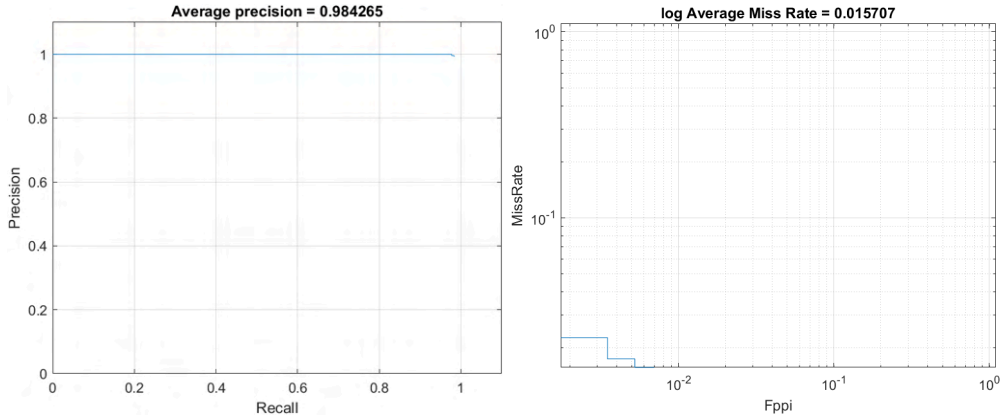


Fig. 4.7 Results with new and trained images

4.3.2.2. ACF

Remember that this network is used for single object detection without too much complexity. The time taken to develop this network was less than 15 min, with 10 stages and 1102 learners.

- Results with only new images:

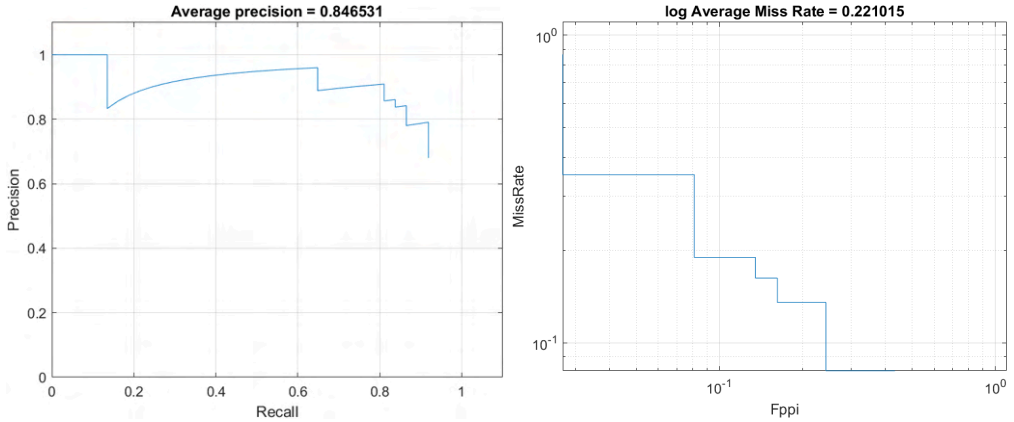


Fig. 4.8 Results with new images

- Results with all the dataset including new images:

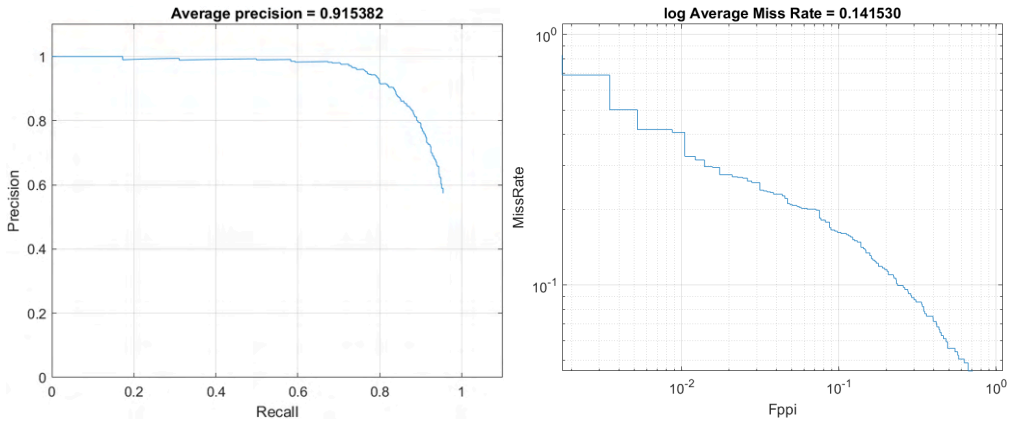


Fig. 4.7 Results with new and trained images

As a conclusion of all this test, there’s a lack of datasource to train and evaluate the network. Although there’s a lack, the results were not perfect but optimal to do the objective. It's necessary to say that not all the labeled sources were perfect, so logically there’s a gap that can cause errors in the training.

It is curious that ACF has more precision and losses less than YOLOv4 in the new dataset. While testing the detectors with all images, YOLOv4 gets more precision and less loss than ACF. Probably that is due network complexity and training time/options. In my opinion the YOLOv4 network is more suitable with multiple scenarios. ACF can also be pretty useful due training time.

As a mere curiosity, other CNN networks like the YOLOv8 network can be imported into Matlab, from github, pytorch, tensorflow or ONNX. In this project YOLOv4 has been used due its simplicity, fast implementation and time dependance; also Matlab itself offers it. It would be nice to test a R-CNN to see how it can affect object dimensionality in the detection, just because the size can matter using YOLOv4.

CHAPTER 5. LOCALIZATION SYSTEMS, FINAL TESTS

5.1. GNSS position drone tracking, Stereo disparity maps and object detection

In these tests the GNSS receiver is a key point, (in this case the BN-880 GPS module was used). This sensor has to be the most resilient one; however, it isn't the case in these results. The overall system consists of a GNSS receiver that gives information to the STM32, then the STM32 gives this data in an appropriate sequence to the raspberry pi. Finally the raspberry pi saves all this data and at the end it delivers all the data to the ground station wireless via ip access.

A few tests have been taken, at first glance in Fig 5.1 it can be seen that the receiver is too sensible when there's no line of sight to the satellites and loses extremely fast its position tracking. What's more, it needs a period of time seeking for satellites before delivering information.

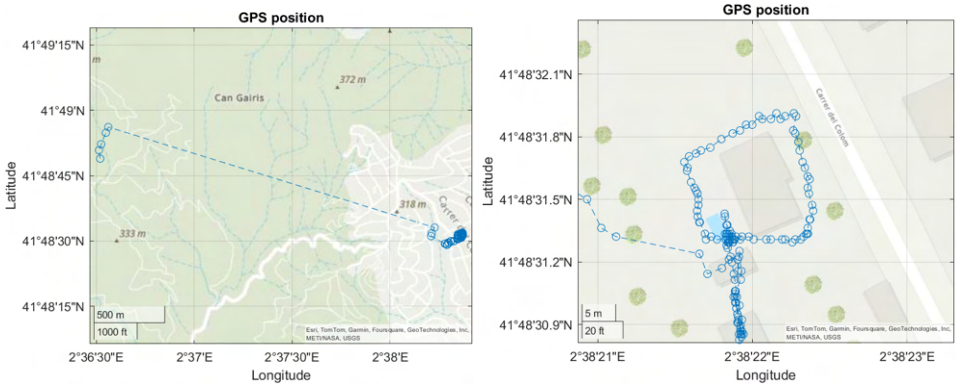


Fig. 5.1 GNSS/GPS Device tracking

Deleting all wrong information about its position, the final position estimation can be seen in Fig 5.2. Satellite tracking and topographic tracking plots are a little bit different, in my personal opinion, satellite tracking seems less exact and displaced from topographic one (less than 2m).

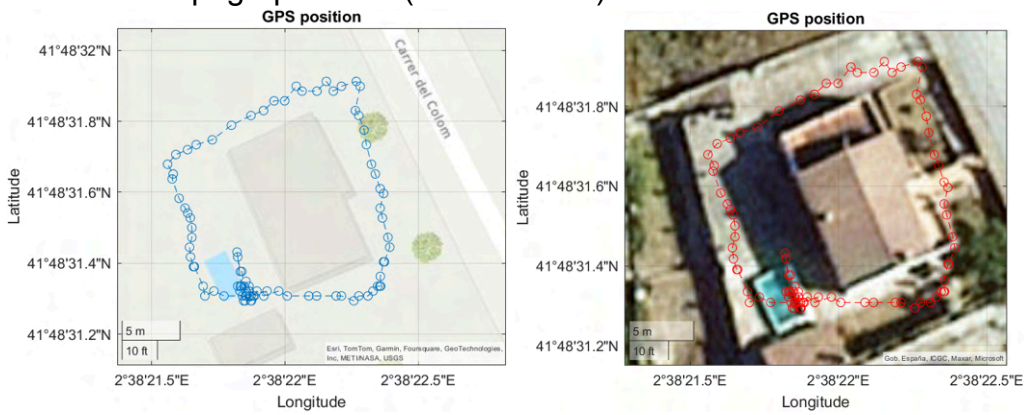


Fig. 5.2 Topographic image (Left), Satellital image (Right)

Respect the altitude values all of them seem to have correlation with the terrain though they have no precision. Regarding the horizontal position, the precision of 2 m said by the receiver makes sense in Fig 5.3, the red line in the right image is the path followed to extract all the data approximately.

Fig. 5.3 Left→ Altitude plot, Right→Position vs expected position

In Fig 5.4 there's an example of distance measurement, showing disparity maps or distance maps in other words. Furthermore, to have a concept of distance, there are 3 plant pots, the first one at 3 m, the second one at 6.5 m and the last one at 11 m.

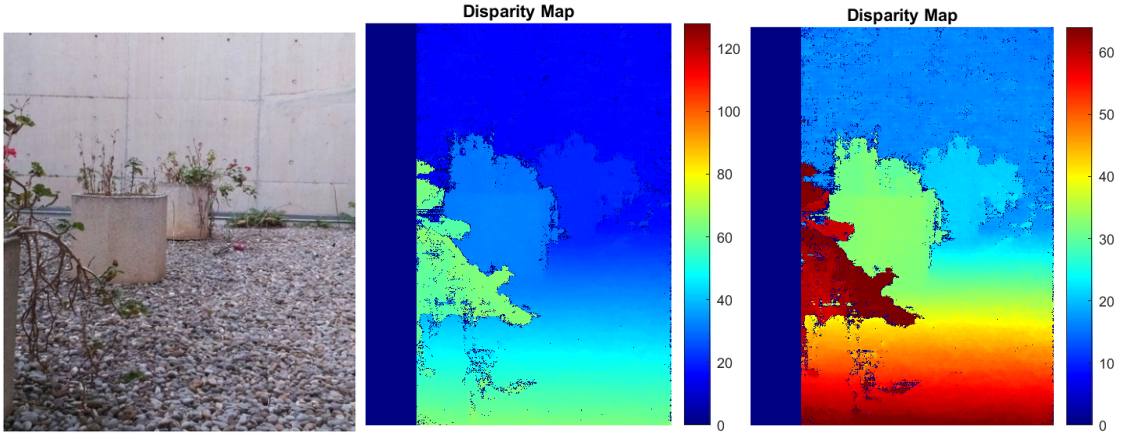


Fig. 5.4 Example Disparity maps (x-axis pixels difference)

The following images (Fig. 5.5) are the fusion of the disparity maps and the object detector giving a confidence of the detection of 25 %, indeed it is so low because the resolution, AI database, and light aren't good enough. Furthermore the images of the dataset are not obtained using the same cameras. Thus it gives an explanation of low coefficient. The distance measurement given by the disparity maps is 5.35 m and the true distance measured with a laser range finder is 4.6 m (a loss around 15 %), of course it can have a lack of precision on the ground truth because the measurement wasn't done using the exact same point respectively.

Fig. 5.5 Object detector detector distance estimator (with image zoom)

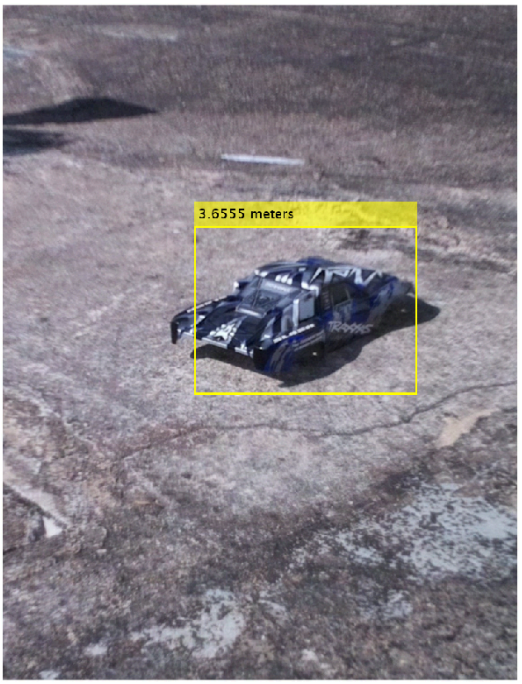


Fig. 5.6 Distance estimation (estimated 3.65 m, indeed 4 m approx in real distance) (with image zoom)

5.2. vSLAM system

The following test is inside a building, the plots are: light, slam parameters and sensor resolution dependant, so the results can be better than they seem to be. The results are scaled to a lower resolution than the real images because it requires computer memory. In the plots there are more points than they need

(they can be removed), furthermore the larger the distance we want to plot, larger errors we get and more difficult would be to track the dispositive.

The main constraint found was only being able to plot small fragments of the video, probably due memory, camera resolution and poor presence of light.

The first test has been done in a straight line inside a corridor, with camera vibrations.

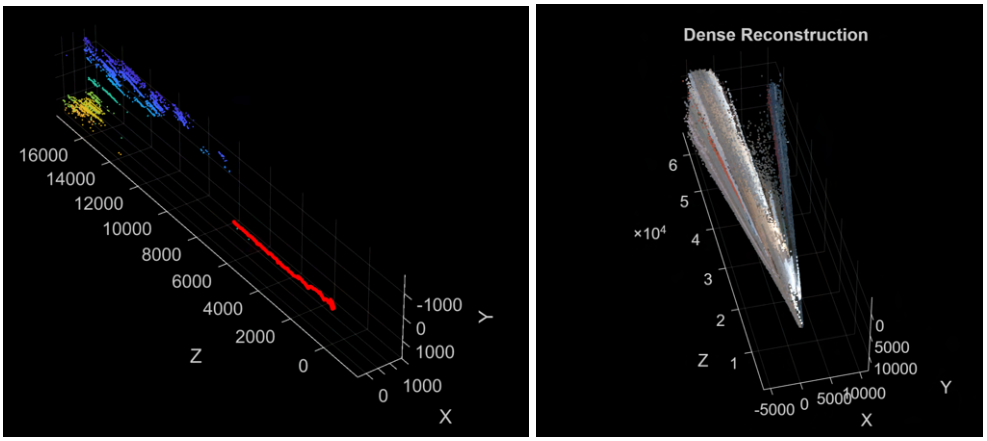


Fig. 5.7 Estimated position and landmarks (Left) (mm)
Dense reconstruction (Right)(mm)

In the plot of Fig 5.8 it can be seen that there's a window in the right side of the reconstruction so that's why there's a branch from the upper sight.

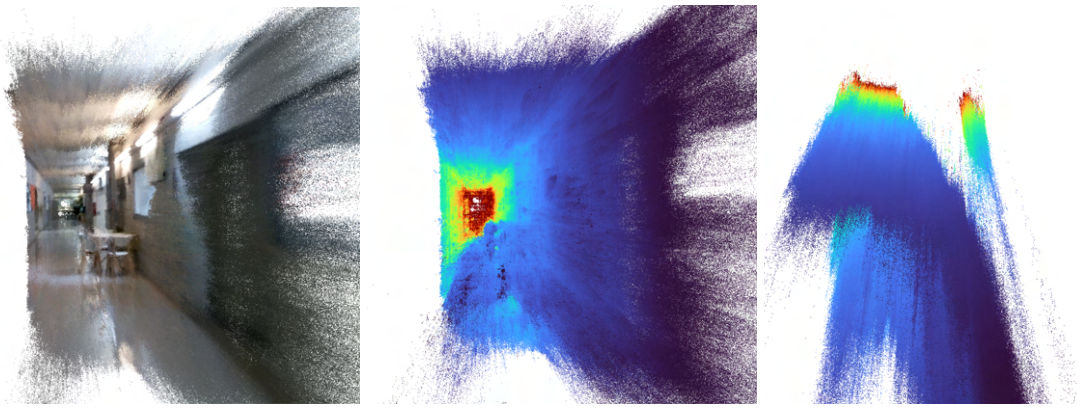


Fig. 5.8 Color and depth reconstruction (right side upper sight)

This other test is more or less the same, the big difference is that there's a move of the camera position scanning the environment as seen in Fig 5.10, being able to see the environment from multiple positions.

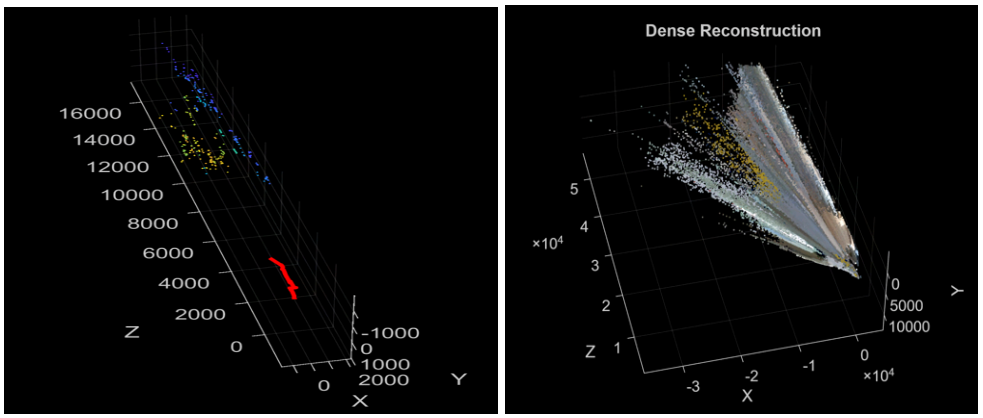


Fig. 5.9 Estimated position and landmarks (Left) (mm)
Dense reconstruction (Right) (mm)

Fig. 5.10 Color reconstruction on different points of view

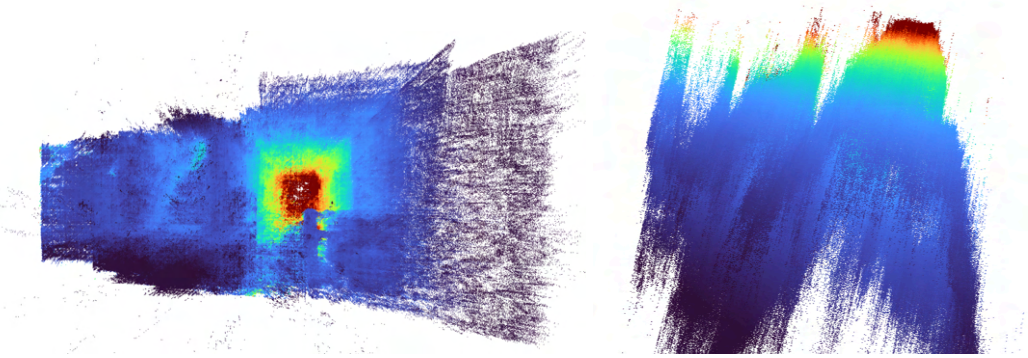


Fig. 5.11 Depth estimation (right side and upper sight)

For every plot around 100 images were used.

5.3. FOV system

The FOV system didn’t use object detectors due lack of confidence, what's more it is only used for horizontal position estimation. The main constraint was GPS mobile integrity. Furthermore poor compass integrity makes the results hard to compute, the most approximated results to the real ground truth is the following one.

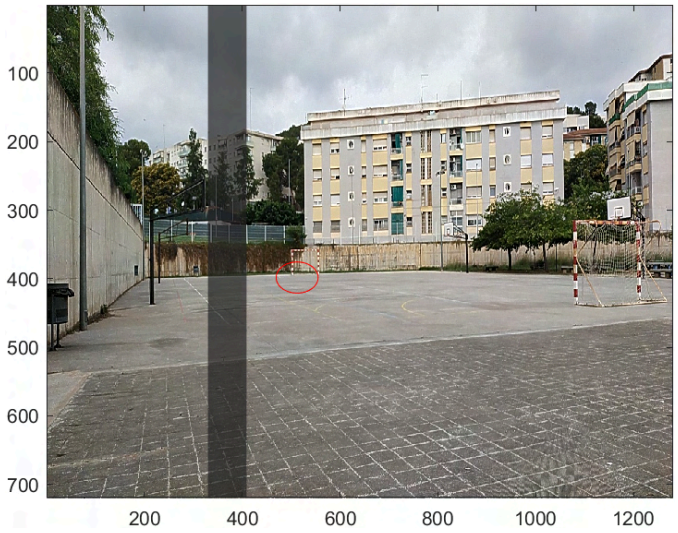


Fig. 5.12 FOV results

In Fig 5.12 it can be seen in red the object position (due large distances it isn't shown) and in a darker bar the estimated position. The variation of distance can be expressed by a few degrees of variation.

The third object results aren't very accurate so they are discarded from this study. With appropriate hardware and error correction they could be plotted in a correct way.

CHAPTER 6. GNSS DATA ACQUISITION AND GPS FLIGHT CONTROL

6.1. Custom GNSS receiver

To control the drone not a simple GPS can be used, thus this section is explained.

6.1.1. GNSS receiver

To control the drone, the BN-880 which was used to do the tests has been updated with the NEO-M8N GPS module. This new module offers an improvement of accuracy of 0.6 m and it needs less time than BN-880 to start to be operable. Furthermore it contains the HMC5883, being more noise resistant than QMC5883. For more information about the BN-880 look annex 4.

Fig. 6.1 Mini Ublox NEO-M8N GPS + HMC5883 module

What's more, this module uses Ublox protocol, this protocol will facilitate the customization of the module, later explained.

6.1.2. NMEA

NMEA indeed means National Marine Electronics Association. Founded in the 60s, this association carried out the normalization and the creation of the NMEA 0183 protocol for any marine electronics such as gyros, autopilots, GNSS receivers, sonars, etc. This protocol transmits the signal at 4800 bauds on a serial connection RS-422 (similar as RS-232) without parity, printable on ASCII characters. The new protocol NMEA 2000 improves the signal velocity up to 250k bauds, and connection capability compatible with CAN bus. For more information look [15].
The vast majority of commercial cheap common GNSS receivers uses this protocol due its simplicity. But how does it work in depth?

Only the Global Positioning System will be explained, other sensor communication protocols can be found on the Internet.

6.1.2.1. Types of NMEA messages

- Informative message:
  These messages start with “$” + 2 letters of the identification of the dispositive + 3 letters type of message. Then every data given is separated by commas. Up to 81 characters the whole message can have, ending with “*”+ 2 hexadecimal characters obtained with an OR operation with the whole sentence (excluding “$*” characters). I.e: “GPGGA,23517… ,0000*5A”.

- Interrogative message:
  These messages start “$” + 2 letters of the identification of the device who asks + 2 letters of the device who’s being asked + letter “Q” meaning a question + “,” + 3 letters of type of question. Then the message in question. I.e: “$CCGPQ,GGA, … “.

- Special message:
  These messages start with “$P” + 3 letters of the manufacturer. Then the manufacturer message goes. I.e: “$PHPQ, … , 000*5F”.

6.1.2.2. GPS NMEA messages

To identify a Global Positioning System module data the first 2 letters must be “GP”. Just identify that the second step is to identify the type of message with the following 3 letters. There are too many, but the most important ones are GGA and GLL alternatively.

Table 6.1. Types of GPS informative messages [16]

$GPBOD	Bearing, origin to destination
$GPBWC	Bearing and distance to waypoint, great circle
$GPGGA	Global Positioning System Fix Data
$GPGLL	Geographic position, latitude / longitude
$GPGSA	GPS DOP and active satellites
$GPGSV	GPS Satellites in view
…	Many more.

- Example: $GPGGA,170834,4124.8963, N,08151.6838, W,1,05,1.5,280.2,M,-34.0,M,,*75”

Table 6.2. Explanation $GPGGA message

Time	170834 →17:08:34 UTC
Latitude	4124.8963, N → 41d 24.8963’N
Longitude	08151.6838, W → 81d 51.6838’W
Fix Quality (0-none/1-GPS/2-DGPS)	1 → GPS fix
Num satellites	05→ 5 satellites
HDOP	1.5
Altitude	280.2,M →280.2 m above SL
Height adobe WGS84	-34.0,M → -34 m
Time since last DGPS	-
Station reference	-
Checksum	*75

- Example: $GPGLL,4916.45,N,12311.12,W,225444,A*28
  If it doesn't have time it will end with a checksum anyways.

Table 6.3. Explanation $GPGLL message

Latitude	4916.45,N → 49d 16.45’ N
Longitude	12311.12,W → 123d 11.12’ W
Time	225444 → 22:54:44 UTC
Status	A → valid
Checksum	*28

To get decimal values it is only necessary to divide per 60, the minutes obtained and add that value to the degrees. Just using degrees, 1º in latitude is 111.321 km and in longitude is equal to 40075 km multiplied by cos(latitude)/360 approximately. Considering spherical earth, though there are many ways like the Haversine formula [17].

6.1.3. Customization via U-Center

Once you connect any GPS via serial you will receive the NMEA. However sometimes if the user doesn't receive any information or even the user wants to change GPS settings special procedures shall be done, and here it comes U-Center. This programme is dedicated for those GNSS sensors with Ublox protocol specially, though probably some other sensors can be used too.

The GPS used in this work uses Ublox protocol, so that's why it is used in this project.

In order to start the modification of the sensor, install the U-center programme and it is necessary to use a usb-ttl adapter to connect the sensor to the computer. Only use serial and power pins. In my case, the green cable was Tx while the yellow cable was Rx.

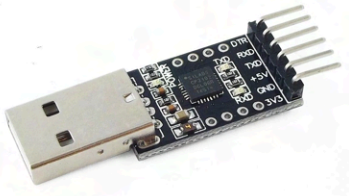


Fig. 6.2 USB-TTL UART Adapter

When you open the programme at first sight, the user can open multiple plots to see how the sensor works, even with Google Maps API, Fig 6.3.

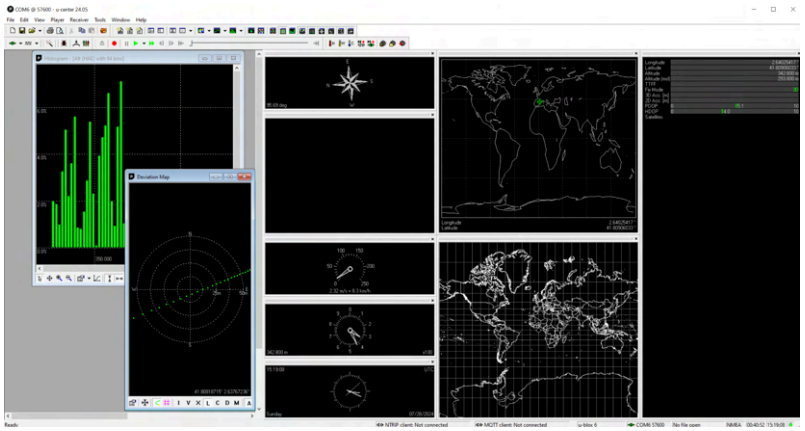


Fig. 6.3 U-Center Programme

Most GNSS sensors work at 4800-9600 bauds and 1 Hz. These parameters can limit the overall code inside a drone, so they must be changed.

Following the steps in Fig 6.4, connect the sensor, find out what velocity uses and look if it works.

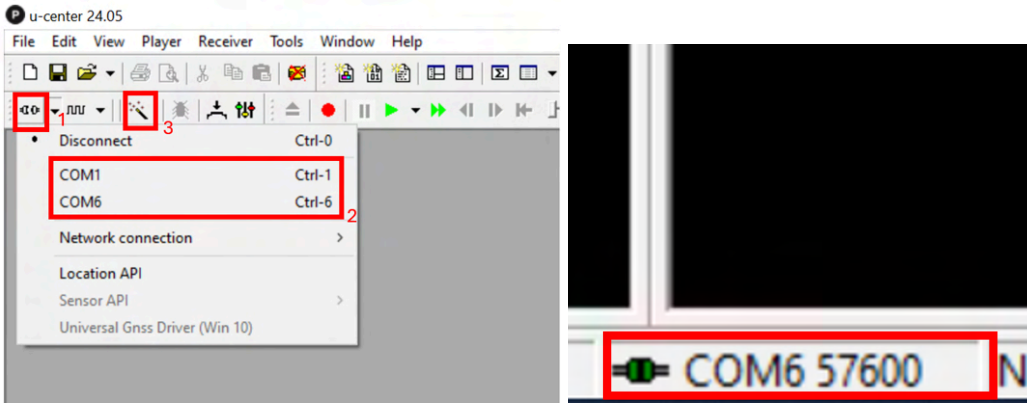


Fig. 6.4 Connection U-Center with baudrate (modified)

Once connected you can see what messages are being sent. To change parameters go to configuration, see Fig 6.5.

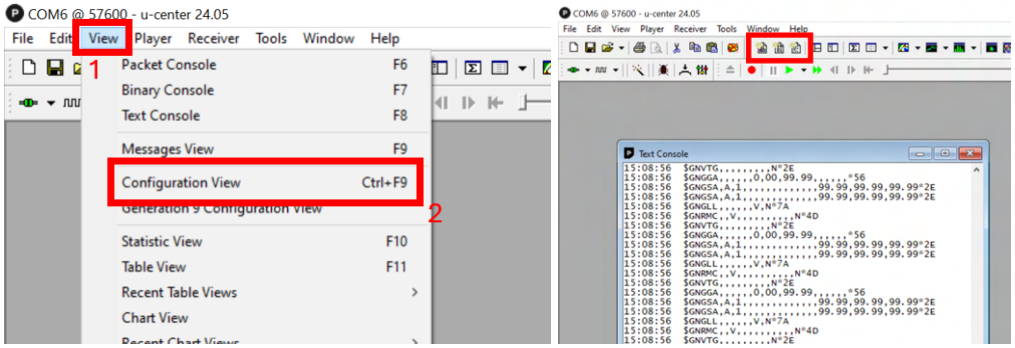


Fig. 6.5 Configuration (Left) and Messages (Right)

Follow the steps in Fig 6.6, to change the baud rate for example 57600, the output must be NMEA and the frequency measurement to 5 Hz and always click on send (it will erase the old configuration of the GPS!!!).

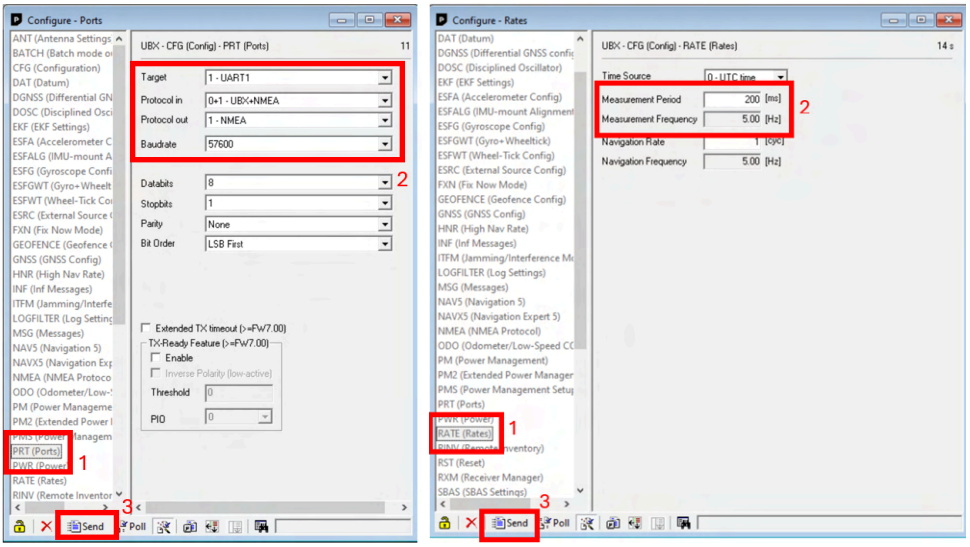


Fig. 6.6 Change GPS Protocol + Baud Rate + Frequency

The code's main source of reference on making the drone in this project has been Brokking.net [1]. This source changes these parameters manually by commands. The method explained here is much more easy for the user. However some extra modifications are needed like the elimination of GSV messages to keep tracking the drone without losing its position, see Fig 6.7. The important data are GGA and GLL messages.

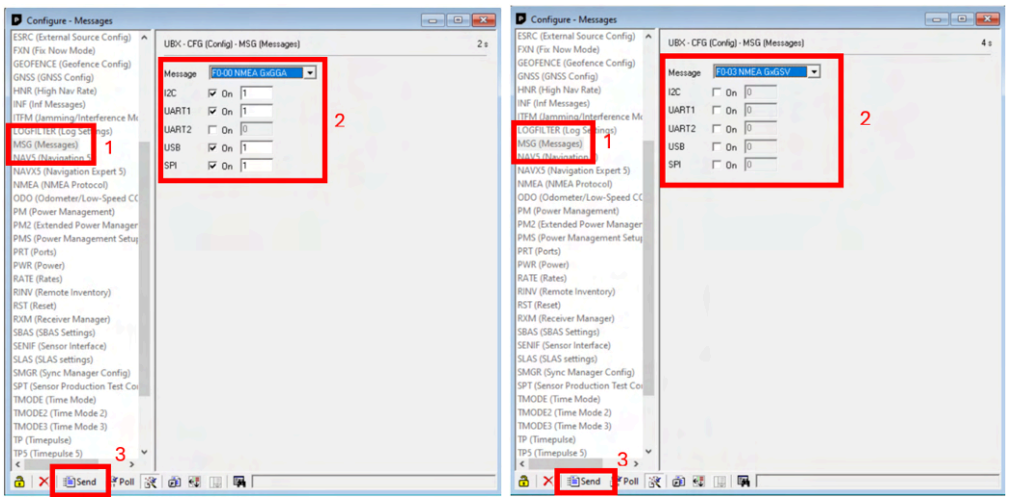


Fig. 6.7 Enable(Left)/Disable(Right) NMEA Messages

Finally the user can see how many constellations are enabled and can be seen. And last but not least save all the modifications done before in the CFG section, see Fig 6.8.

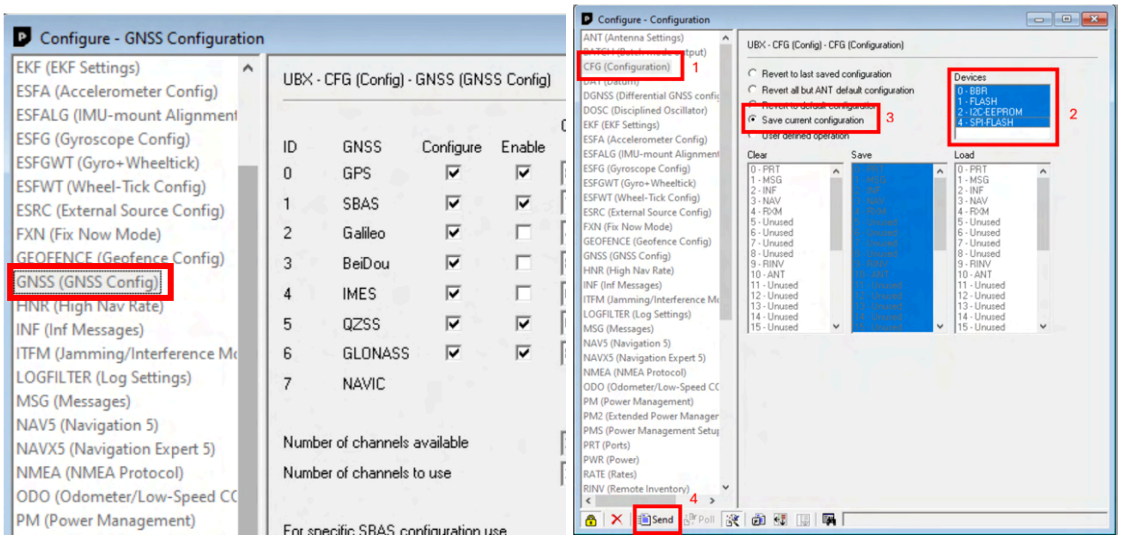


Fig. 6.8 Availability GNSS (Left) and Save change (Right)

6.2. Flight control using GNSS

In order to manage the drone using the modified GNSS receiver, there are several steps. The first step is to read the serial sent by the receiver and look for GGA and GLL messages and obtain the coordinates. And of course these coordinates shall have at least 6 decimals to position the drone in cm (multiplying by 1 million, every last digit change is like 11 cm).

The drone code has been explained in the annexes, any further doubt see the references used to do this part of the project, like [1].

The second step is to create the controller, to do so, the following equations have been used.

$$P_{output} = (GPS_{actual} - GPS_{holding}) \cdot P_{gain} \quad \textbf{(6.1)}$$

$$D_{output} = (GPS_{actual} - GPS_{previous}) \cdot D_{gain} \quad \textbf{(6.2)}$$

But, here comes the problem: the vast majority of arduino coded drones, its loop lasts around 4-6 ms, in this case 4 ms. And the receiver refreshes its data at 5 Hz, in other words, every 200 ms.

To solve this problem Brokking [1] proposes a very good solution, make the position values delay 200 ms and simulate 10 position values (1 every 20 ms) between the delayed position and the actual. Simulating a 50 Hz receiver. This can lead to a better response of the drone though using values with 200 ms delay.

With this simulated position, it can be applied as actual in the controller to make any opportune changes in the drone.

And finally the last step is to apply these PD controller values (longitude adjust and latitude adjust) to the pitch and roll like in the following equations.

$$roll_{adjust} = long_{adjust} \cdot cos(yaw_{angle}) + latt_{adjust} \cdot cos(yaw_{angle} - 90º) \quad \textbf{(6.3)}$$

$$pitch_{adjust} = latt_{adjust} \cdot cos(yaw_{angle}) + long_{adjust} \cdot cos(yaw_{angle} + 90º) \quad \textbf{(6.4)}$$

As the reader can see, this method explained only shows how to correct the drone for only one holding position. If the user wants a drone that uses GNSS self-positioning and to control the drone at the same time, it is possible to do it in 2 possible ways. The first way or the simplest is to enable the GPS mode only in the receiver dead bands (not controlling the drone). And the other method is to disable pitch and roll adjust depending on the control of the user. And of course in every method shall actualize the holding point with every movement. To clarify, this mode will only be enabled if the drone is in the air, if the drone is on the ground it can be dangerous.

CHAPTER 7. BUDGET, ENVIRONMENTAL AND SOCIAL IMPACT

7.1. Budget and environmental impact

Considering all the proceeds taken during this investigation, the summary of all the resources that had been compromised in the recreation of the objective are the following ones (excluding arduino uno set up):

Table 7.1. Budget Basic STM32 Localization system

Material	Cost approx (€)
Matlab	900-262
SolidWorks	1044
Arducam Stereo Hat	135
Pack of antennas	23
Variety of cables, bolts and others	10
Adafruit STM32F405	40
BN-880 (GPS+Compass)	21,66
Raspberry pi 4B	85
Mobile Phone	300-100
3D print	25
TOTAL	2583,66-1745,66
TOTAL without programmes	639,66-439,66
TOTAL without mobile phone and programmes	339,66

Table 7.2. Budget Drone with Localization System

Material	Cost approx (€)
Adafruit Feather STM32F405	40
Frame DJI F450	25
DJI motors and propellers kit	75
ESC kit	25
MPU-6050	2
NEO-M8N GPS module	36
USB-C cable	4
UBEC Cable	2.5
Battery Lipo 3S 2200mA	17.5
XT60 cables	12
Landing Gear for drone	11
FS-I6X radio kit	37
Circuit ELEGOO Plate (not the whole kit)	3
Others (screws, cables …)	5

TOTAL	295
TOTAL with location system (considering basic needed materials)	573

Considering this amount of cost, the average cost on the system can be reduced drastically using cheaper and mass production, though some other components to achieve higher performance can be more expensive.

- RoHS compliant ?
  Arducam Stereo Hat and BN 880 show that are not RoHS compliant, or at least I wasn't able to find it. Any other material used in this project was compliant.

Another side of this study is the electrical consumption and the Co2 produced by this consumption. Supposing 0,246 kgCO2/kWh and 0,0786 €/kWh (in Spain, data from 2019), the following results are being held (Table 7.3.).

Table 7.3. Co2, consumption and cost

Material	**Consumption (approx)**	**Hours of consumption (approx)**	**Cost approx (0.0786 €/kWh)**	**Co2 produced approx (0.246 kg/kWh)**
Laptop	80 W	+500 h	3.144	9.84
Raspberry pi and STM32	5 W	+60 h	0.023	0.074
Soldering	20 W	+16 h	0.025	0.08
		TOTAL:	3.19 €	10 kg

This project hasn't considered CO2 or any other contaminant production in the construction of the drone and system materials. And it has not been contemplated the animal's ambiental dangers, especially by drones.

7.2. Social impact and future expectations

This project has been developed in a simple form, in order to demonstrate the capabilities of having a drone/robotic localization system in self and outer positioning. In case of having better hardware and a more efficient software, the results will be completely different, having a much better performance.

Furthermore a multidrone (swarm) stereo camera system and high resolution cameras could measure farther distances with better precision. Using a single camera in each drone, making a stereo system using a swarm in multiple positions, an external agent can estimate the objective position, Fig 7.1. However, sensor capabilities and self position knowledge are crucial.

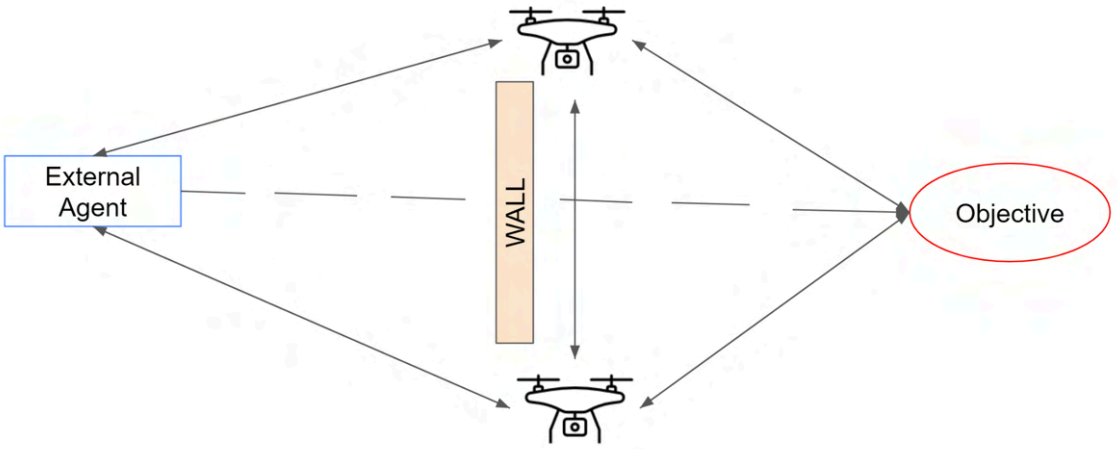


Fig. 7.1 Swarm stereo system

Considering a stereo system with a higher performance, its benefits could fit in:

- Architecture:
  Terrain/location study, building analysis, 3D virtual scenario reconstruction, construction analysis, in drone and first perspective.

- Rescue operations:
  Location of objectives and rescue analysis in dense and high zones in drone and first perspective.

- Military operations:
  Locating targets at greater distances, without the use of lasers, with the location in drone and first perspective.

- Leisure activities:
  Scanning scenarios for 3D virtual scenarios for many applications like game design.

CHAPTER 8. CONCLUSIONS

Completing this work a few things have to be said. This project is a mere approximation, research and creation of localization systems that can be used on multiple vehicles such as drones.

- Concerning hardware and software:

The biggest limitation found on the whole study is the hardware, better hardware and software can really make the difference. However as said before this project does not consist of making the best system, it only consists of a development of a system that can be operable.

Better hardware would be essential to do further and more efficient studies, like higher quality cameras and GNSS receiver. On the role of the software, the extended use of Python or C++ instead of Matlab and parallel or multi-thread performance can be beneficial.

- Concerning multi-dispositive activity:

Furthermore on the development of multi dispositive management, multi node and iterative parallel work can be applied with a useful purpose.

- Concerning my personal opinion:

This work gave me many headaches, I already know many but many times how to not do the things. And I just learned to not trust or rely on hardware systems as I did before.

One of the most concerning factors that I have found is the test and the creation of content which has no source of information or rarely information is found.

Another constraint of this project is to find out that the test can be a little bit complex due the multiple dispositives in a big extension and not all sites are well prepared to hold this scheme, specially for third persons.

The whole objective of this project couldn’t be reached due to the time factor; however, most of them are met, except control the drone using a GNSS receiver, properly.

Finally I can say that the development of this project chat GPT has never been used or any other AI (object detectors are not counted).

I apologize for the inconvenience of any not mentioned source or any error committed in the redaction and production of this work.

Every document created can be found in the following link:
https://github.com/AlexBoa45/Bachelors_Thesis_Alejandro_Boadella.git

REFERENCES

[1] ‘Brokking.net - Project YMFC-32 autonomous - The STM32 Arduino autonomous quadcopter - Home.’ Accessed: Jul. 24, 2024. [Online]. Available: http://www.brokking.net/ymfc-32_auto_main.html#8.2

[2] ‘memoria.pdf’. Accessed: Jul. 24, 2024. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/391713/memoria.pdf?sequence=2&isAllowed=y

[3] ‘Simultaneous localization and mapping’, *Wikipedia*. Mar. 02, 2024. Accessed: Mar. 21, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Simultaneous_localization_and_mapping&oldid=1211448530

[4] Ahmed, ‘The Types of SLAM Algorithms’, Medium. Accessed: Mar. 21, 2024. [Online]. Available: https://medium.com/@nahmed3536/the-types-of-slam-algorithms-356196937e3d

[5] D. Kiss-Illés, C. Barrado Muxí, and E. Salamí San Juan, ‘GPS-SLAM: an augmentation of the ORB-SLAM algorithm’, *Sensors*, vol. 19, no. 22, pp. 1–22, Nov. 2019, doi: 10.3390/s19224973.

[6] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, ‘ORB-SLAM: A Versatile and Accurate Monocular SLAM System’, *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, doi: 10.1109/TRO.2015.2463671.

[7] R. Mur-Artal and J. D. Tardos, ‘ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras’, *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.

[8] ‘Iterative closest point’, *Wikipedia*. Feb. 22, 2024. Accessed: Mar. 24, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Iterative_closest_point&oldid=1209642177

[9] J. Zhang, Y. Yao, and B. Deng, ‘Fast and Robust Iterative Closest Point’, *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2021, doi: 10.1109/TPAMI.2021.3054619.

[10] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, ‘ORB: an efficient alternative to SIFT or SURF’, presented at the Proceedings of the IEEE International Conference on Computer Vision, Nov. 2011, pp. 2564–2571. doi: 10.1109/ICCV.2011.6126544.

[11] D. Tyagi, ‘Introduction to ORB (Oriented FAST and Rotated BRIEF)’, Medium. Accessed: Mar. 25, 2024. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, ‘You Only Look Once: Unified, Real-Time Object Detection’, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[13] J. Redmon and A. Farhadi, ‘YOLOv3: An Incremental Improvement’, Apr. 08, 2018, *arXiv*: arXiv:1804.02767. Accessed: Apr. 06, 2024. [Online]. Available: http://arxiv.org/abs/1804.02767

[14] P. Dollar, R. Appel, S. Belongie, and P. Perona, ‘Fast Feature Pyramids for Object Detection’, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, pp. 1532–1545, Aug. 2014, doi: 10.1109/TPAMI.2014.2300479.
[15] ‘NMEA 0183’, *Wikipedia*. Jun. 12, 2024. Accessed: Jul. 26, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=NMEA_0183&oldid=1228696958
[16] ‘GPS - NMEA sentence information’. Accessed: Jul. 28, 2024. [Online]. Available: https://aprs.gids.nl/nmea/
[17] ‘Haversine formula’, *Wikipedia*. Jul. 18, 2024. Accessed: Jul. 28, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=1235273703
[18] ‘Distance’, *Wikipedia*. Nov. 21, 2023. Accessed: Feb. 01, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Distance&oldid=1186136665
[19] ‘Taxicab geometry’, *Wikipedia*. Jan. 23, 2024. Accessed: Feb. 02, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Taxicab_geometry&oldid=1198273976
[20] ‘Length measurement’, *Wikipedia*. Nov. 09, 2023. Accessed: Feb. 01, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Length_measurement&oldid=1184225945
[21] ‘Radar Block Diagram.pdf’. Accessed: Feb. 03, 2024. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/340878/Radar%20Block%20Diagram.pdf?sequence=1
[22] T. G. Bergman, ‘DESIGN FOR A LASER RANGEFINDER’.
[23] ‘Time-of-flight camera’, *Wikipedia*. Jan. 16, 2024. Accessed: Feb. 03, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Time-of-flight_camera&oldid=1196103867
[24] ‘Laser Range Gating - Long Range & Obscurants | Sensors Unlimited’. Accessed: Feb. 04, 2024. [Online]. Available: https://www.sensorsinc.com/applications/military/laser-range-gating
[25] ‘Interferometry’, *Wikipedia*. Jan. 24, 2024. Accessed: Feb. 05, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Interferometry&oldid=1198663150
[26] ‘Stereo - Camera-wiki.org - The free camera encyclopedia’. Accessed: Feb. 06, 2024. [Online]. Available: http://camera-wiki.org/wiki/Stereo
[27] ‘Stereo camera’, *Wikipedia*. Sep. 12, 2023. Accessed: Feb. 05, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stereo_camera&oldid=1175086591
[28] ‘Stereolabs | AI perception for automation’. Accessed: Feb. 06, 2024. [Online]. Available: https://www.stereolabs.com/
[29] ‘High Resolution Stereo Camera’, *Viquipèdia, l’enciclopèdia lliure*. Jan. 11, 2022. Accessed: Feb. 05, 2024. [Online]. Available: https://ca.wikipedia.org/w/index.php?title=High_Resolution_Stereo_Camera&oldid=29197523
[30] vision team, ‘What is a stereo vision camera?’, e-con Systems.

Accessed: Feb. 10, 2024. [Online]. Available: https://www.e-consystems.com/blog/camera/technology/what-is-a-stereo-vision-camera-2/
[31] ‘What is Structured Light Imaging?’, MoviMED. Accessed: Feb. 10, 2024. [Online]. Available: https://www.movimed.com/knowledgebase/what-is-structured-light-imaging/
[32] ‘Fidler - Depth from Two Views Stereo.pdf’. Accessed: Feb. 08, 2024. [Online]. Available: https://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf
[33] ‘Fidler - Stereo Epipolar Geometry for General Cameras.pdf’. Accessed: Feb. 09, 2024. [Online]. Available: https://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture13_hres.pdf
[34] R. Hartley and A. Zisserman, ‘Multiple View Geometry in Computer Vision, Second Edition’.
[35] ‘Pinhole camera’, *Wikipedia*. Jan. 21, 2024. Accessed: Feb. 11, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pinhole_camera&oldid=1197677006
[36] S. Fidler, ‘Cambridge University Press, 2003’.
[37] ‘lecture09.pdf’. Accessed: Feb. 11, 2024. [Online]. Available: https://www.cs.rice.edu/~vo9/vision/slides/lecture09.pdf
[38] S. Fidler, ‘Back to the Homography: The Why’.
[39] ‘What is Artificial Intelligence (AI) ?’ Accessed: Feb. 23, 2024. [Online]. Available: https://www.ibm.com/topics/artificial-intelligence
[40] ‘Artificial intelligence’, *Wikipedia*. Feb. 21, 2024. Accessed: Feb. 23, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=1209275894
[41] T. Jo, *Deep Learning Foundations*. Cham: Springer International Publishing, 2023. doi: 10.1007/978-3-031-32879-4.
[42] F. M. Salem, *Recurrent Neural Networks: From Simple to Gated Architectures*, 1st ed. Cham: Springer International Publishing, 2022. doi: 10.1007/978-3-030-89929-5.
[43] K. Grauman and B. Leibe, ‘Lecture 14, Sanja Fidler’.
[44] ‘Rectificador (redes neuronales)’, *Wikipedia, la enciclopedia libre*. May 02, 2023. Accessed: Mar. 11, 2024. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Rectificador_(redes_neuronales)&oldid=150909789
[45] ‘Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville (z-lib.org).pdf’. Accessed: Mar. 12, 2024. [Online]. Available: http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf
[46] R. Gandhi, ‘R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms’, Medium. Accessed: Mar. 12, 2024. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
[47] ‘El bus SPI – Prometec’. Accessed: Mar. 09, 2024. [Online]. Available: https://www.prometec.net/bus-spi/
[48] ‘I^2C Primer: What is I^2C? (Part 1) | Analog Devices’. Accessed: Mar. 09,

2024. [Online]. Available: https://www.analog.com/en/resources/technical-articles/i2c-primer-what-is-i2c-part-1.html

[49] ‘UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter | Analog Devices’. Accessed: Mar. 09, 2024. [Online]. Available: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

[50] ‘CAN bus’, *Wikipedia*. Feb. 24, 2024. Accessed: Mar. 09, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=CAN_bus&oldid=1210039147

ANNEX 1: DISTANCE MEASUREMENT

1.1. Length measurement

Since ancient times people used to express the size, amount or degree of things. The standardization of the units took and nowadays takes an important role. Distance measurement is a numerical or occasionally qualitative measurement of how far apart objects or points are [18].

There are different types of distances, in physics and geometry some of them are:

- Euclidean distance:
  Straight line in physical space between 2 points, it is computed using the Pythagorean theorem, as shown in the following formulas. The first formula uses x, y, z as a 3 dimensional space while the second formula, P and Q are different points in n-dimensional real coordinate space.

$$d = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} \quad \text{OR} \quad d(P, Q) = \sqrt{\sum_{i=1}^{n} (Q_i - P_i)^2} \quad \textbf{(A-1.1)}$$

- Manhattan/Taxicab distance:
  Shortest distance in a grid that can be achieved. In the following figure we can observe that all Manhattan distances have the same length. [19]

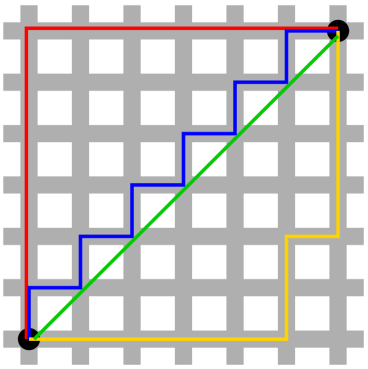


Fig. A-1.1 Green line → Euclidean, Others → Manhattan [18]

$$d = |\Delta x| + |\Delta y| + |\Delta z| \quad \text{OR} \quad d(P, Q) = \sum_{i=1}^{n} |Q_i - P_i| \quad \textbf{(A-1.2)}$$

- Chebyshev distance:
  Known as the chessboard distance, it measures the distance between two points taking the biggest difference among the coordinates.

$$d = max(|\Delta x|, |\Delta y|, |\Delta z|) \quad \text{OR} \quad d(P, Q) = max(\left|Q_i - P_i\right|) \qquad \textbf{(A-1.3)}$$

These distances are susceptible to surface, relativist and other effects.
Other distances not mentioned are Angular, Minkowski, … distances.

1.1.1. Contact devices

As simple as it is, a physical tool used to determine the length, mainly with marks. The most common, the ruler (using the meter unit). Other contact devices are for example:

- Gauge blocks: to calibrate tools, such as calipers, micrometers, etc.
- Caliper: to measure dimensions, usually holes and diameters.
- Feeler gauge: to measure the width of something.
- Graticules: to measure the optical scale of a microscope.
- Opisometer: to measure curved line length on a map.
- Many more.

1.1.2. Non-contact devices

1.1.2.1. Based on time-of-flight

Based on the calculation of the response time to calculate the approximate distance. [20]

$$Length = \frac{v \cdot \Delta t}{2} \qquad \textbf{(A-1.4)}$$

Using v as the velocity of the signal.

- Radar:
  Basically consist of a radio wave sender, the bounced radio waves are received by a receiver which will observe the time that took all the operation determining the object distance.

  There are many types of radar. Mainly they are splitted into “primary radars”, those which are non cooperative, and then “secondary radars”, those which are cooperative. The main difference between them, is that the object who receives the radio wave in secondary radar will send a signal in response to the ground radar with information.

  Inside “primary radar”, the most important radar in distance measurement, there are different types, continuous (CW) and pulse radars. Its name says it all. Inside continuous radars we’ve got modulated or unmodulated radars, and inside pulse radars we’ve got MTI radars or Doppler radars.

The following formula is a simple radar equation to determine the maximum range that can be achieved, many factors are not taken into account due its simplicity.

$$R_{max} = \sqrt[4]{\frac{P_t \cdot G^2 \cdot \lambda^2 \cdot \sigma}{P_s \cdot (4\pi)^3}} \qquad \textbf{(A-1.5)}$$

Where:

$P_t = Transmitted\ power\ (W)$

$P_s = Minimum\ detectable\ signal\ (W)$

$G = Antenna\ gain\ (send\ and\ receive)$

$\lambda = Wavelength\ (m)$

$\sigma = Target\ radar\ cross\ section\ (m^2)$

Radar losses are included inside the minimum detectable signal.

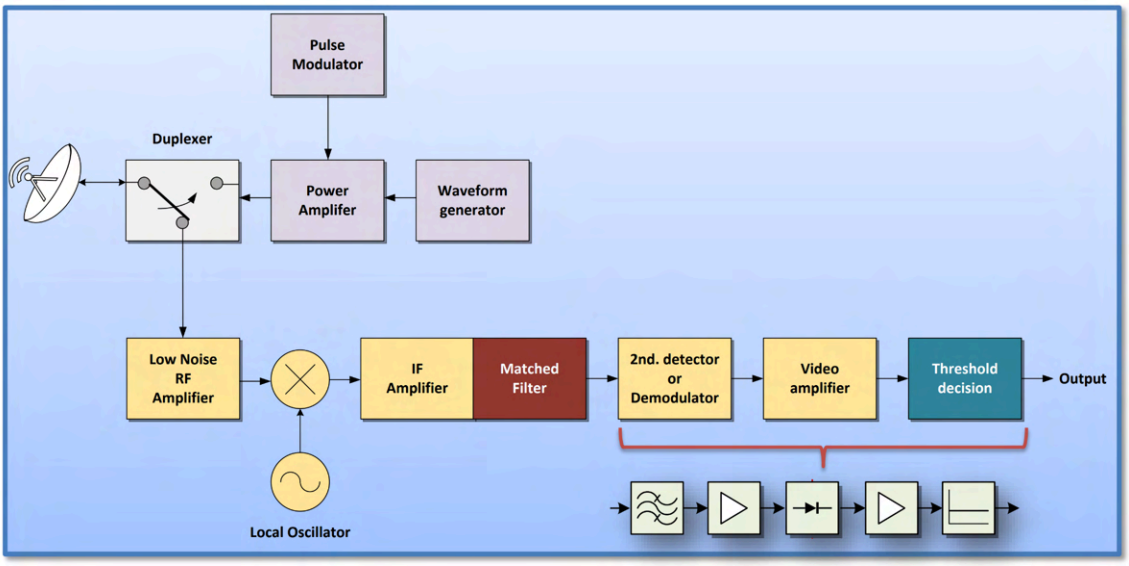


Fig. A-1.2 Diagram block of a Pulsed radar [21]

- Laser rangefinder:
  Pretty similar to radar, a laser beam is projected into an objective, when the reflected laser is received, the time of flight can determine the distance. Not appropriate for high precision measurements. Lasers can give the advantage of having a fixed wavelength, narrow beams, and low noise. Problems can be the clutter, poor collimation, etc.[22]

- Time-of-flight camera:
  A broader group of sensors, one of the most famous for example: LIDAR.

  There are different kinds and technologies [23]:

- Phase changers with RF-modulation:

  Multiple signals with shifts in the phase, can enable a measurement in distance with the autocorrelation process.

  $$Phase = arctan\frac{(A1-A3)}{(A2-A4)} \quad \text{AND} \quad Distance = \frac{(c \cdot Phase)}{(4 \cdot \pi \cdot F_m)} \qquad \textbf{(A-1.6)}$$

  Being A1, A2, A3, A4 optical signals with phase shift 90º and Fm the mean frequency in the IR band, c the velocity of light.

- Range gated imagers:

  While a sender sends a series of pulses, a receiver opens and closes the shutter with the same frequency as the sender, blocking part of the light. Depending on the amount of light blocked, received and the time, it is possible to calculate the distance. For instance the interval of the gate gives the user how near the light has been sended.

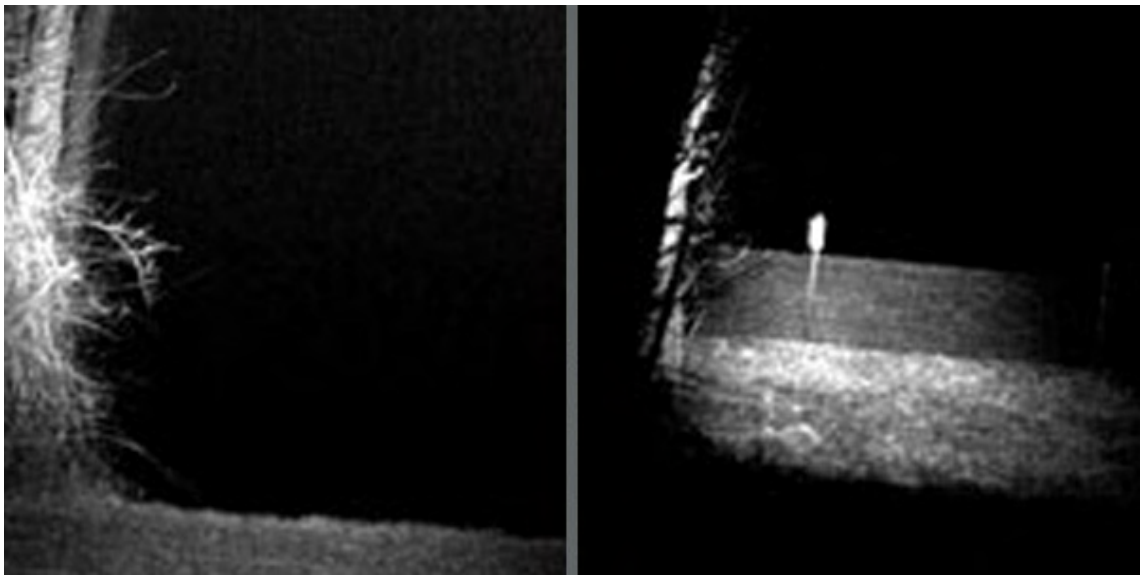


Fig. A-1.3 Gate open early (Left), Gate open after a period of time (Right) [24]

- Direct ToF imagers:

  Like Laser rangefinders, measure a light pulse time reflected on any surface, recording the scatter of points a 3D mapping can be done.

  These dispositives are simple, fast and algorithm efficient with <1% error on the measured distance. However, background light, interference from

other sensors or even multiple reflections can affect the reliability of these sensors.

- Lidar (Laser imaging detection and ranging) (inside ToF camera): Similar as the Laser rangefinder, but with the particularity of making a 3D pulsed scanning of the surroundings. It relies on differential GPS and inertial navigation. Usually mounted on an aerial vehicle, and using 600 nm to 1550 nm wavelengths. Two types, the “incoherent” one reckons the amplitude of the signal while the “coherent” one uses doppler properties on detection.

- Satellite Navigation:
  To sum up, the satellite emits a broadcast information about the self-position, with 4 satellite signals it is possible to calculate the receiver position. Once the signal is received, a correlation, a doppler and a delay acquisition shall be done, later the GPS receiver will precise doppler and delay values to have a better measurement and data recognition. Almanac, relativist effects, doppler effects, correlation, time of light and many more can affect these measures. The error precision of satellites is very variable, changing from meters to less of a pair of centimeters. There are many constellations, services and systems that rely and support satellites.

- Substantial more.

1.1.2.2. Ranging without time-of-flight

- Interferometer:
  There are many kinds of interferometers, a typical one is a Michelson interferometer. The use of a laser into a beam splitter (BS), then the use of corner cubes (CC), in order to have a phase shift depending on the object distance as seen in the Fig A-1.4.

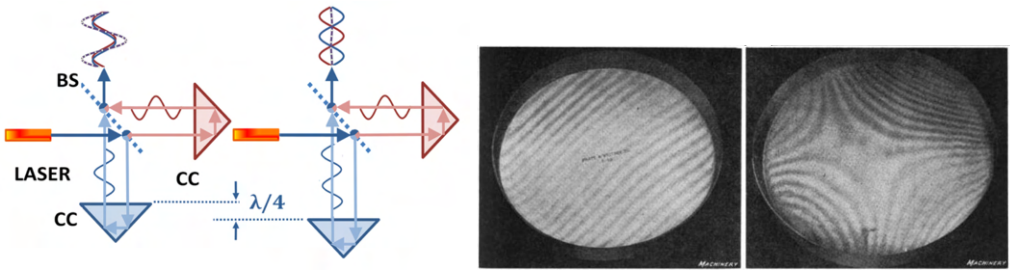


Fig. A-1.4 Left→ Michelson Interferometer, Right → Comparison of a flat and curved surface (another type of interferometer) [25]

- Stadimeter: similar as the sextant, but to determine an estimation of the range of an object. Comparing the size of the object.

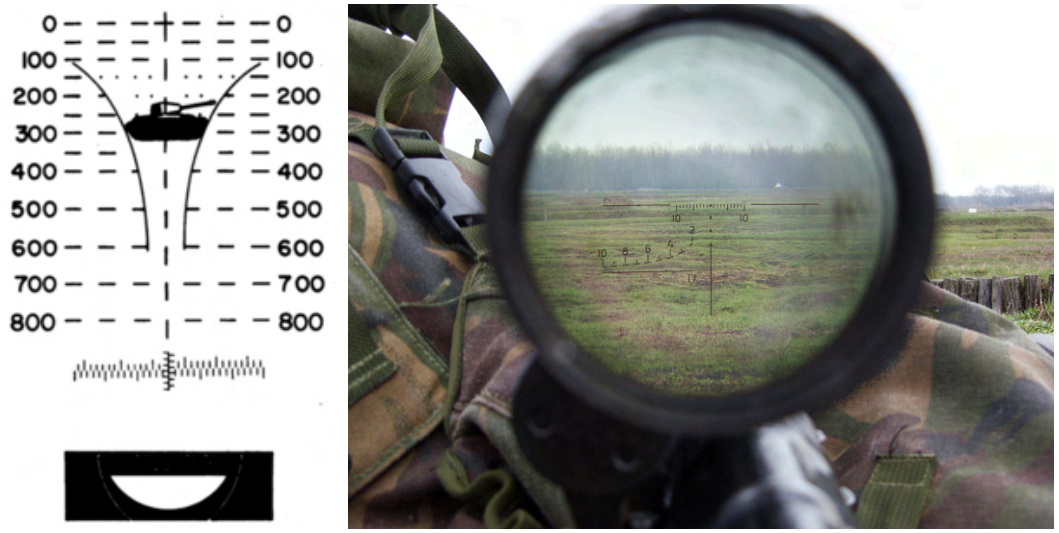


Fig. A-1.5 Left → Stadimeter marks, Right → inside view of the optic

- Diffraction methods:
  The use of diffraction patterns to determine scales from nanometers to millimeters in order to measure particles. One kind of these sensors is for example a He-Ne laser (Fraunhofer theory), where there is a proportional scattered light depending on the size of the particle. Measuring the laser beam angle reflected projected in the particle thus can lead to a size measurement.

- Nuclear Overhauser effect spectroscopy (NOESY):
  As far as i can say, the use of magnetics fields and radiofrequency to determine the length and elucidate atomics molecular structures. With a precision of distance up to 5 Angstrom between 2 protons in proximity.

- Parallax, Cepheids, Supernovae (Redshift and Hubble Law):
  There are different methods to estimate/calculate universe distances, for example Parallax methods consist in a series of observer superpositions like stereo cameras to estimate the position trying to make an isosceles triangle with the vertex being the objective.

- Many more such as corded aperture, structured light, light triangulation, inductive and magnetic sensors, etc.

1.2. Stereoscopic Camera

This will be the main sensor of this project, but what makes this sensor suitable for this research?

1.2.1. What it is and history

Ten years after the first photograph camera, the first stereo camera was created by the Scottish Sir David Brewster and Parisian Jules Dubosq. This camera consists of two cameras displaced between each other. Taking photos from different perspectives can lead to a depth vision imitating human eyes. These cameras usually are separated 6,5 cm as human eyes tend.

In the past, these kinds of cameras weren't so popular except during two periods. Between 19th and 20th Jules Richard popularized these cameras, even in WWI half of the amateur cameras were stereo. And in the 1950s by the stereo realism.

Fig. A-1.6 Kodak Stereo Camera started in 1954 [26], [27]

This not contact device without ToF measurement can be presented in different forms:

- Macro Stereo: The two cameras are very close, giving a better look to closer objects.

- Hyperstereo: The two cameras are quite far away, giving a better look to farther objects.

- Lenticular Stereo: Basically combines different images stripes into a single image.

1.2.2. Actual uses

Nowadays these cameras are widely used; however, they aren't widely known by the population.

- Robotic:
  These cameras can be used to give a 3D situational awareness of unmanned robotic systems. For example: StereoLabs [28]

- Entertainment:
  Many films and many mobile phones have these cameras. For example: iPhone-15 Pro.

- Military purposes:
  Widely used on ground vehicles, unmanned or manned vehicles. On aerial vehicles it's not so common, preferable laser systems on air. For example: Bradley infantry combat vehicle stereo camera 360º awareness.

- Topography and land study:
  Scanning of the environment, making a 3D map, seeing depth. Used on ground stations and in aircraft.

- Medicine:
  Pretty used on special microscopes. Microsurgery, endoscopy procedures, CT scan (computed tomography scan), NMR (Nuclear magnetic resonance), etc. For example: Zeiss.

- Molecular engineering:
  Used in the creation station of new complex molecules, for example in the Widener University.

- Outer space study:
  Used by satellites to map earth and unmanned vehicles. For example: HRSC (high resolution stereo camera) [29]

- Others.

1.2.3. Advantages and drawbacks

I will consider only the advantages and drawbacks inside this project.

- Advantages:
  - More information to analyze when a photo is taken.
  - No aiming is necessary as laser rangefinder needs.
  - Cheaper than a LIDAR.
  - It doesn't jeopardize the objective as it only is a receiver, not a sender, as lasers.

- Drawbacks:
  - Synchronism is essential, 2 photos shall be taken at the same time.
  - Baseline between cameras can affect depth estimation.
  - High computer performances are needed to estimate distances.
  - Camera resolution is an essential perk on depth estimation.
  - It is less accurate than a laser method.

1.2.4. Sensor in use

So in order to study stereo vision the camera to start was: Arducam Stereo Vision Kit 8MP.

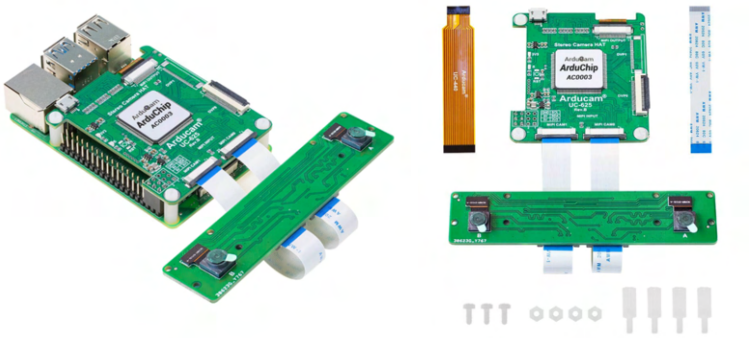


Fig. A-1.7 Arducam Stereo Vision Kit 8MP

- Details:
  - 2 Optic sensors IMX219 8MP
  - Pi HAT board 65 x 65 mm 16g
  - Synchronized photographs
  - ArduChip on board for all processes
  - 62.2° (H) x 48.8° (V) degrees fixed focus
  - Without IR filter
  - Focal length 3,04 mm
  - 80 mm baseline aprox
  - 4k@21fps, 1080p@30fps, 720p@120fps
  - Focus distance +200 mm
  - 300mA each camera separated
  - Cost: 135 €

- Fears about the purchase:
  Its baseline is too short for long distance measurement. The best way to ensure its purpose is to be tested.

ANNEX 2: DEPTH ESTIMATION

2.1. What is depth estimation?

2.1.1. Definition

Stereo depth estimation is a particular form of measurement characterized by the similarity and inspiration from human eyes, focused on range estimation by the correlation and position tracking from different sources of 2D images.

2.1.2. Types

There are different kinds of stereo cameras and techniques [30].

2.1.2.1. Depending on the perception

- Passive stereo:
  Basically it consists of a set of cameras recording images from different perspectives.
  - Advantages: Low cost and performs fine during the day period.
  - Disadvantages: Non-texture scenes and low light can degrade the effectiveness.

- Active stereo:
  Basically it consists of a set of cameras and a structured light system, a laser, etc. There are many types inside this section.
  - Advantages: Low light, non-texture performs quite well. It can be implemented with many other technologies.
  - Disadvantages: Cost, during the day it does not perform really well.

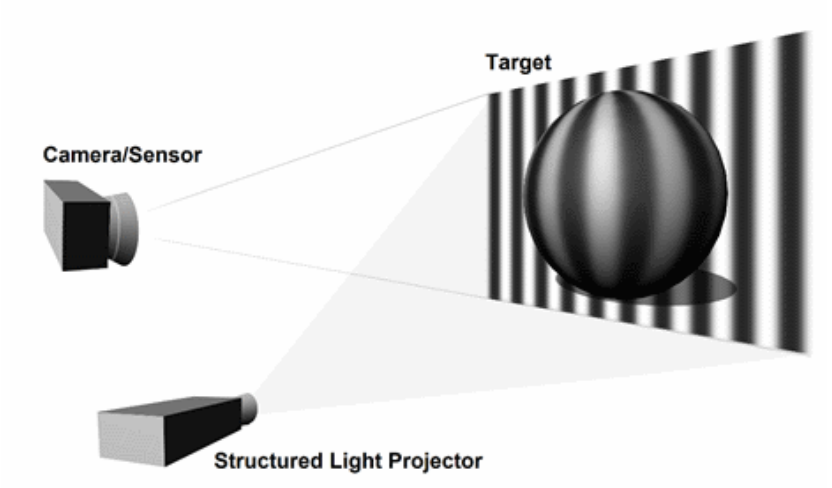


Fig. A-2.1 Structured light method [31]

2.1.2.2. Depending on the camera position

There are 2 kinds of the system, the following ones:

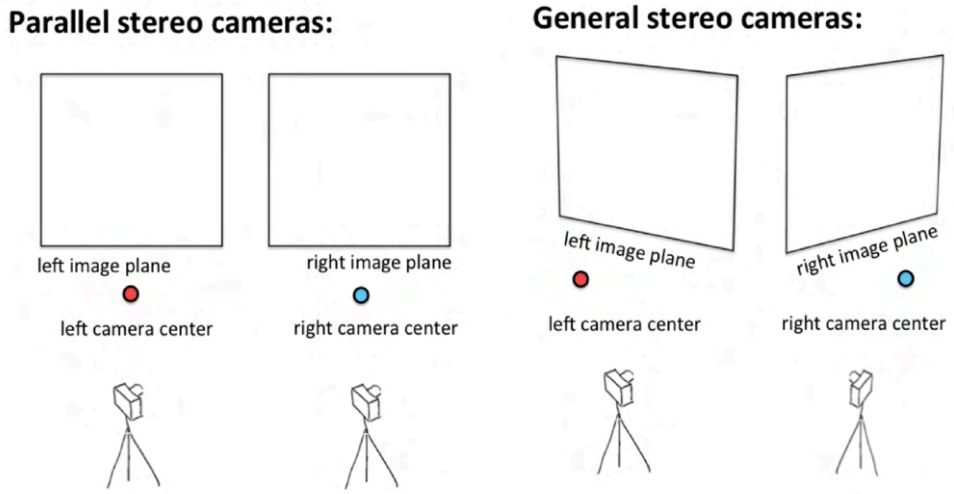


Fig. A-2.2 Types of stereo systems depending on the position [32]

2.2. Parallel stereo analysis

Basic plain model of analysis that I can start from. And most commonly used nowadays on robotics. Everything, supposing an ideal case without the need of rectification and without calibration.

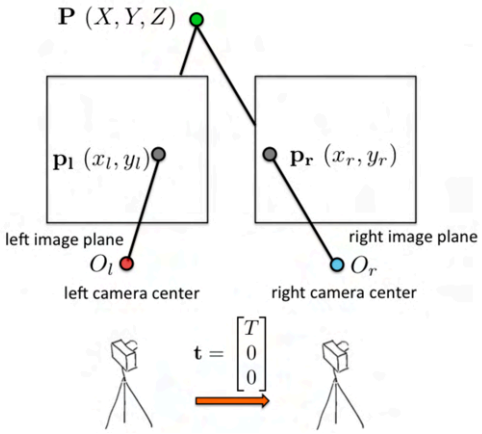


Fig. A-2.3 Parallel stereo camera depth estimation [32]

Considering a parallel set up, we want to know the position P(x, y, z) as shown in Fig A-2.3.

A parallel system can give the system a clue, having parallel images can give us that the Y-coordinates of the left and right images are the same, leading us to a pythagoras problem in an easy way, Fig A-2.4.

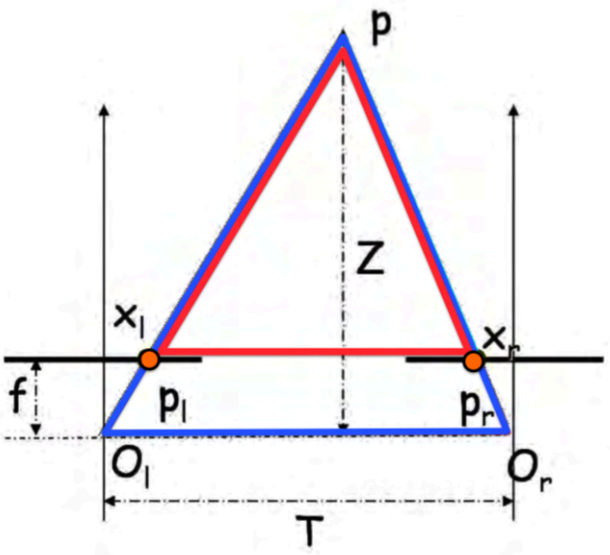


Fig. A-2.4 Depth estimation [32]

Clarification:

$P = Point\ to\ estimate\ depth$
$P_l = Pixel\ point\ on\ left\ image$
$x_l = Position\ of\ the\ left\ pixel\ in\ x - coordinates$
$P_r = Pixel\ point\ on\ left\ image$
$x_r = Position\ of\ the\ right\ pixel\ in\ x - coordinates$
$T = Baseline\ (distance\ between\ cameras)$
$f = Focal\ length$
$P_r = Pixel\ point\ on\ left\ image$
$O_l = Optical\ center\ left$
$O_r = Optical\ center\ right$
$Z = Depth$

Using similar triangles theorem we can reach the following formula:

$$Z = \frac{f \cdot T}{x_l - x_r} \qquad \textbf{(A-2.1)}$$

What can we get from this formula?
The bigger baseline and bigger focal length, the farther we can estimate.
Having a great resolution and a great baseline can also affect the denominator.

The concept of depth estimation is well understood, but how can we match 2 semi-identical points in 2 different images? Here the problems are starting.

There's no simple way to compute. The typical form is to use patch/filter method comparing 2 patches (one from left and one from the right image) at the same time and plotting a disparity function. Clarification: disparity means the difference between images in x position.

As seen in Fig A-2.5, a patch from the left image has been considered the reference, in order to calculate point depth. In the right image different patches are compared to the reference one till the red point. The red point is the x-position of the reference patch on the left image. Why that? Due to geometric conditions, no possible matching can be achieved at the right zone of the red dot. Remember that Y-coordinates shall be the same in every single patch on both images.

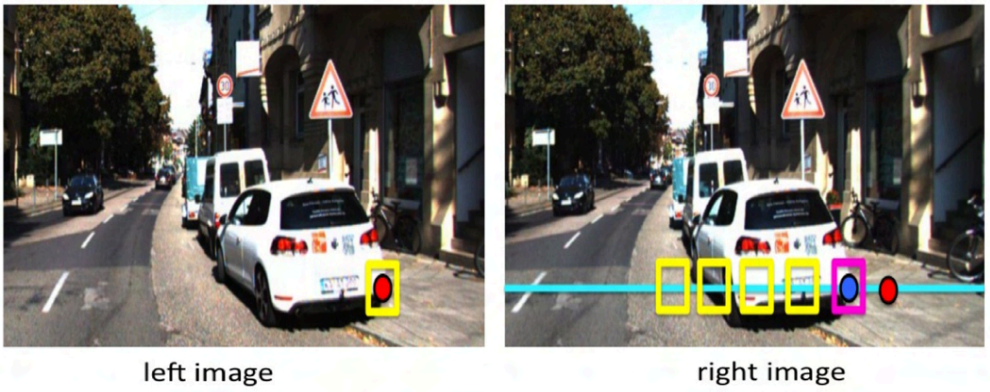


Fig. A-2.5 Depth estimation images [32]

How do we mathematically compare these patches? Supposing that I is a patch [32]:

- SSD (sum squared differences):
  When this method is computed, the best good looking result will be the minimum result.

$$SSD(I_l, I_r) = \sum_{x}^{n}\sum_{y}^{m} (I_l(x,y) - I_r(x,y))^2 \qquad \textbf{(A-2.2)}$$

- Normalized correlation:
  When this method is computed, the best good looking result will be the maximum result.

$$NC(I_l, I_r) = \frac{\sum_{x}^{n}\sum_{y}^{m}(I_l(x,y) - I_r(x,y))}{\|I_l\| \cdot \|I_r\|} \qquad \textbf{(A-2.3)}$$

- Many more.

This method shall be done for every patch in the image in the same Y-coordinate, many times. Patches can affect the result. The smaller the patch, the more detailed estimation we get, however, more time will take and much more probably of having an estimation wrong in non-texture zones (noise). If the patch is bigger, much less time, a better estimate we get, but less detailed scan we get. Furthermore, patches can be presented in different forms such as rectangular, etc.

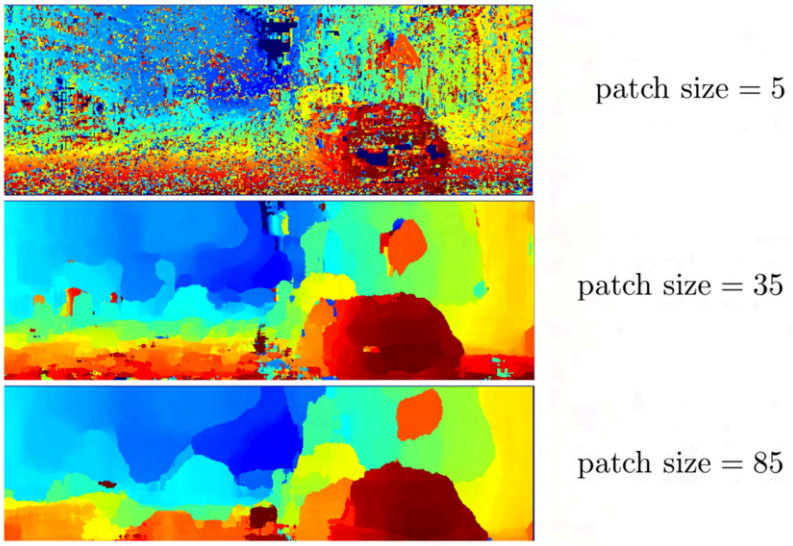


Fig. A-2.6 Patch differences [32]

This problem takes too much complexity due its minimization or maximization, being a NP-completeness problem (nondeterministic polynomial-time completeness), many researchers take chances about heuristics approximations making trade-off about optimal, completeness, accuracy, precision and execution time.

Apart from these methods it has been used also neural networks with the objective of taking advantage of it.

2.3. General stereo analysis

Once the concepts of parallel stereo cameras are understood, the next step is to understand the general stereo system, Fig A-2.7.

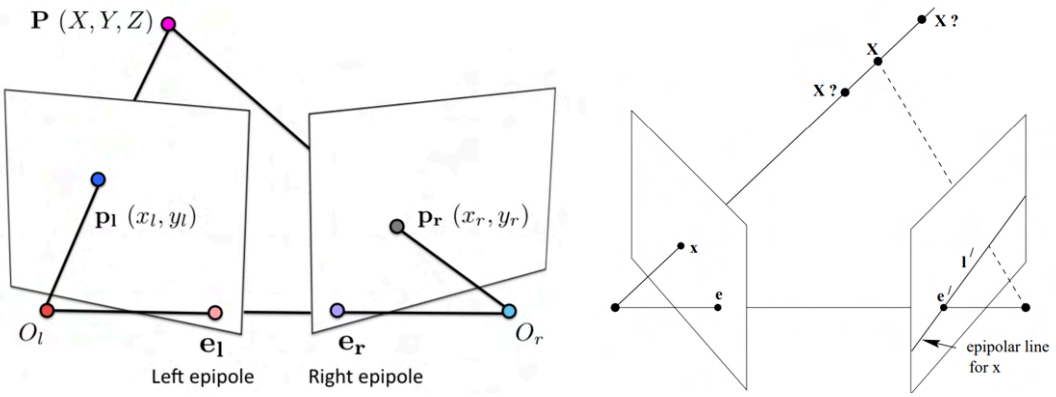


Fig. A-2.7 General stereo system [33], [34]

Same start as parallel stereo but with a difference on the visual angle between cameras. The best approach to a high complexity problem is to make it easier trying to convert the problem into a well known problem, trying to convert it into

a parallel stereo problem. To do so, it is necessary to use epipolar geometry and projection analysis.

As a rough concept, epipolar geometry encompasses epipolar lines and planes, the best form to know what it is, is to see in a figure, for example Fig A-2.7 and Fig A-2.8.

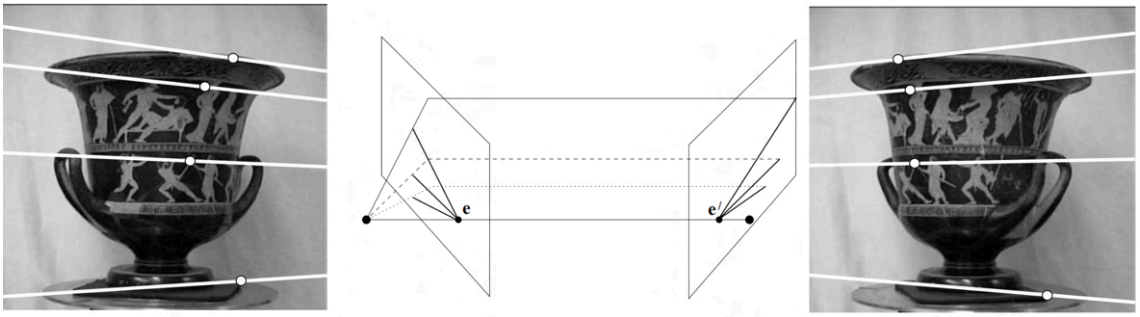


Fig. A-2.8 Epipolar lines and plane [33], [34]

2.3.1. Projection analysis

2.3.1.1. Lens

There are many types of lens, but in fact lenses actuate like a focus, acquiring narrow beam light. The distance from aperture to the object point it's what we call focal length Fig A-2.10.

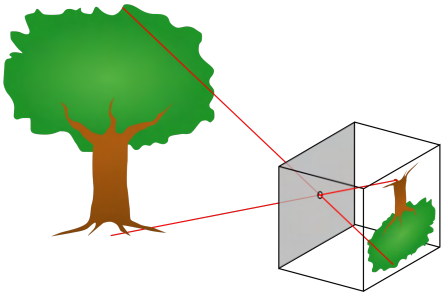


Fig. A-2.9 Pinhole camera scheme [35]

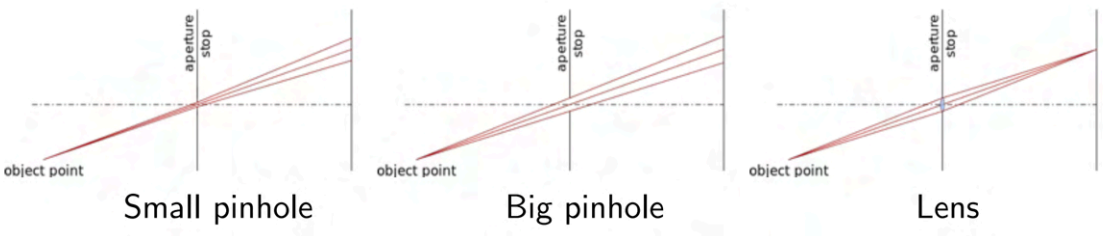


Fig. A-2.10 Pinhole vs lens [36]

The bigger focal length the same scale image we get but more focused and farther the image is. To do the calculations it is better to have the following scene in mind, specially the virtual scheme Fig A-2.11. More information in [36].

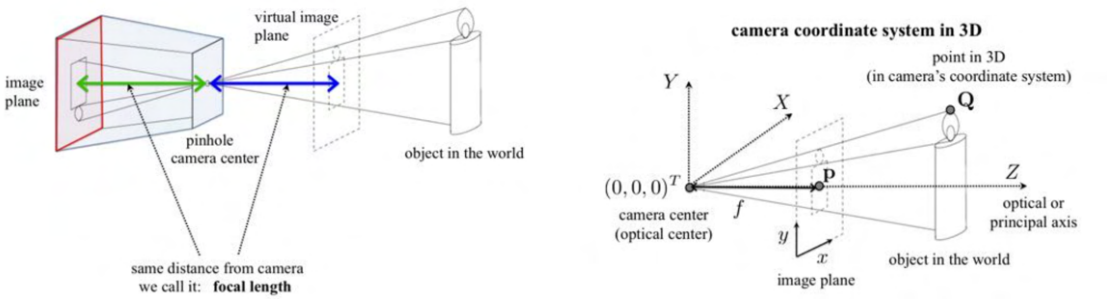


Fig. A-2.11 Scheme projection: overall left, virtual right [36]

Being the pinhole the center point $(0,\ 0,\ 0)^T$. Thus the virtual plane being an orthogonal plane, and with a center called principal point p. The projection line from pinhole to Q will pass through the image plane q point.

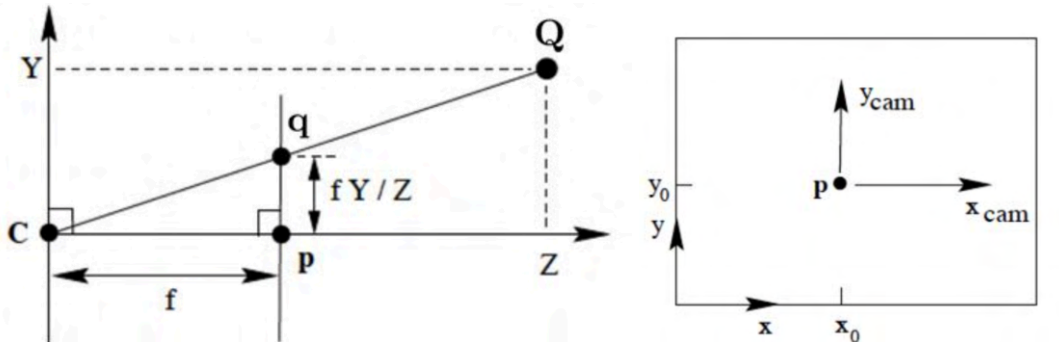


Fig. A-2.12 Scheme projection: 2D scheme left, 2D image plane right [36]

Being $Q = (X,\ Y, Z)^T$, the position from pinhole will be $(\frac{f \cdot X}{Z} + p_x,\ \frac{f \cdot Y}{Z} + p_y,\ f)^T$ and the projection $(\frac{f \cdot X}{Z} + p_x,\ \frac{f \cdot Y}{Z} + p_y)^T$.

From this point, homogeneous coordinates can make it easier, for example: from the projection $(\frac{f \cdot X}{Z} + p_x,\ \frac{f \cdot Y}{Z} + p_y)^T$, the homogeneous coordinates are $(\frac{f \cdot X}{Z} + p_x,\ \frac{f \cdot Y}{Z} + p_y,\ 1)^T$.

Furthermore there are special matrices like K intrinsic calibration matrix or R rotation matrix, T translation matrix that are extrinsic matrices.

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(A-2.4)}$$

2.3.1.2. Projection analysis general stereo

Using Fig A-2.13 as a guidance material. In order to convert (rectify), from general to parallel problem is necessary a fundamental matrix, F.

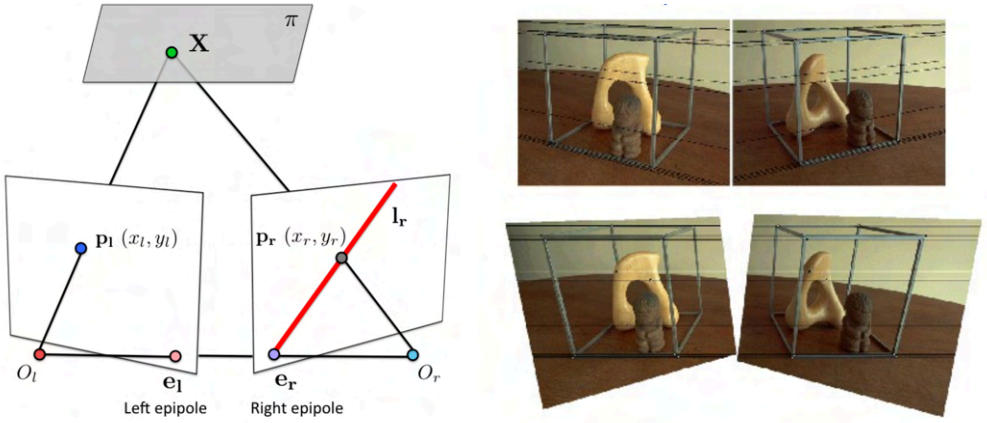


Fig. A-2.13 General stereo scheme [33], [34], [37]

The epipolar line is defined as $l_r = e_r \times p_r$. Using homography we can find that $p_r = H_\pi \cdot p_l$. As a clarification, homography is a technique to reckon a point from a projective plane into another, [36], [38].

This 3x3 matrix is defined as $l_r = F \cdot p_l$, where l_r is the epipole line in the right camera and p_l the pixel in the left camera.

$$l_r = e_r \times p_r = e_r \times H_\pi \cdot p_l = F \cdot p_l \qquad \textbf{(A-2.5)}$$

$$l_r = F \cdot p_l \quad \Leftrightarrow \quad p_r^T \cdot l_r = p_r^T \cdot F \cdot p_l \quad \Leftrightarrow \quad 0 = p_r^T \cdot F \cdot p_l \qquad \textbf{(A-2.6)}$$

How in fact we get the F matrix ?
As a summary of hazardous math, at least 7 matching points between these 2 images shall be considered, and with (Eq. A-2.6) we will get the following matrix (Eq. A-2.7), being able to compute F.

$$\begin{bmatrix} x_{r,1}x_{l,1} & x_{r,1}y_{l,1} & x_{r,1} & y_{r,1}x_{l,1} & y_{r,1}y_{l,1} & y_{r,1} & x_{l,1} & y_{l,1} & 1 \\ & & & \vdots & & & & & \\ x_{r,n}x_{l,n} & x_{r,n}y_{l,n} & x_{r,n} & y_{r,n}x_{l,n} & y_{r,n}y_{l,n} & y_{r,n} & x_{l,n} & y_{l,n} & 1 \end{bmatrix} \mathbf{f} = \mathbf{0} \qquad \textbf{(A-2.7)}$$

Now with F, the projection will be (considering left image as the reference, and P the homogeneous objective coordinates):

$$P_{left} = K \cdot [\, I_{3x3} \mid 0] \cdot P \qquad \text{AND} \quad P_{right} = K \cdot [\, [e_r]_\times F \mid e_r] \cdot P \qquad \textbf{(A-2.8)}$$

Where $[]_\times$:

$$[\mathbf{a}]_x = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \qquad \textbf{(A-2.9)}$$

But we need e_r, that can be computed like: $e_r^T F x_l = 0 \Rightarrow e_r^T F = 0$. Considering the following relation: $l_r = F \cdot x_l$.
As a mere curiosity all calculated before was $F = [T_x] \cdot R$, that uses translation and rotation matrices.

In situations without GNSS coverage, video recording with the use of this technique is pretty useful. Despite being a complex non-linear optimization problem with errors, the errors are usually less than 20 m in accuracy.

Important matrices like calibration matrix, K that use the least square method with image matches, and many others are not explained due its not the purpose of this research, though it's a quite interesting trend.

ANNEX 3: AI

3.1. What is Artificial Intelligence ?

3.1.1. Definition

Called as the intelligence of the machine, is a field of study focused on the development of machine software dedicated to have better performance in areas such as industry, science, etc.

As IBM says: “Artificial intelligence (AI) is technology that enables computers and digital devices to learn, read, write, talk, see, create, make recommendations, and do other things humans do.” [39]

In my personal opinion and as far as i can say, artificial intelligence for me is a mathematical regression method that consists of a series of inputs, outputs and several layers of pulleys connecting these ports. In order to develop/train an AI is to deliver the inputs and outputs adjusting these pulleys repeatedly with many data sources. More or less is like having a function without coefficients only x and y, and trying to guess what these coefficients are, so adjusting pulleys. Once trained, only inputs are delivered and the output will be shown with what was learned on the training.

Much conceptual information has been found and in a detailed explanation, [40].

3.1.2. Actual uses

- Industry:
  Enormous number of applications, such as: energy storage, imperfection detection, intralogistic management, industrial trend analysis and economy, production, etc.

- Generative AI:
  Trending nowadays, has revolutionized in my point of view the concept of artificial intelligence, specially by Chat-GPT and others. These AIs can be unimodal or multimodal, depending on the number of inputs. It can receive, text, images and it will generate, text, code, images, audio, video, and other sort of data. This data can be multipurpose basically.

- Health and medicine:
  In many aspects, for example: tissue identification and detection, molecule approximation, treatment identification, etc.

- Military applications:
  Specially for space, land and airborne applications without rejecting naval warfare. Strategic decision making, simulations, target identification, swarming distribution, security, etc.

- Games, computational software, etc:
  Real time analysis, scenario generation, character development, enhanced graphics, many applications.

3.1.3. Goals

- General intelligence:
  Associated with autonomous learning, the closest approximation is the animal or the human intelligence.

- Social intelligence:
  Any kind of emotions, moods, feelings, anything related with human effective state.

- Perception and language intelligence:
  Sensor data analysis such as microphones, cameras, etc. The most important insight is the data analysis and extraction.

- Learning:
  Artificial intelligence can learn in different ways, supervised (prepared data and analysis), unsupervised (unprepared data), reinforcement learning, etc. The optimization, the data preparation and representation can take the vast majority of complexity of the system.

- Reasoning and knowledge representation:
  Currently reasoning and the representation of these reasons is an unresolved problem in big scales for AI, in order for human understanding.

3.1.4. Techniques

- Artificial neural networks:
  Like the human ones, they try to imitate them. This kind of technique tries to learn about patterns. There are many types, like feedforward networks (FNN) in only one direction whilst recurrent neural networks (RNN) that have memory.

- Deep learning:
  DNN uses several layers of neural networks, giving the user high performance in its purpose.

- Generative pre-trained transformers:
  This technique consists of artificial neural networks trained in large language sets of information. In case the user asks a question, this GPT can answer a prediction to the question; however, it can give false information.

- Classifiers and statistical learning methods:
  There are different methods, such as K-nearest neighbor (KNN), Navier Bayes classifier, Decision Tree, Artificial neural networks, etc. These classifiers, as the name says, try to classify data within a range of categories, like a pattern matching.

- Probabilistic methods:
  Group of methods like Bayesian networks, using probability theory. Many of these methods can be used on Markov models, Kalman filters that can also be used on multi-sensor fusion models.

- Many other techniques.

3.2. Deep Learning Networks Foundations

Multiple Layer Perceptron (MLP), invented in the 50's by Rosenblatt. As a basic concept MLP consists of a set of three layers: input, the hidden and the output layer. This mere concept is the basis of this stage.

Rumelhart in the 80's invented what we call neural networks today. But what are these networks made of?

Despite being a high time consumer in order to train them, they are very high tolerance to noise as well as excellent in performance.

3.2.1. Perceptron

There are various versions of this concept, but I considered the most suitable one to explain from my perspective.

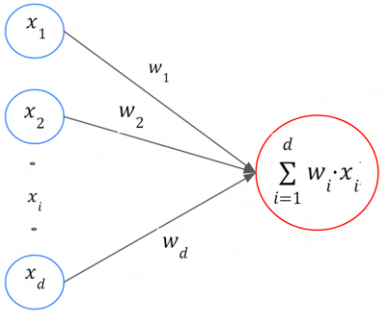


Fig. A-3.1 Perceptron without hidden layer

Starting off from Fig A-3.1, the input is well reckoned by the sum of all inputs multiplied by their connections; however, the output can also be calculated using an activation function (Eq. A-3.0 simplified).

$$y = F(net) = F(\sum_{i=1}^{d} w_i \cdot x_i) \quad \textbf{(A-3.0)}$$

Despite all these layers that can give continuous values, let's say regression networks, classification networks work with output ranges to be feasible in a discretized way.

Considering a loss function where c is the dimensional vector of the output nodes (Eq. A-3.1). This function is used to correct final values using connection weights. And considering a matrix W (Eq. A-3.2) of connection weights, where c (or i) are output nodes and d (or j) are input nodes, (much of this information can be found and has been extracted from [41]). The loss function is not always like this, it can be squared loss, cross-entropy, etc.

Clarification: the i is not the same as before (Eq. A-3.0).

$$L = \frac{1}{2} \cdot \sum_{i=1}^{c} (y_{i_{desired}} - y_{i_{computed}})^2 \quad \text{where} \quad y_{i_{computed}} = \sum_{j=1}^{d} w_{ij} \cdot x_j \quad \textbf{(A-3.1)}$$

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1d} \\ w_{21} & w_{22} & \dots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{c1} & w_{c2} & \dots & w_{cd} \end{pmatrix} \quad \textbf{(A-3.2)}$$

It can be determined that the next weight matrix will be (Eq. A-3.3), with η as a learning rate parameter. In (Eq. A-3.4) the chain rule has been used.

$$W(t+1) = W(t) - \eta \frac{\partial L}{\partial W} \quad \textbf{(A-3.3)}$$

where: $$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_{i_{computed}}} \cdot \frac{\partial y_{i_{computed}}}{\partial w_{ij}} = \dots = -(y_{i_{desired}} - y_{i_{computed}}) x_j \quad \textbf{(A-3.4)}$$

These formulas show that the weight between connections is changed based upon the difference between the target and the local y and the learning rate. In order to have a successful learning, the loss function shall be minimized, but it can fall in local minima.

- What happens when we include hidden layers between input and output? Let's compute only 1 hidden layer.

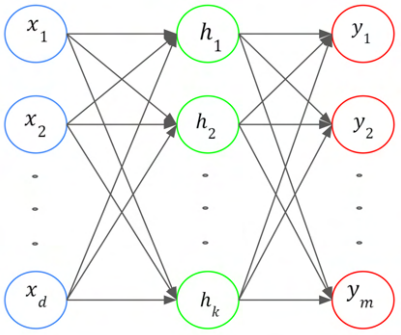


Fig. A-3.2 Perceptron with hidden layer

As seen in the Fig A-3.2, the number of connections reached can be calculated by $(d \times k) + (k \times m)$, d vector of inputs, k vector/number of nodes in hidden layer and m the output. The activation function is also applied in the input values, it is usually a tanh, max, sigmoid, linear function.

The weight matrix for input and only 1 hidden layer is exactly the same matrix as (Eq. A-3.2), but instead c lines of output nodes, k lines of hidden layer nodes.

As an example the output value for h node is the sum of w connections of x input nodes where j are the number of inputs and i the number of hidden nudes (Eq. A-3.5).

$$h_i = F(\sum_{j=1}^{d} w_{ji} \cdot x_j) \qquad \textbf{(A-3.5)}$$

The weight matrix for 1 hidden layer and output is exactly the same matrix as (Eq. A-3.2), instead of d columns it has got k columns of hidden nodes and m lines for output nodes.

Somehow the hidden layer can be as input or as output in the matrices and equations (Eq. A-3.6), the equations are the same, the only thing that changes are the target variables for every pair of layers. In case of having multiple hidden layers it will be the same, only changing variables.

$$y_{i_{computed}} = F(\sum_{j=1}^{k} w_{ji} \cdot h_j) \qquad \textbf{(A-3.6)}$$

- But does the loss function affect MLP?

While the values are computed forwards the weights are adjusted backwards the net. Considering always a 3 layer network. The loss function is the same as before, but the weight update is different.

Between output and hidden layer, the weight matrix is calculated (Eq. A-3.7):

$$W^2(t+1) = W^2(t) - \eta \frac{\partial L}{\partial W^2} \quad \textbf{(A-3.7)}$$

where:

$$\frac{\partial L}{\partial w^2_{ji}} = \frac{\partial L}{\partial y_{i_{computed}}} \cdot \frac{\partial y_{i_{computed}}}{\partial net^2_i} \cdot \frac{\partial net^2_i}{\partial w^2_{ji}} \quad \textbf{(A-3.8)}$$

The net variable means the value given by the node behind that connection, for example for y the net will be $w \cdot h$. To know what net there is, just look for dependencies in the chain rule used in these equations.
This matrix needs the activation function in order to be developed. One often used is the sigmoid function (Eq. A-3.9).

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \quad \textbf{(A-3.9)}$$

In case of using this sigmoid function it will be line (Eq. A-3.10). In case of linear function (Eq. A-3.11). In the case of vectors there will be outer products.

$$\frac{\partial L}{\partial w^2_{ji}} = (y_{i_{desired}} - y_{i_{computed}}) \cdot y_{i_{computed}} \cdot (1 - y_{i_{computed}}) \cdot h_j \quad \textbf{(A-3.10)}$$

$$\frac{\partial L}{\partial w^2_{ij}} = (y_{i_{desired}} - y_{i_{computed}}) \cdot 1 \cdot h_j \quad \textbf{(A-3.11)}$$

Between the hidden layer and the input, the weight matrix is calculated using (Eq. A-3.12). Due w^2 are influenced by w^1, these weights are summed in order to adjust w^1.

$$W^1(t+1) = W^1(t) - \eta \frac{\partial L}{\partial W^1} \quad \textbf{(A-3.12)}$$

where:

$$\frac{\partial L}{\partial w^1_{ji}} = \sum_{k=1}^{m} \frac{\partial L}{\partial y_{k_{computed}}} \cdot \frac{\partial y_{k_{computed}}}{\partial net^2_k} \cdot \frac{\partial net^2_k}{\partial h_i} \cdot \frac{\partial h_i}{\partial net^1_i} \cdot \frac{\partial net^1_i}{\partial w^1_{ji}} \quad \textbf{(A-3.13)}$$

As a clarification k index is the index of the output layer and m is the max number of output nodes in (Eq. A-3.13).

Using the linear function we can find (Eq. A-3.14).

$$W^1_{ji}(t+1) = W^1_{ji} - \eta \sum_{k=1}^{m} x_j \cdot w^2_{ik} \cdot (y_{k_{desired}} - y_{k_{computed}}) \quad \textbf{(A-3.14)}$$

As seen, to calculate the weights of the input layer and hidden layer, the weight between hidden and output layer are used, as well as the function at the output.

Just remember if indexes are not well understood in (Eq. A-3.14), i is the hidden layer, k the output layer and j is the input layer. These formulas can be quite complex but once you see the main idea, they are quite easy to understand.
Another thing to comment on, is that learning at a specific learning rate can be slow. And so in order to improve rapidly there is stochastic gradient descent (Eq. A-3.15).

$$W(t+1) = W(t) - \eta \cdot (1 - \Delta e \cdot \beta) \cdot \frac{\partial L}{\partial W} \qquad \textbf{(A-3.15)}$$

Where Δe is the gradient between current and previous error (input/output) and β is a constant 0 to 1 for smoothing error change.

3.2.2. Recurrent structure in neural networks

- What is a recurrent structure?

In basic neural networks, there is an input and then an output, every calculus is only in one direction, that is more or less what is called feedforward. An example of this is the MLP, done before.

Recurrent structure is a kind of structure where the output of the node is used as the input of the node itself. Temporal sequencing is quite important in this structure.

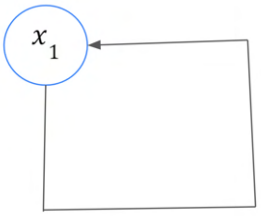


Fig. A-3.3 Recurrent structure

In a recurrent structure the value of the node will be the following (Eq. A-3.16), in case of a linear activation function, it will be only: $x_i(t) = w_i \cdot x_i(t-1)$.

$$x_i(t) = F(w_i \cdot x_i(t-1)) \qquad \textbf{(A-3.16)}$$

Fig. A-3.4 Hybrid structure

Supposing Fig A-3.4 we can deduce the following equations.

$$x_1(t) = F(w_1 \cdot x_1(t-1)) \qquad \textbf{(A-3.17)}$$

$$x_2(t) = F(w_{21} \cdot x_1(t-1) + w_2 \cdot x_2(t-1)) \qquad \textbf{(A-3.18)}$$

Any special configuration of connection and nodes is resolved in the same way. Recurrent connections usually are located inside hidden layers.

3.2.3. Recurrent neural network (RNN)

There are many types of RNN like the long short-term memory (LSTM) or GRU but this frame is dedicated to a basic understanding of RNN networks to have a starting point.

This kind of neural networks can give the advantage of processing the data based on time, depending data processed in $t-1$, the results can change in time t . Used specially in temporal dynamic behavior areas.

Some authors' explanations of RNN have a little bit different notation respect before, but basically it has the same meaning, the only thing that changes is the form of notation. There are many RNN, the most common like Hopfield, Jordan or this one (Eq. A-3.19) Elman network.

$$h_t = \sigma_h(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h) \quad \text{AND} \quad y_t = \sigma_y(W_y \cdot h_t + b_y) \qquad \textbf{(A-3.19)}$$

where:

x, h, y are the same as before
W, U are weight matrices
σ is the activation function
b is the bias of the node

In contrast, an extended RNN model can be with multiple and different inputs and parameters for example (Eq. A-3.20) (including $s(t)$ as input).

$$h_t = \sigma_h(A_h \cdot x_t + U_h \cdot h_{t-1} + W_h \cdot s_t + b_h) \quad \text{AND} \quad y_t = \sigma_y(V_y \cdot h_t + D \cdot s_t + b_y) \textbf{(A-3.20)}$$

The formation of these structures is pretty easy, as explained before. But the problem comes at backpropagation methods, time takes an important role. For more information and co-state dynamics (linearized sensitivity) in Optimization Theory see [42] in backpropagation.

To start with, the following scheme (Fig. A-3.5), it has the same formulation as (Eq. A-3.19) without including the bias.

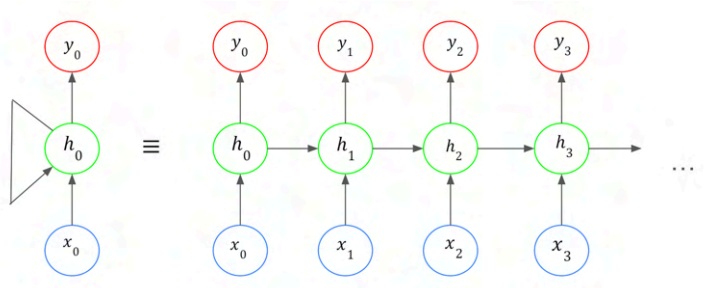


Fig. A-3.5 Simple recurrent neural network

To compute the total loss during a period of time we can use (Eq. A-3.21). Recall that the loss function is computed as in (Eq. A-3.3).

$$L = \frac{1}{N+1}\sum_{t=0}^{N} L^t \qquad \text{where} \qquad t = 0, \dots, N \qquad \textbf{(A-3.21)}$$

In order to compute the weight matrices, these matrices depend on past values, let's see an example (Fig. A-3.6).

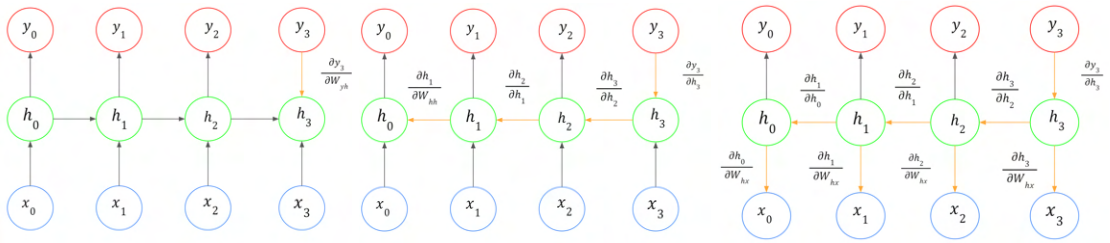


Fig. A-3.6 Chain rule in RNN, 3 cases

To compute the matrix W_{yh}, see (Eq. A-3.22).

$$\frac{\partial L_3}{\partial W_{yh}} = \frac{\partial L_3}{\partial y_{computed}} \cdot \frac{\partial y_{computed}}{\partial net} \cdot \frac{\partial net}{\partial W_{yh}} = \frac{\partial L_3}{\partial y_{computed}} \cdot \frac{\partial y_{computed}}{\partial W_{yh}} \qquad \textbf{(A-3.22)}$$

To compute W_{hh}, the index of $i = 0$ only if there isn't another connection at h_0, (Eq. A-3.23). To compute W_{hx}, (Eq. A-3.24), (Fig A-3.6 can be helpful).

$$\frac{\partial L_3}{\partial W_{hh}} = \sum_{i=0}^{N} \frac{\partial L_3}{\partial y_{computed}} \frac{\partial y_{computed}}{\partial h_N} \left(\prod_{j=i+1}^{N} \frac{\partial h_j}{\partial W_{j-1}} \right) \frac{\partial h_i}{\partial W_{hh}} \approx \sum_{i=0}^{N} \frac{\partial L_3}{\partial y_{computed}} \frac{\partial y_{computed}}{\partial h_i} \frac{\partial h_i}{\partial W_{hh}} \qquad \textbf{(A-3.23)}$$

$$\frac{\partial L_3}{\partial W_{hx}} = \sum_{i=0}^{N} \frac{\partial L_3}{\partial y_{computed}} \cdot \frac{\partial y_{computed}}{\partial h_i} \cdot \frac{\partial h_i}{\partial W_{hx}} \qquad \textbf{(A-3.24)}$$

Basically follow a tree distribution using the chain rule. Nevertheless, this system has short memory (long term weight disappears, tends to 0) and if weights are too high, clipping is used (make boundaries in the weight matrices).

3.4. Convolutional neural network (CNN)

One of the most important neural networks in image processing. From estimating distances to identify objects in pictures, it can take a variety of jobs. [43]

The most common activation functions in these CNNs are sigmoid, tanh and especially ReLU (rectified linear unit), for more information about ReLU [44].

Although the input in these networks are images, these images have different colors (RGB colors) and parameters, so input and the structure of these neural networks have multi-dimensional proportions. These neural networks can be considered as divided per blocks of neurons, each block having different actions and causing different properties to the overall network. In fact they are not called blocks, they are called tensors.

The name of these networks comes from a mathematical operation (convolution). These operations are used with a filtering action, in order to obtain useful data from the input. Many tensors in these networks are acting as filtering banks applying convolution to their inputs (Fig. A-3.7). Another important function used in these networks is pooling, there are many types of pooling. The most used, max pooling, in a set of values, the output value will be the greatest of the set.

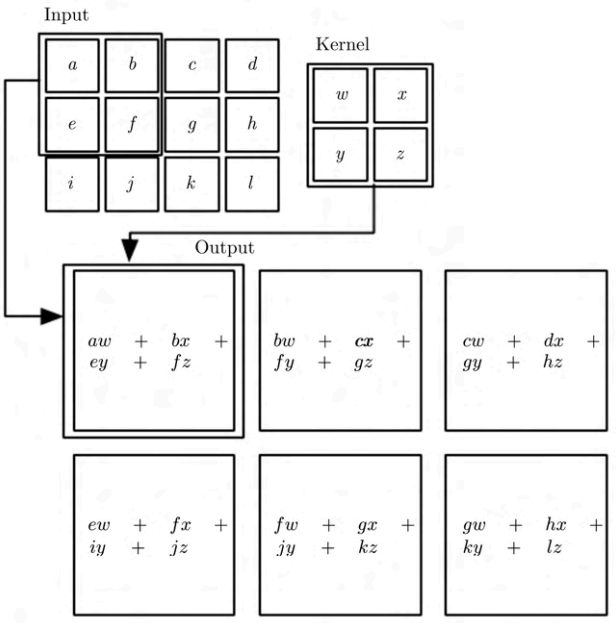


Fig. A-3.7 Convolution operation right [45]

ReLU in a perceptron takes the role of classification operation while the pooling takes feature extraction. Furthermore, softmax is used to do the final feature classification. Recall that there are many kinds of networks, regression and classifiers in this case (Fig. A-3.8).

Many of these networks are sets of tensors with different sizes, mixing ReLU, pooling and convolutional operations. The problem comes from the training time, many nets can last weeks of training due the size of the network. Many people use pretrained networks changing only the classifier (last layer) on arbitrary classes, leaving only the full network, the feature vector and a new classifier, and that works.

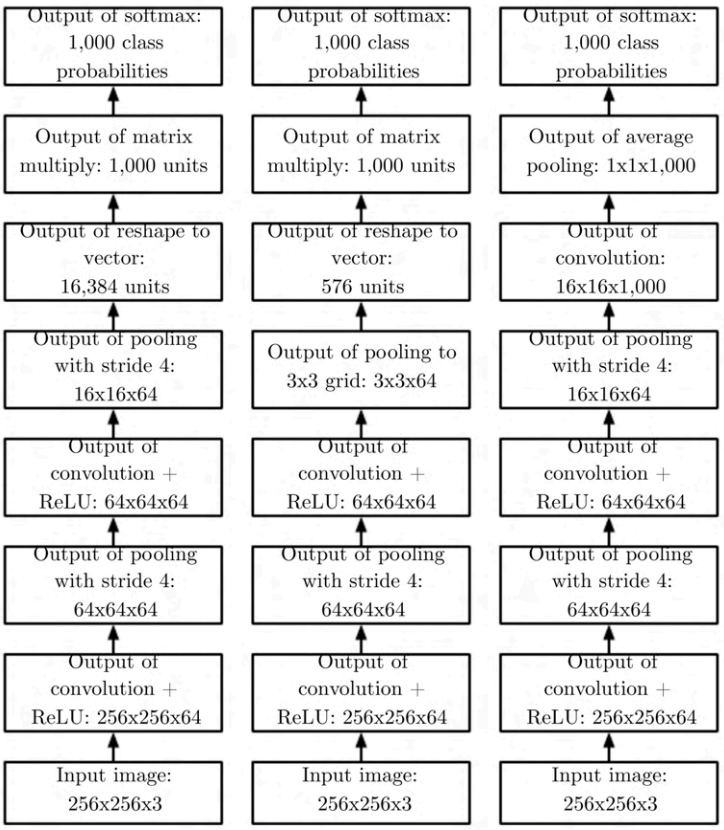


Fig. A-3.8 Examples of CNN [45]

Now I can identify an object, but how can I identify where it is?

- R-CNN:
  Using a searching algorithm (+2000 regions), crop the image into a smaller one and pass this image through the network. However, this can lead to a low performance in time.

- Fast R-CNN:
  Pass the image into a convolutional network to generate a feature map, and with ROI pooling layer, resize and feed the network to classify.
  Faster than R-CNN, but bottlenecked by the regional proposals.
  As seen in the image this network can classify and estimate the boxes.

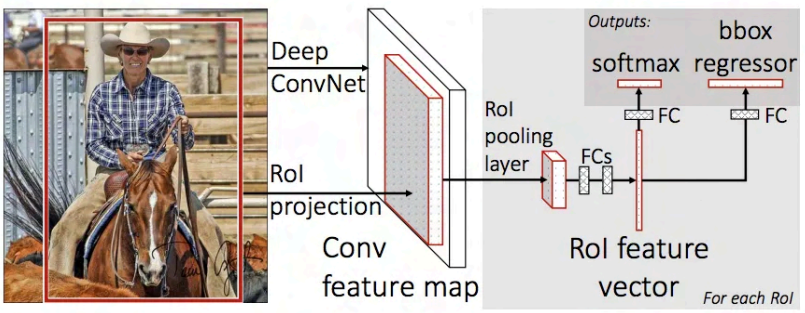


Fig. A-3.9 Fast R-CNN scheme [46]

- Faster R-CNN:
  Same as Fast R-CNN, but a separated network is used, a regional proposal network predicts the regions. Then these regions pass through a ROI pooling layer to resize and feed the network to classify.

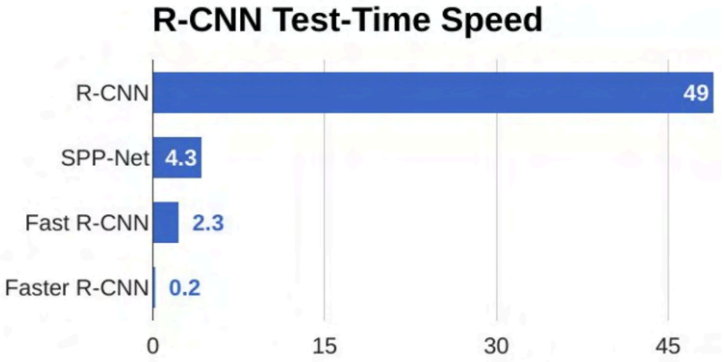


Fig. A-3.10 CNN comparison [46]

- YOLO (You Only Look Once):
  Splitting the image in a grid of boxes, a convolutional network makes predictions with these boxes, and puts offsets in these bounding boxes. The boxes above the threshold are used to predict the object. Pretty fast, around 45 frames per second. Nevertheless, it struggles very much with small objects.

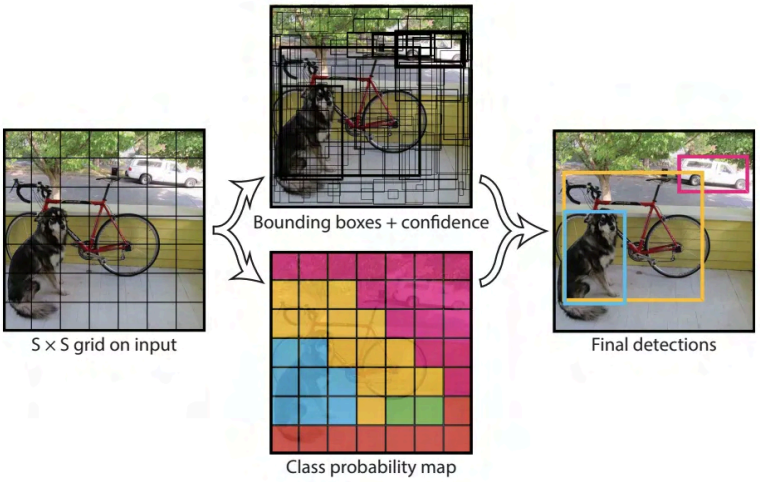


Fig. A-3.11 YOLO [46]

ANNEX 4: SENSORS AND BUS CONNECTION

4.1. Physical connection

This part of the project is oriented to make a system read bus with a STM32 or even with any arduino board. Different connections and codes can be implemented.

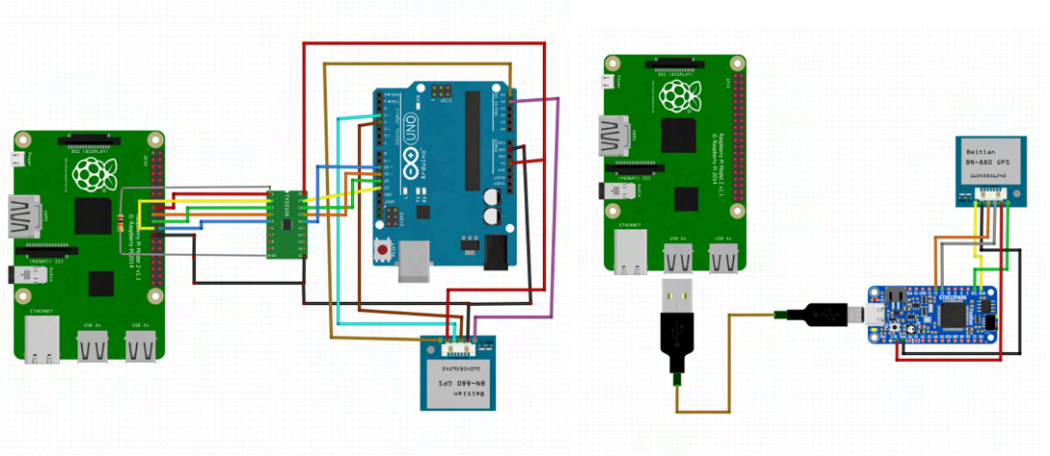


Fig. A-4.1 Example of connections using Arduino UNO and STM32 via Fritzing

The Raspberry pi can be powered via 5V and GND GPIO pins. For the connection with the Arduino a SPI bus has been done and the connection with the STM32 a serial bus has been done. Furthermore the STM32 doesn't need to be powered, because it is powered using the bus. GPS uses serial and I2C buses.

Apart from this, the steps to retrieve the data from the mobile phone are explained within the code. The main concepts are: install the Matlab App, configure the App (it is easy) and use a series of functions to extract the data.

4.1.1. Sensors

The sensor used to determine the position of the object is the Beitian BN-880 GPS (after using this sensor, Mini Ublox NEO-M8N GPS was used due better precision and in order to create the drone). It is composed of a GNSS receiver and a compass. The whole system is fed at 3.3-5V.

The GNSS receiver can receive data from GPS, Galileo, Glonass, BDS, SBAS, QZSS. With a sensitivity of -160dBm, its accuracy is 2 m and a velocity of 0.5 m/s. Furthermore it has 72 search channels. Connected with a serial connection.

The compass (magnetometer) is the HMC5883, it can offer a precision of 2º in its measure. Connected within an I2C bus. This module can be influenced by radiation of the surrounding electronics.

However, some distributors sell the same sensor but with QMC5883 without saying it, a Chinese version of the Honeywell compass with relatively worse performance. In case of doubt, the Chinese version of the compass, its address is “0x0D” instead of “0x1E”. In this project QMC5883 has been used.

Fig. A-4.2 BN-880 GPS + Compass H/QMC5883

4.1.2. Buses and connections

There are several types of buses used or tried to implement in this project. Especially SPI, I2C and Serial.

4.1.2.1. SPI Bus

This bus configuration was invented in 1982 by Motorola. Short distance, full duplex with high data transmission velocity, synchronized. It consists of 4 threads: a clock determining when to read by the master, a MOSI thread that sends master data, a MISO thread that sends slave data and finally SS (slave selector), just by the name the meaning is understood.

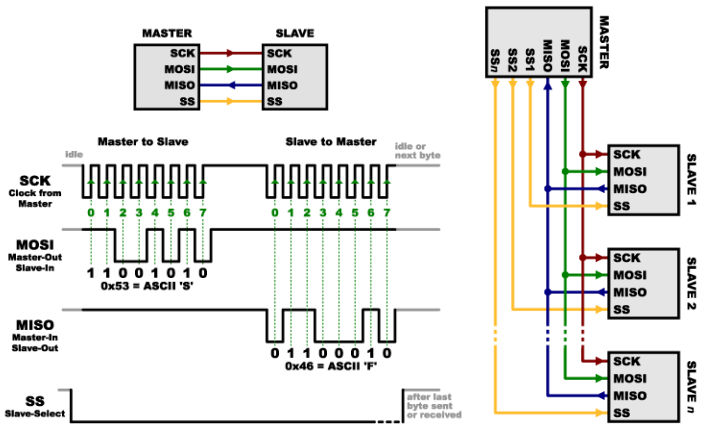


Fig. A-4.3 SPI bus structure [47]

4.1.2.2. I2C Bus

This bus configuration was invented in 1992 by Philips Corporation. Its data transmission can handle 100 kbits/s or 400 kbits/s, structured as a master-slave, 2 threads (SDA for data transmission and SCL as a master clock).

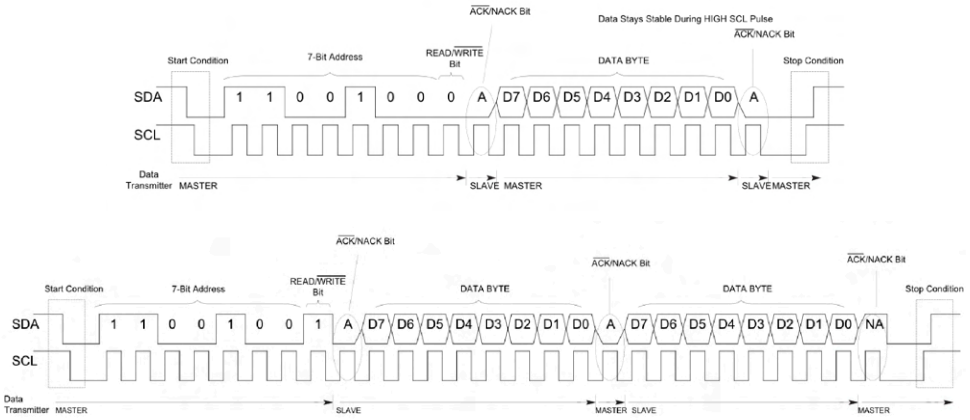


Fig. A-4.4 I2C structure (up sending, down receiving)[48]

I2C can have a multirole function in order to communicate with multiple sensors inside a drone. In particular, this bus has been implemented to receive the compass data. What's more, it has been tested with multiple sensors at the same time using the compass of the BN-880 module and the MPU6050 GY-521 IMU sensor, seeking for bus resilience. Using the library Wire, data extraction and communication with both sensors has been achieved, making it possible to implement into a bigger system like a drone.

4.1.2.3. Serial connection

Unsynchronised, 1 thread connection from TX to RX, sends 8 bits with 2 added bits (start and stop) for clock synchronism. Typically the serial port is the RS-232 interface.

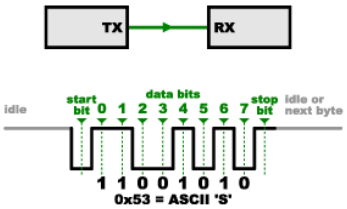


Fig. A-4.5 Serial structure [47]

4.1.2.4. UART connection

Basically it's a serial connection, the big differences come from the frame, protocol, and others. It encompasses many connections like RS-232, RS-485, etc.

Fig. A-4.6 UART frame [49]

4.1.2.5. CAN Bus

This bus configuration started in 1983 at Robert Bosch GmbH, then in 1987 the bus was introduced by Intel and Philips. Applied mainly in vehicles and instrumental platforms, this bus has been one of the most important buses in the world. Standardized in 1993, it has different parts: ISO 11898-1 data link layer, ISO 11898-2 physical layer for high speed, ISO 11898-3 physical layer for low speed and error tolerance and others. [50]

Data transmission up to 5-1 Mbits/s in high speed, up to 125 kbits/s in low speed. Twisted cables with 120 ohms resistors at the end, both cables send the same signal but reverse polarized due noise correction.

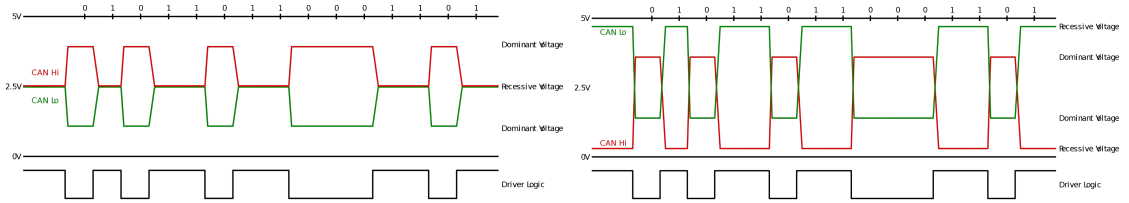


Fig. A-4.7 CAN transmission (left high speed, right low speed) [50]

There are different kinds of CAN frames, but the standard one consist in:

- 1 bit, start frame
- 11 bits, identification
- 1 bit, stuff synchronism
- 1 bit, frame type (remote frame or data frame)
- 1 bit, identifier extension
- 1 bit, reserved bit
- 4 bits, data length
- 0-8 bytes, data
- 16 bits, cyclic redundancy check
- 2 bits, acknowledgement
- 7 bits, end of frame
- 3 bits, spacing

In order to connect STM32 and other devices using a bus can, the MCP2515 chip module was tested at 5V; however, it can’t be used within a direct system with raspberry pi due high voltage. To use this whole module directly at 3V, there are 2 ways, connecting the external cable to the TJA1050 chip at 5V or changing the TJA1050 chip to another SN65HVD230 chip that can be fed at 3V. This module uses SPI communication to develop the bus can. In this project has been achieved can bus communication between arduino modules, but due inexplicable deeds raspberry pi after a long time of use, bus can wasn't enable to continue operating.

Fig. A-4.8 MCP2515, CAN module

4.1.2.6. Others

In order to establish a communication bus between raspberry pi and arduino/STM32, both modules work at different voltages, the extension module TXS0108E can be a solution to connect raspberry pi and arduino. This particular chip has been chosen because it can transmit data at high velocity differentiating with respect to other modules in the market. Also it is recommended, put 0.1 mF capacitors between VCC and ground. A pull up 10k resistor is needed between VCC and E0 in the lower voltage part of the module.

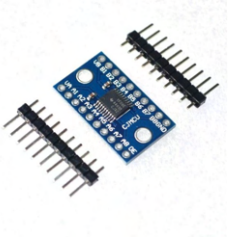


Fig. A-4.9 TXS0108E module

4.2. Software

Not all the software of all this project will be explained thoroughly, but some key insight must be shown. It has developed 2 softwares, one for Arduino UNO with SPI communication and the other one with the STM32 on Serial communication.

The Arduino UNO software doesn't take much time in the loop, SPI bus around 1000 microseconds, GPS around 3000 and compass (the most time eaten) can give the whole software loop as much as around 23000 microseconds in Arduino UNO.

In case of using an STM32, the time taken is much less, around 1000 microseconds the whole loop enabling the use inside a drone control loop. The frequency of the STM32f405 is 168MHz compared to the Arduino UNO at 16MHz. Furthermore some sensors or all them can be assigned to take role periodically at a specific time and not interfere all the time in the loop.

4.2.1. Compass

Before starting to extract data from the compass directly, the compass has to be calibrated, the library QMC5883LCompass hands a document especially to obtain the calibration parameters.

Once with these parameters in a general code is simple as read data, apply parameters and generate the angle starting from 0 to 359. After that, pass this data into a byte form to generate the message. This message contains the magnetic north, the true north has to be computed using the magnetic deviation (approx 2º in case of Barcelona). In this project the true north is not going to be used directly. All data is obtained via I2C bus.

4.2.2. GNSS receiver

In the case of the GPS is a little bit more tricky, in order to receive data, a periodical interaction (read/send) has to be done, specially to receive altitude data.

With Arduino UNO, once with the data, the latitude and longitude can be negative or positive, depending on the sign of the data an extra byte will be generated to differentiate on the final message the real values. The ideal method to use is the IEEE 754 format. Apart from this, to make it simple, all values will be positive and multiplied by a high value. This is done to have the highest resolution/accuracy available, just because all the data have decimals, and values with decimals are complex to send via bus; however, integer big numbers are more simple to send. To store these numbers, big data formats are used like uint_64 or float, instead of habitual integers. Finally all this data will be converted into bytes and located in a message array.

Respect STM32, all the data will be floats, not converting it directly into bytes.

At least around 6 decimal places on GPS data are required for precision/resolution less than 1.1 m in the data; though the maximum resolution of the sensor is 2 m.

4.2.3. Arduino UNO SPI bus

With the message computed, a preface composed of 0x01, 0x02, 0x03 and 0x04 bytes will be sent to know when the data starts. Every time the Raspberry pi asks for information an individual byte of the whole message will be sent.

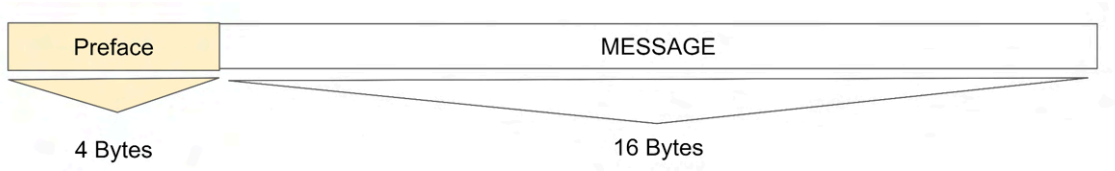


Fig. A-4.10 Whole message structure

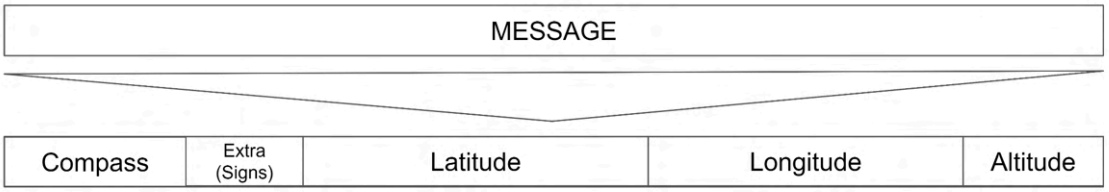


Fig. A-4.11 Simple message structure

Furthermore, this bus shall be set at a frequency of less than 8 Mhz in Arduino Uno, to not have errors in transmission. The code developed for this bus has no SS pin interference due its unnecessary purpose.

4.2.4. STM32 Serial bus

In order to send the data in the simplest way as possible, and not using any conversor, a usb to usb-c connection shall be done with the raspberry pi and the STM32. Printing in serial with the STM32 coded with Arduino can make the whole process easier, because it uses the format ASCII (American Standard Code for Information Interchange).

At first instances, to ensure reception and upload of the code of some STM32s; it would be useful to download Arduino on the Raspberry pi. To do so on Debian, execute the following commands in the terminal:

1. sudo apt update
2. sudo apt upgrade
3. sudo apt install arduino
4. sudo usermod -aG dialout $(whoami)

The last command is for granting the execution permissions, and remember to always reboot!!

Once inside Arduino IDE, open preferences and put the following link from GitHub:
https://raw.githubusercontent.com/koendv/stm32duino-raspberrypi/master/BoardManagerFiles/package_stm_index.json

Then install inside TOOLS, the board packages of the STM32 family, and finally configure the setting of the personal STM32.

This setup is able to read serial data and manage every input and output of the arduino code inside the STM32, even upload code. Thought it is very useful, some versions of STM32 aren't well supported, and depending on the version of the OS of the Raspberry pi, the dfu-utils library version 0.9 and ARM libraries aren't updated (the case of this project), so some STM32's serial can only be read.

Stepping back away without the knowledge of the implementation of Arduino inside the rasp. The communication has been done with the following terminal commands.

To know what devices are connected use command 1. To know serial connections use command 2. To establish communication it is necessary to use the command 3 that implements the baud rate in numbers and the serial port “/dev/ttyACM0” in the case of this project. Command 4 only makes the output of the data receiver into a file DATA.csv, it can be even a text file, it uses >> that means every information received will be saved in a different line inside the csv. Furthermore there are other ways to implement this, using the command tee like 5 that even shows the data on the terminal. The sixth command shows the data in hexadecimal on the screen. And the last one is optional, it sends the char “ON” to the STM32.

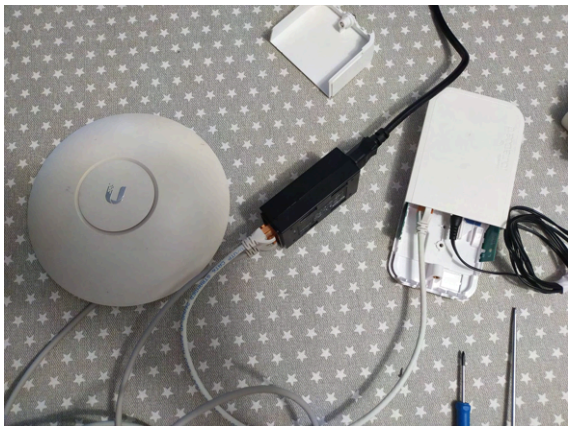
1. lsusb
2. dmesg | grep tty
3. stty 115200 -F /dev/ttyACM0 raw -echo
4. cat /dev/ttyACM0 >> DATA.csv
5. cat /dev/ttyACM0|tee DATA.csv
6. cat /dev/ttyACM0|hexdump -C
7. echo "ON">/dev/ttyACM0

After the creation of this csv, the download using the IP address via Matlab and the deletion of it inside the rasp is done.

A drawback of this communication is that the terminal/Matlab code shall be always running to extract data at that moment. Another way to make it more efficient is to create this connection using Python via a pyserial library and run that code externally using Matlab, Python or PUTTY. And from the ground station any operation on receiving or sending data can be done at any time without interfering with others.

ANNEX 5: LAN CREATION

In order to make the system reliable and feasible a good LAN has to be developed. Specifically for long range applications a simple LAN won't work, so this LAN is created for distances larger than 100 meters. Also the receiver antenna shall be powerful enough to receive and send back the data (the antenna within the drone or vehicle).



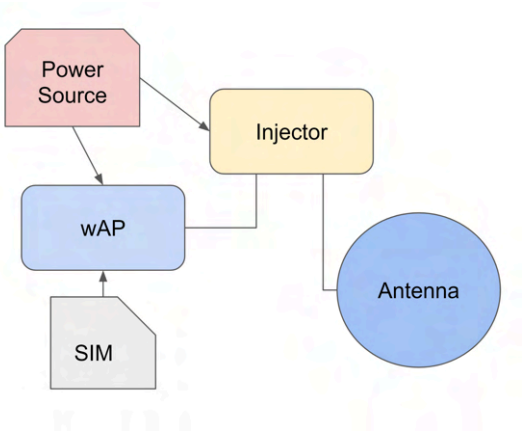


Fig. A-5.1 Overall LAN System

5.1 Hardware

- Mikrotik wAP LR8 (180€ approx)

Fig. A-5.2 Mikrotik wAP LR8

- Ubiquiti UAP-AC-LR 2.4-5GHz access point 1Gbps (183 m range) (145€)

Fig. A-5.3 UAP-AC-LR 2.4-5GHz

- Ubiquiti Networks POE-48-24W Adapter/injector (13€)

Fig. A-5.4 POE-48-24W Adapter/injector

- Ethernet cables (different sizes, prices depending on the purpose)
- SIM mobile card

5.2 Hardware connections

Pretty easy, the wAP needs: to be powered, a SIM card and a connection of an ethernet cable. This dispositive enables a LAN with 2 antennas; however, this LAN is range limited. All connections can be seen in Fig A-5.5.

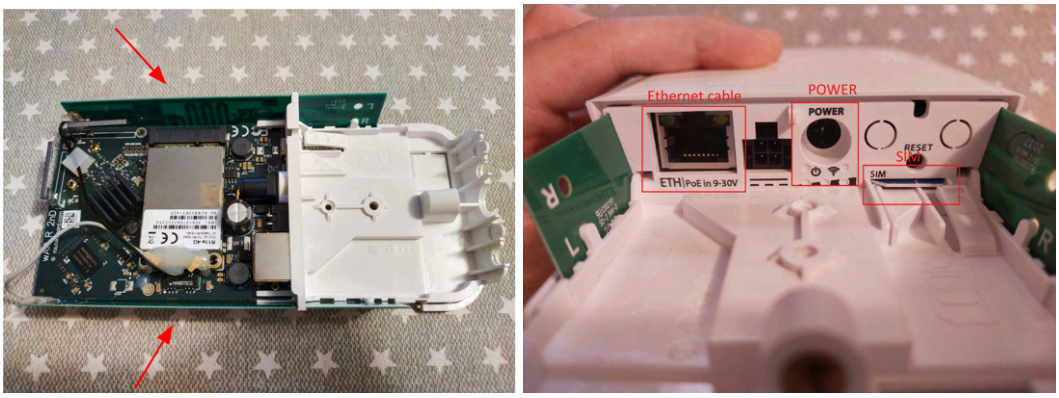


Fig. A-5.5 POE-48-24W Adapter/injector (showing ports and antennas)

The other head of the ethernet cable will be connected to the LAN port of the adapter/injector and also another cable shall be connected to the POE port (powered over ethernet), Fig A-5.6. This connection will power the ubiquiti antenna using the ethernet cable.

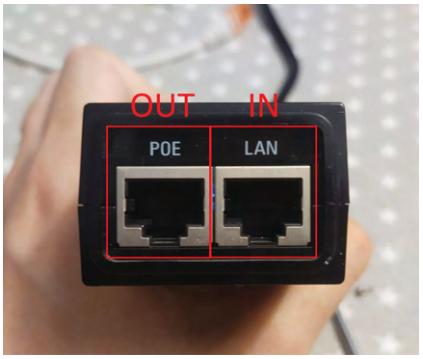


Fig. A-5.6 POE-48-24W Adapter/injector

Finally, the outer head of the cable connected from the POE port will be connected to the antenna.

Fig. A-5.7 Ethernet port Ubiquiti antenna

5.3 Software

In this system 1 LAN and 2 wifi sources will be created, the first wifi source comes from the wAP and the second one comes from the Ubiquiti antenna connected to the wAP source.

Firstly, to create the LAN, the wAP Mikrotik program should be installed (WinBox). With a cable connection to the wAP open WinBox on your computer. Search for the label neighbors and double click on the MAC address of the wAP. To connect the wAP just put it as the username “admin” and then connect to pop a new window (password shall be empty at the first time, see for instruction of the product if any problem appears).

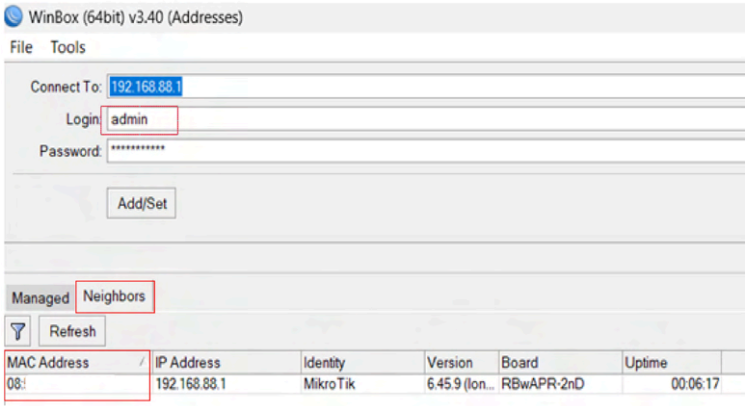


Fig. A-5.8 Connection with the device

To create the APN (connection with the mobile internet services, in my case Digi Mobil), inside this new window go to interfaces (in the left column), then go to the LTE section. Click inside LTE APNs or the line beneath. Create a new line in LTE APNs (always would be 1 line created as default), this new line name it whatever you want (in my case “internet”), put the APN and the authentication type (these values can be found by the mobile internet company you have contracted). Finally apply these parameters.

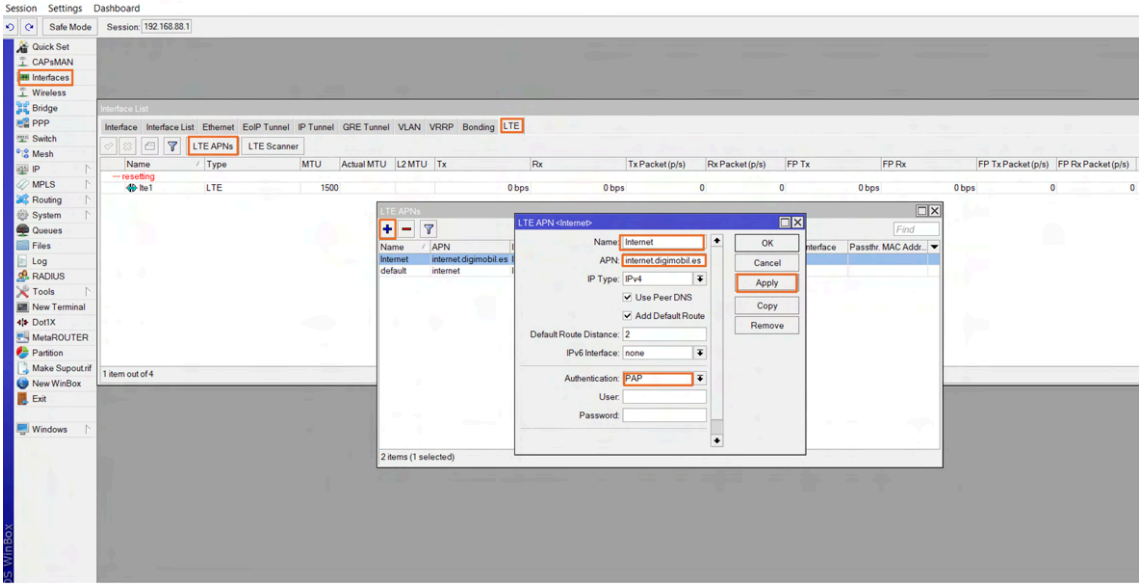


Fig. A-5.9 Creation of the APN

Right now internet connection has been achieved. The last things to do are enable this connection and create the LAN. Go back to interfaces and double click on the LTE line. If it doesn't appear, reboot the program or create a new line.

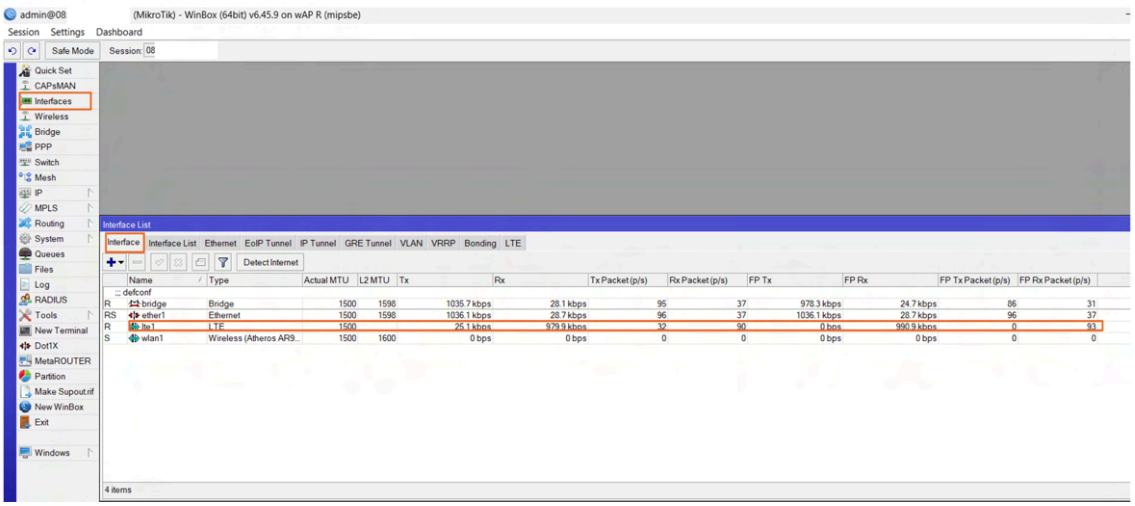


Fig. A-5.10 Enabling LAN

This would open a new window. Inside this new window put the pin of the SIM and the APN name you put before, and then apply.

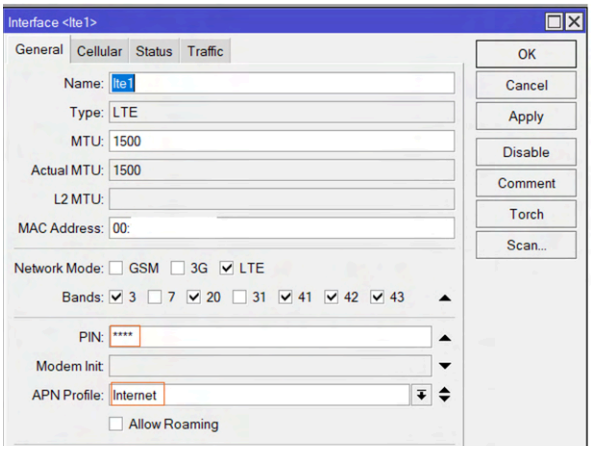


Fig. A-5.11 Enabling internet

Go back, open the wlan line and put ap bridge and the SSID (name of the access point), then apply.

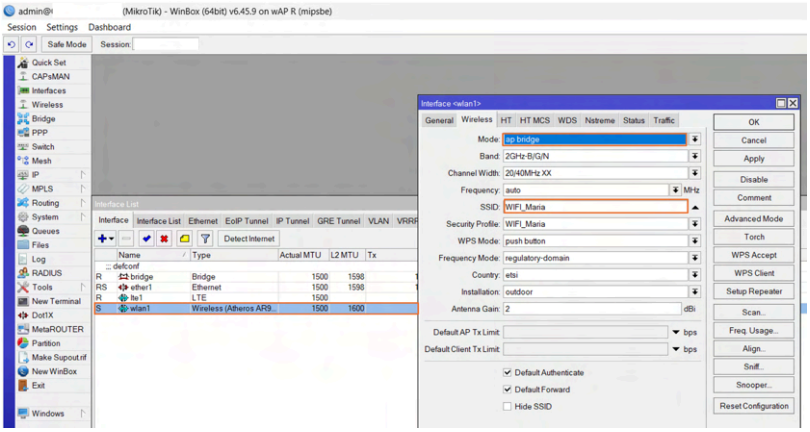


Fig. A-5.12 SSID and others

Finally to change the password for wireless devices, click on the left column Wireless, inside this new window open Security Profile and click the line. It will open a new window, inside this new window, the WPA Pre-Shared Key is the password for wireless devices, finally apply all these parameters.

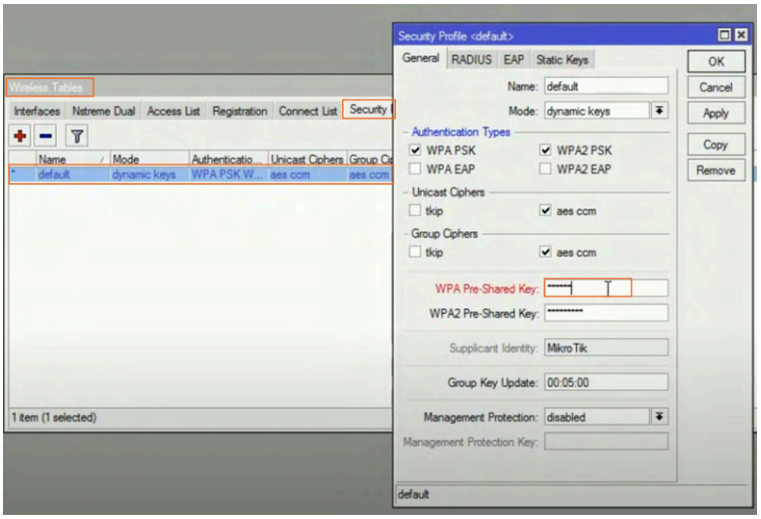


Fig. A-5.13 Password of the LAN

The wAP section has been covered, now the long range antenna. Connect the antenna to the wAP using the injector. Download the UniFi Network Server program and create an account (stay inside the wireless connection of the wAP). Once you open the program it will redirect you into an internet window to manage the device.

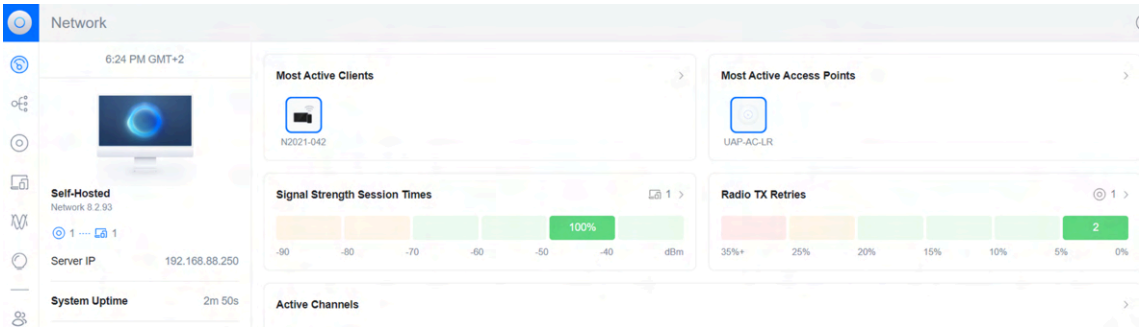


Fig. A-5.14 UniFi/Ubiquiti program

Adopt the new antenna following the instructions in Fig A-5.15, if it was adopted before, it won't appear.

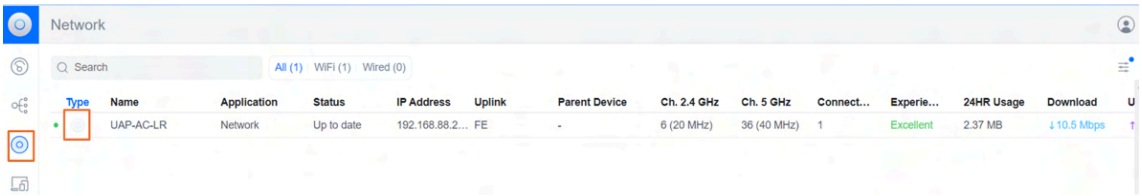


Fig. A-5.15 Adopting antenna.

As a curiosity, any dispositive connection to this network can be found using this program.

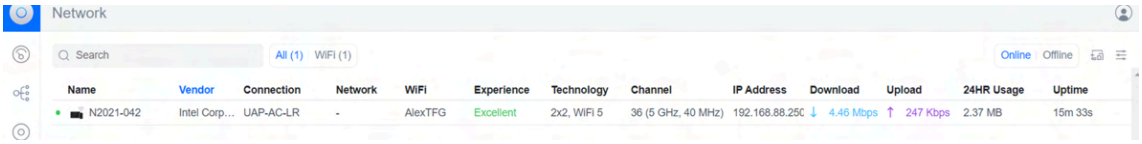


Fig. A-5.16 Connected dispositives

Lastly, inside the settings, open the new network and put the antenna Wifi name and the password.

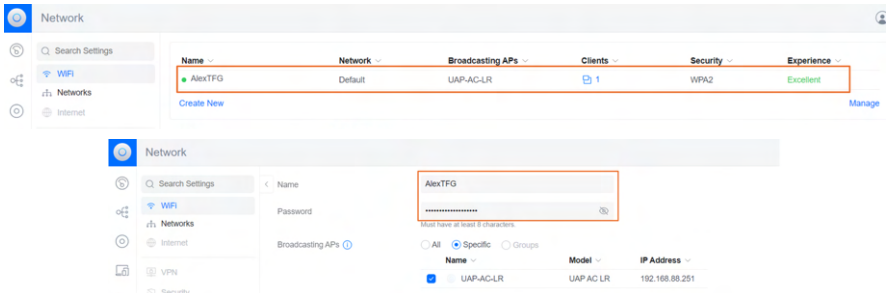


Fig. A-5.17 Change antenna name and password

ANNEX 6: MATLAB AI CODE

Training and test document for object detection

Created by: Alejandro Boadella Aguado (EETAC-UPC)

References: MATLAB sources and examples.

Location of the files and labeled ground-truth data.

```
photos2 = load("C:\Users\Alex\Desktop\photos2\photos2.mat");
gTruth2 = photos2.gTruth;

photos3 = load("C:\Users\Alex\Desktop\photos3\photos3.mat");
gTruth3 = photos3.gTruth;

photos4 = load("C:\Users\Alex\Desktop\photos4\photos4.mat");
gTruth4 = photos4.gTruth;

photos5 = load("C:\Users\Alex\Desktop\photos5\photos5.mat");
gTruth5 = photos5.gTruth;

photos6 = load("C:\Users\Alex\Desktop\photos6\photos6.mat");
gTruth6 = photos6.gTruth;

photos7 = load("C:\Users\Alex\Desktop\photos7\photos7.mat");
gTruth7 = photos7.gTruth;

photos8 = load("C:\Users\Alex\Desktop\photos8\photos8.mat");
gTruth8 = photos8.gTruth;

photos9 = load("C:\Users\Alex\Desktop\photos9\photos9.mat");
gTruth9 = photos9.gTruth;

photos10 = load("C:\Users\Alex\Desktop\rest_photos\rest_photos.mat");
gTruth10 = photos10.gTruth;
```

Combination of ground-truth and generation of anchors for heading of the YOLOv4 "tiny version" and ACF.

```
[imList,boxLabels]=objectDetectorTrainingData([gTruth2 gTruth3 gTruth4 gTruth5 gTruth6 gTruth7 gTruth8 gTruth9 gTruth10]);
TrainingData=combine(imList,boxLabels);
className = "car";

numAnchors = 6;
[anchors,meanIoU] = estimateAnchorBoxes(TrainingData,numAnchors);
area = anchors(:, 1).*anchors(:,2);
[~,idx] = sort(area,"descend");
anchors = anchors(idx,:);
anchorBoxes = {anchors(1:3,:) anchors(4:6,:) };
```

In case of the big version of YOLOv4 is requested use this code for anchors.

The big version has more heads (anchors) in the structural part as explained in the project.

```
numAnchors = 9;
[anchors,meanIoU] = estimateAnchorBoxes(TrainingData,numAnchors);
area = anchors(:, 1).*anchors(:,2);
[~,idx] = sort(area,"descend");
anchors = anchors(idx,:);
anchorBoxes = {anchors(1:3,:) anchors(4:6,:) anchors(7:9,:)};
```

ACF detector trained

```
detector = trainACFObjectDetector(TrainingData,NumStages=10)
```

Code to obtain the YOLOv4 detector (using the tiny version)

```
detector = yolov4ObjectDetector("tiny-yolov4-coco",className,anchorBoxes);
```

Options for training procedure.

```
options = trainingOptions('sgdm',...
          InitialLearnRate = 0.0001,...
          Verbose = true,...
          MiniBatchSize = 16,...
          MaxEpochs = 260,...
          VerboseFrequency = 30, ...
          l2Regularization = 0.0005, ...
          Shuffle ='every-epoch', ...
          ResetInputNormalization=true, ...
          ExecutionEnvironment='gpu');
```

Training of the detector

```
[detector2,info] = trainYOLOv4ObjectDetector(TrainingData,detector,options);
```

Load saved detectors, created before.

```
load AI3.mat
load ACF3.mat
```

Load new images to evaluate detector.

```
photos1 = load("C:\Users\Alex\Desktop\photos1\photos1.mat");
gTruth1 = photos1.gTruth;
[imList,boxLabels]=objectDetectorTrainingData(gTruth1);
```

Load new and old images to evaluate detector (needs previous codes to be activated)

```
[imList,boxLabels]=objectDetectorTrainingData([gTruth1 gTruth2 gTruth3 gTruth4 gTruth5 gTruth6 gTruth7 gTruth8 gTruth9 gTruth10]);
```

Detection

```
results = detect(detector,imList);
```

Evaluation of the results (using an overalp of 0.35 out of 1)

```
Overlap=0.35;
[ap, recall, precision] = evaluateDetectionPrecision(results, boxLabels,Overlap);
[am, fppi, missRate] = evaluateDetectionMissRate(results, boxLabels,Overlap);

figure(1)
plot(recall, precision);
grid on
title(sprintf('Average precision = %f', ap))
xlabel('Recall')
ylabel('Precision')
xlim([0 1.1])
ylim([0 1.1])

figure(2)
loglog(fppi, missRate);
grid on
title(sprintf('log Average Miss Rate = %f', am))
xlabel('Fppi')
ylabel('MissRate')
xlim([0 1.1])
ylim([0 1.1])
```

Test the detector on an image (use this code or the following one, both are pretty similar)

```
img = imread("C:\Users\Alex\Desktop\photos4\VID-20231114-WA0006_060.png");
[bboxes,scores] = detect(detector2,img);
[selectedBbox,selectedScore] = selectStrongestBbox(bboxes,scores,"NumStrongest",1);
frameLabeled=insertObjectAnnotation(img,"rectangle",selectedBbox,"Car_coeff:"+ selectedScore);
figure
imshow(frameLabeled)

```

```
img = imread("C:\Users\Alex\Desktop\photos4\VID-20231114-WA0006_060.png");
[bboxes,scores] = detect(detector,img);
bboxes=bboxes(scores>15,:);
scores=scores(scores>15,:);
[selectedBbox,selectedScore] = selectStrongestBbox(bboxes,scores,"NumStrongest",1);
frameLabeled=insertObjectAnnotation(img,"rectangle",selectedBbox,"Car_coeff:"+ selectedScore);

figure
imshow(frameLabeled)
```

ANNEX 7: MATLAB GENERAL CODE

General Code Doc

Created by: Alejandro Boadella (EETAC-UPC)

References: Matlab sources and examples.

```
clear all
format long
```

Load calibration parameters and artificial intelligence.

```
load calibracion4.mat
load AI3.mat
load ACF3.mat
```

Mobile and Raspberry connection:

Create connection with mobile phone and use the camera. To connect the mobile, MATLAB Mobile App must be downloaded and configured to do so. Sometimes an error pops, probably it can be caused by the MATLAB general path, it should be located in MATLAB Drive.

If there is any error, use the following path once and update everything (in some computers the path can be different): C:\Program Files\MATLAB\R2024a\bin\win64

The error probably is caused by the root establishment with multiple MATLAB versions.

```
phone=mobiledev;
```

```
cam_phone = camera(phone,'back');
cam_phone.Resolution = '1280x720';
```

Connection with the Raspberry Pi using IP address, User and Password to do SSH connection.

```
mypi = raspi('10.3.6.180','AlexTFG','TFG')
camera2 = cameraboard(mypi,'Resolution','1920x1080','Quality',100,'FrameRate',30);
```

Take a snapshot with the mobile camera, obtain mobile data like orientation, GPS and others. Finally discard the data saved in the mobile and calibrate the compass.

```
phone.Logging = 1;
pause(1);
[lat_mo, lon_mo, timestamp, speed, course, alt_mo, horizac_precision] = poslog(phone);
[mo_orientation, timestamp] = orientlog(phone);
phone.Logging = 0;
discardlogs(phone);
```

```
photo_phone = snapshot(cam_phone,'immediate');
Correction_value_azimuth_mo=90; % Change the azimuth value by its horitzontal to verical position
Compass_mo=mo_orientation(:,1) + Correction_value_azimuth_mo;
Elevation_mo=mo_orientation(:,2);
Inclination=mo_orientation(:,3);
```

STM32 Serial Bus:

Listen the serial bus. Stop when the user doesn't want to listen more. In order to improve this part it is necessary to do a Python code that records the data and MATLAB retrieves the information from Python; however, this isn't the objective of this work.

```
system(mypi,'stty 115200 -F /dev/ttyACM0 raw -echo');
system(mypi,'cat /dev/ttyACM0 >> DATA.csv');
%system(mypi,'cat /dev/ttyACM0|tee DATA.csv')
```

Stop recording and download the data. Once downloaded, retrieve the data.

```
getFile(mypi,'DATA.csv','C:\Users\alexb\Dropbox\ALEX\UPC\TFG');
deleteFile(mypi,'DATA.csv');
```

```
data=readmatrix('C:\Users\alexb\Dropbox\ALEX\UPC\TFG\DATA.csv');

compass = data(end,1);
latt = data(end,2);
longi = data(end,3);
alt = data(end,4);
```

Arduino UNO SPI Bus:

Creation of SPI bus.

```
myspidevice = spidev(mypi,'CE0',0,400000,8);
```

Record some data with the Raspberry pi.

```
record={};
for i=1:20*2
    record{i,1} = dec2hex(writeRead(myspidevice,[hex2dec('08')]),2);
end
```

Locate the preface of the message, to know where the whole frame is located.

```
for i=1:(length(record)/2+4)

     if hex2dec(record{i,1})==1 && hex2dec(record{i+1,1})==2 && hex2dec(record{i+2,1})==3 && hex2dec(record{i+3,1})==4
            start_message=i+4;
     end

end
```

Using bytes received, transform the bits/bytes into real number data.

```
compass = hex2dec(strcat(record{start_message+1}, record{start_message}));

latt = hex2dec(strcat(record{start_message+8}, record{start_message+7} ...
    ,record{start_message+6}, record{start_message+5}, record{start_message+4} ...
    ,record{start_message+3}))/10^10;

longi = hex2dec(strcat(record{start_message+14} ,record{start_message+13} ...
    ,record{start_message+12} ,record{start_message+11} ,record{start_message+10} ...
    ,record{start_message+9}))/10^10;

symbol = hex2dec(record{start_message+2});

alt = hex2dec(strcat(record{start_message+16}, record{start_message+15}));
```

Plot GPS route using any kind of saved data (like .csv):

Considering data(:,2)=vector latitude, data(:,3)=vector longitude and data(:,4)=vector height.

This first plot is a topographic map.

```
figure(1)
geoplot(data(:,2),data(:,3),'o--','Color','r')
geobasemap topographic
title('GNSS position')
```

This first plot is a satellite map.

```
figure(2)
geoplot(data(:,2),data(:,3),'o--','Color','r')
geobasemap satellite
title('GNSS position')
```

This plot is a 3D map position.

```
uif = uifigure;
g = geoglobe(uif);
geoplot3(g,data(:,2),data(:,3),data(:,4),'Color','r')
```

Object Detection and Distance Estimation:

Take a photo with the Raspberry Pi, rectify the images of both cameras and create a disparity map. This code detects only a single object, multiple object also can be detected with only changing a few lines of the code.

Then the next step is to detect the object in the image. In case of detection the centroid of the detected label is analyzed depending the disparity map obtaining the object distance.

Finally a label with the distance is shown using the image of the left camera.

```
img = snapshot(camera2);
img_left =img(:, 1:end/2,:);
img_right = img(:, end/2+1:end,:);
[frameLeftRect, frameRightRect, reprojectionMatrix] = rectifyStereoImages(img_left, img_right, stereoParams,"linear");
frameLeftGray  = im2gray(frameLeftRect);
frameRightGray = im2gray(frameRightRect);
disparityMap = disparitySGM(frameLeftGray,frameRightGray,DisparityRange=[0 128],UniquenessThreshold=7);
imshow(disparityMap,[0 128]);
colormap jet

points3D = reconstructScene(disparityMap, reprojectionMatrix);
points3D = points3D ./ 1000;

[bboxes,scores]  = detect(detector,frameLeftRect);
[selectedBbox,selectedScore] = selectStrongestBbox(bboxes,scores,"NumStrongest",1);
frameLabeled=insertObjectAnnotation(frameLeftRect,"rectangle",selectedBbox,"Car_coeff:"+ selectedScore);
imshow(frameLabeled);

 if ~isempty(selectedBbox)

    % Find the centroids of the detected objects.
    centroids = [round(selectedBbox(:, 1) + selectedBbox(:, 3) / 2), ...
        round(selectedBbox(:, 2) + selectedBbox(:, 4) / 2)];

    % Find the 3-D world coordinates of the centroids.
    centroidsIdx = sub2ind(size(disparityMap), centroids(:, 2), centroids(:, 1));
    X = points3D(:, :, 1);
    Y = points3D(:, :, 2);
    Z = points3D(:, :, 3);
    centroids3D = [X(centroidsIdx), Y(centroidsIdx), Z(centroidsIdx)];

    % Find the distances from the camera in meters.
    dists = sqrt(sum(centroids3D .^ 2, 2));

    % Display the detected objects and their distances.
    labels = dists +" meters";
    dispFrame = insertObjectAnnotation(frameLeftRect, "rectangle", selectedBbox, labels,'LineWidth',3,'FontSize',20);

 else

    dispFrame = frameLeftRect;

 end

imshow(dispFrame);
```

FOV System (drone positioning in horizontal axis):

This code it isn't well conditioned so it can contain little error depending also on the hardware.

Give the degrees that the camera can observe horizontally, the magnetic deviation and a calibration that it is the variation between the compasses to eliminate offset error. The extra code is for convert the magnetic north to the real north.

```
Camera_width_h=66;
Magnetic_deviation=2;
Magnetic_offset_calibration=0;

% compass = compass - Magnetic_deviation;
% Compass_mo = Compass_mo - Magnetic_deviation;

```

Put mobile coordinate as a center coordinate axis. Then reckon the local values (in meters) of the GPS on the drone.

Calculate the angle to observe. Convert the x-axis values to north-axis values of compass and finally pass the values into magnetic compass values using magnetic deviation.

```
origin=[lat_mo lon_mo 0];
[xEast,yNorth,zUp] = latlon2local(latt,longi,alt,origin);
alpha=atan2d(yNorth,xEast);

alpha= 90 - alpha;
alpha_mag=alpha + Magnetic_deviation;

```

See how many pixels on the camera has each degree. See the variation of the mobile compass and the estimated position.

Depending of the value of the estimated position it will plot on the right or left side of the image considering 1280 pixels of the horizontal image axis.

A bar will be plotted on the image covering +20 pixel of width.

```
camera_variantion=1280/Camera_width_h;
degree_variation = rad2deg(angdiff(deg2rad(Compass_mo),deg2rad(alpha)));

image(photo_phone)
hold on
if degree_variation<=0
    xline( (1280/2) - camera_variantion*degree_variation,'LineWidth',20)
else
    xline( (1280/2) + camera_variantion*degree_variation,'LineWidth',20)
end
hold off

```

ANNEX 8: MATLAB SLAM CODE

Reconstruction vSLAM Doc

Created by: Alejandro Boadella (EETAC-UPC)

References: MATLAB sources and examples.

In this code, fast movement of the camera and lack of landmarks can make the code loss the track of the hardware, leading a non or a partial reconstruction of the scenario.

```
clear all
format long
```

Connect raspberry pi via ip address, put user and password to do SSH connection.

Enable camera, load camera calibration matrix and allocate folders.

```
mypi = raspi('192.168.2.100','AlexTFG','TFG');
camera = cameraboard(mypi,'Resolution','1920x1080','Quality',100,'FrameRate',30);
load calibracion4.mat
```

Option 1:

Loop: Take a snapshot, divide the image into 2 (2 cameras), rectify the images and save them into a folder.

```
ImageFolder_left ='C:\Users\alexb\Desktop\Left_video';
ImageFolder_right ='C:\Users\alexb\Desktop\Right_video';
k=1;
```

```
while true

    img = snapshot(camera);
    img_left =img(:, 1:end/2,:);
    img_right = img(:, end/2+1:end,:);
    [frameLeftRect, frameRightRect, reprojectionMatrix] = rectifyStereoImages(img_left, img_right, ...
        stereoParams,"linear");
    file_name_1 = sprintf('Image%d.jpg',k);
    file_name_2 = sprintf('Image%d.jpg',k);
    imgName_1 = fullfile(ImageFolder_left,file_name_1);
    imwrite(frameLeftRect,imgName_1,'Quality',100);
    imgName_2 = fullfile(ImageFolder_right,file_name_2) ;
    imwrite(frameRightRect,imgName_2,'Quality',100);
    k=k+1;

end
```

Option 2: (best option)

Instead of taking pictures, record with the raspberry pi video. Another method to record a video is using raspberry pi commands. Furthermore resolutions can be changed using different command structure.

For example: raspivid -t seconds*1000 -o video.h264

MATLAB command to send: system(mypi,'raspivid -t seconds*1000 -o video.h264');

```
seconds=40;
record(camera,'myvideo.h264',seconds);
```

Obtain the h264 file.

```
getFile(mypi,'myvideo.h264','C:\Users\alexb\Dropbox\ALEX\UPC\TFG');
```

Delete the file inside the raspberry pi.

```
deleteFile(mypi,'myvideo.h264');
```

MATLAB is unable to read directly an h264 video format, so mp4 conversion shall be done. This is done with FFMPEG. (newest version might be playable in MATLAB) (It only works for small video less than 10 seconds. The best option is to test it. Use other programmes conversions if the length of the video is too high!!!)

Specify the h264 path and video file, specify the path and file of the desired mp4 video.

Select the frame rate , specify the folder of the ffmpeg program (this program is free, just search and download from internet). Then a windows command will be executed and the status will tell if the conversion was successful.

```
h264FilePath = 'C:\Users\alexb\Dropbox\ALEX\UPC\TFG\myvideo.h264';
mp4FilePath = 'C:\Users\alexb\Dropbox\ALEX\UPC\TFG\file.mp4';
frame_rate='24';
ffmpegDir = 'C:\Users\alexb\Desktop\ffmpeg-master-latest-win64-gpl\ffmpeg-master-latest-win64-gpl';
ff = fullfile(ffmpegDir, 'bin', 'ffmpeg.exe');
ffmpegCommand = sprintf('%s  -framerate %s -i %s -c copy %s', ff, frame_rate, h264FilePath, mp4FilePath)
status = system(ffmpegCommand);
if status == 0
    disp('Conversion completed successfully.');
else
    disp('Conversion failed.');
end
```

Create 2 folder for the 2 cameras. The code reads the mp4 video (video path) and takes a picture, after that, the picture is rectified and saved in those folders.

```
ImageFolder_left ='C:\Users\Alex\Desktop\Vid2\Left';
ImageFolder_right ='C:\Users\Alex\Desktop\Vid2\Right';

vid = VideoReader('C:\Users\Alex\Dropbox\ALEX\UPC\TFG\Videos\myvideo_2.mp4')
for k = 1:vid.NumberOfFrames
    img=read(vid, k);
    img_left =img(:, 1:end/2,:);
    img_right = img(:, end/2+1:end,:);
    [frameLeftRect, frameRightRect, reprojectionMatrix] = rectifyStereoImages(img_left, img_right, ...
        stereoParams,"linear");
    file_name_1 = sprintf('Image%d.jpg',k);
    file_name_2 = sprintf('Image%d.jpg',k);
    imgName_1 = fullfile(ImageFolder_left,file_name_1);
    imwrite(frameLeftRect,imgName_1,'Quality',100);
    imgName_2 = fullfile(ImageFolder_right,file_name_2) ;
    imwrite(frameRightRect,imgName_2,'Quality',100);
    k=k+1;
end

```

Preparing datastores:

Load the re-projection matrix, that can be obtained with the calibration parameters.

Obtain the path where video images are or individual taken photos are (fullfile function in case of several paths is used). Create an image datastore to manage all the images with less computational work. Finally a sorting algorithm in case of indexing image error in the datastore. In order to sort these files, Natsortfiles Add-on from MATLAB has been used. The indexing error is a common error though sometimes doesn't appear, depending how the images are named.

```
load reprojectionMatrix_2.mat

imgFoldersLeft = fullfile('C:\Users\Alex\Desktop\Vid\Left');
imgFoldersRight = fullfile('C:\Users\Alex\Desktop\Vid\Right');
imdsLeft = imageDatastore(imgFoldersLeft);
imdsRight = imageDatastore(imgFoldersRight);
imdsLeft.Files = natsortfiles(imdsLeft.Files);
imdsRight.Files = natsortfiles(imdsRight.Files);

```

Test code (no importance):

This code is only for testing the images disparity of the images of the datastore.

It hasn't got an important role in the reconstruction. Disparity parameter can be changed, specially in the threshold aspect to view the patch size in the disparity map.

```
frameLeftRect = readimage(imdsLeft,28);
frameRightRect = readimage(imdsRight,28);
frameLeftGray  = im2gray(frameLeftRect);
frameRightGray = im2gray(frameRightRect);

f=figure(1);
set(f, 'Visible', 'on');
combinationimages= stereoAnaglyph(frameLeftRect, frameRightRect);
disparityMap = disparitySGM(frameLeftGray,frameRightGray,DisparityRange=[0 128],UniquenessThreshold=7);
hold on;
subplot(1,3,1);
imshow(frameLeftRect);
subplot(1,3,2);
imshow(combinationimages);
subplot(1,3,3);
imshow(disparityMap,[0 128]);
colormap jet
colorbar
hold off;
```

Starting vSLAM, Detecting features/landmarks and tracking:

Selecting in what frame we want to start ("start"). Then analyzing how big the images are.

Creating an slam object (14 levels of pyramidal detection, recall ORB), other configuration not mention, very sensitive in parameters changing.

This code detects a useful key frame to detect the position of the hardware and its surrounding, then passes this frame to the slam object, which calculates and determines the position/track and all the features. Some code is deactivated, it can be activated for curiosity. But no important role is occupied.

```
start=500;
imageSize = size(readimage(imdsLeft,1),[1,2]);% in pixels [mrows, ncols]

vslam = stereovslam(reprojectionMatrix, imageSize, SkipMaxFrames=5, NumLevels=14, UniquenessThreshold=50);

for i = start:numel(imdsLeft.Files)

    frameLeftRect = readimage(imdsLeft,i);
    frameRightRect = readimage(imdsRight,i);
    % Pass each stereo image frame to the vSLAM algorithm.
    addFrame(vslam,frameLeftRect,frameRightRect);

    if hasNewKeyFrame(vslam)
        % Display 3D map points and camera trajectory
        ax = plot(vslam);

    end
    %status = checkStatus(vslam)
end

% while ~isDone(vslam)
%     if hasNewKeyFrame(vslam)
%         ax = plot(vslam);
%     end
% end
figure(1)
% Retrieve camera poses and key frame
[camPoses,keyFrameIDs] = poses(vslam);
estimatedTrajectory = vertcat(camPoses.Translation);

numKeyFrames = numel(keyFrameIDs);
ptClouds = repmat(pointCloud(zeros(1,3)),numKeyFrames,1);
```

Reconstruction:

Predefining a disparity range of the pixels.

Basically this code is the creation with the key frames of disparity maps using the images. Then from this disparity maps with the re-projection matrix create a point cloud (a set of pixels point in the space xyz. Apply color from the left image to this cloud point, and finally a stitching the different cloud points of the key frames to generate a main cloud point.

```
disparityRange = [0 128];
i=1;
for i = 1:numKeyFrames

    frameLeftRect = readimage(imdsLeft,double(keyFrameIDs(i)+start-1));
    frameRightRect = readimage(imdsRight,double(keyFrameIDs(i)+start-1));
    imageSize = size(frameLeftRect,[1,2]);% in pixels [mrows, ncols]

    frameLeftGray = im2gray(frameLeftRect);
    frameRightGray = im2gray(frameRightRect);
    % Reconstruct scene from disparity
    disparityMap = disparitySGM(frameLeftGray,frameRightGray,DisparityRange=disparityRange, UniquenessThreshold=7);
    xyzPoints = reconstructScene(disparityMap,reprojectionMatrix);

    xyzPoints(1:100,:,:) = NaN;

    xyzPoints  = reshape(xyzPoints, [imageSize(1)*imageSize(2), 3]);

    % Get color from the color image
    colors = reshape(frameLeftRect, [imageSize(1)*imageSize(2), 3]);

    % Remove world points that are behind
    % Distant points can be annoying
    % Delete them using and varying the following example in validIndex:
    %   [ & xyzPoints(:,3) < 100/reprojectionMatrix(4,3)];
    % If the user want to remove some the pixels modify validIndex
    validIndex = xyzPoints(:,3) > 0 & xyzPoints(:,3) < 700/reprojectionMatrix(4,3) ;
    xyzPoints = xyzPoints(validIndex,:);
    colors = colors(validIndex,:);

    % Transform world points to the world coordinates
    xyzPoints  = transformPointsForward(camPoses(i),xyzPoints);
    ptClouds(i) = pointCloud(xyzPoints, Color=colors);

    % Downsample the point cloud
    ptClouds(i) = pcdownsample(ptClouds(i),random=0.5);
end

% Concatenate the point clouds into a single
pointCloudsAll = pccat(ptClouds);

figure(2)
ax = pcshow(pointCloudsAll,VerticalAxis="y", VerticalAxisDir="down");
xlabel('X')
ylabel('Y')
zlabel('Z')
title('Dense Reconstruction')

```

This code is the visualization in first point of view and in 3D perspective of the recorded data.

Pretty useful indeed, to generate and visualize distance depth.

```
% Visualize the point cloud
viewer = pcviewer(pointCloudsAll,VerticalAxis="YDown");

% Navigate to the first camera pose
viewer.CameraUpVector = [0 -1 0];
for i = 1:numKeyFrames
    position = camPoses(i).Translation;
    target = [0 0 1 1]*camPoses(i).A';
    viewer.CameraPosition = position;
    viewer.CameraTarget = target(1:3);
    pause(0.1);
end

```

ANNEX 9: ARDUINO UNO CODE

```
// Tested code with Arduino UNO (also with multiple I2C sensors)
// Code done by Alejandro Boadella EETAC student

//Download libraries: Compass, GPS, "Fake/imitation" Serial Sender, SPI bus, I2C bus
#include <QMC5883LCompass.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <SPI.h>
#include <Wire.h>

// Creation of compass object
QMC5883LCompass compass;

// GPS bauds, gps object and serial connection to pins 4 and 3
static const int RXPin = 4, TXPin = 3;
static const uint32_t GPSBaud = 9600;
TinyGPSPlus gps;
SoftwareSerial ss(RXPin, TXPin);

//Declaration of variables and Timer
static const float latt;
static const float longi;
static const float alt;
unsigned long previousMillis=millis();

// Vector that will contain all the data that will be send and a counter for the vector
byte  byteArray[]={0x01,0x02,0x03,0x04};
int counter=0;

// Setup loop only run once
void setup() {
   //Serial bus communication bus the rasp at 115200 bauds
  Serial.begin(115200);
  // Initiation of the compass
  compass.init();
  // Creation of the 'fake' serial bus with the GPS at 9600 bauds
  ss.begin(GPSBaud);
    // Calibrate the compass, in order to change and recalibrate the compass: use the library of
the compass --> QMC5883LCompass.h
  compass.setCalibrationOffsets(1221.00, -4.00, -1818.00);
  compass.setCalibrationScales(1.13, 1.19, 0.78);

  // Declaration of Miso output for SPI bus and setting device as a slave
  pinMode(MISO, OUTPUT);
  SPCR |= _BV(SPE);
}

static void measure_compass() {
     // Read compass values (the output will be the magnetic north, not the true north!!, 2º of
variation respect Catalonia)
    compass.read();
    // Return Azimuth reading, from 0 to 359 degrees
    int longInt = compass.getAzimuth();
    if (longInt<0){
      longInt=360+longInt;
```

```
    }
    // Save the information in the data vector in byte form

    //Serial.print("Compass: ");
    //Serial.println(longInt);
    byteArray[5] = (int)((longInt >> 8) & 0XFF);
    byteArray[4] = (int)((longInt & 0XFF));
    //Serial.println(byteArray[5],HEX);
    //Serial.println(byteArray[4],HEX);
}

static void measure_gps() {
    // See if the gps is available and read data
    if (ss.available()){
       gps.encode(ss.read());
     }
     // If the location is valid, read the data and save in the vector data the latitude and the
longitude
    if (!(gps.location.isValid())){
    }
    else{
      float latt=gps.location.lat();
      float longi=gps.location.lng();
      //Creation of an extra byte of information to determine signs and others
      if (latt<0 and longi<0){
        byteArray[6]=0x00;
      }

      if (latt<0 and longi>0){
        byteArray[6]=0x01;
      }

      if (latt>0 and longi<0){
        byteArray[6]=0x02;
      }
      else{
        byteArray[6]=0x03;
      }
      //save the data in another format and multiply the data by 10^10
      uint64_t latts= trunc((abs(latt*pow(10,10))));
      uint64_t longis= trunc(abs(longi*pow(10,10)));

      //Pass the data from integer to byte format and save the data inside the vector
      byteArray[12] = ((latts >> 40) & 0XFF);
      byteArray[11] = ((latts >> 32) & 0XFF);
      byteArray[10] = ((latts >> 24) & 0XFF);
      byteArray[9] = ((latts >> 16) & 0XFF);
      byteArray[8] = ((latts >> 8) & 0XFF);
      byteArray[7] = ((latts & 0XFF));

      //Serial.println(latt,20);
      //Serial.println(byteArray[12],HEX);
      //Serial.println(byteArray[11],HEX);
      //Serial.println(byteArray[10],HEX);
      //Serial.println(byteArray[9],HEX);
      //Serial.println(byteArray[8],HEX);
      //Serial.println(byteArray[7],HEX);

      //Pass the data from integer to byte format and save the data inside the vector
```

```
      byteArray[18] = ((longis >> 40) & 0XFF);
      byteArray[17] = ((longis >> 32) & 0XFF);
      byteArray[16] = ((longis >> 24) & 0XFF);
      byteArray[15] = ((longis >> 16) & 0XFF);
      byteArray[14] = ((longis >> 8) & 0XFF);
      byteArray[13] = ((longis & 0XFF));

      //Serial.println(longi,20);
      //Serial.println(byteArray[18],HEX);
      //Serial.println(byteArray[17],HEX);
      //Serial.println(byteArray[16],HEX);
      //Serial.println(byteArray[15],HEX);
      //Serial.println(byteArray[14],HEX);
      //Serial.println(byteArray[13],HEX);

    }
}

static void measure_height(){
    // Smartdelay is used to not lose connection and see if it is available
    smartDelay(0);
    if (!(gps.altitude.isValid())){
    }
    else{
      float alt=gps.altitude.meters();
      int alti= (int) alt;
      byteArray[20] = (int)((alti >> 8) & 0XFF);
      byteArray[19] = (int)((alti & 0XFF));
      //Serial.print("ALT: ");
      //Serial.println(alti);
    }
    // Finally save the height data inside the vector
}

static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}

static void send_bus(){
      //If the SPI bus detects signal from the raspberry pi it will send a byte every loop of the
arduino, thus the counter.
    if ((SPSR & (1 << SPIF)) != 0)
    {
      // Send the byte of the data vector to the rasp and then change into another byte
      SPDR = byteArray[counter];
      counter++;

      if (counter==21 ){
        counter=0;
      }
    }
}
//Main loop
```

```
void loop() {
  // With the timer, every 1s measure compass, gps
  // Measure height must be done periodically with the objective of receiving data
  if (millis()- previousMillis> 1000ul)
  {
    measure_compass();
    measure_gps();
    previousMillis = millis();
  }
  // Send 1 byte data through the SPI bus every loop
  measure_height();
  send_bus();
}
```

ANNEX 10: STM32 BASIC CODE

```
// Tested code with STM32F405 (also with multiple I2C sensors)
// Code done by Alejandro Boadella EETAC student

//Download libraries: Compass, GPS, "Fake/imitation" Serial Sender,I2C bus
#include <QMC5883LCompass.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <Wire.h>

// Creation of compass object
QMC5883LCompass compass;

// GPS bauds, gps object and serial connection to pins RX=PC2 and TX=PC3
static const uint32_t GPSBaud = 9600;
TinyGPSPlus gps;
SoftwareSerial ss(PC2, PC3);

// Creation of the vector that contains all the data transmitted to the raspberry pi
float DATA[3]={};
// Timer
unsigned long previousMillis=millis();

// Setup loop only run once
void setup() {
  //Serial bus comunication bus the rasp at 115200 bauds
  Serial.begin(115200);
  // Initiation of the compass
  compass.init();
  // Creation of the 'fake' serial bus with the GPS at 9600 bauds
  ss.begin(GPSBaud);
   // Calibrate the compass, in order to change and recalibrate the compass: use the library of
the compass --> QMC5883LCompass.h
  compass.setCalibrationOffsets(1006.00, 75.00, -1510.00);
  compass.setCalibrationScales(1.15, 1.22, 0.76);

}

void calib_compass()
{
  delay(5000);
  Serial.println("CALIBRATING. Move your sensor!!");
  compass.calibrate();

compass.setCalibrationOffsets(compass.getCalibrationOffset(0),compass.getCalibrationOffse
t(1),compass.getCalibrationOffset(2));
                                    compass.setCalibrationScales(compass.getCalibrationScale(0),
compass.getCalibrationScale(1),compass.getCalibrationScale(2));
  Serial.println("DONE");
}

void measure_compass() {

     // Read compass values (the output will be the magnetic north, not the true north!!, 2º of
variation respect Catalonia)
    compass.read();
```

```
    // Return Azimuth reading, from 0 to 359 degrees
    float comp = compass.getAzimuth();
    //if (comp<0){
    //  comp=360+comp;
    //}
    // Save the information in the data vector
    DATA[0]=comp+180;
}

void measure_gps() {
    // See if the gps is available and read data
    if (ss.available()){
        gps.encode(ss.read());
      }
      // If the location is valid, read the data and save in the vector data the latitude and the
longitude
    if (!(gps.location.isValid())){

    }
    else{
      DATA[1]=gps.location.lat();
      DATA[2]=gps.location.lng();
    }
    DATA[1]=gps.location.lat();
    DATA[2]=gps.location.lng();
}

void measure_height(){
    // Smartdelay is used to not lose connection and see if it is available
    smartDelay(0);
    if (!(gps.altitude.isValid())){
    }
    else{
      DATA[3]=gps.altitude.meters();
    }
    // Finally save the height data inside the vector
}

void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}

void send_bus(){
    // Send via serial and ASCII all the data retained in the vector
    Serial.print(DATA[0]);
    Serial.print(",");
    Serial.print(DATA[1],7);
    Serial.print(",");
    Serial.print(DATA[2],7);
    Serial.print(",");
    Serial.println(DATA[3],1);
}
```

```
//Main loop
void loop() {
  // With the timer, every 1,5s measure compass, gps and send all the data through the serial bus
  // Measure height must be done periodically with the objective of receiving data
  if (millis()- previousMillis> 1500ul)
  {
    measure_compass();
    measure_gps();
    send_bus();
    previousMillis = millis();

  }
  measure_height();
}
```

ANNEX 11: STM32 DRONE CODE

- Main, where the setup and the main loop acts:

```
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//This code has been made and compiled by Alejandro Boadella (EETAC-UPC student).
// Main source:
// - Brokking.net Link: http://www.brokking.net/ymfc-32_auto_main.html#8.2
// Secondary sources:
// - Design and implementation of a new quadcopter drone prototype
// controlled through STM32 using the Arduino environment by León Enrique Prieto Bailo.
// - Arduproject.es
// It works only on stabilize mode. GPS mode doesn't work properly, little changes have to be
made.
// This is due by the time factor on the development of this project. Some variables are not
needed.
// I apologize for the inconvenience of any not mentioned source or any error committed
// in the readaction of the code.
////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////

//Libraries used:
#include <Wire.h>                 // I2C bus library.
#include <SoftwareSerial.h>          // Serial bus for GPS "digital serial".

// Flight Mode Enumeration (i.e: FM_disabled==flight mode 0)
enum FlightMode{
  FM_disabled,                     // Disabled mode.
  FM_mounting,                      // Starts to be prepared for take off mode.
  FM_stable,                       // No balance fly mode.
  FM_stable_battery,                  // Stable mode + Battery corrector.
  FM_gps_hold                      // Mode that uses GPS to be stable.
};

 FlightMode fm = FM_disabled;           //Starting flight mode (fm)

// Serial connection to pins RX=PC2/Pin 12 and TX=PC3/Pin 11 for the GNSS receviver
SoftwareSerial ss(PC2, PC3);

// LED
volatile long led_timer;

// MOTORS
long time_motores_start, time_1, time_2, time_ON;

// FlightSky i6
#define pin_PPM PA4                        // Pin radio PPM.
#define number_channels 8                     // 6 channels and 2 phantom channels.
                                  //if you increase the number of these channels, the phantom
channels won't be there.
uint64_t pulse_instant[number_channels * 2 + 2];    // 8 channels that contains 8 rises and 8
falls in the PPM = 16 + 2 for the markers.
uint16_t remote_channel[number_channels];          // Vector of values of channels.
uint8_t flank_count = 1;                     // Counter.
int16_t throttle;
int32_t pid_roll_setpoint_base;
int32_t pid_pitch_setpoint_base;
```

```
// PID: Variables
float roll_level_adjust, pitch_level_adjust;
float pid_error_temp;
float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;
float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch,
pid_last_pitch_d_error;
float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw,
pid_last_yaw_d_error;

// Check:
float pid_t_control_error, pid_t_output, pid_rate_control_error, pid_i_mem_rate,
pid_rate_error_prev, pid_output_rate;

// PID: Roll
float pid_p_gain_roll = 0.9;      // (By Default Brokking uses) p=1.3 i=0.04 d=18
float pid_i_gain_roll = 0.009;
float pid_d_gain_roll = 4;
int pid_max_roll = 400;

// PID: Pitch
float pid_p_gain_pitch = pid_p_gain_roll;
float pid_i_gain_pitch = pid_i_gain_roll;
float pid_d_gain_pitch = pid_d_gain_roll;
int pid_max_pitch = 400;

// PID: Yaw
float pid_p_gain_yaw = 0.75;
float pid_i_gain_yaw = 0.01;
float pid_d_gain_yaw = 0;
int pid_max_yaw = 400;

// MPU6050:
#define MPU6050_ADDRESS 0x68
#define MPU6050_ACCEL_XOUT_H 0x3B
#define MPU6050_PWR_MGMT_1 0x6B
#define MPU6050_GYRO_CONFIG 0x1B
#define MPU6050_ACCEL_CONFIG 0x1C
#define MPU6050_CONFIG 0x1A
int16_t manual_gyro_pitch_cal_value = 0;
int16_t manual_gyro_roll_cal_value = 0;
int16_t manual_gyro_yaw_cal_value = 0;
int16_t manual_x_cal_value = 0;
int16_t manual_y_cal_value = 0;
int16_t manual_z_cal_value = 0;
boolean auto_level = true;                  // This makes that it calibrates automatically the IMU
every time the code starts.
uint8_t use_manual_calibration = false;        // No special use. Change if you want manual
calibration.
int32_t acc_total_vector, acc_total_vector_at_start;
int32_t gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;
int16_t acc_pitch_cal_value;
int16_t acc_roll_cal_value;
int16_t cal_int;
int16_t temperature;
int16_t acc_x, acc_y, acc_z;
int16_t gyro_pitch, gyro_roll, gyro_yaw;
int32_t acc_x_cal, acc_y_cal, acc_z_cal;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll, angle_yaw;
```

```
// ESC
#define pin_motor1 PC6                    // Pin motor 1  GPIO 6
#define pin_motor2 PC7                    // Pin motor 2  GPIO 5
#define pin_motor3 PB9                    // Pin motor 3  GPIO 10
#define pin_motor4 PB8                    // Pin motor 4  GPIO 9
int16_t esc_1, esc_2, esc_3, esc_4;         // Values for outputs.

TIM_TypeDef *TIM_DEF_M1_M2 = TIM3;          // This code defines 2 hardware timers that
will take the role of send PWM pulses to the motors.
TIM_TypeDef *TIM_DEF_M3_M4 = TIM4;

uint32_t channel_motor1 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor1), PinMap_PWM));
// Defines a PWM output.
uint32_t channel_motor2 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor2), PinMap_PWM));
uint32_t channel_motor3 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor3), PinMap_PWM));
uint32_t channel_motor4 =
STM_PIN_CHANNEL(pinmap_function(digitalPinToPinName(pin_motor4), PinMap_PWM));

HardwareTimer *TIM_M1_M2 = new HardwareTimer(TIM_DEF_M1_M2); // creation of the
timers.
HardwareTimer *TIM_M3_M4 = new HardwareTimer(TIM_DEF_M3_M4);

// Loop timer
uint32_t loop_timer;

// Error signal
uint8_t error;

// GPS
boolean gps_status = false;               // It start in false, then if the user wants to use GPS a
previous calibration shall be done to use it.
float gps_p_gain = 2.7;                   //Gain setting for the GPS P-controller (default = 2.7).
float gps_d_gain = 6.5;                   //Gain setting for the GPS D-controller (default = 6.5).
float declination = 0.0;                 //Set the declination between the magnetic and geographic
north. (Barcelona has 2 degress approx in 2024).

uint8_t read_serial_byte, incomming_message[100], number_used_sats, fix_type;
uint8_t waypoint_set = 0, latitude_north, longiude_east ;
uint16_t message_counter;
int16_t gps_add_counter;
int32_t l_lat_gps, l_lon_gps, lat_gps_previous, lon_gps_previous;
int32_t lat_gps_actual, lon_gps_actual, l_lat_waypoint, l_lon_waypoint;
float gps_pitch_adjust_north, gps_pitch_adjust, gps_roll_adjust_north, gps_roll_adjust;
float lat_gps_loop_add, lon_gps_loop_add, lat_gps_add, lon_gps_add;
uint8_t new_line_found, new_gps_data_available, new_gps_data_counter;
uint8_t gps_rotating_mem_location, return_to_home_step;
int32_t gps_lat_total_avarage, gps_lon_total_avarage;
int32_t gps_lat_rotating_mem[40], gps_lon_rotating_mem[40];
int32_t gps_lat_error, gps_lon_error;
int32_t gps_lat_error_previous, gps_lon_error_previous;
uint32_t gps_watchdog_timer;

float l_lon_gps_float_adjust, l_lat_gps_float_adjust, gps_man_adjust_heading;
float return_to_home_lat_factor, return_to_home_lon_factor, return_to_home_move_factor;
uint8_t home_point_recorded;
```

```
int32_t lat_gps_home, lon_gps_home;

// Compass HMC5883L:
boolean hold_heading_status = false;      // It makes a first measure of the compass to set the heading and enables GPS.
uint8_t compass_address = 0x1E;
uint8_t compass_calibration_on, heading_lock;
int16_t compass_x, compass_y, compass_z;
int16_t compass_cal_values[6];
float compass_x_horizontal, compass_y_horizontal, actual_compass_heading;
float compass_scale_y, compass_scale_z;
int16_t compass_offset_x, compass_offset_y, compass_offset_z;
float course_a, course_b, course_c, base_course_mirrored, actual_course_mirrored;
float course_lock_heading, heading_lock_course_deviation;


// Battery: (It needs to be a 3S battery or the code should be modified):
float low_battery_warning = 10.5;         //Set the battery warning at 10.5V (default = 10.5V).
float battery_voltage;
float battery_compensation = 40.0;       // Increment if the drone falls when the battery voltage falls.
                                         // Decrement if the drone rises when the battery voltage falls.

// Timer to print values via serial:
int long timp;

////////////////////////////////////////////////////////////////////////////////////////////////////
// ////////////////////////////////////       Setup routine       ////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(115200);                  // Start serial communication at 115200 bauds.
  Wire.begin();                          // Start I2C bus at 400khz (high velocity).
  Wire.setClock(400000);
  delay(1000);

  init_components();                     // Start basic components.
  read_rc();                             // Read radio.
  led_on();                              // Show that the drone is activated.
  delay(250);

  if (remote_channel[5] < 1100){         // While and if the drone channel 5 is lower than 1100us, activate ESC calibration.
    read_rc();
    while (remote_channel[5] < 1100){
      read_rc();

       esc_1 = remote_channel[3];        // Read radio and see throttle channel.
       esc_2 = remote_channel[3];
       esc_3 = remote_channel[3];
       esc_4 = remote_channel[3];

       if (esc_1 < 1000) esc_1 = 1000;   // Enable lower and upper pulse limits.
       if (esc_2 < 1000) esc_2 = 1000;
       if (esc_3 < 1000) esc_3 = 1000;
       if (esc_4 < 1000) esc_4 = 1000;

       if (esc_1 > 2000) esc_1 = 2000;
       if (esc_2 > 2000) esc_2 = 2000;
       if (esc_3 > 2000) esc_3 = 2000;
```

```
        if (esc_4 > 2000) esc_4 = 2000;

      TIM_M1_M2->setCaptureCompare(channel_motor1, esc_1,
MICROSEC_COMPARE_FORMAT);   // Send output data to the motors.
      TIM_M1_M2->setCaptureCompare(channel_motor2, esc_2,
MICROSEC_COMPARE_FORMAT);
      TIM_M3_M4->setCaptureCompare(channel_motor3, esc_3,
MICROSEC_COMPARE_FORMAT);
      TIM_M3_M4->setCaptureCompare(channel_motor4, esc_4,
MICROSEC_COMPARE_FORMAT);
   }
 }

 if (remote_channel[5] > 1900){    // if the channel 5 is higher than 1900us calibrate compass.
  led_off();                      // Show motors are not enable.
  delay(200);
  setup_compass();
  calibrate_compass();            // Initialize the compass and set the correct registers.
  read_compass();                 // Read and calculate the compass data.
  gps_status=true;                // Enable GPS
  delay(200);
  led_on();                       // Enable drone
  delay(200);
 }

 led_off();
 delay(200);
 init_imu();                      // Calibrate IMU or MPU-6050.
 delay(200);
 led_on();
 loop_timer = micros();
 Serial.println("Setup finished");
 timp=millis();
 read_process_units();            // Make basic measurement and calculus.
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////       Main routine    ////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
void loop() {

 reference_computation();                                  // Seeking for fligh mode

     if (fm == FM_gps_hold && hold_heading_status == false){        // If the GPS mode is set,
enable it and get the heading angle.
       gps_setup();                                       // Set the baud rate and output refresh rate
of the GPS module.    ¡¡¡ If the code stops probably read flush serial buffer !!!
       read_compass();                                   // Read and calculate the compass data.
       angle_yaw = actual_compass_heading;                       // Set the initial compass
heading.
       course_lock_heading = angle_yaw;                          // Set the heading as the main
angle in reference computation.
       hold_heading_status=true;
     }

  read_process_units();                                  // Process main units.

    if (fm == FM_gps_hold && hold_heading_status == true){          // This code enables
GPS positionaing and also at the same time heading correction
```

```
        read_compass();                                          // Read and calculate the
compass data.
        if (gps_add_counter >= 0)gps_add_counter --;

        read_gps();

        angle_yaw -= course_deviation(angle_yaw, actual_compass_heading) / 1200.0;       //
Calculate the difference between the gyro and compass heading and make a small
correction.
        if (angle_yaw < 0) angle_yaw += 360;                                  // If the compass
heading becomes smaller then 0, 360 is added to keep it in the 0 till 360 degrees range.
        else if (angle_yaw >= 360) angle_yaw -= 360;                          // If the compass
heading becomes larger then 360, 360 is subtracted to keep it in the 0 till 360 degrees range.

        gps_man_adjust_heading = angle_yaw;                                   // Send GPS
heading correction.

        //When the heading_lock mode is activated the roll and pitch pid setpoints are heading
dependent.
        //At startup the heading is registerd in the variable course_lock_heading.
        //First the course deviation is calculated between the current heading and the
course_lock_heading.
        //Based on this deviation the pitch and roll controls are calculated so the responce is
the same as on startup.

        if (hold_heading_status == true) {                                    // This code will only be
activated if GPS at that instance isn't enable due loss of GPS information. Put false to
desactivate!!!!!!!!!
                                                                              // Put false if you dont want to use it.
It doesn't correct the heading, only corrects the movement of the drone respect the pilot
orientation.
          heading_lock_course_deviation = course_deviation(angle_yaw,
course_lock_heading);
          remote_channel[1] = 1500 + ((float)(remote_channel[1] - 1500) *
cos(heading_lock_course_deviation * 0.017453)) + ((float)(remote_channel[2] - 1500) *
cos((heading_lock_course_deviation - 90) * 0.017453));
          remote_channel[2] = 1500 + ((float)(remote_channel[2] - 1500) *
cos(heading_lock_course_deviation * 0.017453)) + ((float)(remote_channel[1] - 1500) *
cos((heading_lock_course_deviation + 90) * 0.017453));
          gps_man_adjust_heading = course_lock_heading;
        }
    }


    if (fm == FM_gps_hold && waypoint_set == 1) {     // If the drone is in GPS mode and
there's a waypoint.
      pid_roll_setpoint_base = 1500 + gps_roll_adjust;
      pid_pitch_setpoint_base = 1500 + gps_pitch_adjust;
    }
    else {
      pid_roll_setpoint_base = remote_channel[1];
      pid_pitch_setpoint_base = remote_channel[2];
    }

    //Because we added the GPS adjust values we need to make sure that the control limits
are not exceded.
    if (pid_roll_setpoint_base > 2000)pid_roll_setpoint_base = 2000;
    if (pid_roll_setpoint_base < 1000)pid_roll_setpoint_base = 1000;
```

```
    if (pid_pitch_setpoint_base > 2000)pid_pitch_setpoint_base = 2000;
    if (pid_pitch_setpoint_base < 1000)pid_pitch_setpoint_base = 1000;

  controllers();  // Apply PID corrections.

  actuators();    // Send signals to the motors.

  if (micros() - loop_timer > 4050) {     // It the loop is too slow, show it.
    Serial.println("LOOP SLOW");
  }

  while (micros() - loop_timer < 4000);   //We wait until 4000us are passed.
  loop_timer = micros();

  /*if (millis() - timp > 1000) {          // Every second send information of anything the user want
like, compass angle, GPS position, etc.

    Serial.print("FM:");
    Serial.println(fm);

    timp=millis();
  }*/

}
```

- Actuators, delivers the output:

```
void actuators() {
  act_esc_outputs();
  act_esc_PWM();
}

void act_esc_outputs() {
  if (fm >=2) {
    if (throttle > 1800) throttle = 1800;                                    // Limit max throttle.
    esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw;        //Calculate the
pulse for esc 1 (front-right - CCW).
    esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw;        //Calculate the
pulse for esc 2 (rear-right - CW).
    esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw;        //Calculate the
pulse for esc 3 (rear-left - CCW).
    esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw;        //Calculate the
pulse for esc 4 (front-left - CW).

    if (battery_voltage < 12.40 && battery_voltage > 6.0 && fm == FM_stable_battery) {
//Is the battery connected and in the battery mode?
      esc_1 += (12.40 - battery_voltage) * battery_compensation;                //Compensate the
esc-1 pulse for voltage drop.
      esc_2 += (12.40 - battery_voltage) * battery_compensation;                //Compensate the
esc-2 pulse for voltage drop.
      esc_3 += (12.40 - battery_voltage) * battery_compensation;                //Compensate the
esc-3 pulse for voltage drop.
      esc_4 += (12.40 - battery_voltage) * battery_compensation;                //Compensate the
esc-4 pulse for voltage drop.
    }

  }
  else {
    esc_1 = 1000;
    esc_2 = 1000;
    esc_3 = 1000;
    esc_4 = 1000;
  }

  if (esc_1 < 1000) esc_1 = 950;              //Limit the esc's pulse to 950us.
  if (esc_2 < 1000) esc_2 = 950;
  if (esc_3 < 1000) esc_3 = 950;
  if (esc_4 < 1000) esc_4 = 950;

  if (esc_1 > 2000) esc_1 = 2000;             //Limit the esc's pulse to 2000us.
  if (esc_2 > 2000) esc_2 = 2000;
  if (esc_3 > 2000) esc_3 = 2000;
  if (esc_4 > 2000) esc_4 = 2000;
}
void act_esc_PWM(){
  TIM_M1_M2->setCaptureCompare(channel_motor1, esc_1,
MICROSEC_COMPARE_FORMAT);   // Send pulse to the motors.
  TIM_M1_M2->setCaptureCompare(channel_motor2, esc_2,
MICROSEC_COMPARE_FORMAT);
  TIM_M3_M4->setCaptureCompare(channel_motor3, esc_3,
MICROSEC_COMPARE_FORMAT);
  TIM_M3_M4->setCaptureCompare(channel_motor4, esc_4,
MICROSEC_COMPARE_FORMAT);
}
```

- Controller, calculates the PID output:

```
void controllers() {
  cnt_attitude_sp();
  cnt_attitude_pid();
}

void cnt_attitude_sp() {    // Some of this section has been modified, though it isn't too much noticeable.

  //The PID set point in degrees per second is determined by the roll receiver input.
  //In the case of deviding by 3 the max roll rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
  pid_roll_setpoint = 0;
  //We need a little dead band of 16us for better results.
  if (pid_roll_setpoint_base > 1501) pid_roll_setpoint = pid_roll_setpoint_base - 1501;
  else if (pid_roll_setpoint_base < 1499) pid_roll_setpoint = pid_roll_setpoint_base - 1499;

  pid_roll_setpoint -= roll_level_adjust;  //Subtract the angle correction from the standardized receiver roll input value.
  pid_roll_setpoint /= 3.0;                //Divide the setpoint for the PID roll controller by 3 to get angles in degrees.


  //The PID set point in degrees per second is determined by the pitch receiver input.
  //In the case of deviding by 3 the max pitch rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
  pid_pitch_setpoint = 0;
  //We need a little dead band of 16us for better results.
  if (pid_pitch_setpoint_base > 1501) pid_pitch_setpoint = pid_pitch_setpoint_base - 1501;
  else if (pid_pitch_setpoint_base < 1499) pid_pitch_setpoint = pid_pitch_setpoint_base - 1499;

  pid_pitch_setpoint -= pitch_level_adjust;  //Subtract the angle correction from the standardized receiver pitch input value.
  pid_pitch_setpoint /= 3.0;                 //Divide the setpoint for the PID pitch controller by 3 to get angles in degrees.

  //The PID set point in degrees per second is determined by the yaw receiver input.
  //In the case of deviding by 3 the max yaw rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
  pid_yaw_setpoint = 0;
  //We need a little dead band of 16us for better results.
  if (remote_channel[3] > 1050) {  //Do not yaw when turning off the motors.
    if (remote_channel[4] > 1550) pid_yaw_setpoint = (remote_channel[4] - 1550) / 3.0;
    else if (remote_channel[4] < 1450) pid_yaw_setpoint = (remote_channel[4] - 1450) / 3.0;
  }
}

void cnt_attitude_pid() {
  //Roll calculations
  pid_error_temp = gyro_roll_input - pid_roll_setpoint;
  pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
  if (pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
  else if (pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;

  pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll * (pid_error_temp - pid_last_roll_d_error);
  if (pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
```

```
  else if (pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;

  pid_last_roll_d_error = pid_error_temp;

  //Pitch calculations
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if (pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
  else if (pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch * -1;

  pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch +
pid_d_gain_pitch * (pid_error_temp - pid_last_pitch_d_error);
  if (pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
  else if (pid_output_pitch < pid_max_pitch * -1)pid_output_pitch = pid_max_pitch * -1;

  pid_last_pitch_d_error = pid_error_temp;

  //Yaw calculations
  pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
  pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
  if (pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
  else if (pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;

  pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw *
(pid_error_temp - pid_last_yaw_d_error);
  if (pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
  else if (pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;

  pid_last_yaw_d_error = pid_error_temp;
}
```

- Others, turn on/off the board led:

```
// LED ON
void led_on() {
  digitalWrite(PC1, HIGH);
}

// 4 LED BLINKS / SEC
void led_fast() {
  if ((digitalRead(PC1) == LOW) && (millis() - led_timer > 250)) {
    digitalWrite(PC1, HIGH);
    led_timer = millis();
  } else if ((digitalRead(PC1) == HIGH) && (millis() - led_timer > 250)) {
    digitalWrite(PC1, LOW);
    led_timer = millis();
  }
}

// 1 LED BLINK / SEC
void led_slow() {
  if ((digitalRead(PC1) == LOW) && (millis() - led_timer > 1000)) {
    digitalWrite(PC1, HIGH);
    led_timer = millis();
  } else if ((digitalRead(PC1) == HIGH) && (millis() - led_timer > 1000)) {
```

```
    digitalWrite(PC1, LOW);
    led_timer = millis();
  }
}

// LED OFF
void led_off() {
  digitalWrite(PC1, LOW);
}
```

- Initiation and calibration:

```
void init_components() {
  init_led();
  init_rc();
  init_esc();
}

// Led output declaration:
void init_led() {
  pinMode(PC1, OUTPUT);
}

// Radio control PPM read setup:
void init_rc() {
  pinMode(pin_PPM, INPUT);                                    // Declare the pin as an input.
  attachInterrupt(digitalPinToInterrupt(pin_PPM), read_PPM, CHANGE);        // Declare that
every time the signal changes read_PPM will be applied as an interruption.
}

// Reads the PPM signal: (IT DOESN'T PROCESSES)
void read_PPM() {
  if (micros() - pulse_instant[flank_count - 1] > 2500) flank_count = 0;      // If the changer of
the signal timer is higher than 2500us the PPM whole signal restarts leaving a counter will 0
as a value.
  pulse_instant[flank_count] = micros();                          // Safe in a vector the time
when the signal rises or falls.
  flank_count++;                                                // Counter increment when the signal
rises or falls.
}

// Declaration motor PWM with the Timers:
void init_esc() {
  TIM_M1_M2->setPWM(channel_motor1, pin_motor1, 250, 0);
  TIM_M1_M2->setPWM(channel_motor2, pin_motor2, 250, 0);
  TIM_M3_M4->setPWM(channel_motor3, pin_motor3, 250, 0);
  TIM_M3_M4->setPWM(channel_motor4, pin_motor4, 250, 0);
}

// GPS setup:
void gps_setup(void){

  ss.begin(57600);                                              // GPS setup at 57600 bauds, it could
be more for more efficient code.
```

```
  delay(250);

}

// Compass calibration:
void calibrate_compass(void) {
  compass_calibration_on = 1;                                  // Set the
compass_calibration_on variable to disable the adjustment of the raw compass values.
  while (remote_channel[2] < 1900) {                           // Stay in this loop until the pilot
pulls up the pitch stick of the transmitter.
    delayMicroseconds(3700);                                   // Simulate a 250Hz program
loop.
    read_compass();                                            // Read the raw compass values.
    read_rc();
    // In the following lines the maximum and minimum compass values are detected and
stored.
    if (compass_x < compass_cal_values[0])compass_cal_values[0] = compass_x;
    if (compass_x > compass_cal_values[1])compass_cal_values[1] = compass_x;
    if (compass_y < compass_cal_values[2])compass_cal_values[2] = compass_y;
    if (compass_y > compass_cal_values[3])compass_cal_values[3] = compass_y;
    if (compass_z < compass_cal_values[4])compass_cal_values[4] = compass_z;
    if (compass_z > compass_cal_values[5])compass_cal_values[5] = compass_z;
  }
  compass_calibration_on = 0;                                  // Reset the
compass_calibration_on variable.

  setup_compass();                                             // Initiallize the compass and set the
correct registers.
  read_compass();                                              // Read and calculate the compass
data.
  angle_yaw = actual_compass_heading;                          // Set the initial compass
heading.

  loop_timer = micros();                                       //Set the timer for the next loop.
}

//At startup the registers of the compass need to be set. After that the calibration offset and
scale values are calculated.
void setup_compass() {
  Wire.beginTransmission(compass_address);                     // Start communication with the
compass.
  Wire.write(0x00);                                            // We want to write to the Configuration
Register A (00 hex).
  Wire.write(0x78);                                            // Set the Configuration Regiser A bits as
01111000 to set sample rate (average of 8 at 75Hz).
  Wire.write(0x20);                                            // Set the Configuration Regiser B bits as
00100000 to set the gain at +/-1.3Ga.
  Wire.write(0x00);                                            // Set the Mode Regiser bits as 00000000 to set
Continues-Measurement Mode.
  Wire.endTransmission();                                      // End the transmission with the compass.

  // Calculate the calibration offset and scale values
  compass_scale_y = ((float)compass_cal_values[1] - compass_cal_values[0]) /
(compass_cal_values[3] - compass_cal_values[2]);
  compass_scale_z = ((float)compass_cal_values[1] - compass_cal_values[0]) /
(compass_cal_values[5] - compass_cal_values[4]);

  compass_offset_x = (compass_cal_values[1] - compass_cal_values[0]) / 2 -
compass_cal_values[1];
```

```
  compass_offset_y = (((float)compass_cal_values[3] - compass_cal_values[2]) / 2 -
compass_cal_values[3]) * compass_scale_y;
  compass_offset_z = (((float)compass_cal_values[5] - compass_cal_values[4]) / 2 -
compass_cal_values[5]) * compass_scale_z;
}

// Start the IMU/MPU-6050:
void init_imu(void) {
  Wire.beginTransmission(MPU6050_ADDRESS);
  error = Wire.endTransmission();
  while (error != 0) {
    delay(4);
  }

  Wire.beginTransmission(MPU6050_ADDRESS);            // Start communication with the
MPU-6050.
  Wire.write(MPU6050_PWR_MGMT_1);                     // We want to write to the
PWR_MGMT_1 register (6B hex).
  Wire.write(0x00);                                   // Set the register bits as 00000000 to activate the
gyro.
  Wire.endTransmission();                             // End the transmission with the gyro.

  Wire.beginTransmission(MPU6050_ADDRESS);
  Wire.write(MPU6050_GYRO_CONFIG);                    // We want to write to the
GYRO_CONFIG register (1B hex)
  Wire.write(0x08);                                   // Set the register bits as 00001000 (500dps full
scale).
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_ADDRESS);
  Wire.write(MPU6050_ACCEL_CONFIG);                   // We want to write to the
ACCEL_CONFIG register (1A hex).
  Wire.write(0x10);                                   // Set the register bits as 00010000 (+/- 8g full scale
range).
  Wire.endTransmission();

  Wire.beginTransmission(MPU6050_ADDRESS);
  Wire.write(MPU6050_CONFIG);                         // We want to write to the CONFIG register
(1A hex).
  Wire.write(0x03);                                   // Set the register bits as 00000011 (Set Digital Low
Pass Filter to ~43Hz).
  Wire.endTransmission();

  if (use_manual_calibration) cal_int = 2000;
  else {
    cal_int = 0;
    gyro_pitch_cal = 0;
    gyro_roll_cal = 0;
    gyro_yaw_cal = 0;
  }

  // Let's take multiple gyro data samples so we can determine the calibration parameters.
  if (cal_int != 2000) {                              // If it isn't 2000 value.
    for (cal_int = 0; cal_int < 2000; cal_int++) {    // Take 2000 readings for calibration.

      read_imu();                                     // Read the gyro output.

      gyro_roll_cal += gyro_roll;                     // Ad roll value to gyro_roll_cal.
      gyro_pitch_cal += gyro_pitch;
```

```
    gyro_yaw_cal += gyro_yaw;

    acc_x_cal += acc_x;
    acc_y_cal += acc_y;
    acc_z_cal += acc_z;
    delay(4);
  }
  gyro_roll_cal /= 2000;                          // Divide the roll total by 2000.
  gyro_pitch_cal /= 2000;
  gyro_yaw_cal /= 2000;
  acc_x_cal /= 2000;
  acc_y_cal /= 2000;
  acc_z_cal /= 2000;
  manual_gyro_pitch_cal_value = gyro_pitch_cal;       // Pass the data to the calibration
values.
  manual_gyro_roll_cal_value = gyro_roll_cal;
  manual_gyro_yaw_cal_value = gyro_yaw_cal;
  manual_x_cal_value = acc_x_cal;
  manual_y_cal_value = acc_y_cal;
  //manual_z_cal_value = acc_z_cal - 4096;            // Reduce value due error factor. (yaw
not needed)

  read_imu();

  acc_total_vector = sqrt((acc_x * acc_x) + (acc_y * acc_y) + (acc_z * acc_z));   //Calculate
the total accelerometer vector.

  if (abs(acc_y) < acc_total_vector) {                                  //Prevent the asin function
to produce a NaN.
    angle_pitch_acc = asin((float)acc_y / acc_total_vector) * 57.296;          //Calculate the
pitch angle.
  }
  if (abs(acc_x) < acc_total_vector) {                                  //Prevent the asin function
to produce a NaN.
    angle_roll_acc = asin((float)acc_x / acc_total_vector) * 57.296;           //Calculate the
roll angle.
  }
  angle_pitch = angle_pitch_acc;                                        //Set the gyro pitch angle
equal to the accelerometer pitch angle when the quadcopter is started.
  angle_roll = angle_roll_acc;
 }
}
```

- Reads and process the units:

```
void read_process_units() {
  read_rc();
  read_imu();
  read_battery();
  process_imu();

}

// Process radio control data:
void read_rc() {
  if (flank_count == 18) {                                                    // If the count
reaches 18 = 2*channels + 2, the whole signal has been read.
    for (int i = 1; i <= number_channels; i++) {
      remote_channel[i] = map(pulse_instant[2 * i] - pulse_instant[2 * i - 1], 600, 1600, 1000,
2000);      // It reads the fall and rise of the signal, does the variation and maps the variation
into an interval from 1000-2000us.
    }
  }
}

// Reads battery voltage:
void read_battery() {

  // Load the battery voltage to the battery_voltage variable.
  // The STM32 uses a 12 bit analog to digital converter.
  // analogRead => 0 = 0V ..... 4095 = 3.3V
  // The voltage divider (1k & 10k) is 1:11.
  // analogRead => 0 = 0V ..... 4095 = 36.3V
  // 36.3 / 4095 = 112.81.
  // The variable battery_voltage holds 1050 if the battery voltage is 10.5V.
  battery_voltage = (float)analogRead(PA5) / 112.81;

  // The battery voltage is needed for compensation.
  // A complementary filter is used to reduce noise.
  // 1410.1 = 112.81 / 0.08.
  battery_voltage = battery_voltage * 0.92 + ((float)analogRead(PA5) / 1410.1);
// If it gives a lower value, map the output to the real values!!!

  // Default setting is 10.5V 3S.
  if (battery_voltage > 6.0 && battery_voltage < low_battery_warning && error ==
0)Serial.println("error bat");

}

// Reads IMU/MPU-6050 data:
void read_imu(void) {
  Wire.beginTransmission(MPU6050_ADDRESS);                  // Start communication with
the gyro.
  Wire.write(MPU6050_ACCEL_XOUT_H);                         // Start reading @ register 43h
and auto increment with every read.
  Wire.endTransmission();                                   // End the transmission.
  Wire.requestFrom(MPU6050_ADDRESS, 14);                    // Request 14 bytes from the
MPU 6050.
  acc_y = Wire.read() << 8 | Wire.read();
  acc_x = Wire.read() << 8 | Wire.read();                   // Add the low and high byte to the acc
variable.
```

```
  acc_z = Wire.read() << 8 | Wire.read();
  temperature = Wire.read() << 8 | Wire.read();
  gyro_roll = Wire.read() << 8 | Wire.read();               // Read high and low part of the
angular data.
  gyro_pitch = Wire.read() << 8 | Wire.read();
  gyro_yaw = Wire.read() << 8 | Wire.read();
  gyro_pitch *= -1;                                         // Invert the direction of the axis. See the
angles, if they are not right, comment this.
  gyro_yaw *= -1;

  acc_x -= manual_x_cal_value;                              // If it isn't in the calibration mode it won't
have 0 as a value.
  acc_y -= manual_y_cal_value;
  acc_z -= manual_z_cal_value;
  gyro_roll -= manual_gyro_roll_cal_value;                  // Subtact the manual gyro roll
calibration value.
  gyro_pitch -= manual_gyro_pitch_cal_value;
  gyro_yaw -= manual_gyro_yaw_cal_value;
}

// Process the IMU:
void process_imu() {
 // 65.5 = 1 deg/sec (check the datasheet of the MPU-6050 for more information).
  gyro_roll_input = (gyro_roll_input * 0.7) + (((float)gyro_roll / 65.5) * 0.3);     // Gyro pid input
is deg/sec.
  gyro_pitch_input = (gyro_pitch_input * 0.7) + (((float)gyro_pitch / 65.5) * 0.3);
  gyro_yaw_input = (gyro_yaw_input * 0.7) + (((float)gyro_yaw / 65.5) * 0.3);

  // Gyro angle calculations
  // 0.0000611 = 1 / (250Hz / 65.5)
  angle_pitch += (float)gyro_pitch * 0.0000611;                                  // Calculate the traveled
pitch angle and add this to the angle_pitch variable.
  angle_roll += (float)gyro_roll * 0.0000611;
  angle_yaw += (float)gyro_yaw * 0.0000611;
  if (angle_yaw < 0) angle_yaw += 360;                                           // If the compass heading
becomes smaller then 0, 360 is added to keep it in the 0 till 360 degrees range.
  else if (angle_yaw >= 360) angle_yaw -= 360;                                   // If the compass
heading becomes larger then 360, 360 is subtracted to keep it in the 0 till 360 degrees range.

 // 0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is in radians
and not degrees.
  angle_pitch -= angle_roll * sin((float)gyro_yaw * 0.000001066);                // If the IMU has
yawed transfer the roll angle to the pitch angel.
  angle_roll += angle_pitch * sin((float)gyro_yaw * 0.000001066);                // If the IMU has
yawed transfer the pitch angle to the roll angel.

  // Accelerometer angle calculations
  acc_total_vector = sqrt((acc_x * acc_x) + (acc_y * acc_y) + (acc_z * acc_z));    // Calculate
the total accelerometer vector.

  if (abs(acc_y) < acc_total_vector) {                                           // Prevent the asin function to
produce a NaN.
    angle_pitch_acc = asin((float)acc_y / acc_total_vector) * 57.296;            // Calculate the
pitch angle.
  }
  if (abs(acc_x) < acc_total_vector) {                                           // Prevent the asin function to
produce a NaN.
    angle_roll_acc = asin((float)acc_x / acc_total_vector) * 57.296;             // Calculate the roll
angle.
```

```
  }

  angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;                 //Correct the drift
of the gyro pitch angle with the accelerometer pitch angle.
  angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;                    //Correct the drift of
the gyro roll angle with the accelerometer roll angle.

  pitch_level_adjust = angle_pitch * 15;                                         //Calculate the pitch angle
correction.
  roll_level_adjust = angle_roll * 15;                                           //Calculate the roll angle
correction.

  if (!auto_level) {                                                             //If the quadcopter is not in auto-level
mode
    pitch_level_adjust = 0;                                                      //Set the pitch angle correction to
zero.
    roll_level_adjust = 0;                                                       //Set the roll angle correcion to
zero.
  }

    //Serial.println(angle_pitch);
    //Serial.println(angle_roll);
    //Serial.println(angle_yaw);
    //Serial.println(acc_y);
    //Serial.println(acc_x);

}


// The following subrouting calculates the smallest difference between two heading values:
float course_deviation(float course_b, float course_c) {
  course_a = course_b - course_c;
  if (course_a < -180 || course_a > 180) {
    if (course_c > 180)base_course_mirrored = course_c - 180;
    else base_course_mirrored = course_c + 180;
    if (course_b > 180)actual_course_mirrored = course_b - 180;
    else actual_course_mirrored = course_b + 180;
    course_a = actual_course_mirrored - base_course_mirrored;
  }
  return course_a;
}

// Reads the compass:
void read_compass() {
  Wire.beginTransmission(compass_address);                  // Start communication with the
compass.
  Wire.write(0x03);                                        // We want to start reading at the hexadecimal
location 0x03.
  Wire.endTransmission();                                   // End the transmission with the gyro.

  Wire.requestFrom(compass_address, 6);                     // Request 6 bytes from the
compass.
  compass_y = Wire.read() << 8 | Wire.read();               // Add the low and high byte to the
compass_y variable.
  compass_y *= -1;                                         // Invert the direction of the axis.
  compass_z = Wire.read() << 8 | Wire.read();               // Add the low and high byte to the
compass_z variable.;
  compass_x = Wire.read() << 8 | Wire.read();               // Add the low and high byte to the
compass_x variable.;
```

```
  compass_x *= -1;                                        // Invert the direction of the axis.

  // Before the compass can give accurate measurements it needs to be calibrated. At startup the compass_offset and compass_scale
  // variables are calculated. The following part will adjust the raw compas values so they can be used for the calculation of the heading.

  if (compass_calibration_on == 0) {                      // When the compass is not beeing calibrated.
    compass_y += compass_offset_y;                        // Add the y-offset to the raw value.
    compass_y *= compass_scale_y;                         // Scale the y-value so it matches the other axis.
    compass_z += compass_offset_z;                        // Add the z-offset to the raw value.
    compass_z *= compass_scale_z;                         // Scale the z-value so it matches the other axis.
    compass_x += compass_offset_x;                        // Add the x-offset to the raw value.
  }

  // The compass values change when the roll and pitch angle of the quadcopter changes. That's the reason that the x and y values need to calculated for a virtual horizontal position.
  // The 0.0174533 value is phi/180 as the functions are in radians in stead of degrees.

  compass_x_horizontal = (float)compass_x * cos(angle_pitch * -0.0174533) + (float)compass_y * sin(angle_roll * 0.0174533) * sin(angle_pitch * -0.0174533) - (float)compass_z * cos(angle_roll * 0.0174533) * sin(angle_pitch * -0.0174533);
  compass_y_horizontal = (float)compass_y * cos(angle_roll * 0.0174533) + (float)compass_z * sin(angle_roll * 0.0174533);

  // Now that the horizontal values are known the heading can be calculated. With the following lines of code the heading is calculated in degrees.
  // Please note that the atan2 uses radians in stead of degrees. That is why the 180/3.14 is used.
  if (compass_y_horizontal < 0)actual_compass_heading = 180 + (180 + ((atan2(compass_y_horizontal, compass_x_horizontal)) * (180 / 3.14)));
  else actual_compass_heading = (atan2(compass_y_horizontal, compass_x_horizontal)) * (180 / 3.14);

  actual_compass_heading += declination;                  // Add the declination to the magnetic compass heading to get the geographic north.
  if (actual_compass_heading < 0) actual_compass_heading += 360;    // If the compass heading becomes smaller then 0, 360 is added to keep it in the 0 till 360 degrees range.
  else if (actual_compass_heading >= 360) actual_compass_heading -= 360; // If the compass heading becomes larger then 360, 360 is subtracted to keep it in the 0 till 360 degrees range.
}

// Reads GPS data:
void read_gps(void) {

  while (ss.available() && new_line_found == 0) {         // Stay in this loop as long as there is serial information from the GPS available.
    char read_serial_byte = ss.read();                    // Load a new serial byte in the read_serial_byte variable.
    if (read_serial_byte == '$') {                        // If the new byte equals a $ character.
      for (message_counter = 0; message_counter <= 99; message_counter ++) {  // Clear the old data from the incomming buffer array.
        incomming_message[message_counter] = '-';         // Write a - at every position.
```

```
    }
    message_counter = 0;                                                    // Reset the
message_counter variable because we want to start writing at the begin of the array.
  }
  else if (message_counter <= 99)message_counter ++;                                    // If
the received byte does not equal a $ character, increase the message_counter variable.
  incomming_message[message_counter] = read_serial_byte;                                //
Write the new received byte to the new position in the incomming_message array.
  if (read_serial_byte == '*') new_line_found = 1;                                      // Every
NMEA line end with a *. If this character is detected the new_line_found variable is set to 1.
 }

 // If the software has detected a new NMEA line it will check if it's a valid line that can be
used.
 if (new_line_found == 1) {                                                 // If a new NMEA
line is found.
  new_line_found = 0;                                                   // Reset the
new_line_found variable for the next line.
  if (incomming_message[4] == 'L' && incomming_message[5] == 'L' &&
incomming_message[7] == ',') {    // When there is no GPS fix or latitude/longitude
information available.
    // Set some variables to 0 if no valid information is found by the GPS module. This is
needed for GPS lost when flying.
    l_lat_gps = 0;
    l_lon_gps = 0;
    lat_gps_previous = 0;
    lon_gps_previous = 0;
    number_used_sats = 0;
  }

  // If the line starts with GA and if there is a GPS fix we can scan the line for the latitude,
longitude and number of satellites.
  if (incomming_message[4] == 'G' && incomming_message[5] == 'A' &&
(incomming_message[44] == '1' || incomming_message[44] == '2')) {
    lat_gps_actual = ((int)incomming_message[19] - 48) *  (long)10000000;
// Filter the minutes for the GGA line multiplied by 10.
    lat_gps_actual += ((int)incomming_message[20] - 48) * (long)1000000;
// Filter the minutes for the GGA line multiplied by 10.
    lat_gps_actual += ((int)incomming_message[22] - 48) * (long)100000;
// Filter the minutes for the GGA line multiplied by 10.
    lat_gps_actual += ((int)incomming_message[23] - 48) * (long)10000;
// Filter the minutes for the GGA line multiplied by 10.
    lat_gps_actual += ((int)incomming_message[24] - 48) * (long)1000;                   //
Filter the minutes for the GGA line multiplied by 10.
    lat_gps_actual += ((int)incomming_message[25] - 48) * (long)100;                    //
Filter the minutes for the GGA line multiplied by 10.
    lat_gps_actual += ((int)incomming_message[26] - 48) * (long)10;                     //
Filter the minutes for the GGA line multiplied by 10.
    lat_gps_actual /= (long)6;                                              // To convert the
minutes to degrees we need to divide the minutes by 6.
    lat_gps_actual += ((int)incomming_message[17] - 48) *  (long)100000000;
// Add the degrees multiplied by 10.
    lat_gps_actual += ((int)incomming_message[18] - 48) *  (long)10000000;
// Add the degrees multiplied by 10.
    lat_gps_actual /= 10;                                                   // Divide everything
by 10.

    lon_gps_actual = ((int)incomming_message[33] - 48) *  (long)10000000;
// Filter the minutes for the GGA line multiplied by 10.
```

```
    lon_gps_actual += ((int)incomming_message[34] - 48) * (long)1000000;
// Filter the minutes for the GGA line multiplied by 10.
    lon_gps_actual += ((int)incomming_message[36] - 48) * (long)100000;
// Filter the minutes for the GGA line multiplied by 10.
    lon_gps_actual += ((int)incomming_message[37] - 48) * (long)10000;
// Filter the minutes for the GGA line multiplied by 10.
    lon_gps_actual += ((int)incomming_message[38] - 48) * (long)1000;
// Filter the minutes for the GGA line multiplied by 10.
    lon_gps_actual += ((int)incomming_message[39] - 48) * (long)100;                    //
Filter the minutes for the GGA line multiplied by 10.
    lon_gps_actual += ((int)incomming_message[40] - 48) * (long)10;                     //
Filter the minutes for the GGA line multiplied by 10.
    lon_gps_actual /= (long)6;                                          // To convert the
minutes to degrees we need to divide the minutes by 6.
    lon_gps_actual += ((int)incomming_message[30] - 48) * (long)1000000000;
// Add the degrees multiplied by 10.
    lon_gps_actual += ((int)incomming_message[31] - 48) * (long)100000000;
// Add the degrees multiplied by 10.
    lon_gps_actual += ((int)incomming_message[32] - 48) * (long)10000000;
// Add the degrees multiplied by 10.
    lon_gps_actual /= 10;                                               // Divide everything
by 10.

    if (incomming_message[28] == 'N')latitude_north = 1;                                //
When flying north of the equator the latitude_north variable will be set to 1.
    else latitude_north = 0;                                            // When flying south
of the equator the latitude_north variable will be set to 0.

    if (incomming_message[42] == 'E')longiude_east = 1;                                 //
When flying east of the prime meridian the longiude_east variable will be set to 1.
    else longiude_east = 0;                                             // When flying west
of the prime meridian the longiude_east variable will be set to 0.

    number_used_sats = ((int)incomming_message[46] - 48) * (long)10;
// Filter the number of satellites from the GGA line.
    number_used_sats += (int)incomming_message[47] - 48;                                //
Filter the number of satellites from the GGA line.

    if (lat_gps_previous == 0 && lon_gps_previous == 0) {                               // If
this is the first time the GPS code is used.
     lat_gps_previous = lat_gps_actual;                                 // Set the
lat_gps_previous variable to the lat_gps_actual variable.
     lon_gps_previous = lon_gps_actual;                                 // Set the
lon_gps_previous variable to the lon_gps_actual variable.
    }

    lat_gps_loop_add = (float)(lat_gps_actual - lat_gps_previous) / 10.0;               //
Divide the difference between the new and previous latitude by ten.
    lon_gps_loop_add = (float)(lon_gps_actual - lon_gps_previous) / 10.0;
// Divide the difference between the new and previous longitude by ten.

    l_lat_gps = lat_gps_previous;                                       // Set the
l_lat_gps variable to the previous latitude value.
    l_lon_gps = lon_gps_previous;                                       // Set the
l_lon_gps variable to the previous longitude value.

    lat_gps_previous = lat_gps_actual;                                  // Remember
the new latitude value in the lat_gps_previous variable for the next loop.
    lon_gps_previous = lon_gps_actual;                                  // Remember
```

```
the new longitude value in the lat_gps_previous variable for the next loop.

      // The GPS is set to a 5 Hz refresh rate. Between every 2 GPS measurements, 9 GPS
values are simulated.
      gps_add_counter = 5;                                                    // Set the
gps_add_counter variable to 5 as a count down loop timer
      new_gps_data_counter = 9;                                               // Set the
new_gps_data_counter to 9. This is the number of simulated values between 2 GPS
measurements.
      lat_gps_add = 0;                                                        // Reset the
lat_gps_add variable.
      lon_gps_add = 0;                                                        // Reset the
lon_gps_add variable.
      new_gps_data_available = 1;                                             // Set the
new_gps_data_available to indicate that there is new data available.
    }

    //If the line starts with SA and if there is a GPS fix we can scan the line for the fix type
(none, 2D or 3D).
    if (incomming_message[4] == 'S' && incomming_message[5] == 'A')fix_type =
(int)incomming_message[9] - 48;

  }

  // After 5 program loops 5 x 4ms = 20ms the gps_add_counter is 0.
  if (gps_add_counter == 0 && new_gps_data_counter > 0) {                      //
If gps_add_counter is 0 and there are new GPS simulations needed.
    new_gps_data_available = 1;                                               // Set the
new_gps_data_available to indicate that there is new data available.
    new_gps_data_counter --;                                                  // Decrement
the new_gps_data_counter so there will only be 9 simulations
    gps_add_counter = 5;                                                      // Set the
gps_add_counter variable to 5 as a count down loop timer

    lat_gps_add += lat_gps_loop_add;                                          // Add the
simulated part to a buffer float variable because the l_lat_gps can only hold integers.
    if (abs(lat_gps_add) >= 1) {                                              // If the absolute
value of lat_gps_add is larger then 1.
      l_lat_gps += (int)lat_gps_add;                                          // Increment the
lat_gps_add value with the lat_gps_add value as an integer. So no decimal part.
      lat_gps_add -= (int)lat_gps_add;                                        // Subtract the
lat_gps_add value as an integer so the decimal value remains.
    }

    lon_gps_add += lon_gps_loop_add;                                          // Add the
simulated part to a buffer float variable because the l_lat_gps can only hold integers.
    if (abs(lon_gps_add) >= 1) {                                              // If the absolute
value of lat_gps_add is larger then 1.
      l_lon_gps += (int)lon_gps_add;                                          // Increment
the lat_gps_add value with the lat_gps_add value as an integer. So no decimal part.
      lon_gps_add -= (int)lon_gps_add;                                        // Subtract
the lat_gps_add value as an integer so the decimal value remains.
    }
  }

  if (new_gps_data_available) {                                               // If there is a
new set of GPS data available.
    gps_watchdog_timer = millis();                                            // Reset the
GPS watchdog timer.
```

```
    new_gps_data_available = 0;                                                 // Reset the
new_gps_data_available variable.

    if (fm == FM_gps_hold && waypoint_set == 0) {                                   // If the
flight mode is 3 (GPS hold) and no waypoints are set.
      waypoint_set = 1;                                                     // Indicate that the
waypoints are set.
      l_lat_waypoint = l_lat_gps;                                           // Remember the
current latitude as GPS hold waypoint.
      l_lon_waypoint = l_lon_gps;                                           // Remember
the current longitude as GPS hold waypoint.
    }

    if (fm ==FM_gps_hold && waypoint_set == 1) {                                    // If the
GPS hold mode and the waypoints are stored and take off detected.
      // GPS stick move adjustments
      if (fm == FM_gps_hold && remote_channel[3] >= 1400) {                             //
¡¡¡¡¡ 1400us is considered when the drone isn't touching the ground, consider changing !!!!
        if (!latitude_north) {
          l_lat_gps_float_adjust += 0.0015 * (((remote_channel[2] - 1500) *
cos(gps_man_adjust_heading * 0.017453)) + ((remote_channel[1] - 1500) *
cos((gps_man_adjust_heading - 90) * 0.017453))); //South correction
        }
        else {
          l_lat_gps_float_adjust -= 0.0015 * (((remote_channel[2] - 1500) *
cos(gps_man_adjust_heading * 0.017453)) + ((remote_channel[1] - 1500) *
cos((gps_man_adjust_heading - 90) * 0.017453))); //North correction
        }

        if (!longiude_east) {
          l_lon_gps_float_adjust -= (0.0015 * (((remote_channel[1] - 1500) *
cos(gps_man_adjust_heading * 0.017453)) + ((remote_channel[2] - 1500) *
cos((gps_man_adjust_heading + 90) * 0.017453)))) / cos(((float)l_lat_gps / 1000000.0) *
0.017453); //West correction
        }

        else {
          l_lon_gps_float_adjust += (0.0015 * (((remote_channel[1] - 1500) *
cos(gps_man_adjust_heading * 0.017453)) + ((remote_channel[2] - 1500) *
cos((gps_man_adjust_heading + 90) * 0.017453)))) / cos(((float)l_lat_gps / 1000000.0) *
0.017453); //East correction
        }
      }

      if (l_lat_gps_float_adjust > 1) {
        l_lat_waypoint ++;
        l_lat_gps_float_adjust --;
      }
      if (l_lat_gps_float_adjust < -1) {
        l_lat_waypoint --;
        l_lat_gps_float_adjust ++;
      }

      if (l_lon_gps_float_adjust > 1) {
        l_lon_waypoint ++;
        l_lon_gps_float_adjust --;
      }
      if (l_lon_gps_float_adjust < -1) {
        l_lon_waypoint --;
```

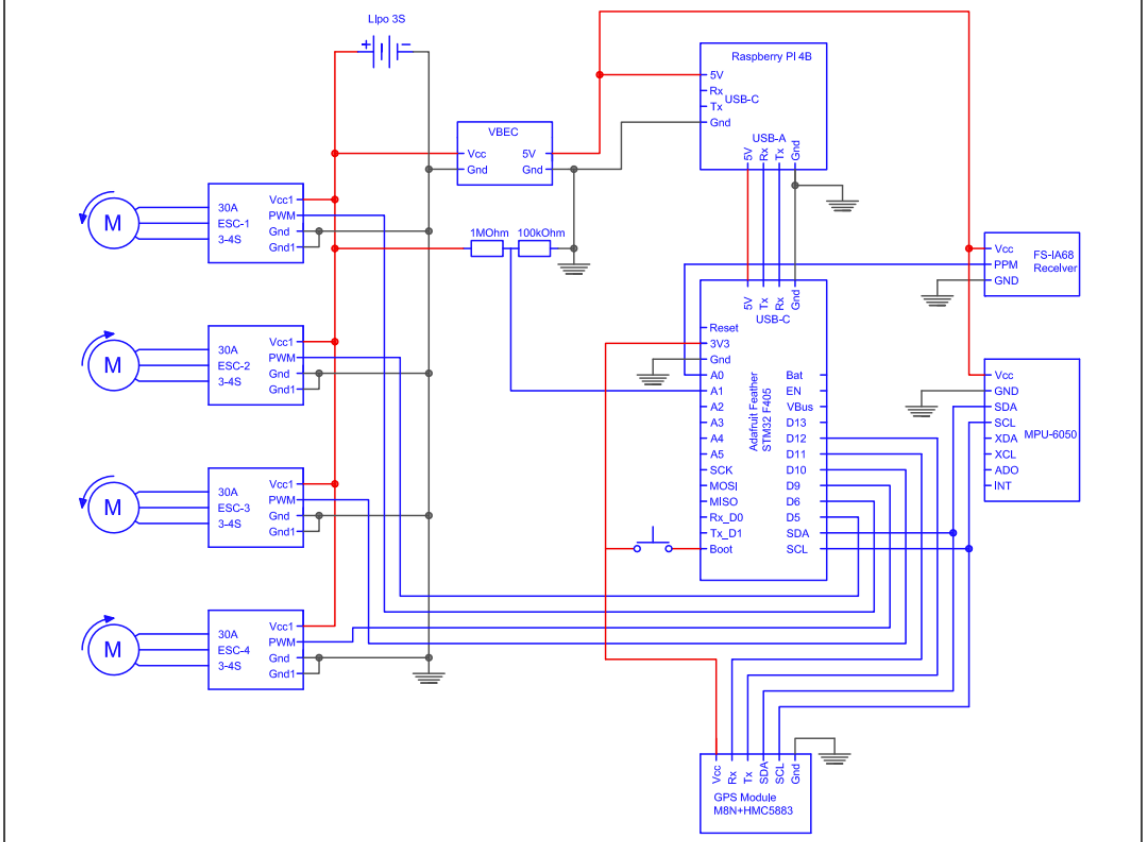
```
      l_lon_gps_float_adjust ++;
    }

    gps_lon_error = l_lon_waypoint - l_lon_gps;                                    //
Calculate the latitude error between waypoint and actual position.
    gps_lat_error = l_lat_gps - l_lat_waypoint;                                    // Calculate
the longitude error between waypoint and actual position.

    gps_lat_total_avarage -=  gps_lat_rotating_mem[ gps_rotating_mem_location];
// Subtract the current memory position to make room for the new value.
    gps_lat_rotating_mem[ gps_rotating_mem_location] = gps_lat_error -
gps_lat_error_previous;         // Calculate the new change between the actual pressure and
the previous measurement.
    gps_lat_total_avarage +=  gps_lat_rotating_mem[ gps_rotating_mem_location];
// Add the new value to the long term average value.

    gps_lon_total_avarage -=  gps_lon_rotating_mem[ gps_rotating_mem_location];
// Subtract the current memory position to make room for the new value.
    gps_lon_rotating_mem[ gps_rotating_mem_location] = gps_lon_error -
gps_lon_error_previous;         // Calculate the new change between the actual pressure and
the previous measurement.
    gps_lon_total_avarage +=  gps_lon_rotating_mem[ gps_rotating_mem_location];
// Add the new value to the long term average value.
    gps_rotating_mem_location++;                                                   // Increase
the rotating memory location.
    if ( gps_rotating_mem_location == 35) gps_rotating_mem_location = 0;
// Start at 0 when the memory location 35 is reached.

    gps_lat_error_previous = gps_lat_error;                                        //
Remember the error for the next loop.
    gps_lon_error_previous = gps_lon_error;                                        //
Remember the error for the next loop.

    // Calculate the GPS pitch and roll correction as if the nose of the multicopter is facing
north.
    // The Proportional part = (float)gps_lat_error * gps_p_gain.
    // The Derivative part = (float)gps_lat_total_avarage * gps_d_gain.
    gps_pitch_adjust_north = (float)gps_lat_error * gps_p_gain + (float)gps_lat_total_avarage
* gps_d_gain;
    gps_roll_adjust_north = (float)gps_lon_error * gps_p_gain + (float)gps_lon_total_avarage
* gps_d_gain;

    if (!latitude_north)gps_pitch_adjust_north *= -1;                              // Invert
the pitch adjustment because the quadcopter is flying south of the equator.
    if (!longiude_east)gps_roll_adjust_north *= -1;                                // Invert
the roll adjustment because the quadcopter is flying west of the prime meridian.

    // Because the correction is calculated as if the nose was facing north, we need to convert
it for the current heading.
    gps_roll_adjust = ((float)gps_roll_adjust_north * cos(angle_yaw * 0.017453)) +
((float)gps_pitch_adjust_north * cos((angle_yaw - 90) * 0.017453));
    gps_pitch_adjust = ((float)gps_pitch_adjust_north * cos(angle_yaw * 0.017453)) +
((float)gps_roll_adjust_north * cos((angle_yaw + 90) * 0.017453));

    // Limit the maximum correction to 300. This way we still have full control with the pitch
and roll stick on the transmitter.
    if (gps_roll_adjust > 300) gps_roll_adjust = 300;
    if (gps_roll_adjust < -300) gps_roll_adjust = -300;
    if (gps_pitch_adjust > 300) gps_pitch_adjust = 300;
```

```
      if (gps_pitch_adjust < -300) gps_pitch_adjust = -300;
    }
  }

  if (gps_watchdog_timer + 1000 < millis()) {                                  // If the
watchdog timer is exceeded the GPS signal is missing.
    if (fm == FM_gps_hold) {                                                   // If flight mode is
set to (GPS hold).
      fm = FM_stable;                                                          // Set the flight
mode to 2.
      //Serial.println("Error GPS");
    }
  }

  if (fm < 4 && waypoint_set > 0) {                                            // If the GPS
hold mode is disabled and the waypoints are set.
    gps_roll_adjust = 0;                                                       // Reset the
gps_roll_adjust variable to disable the correction.
    gps_pitch_adjust = 0;                                                      // Reset the
gps_pitch_adjust variable to disable the correction.
    if (waypoint_set == 1) {                                                   // If the waypoints
are stored
      gps_rotating_mem_location = 0;                                           // Set the
gps_rotating_mem_location to zero so we can empty the
      waypoint_set = 2;                                                        // Set the
waypoint_set variable to 2 as an indication that the buffer is not cleared.
    }
    gps_lon_rotating_mem[ gps_rotating_mem_location] = 0;                      //
Reset the current gps_lon_rotating_mem location.
    gps_lat_rotating_mem[ gps_rotating_mem_location] = 0;                      //
Reset the current gps_lon_rotating_mem location.
    gps_rotating_mem_location++;                                               // Increment
the gps_rotating_mem_location variable for the next loop.
    if (gps_rotating_mem_location == 36) {                                     // If the
gps_rotating_mem_location equals 36, all the buffer locations are cleared.
      waypoint_set = 0;                                                        // Reset the
waypoint_set variable to 0.
      //Reset the variables that are used for the D-controller.
      gps_lat_error_previous = 0;
      gps_lon_error_previous = 0;
      gps_lat_total_avarage = 0;
      gps_lon_total_avarage = 0;
      gps_rotating_mem_location = 0;
      //Reset the waypoints.
      l_lat_waypoint = 0;
      l_lon_waypoint = 0;
    }
  }
}
```

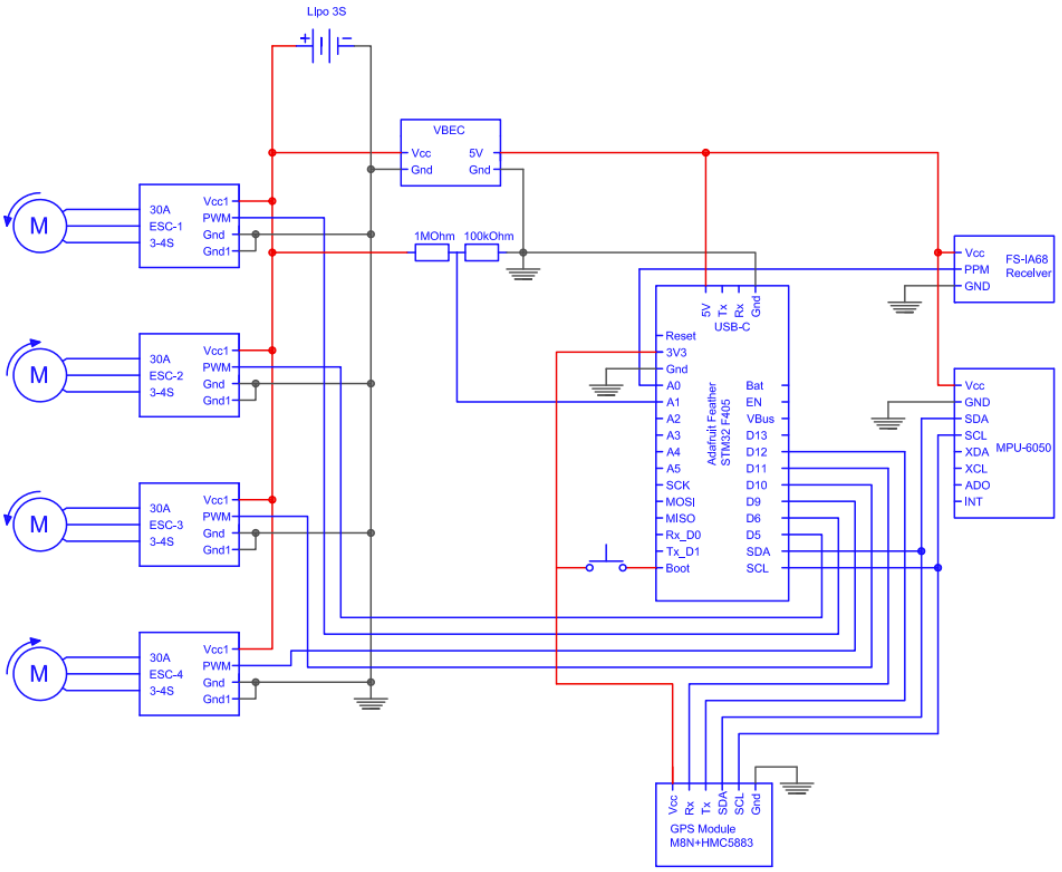
- Computes the reference drone mode:

```
void reference_computation(){
        ref_mode_management();
        ref_gen();
}

void ref_mode_management(){

  if (remote_channel[3] < 1100 && remote_channel[4] < 1100) fm = FM_mounting;
// Enables mounting mode.
  if (fm == FM_mounting && remote_channel[3] < 1100 && remote_channel[4] > 1450) fm =
FM_stable;          // Enables stable mode after mounting mode has been reached.
  if (fm >=2 && remote_channel[3] < 1050 && remote_channel[4] > 1950) fm = FM_disabled;
// Enables disabled mode.
  if (fm >=2 && remote_channel[6] < 1100) fm = FM_stable;                                    //
Enables stable mode.
  if (fm >=2 && remote_channel[6] >= 1100 && remote_channel[6] <= 1900) fm =
FM_stable_battery;          // Enables stable battery mode.
  if (fm >=2 && remote_channel[6] > 1900) fm = FM_gps_hold;                                  //
Enables gps mode.
}

void ref_gen(){

        if(fm == FM_mounting){                    // Starts the drone parameters.
                throttle = 950;
                angle_pitch = angle_pitch_acc;
           angle_roll = angle_roll_acc;
    pid_i_mem_roll = 0;
    pid_last_roll_d_error = 0;
    pid_output_roll = 0;
    pid_i_mem_pitch = 0;
    pid_last_pitch_d_error = 0;
    pid_output_pitch = 0;
    pid_i_mem_yaw = 0;
    pid_last_yaw_d_error = 0;
    pid_output_yaw = 0;
        }

        if(fm >=2){                               // Considers the throttle with the channel 3.
                throttle = remote_channel[3];
        }
}
```

ANNEX 11: DRONE CIRCUITS



CLIENT:	UPC EETAC	Planell:	2024_01
CONJUNT:	TFG	Data:	29-07-24
DENOMINACIÓ:	Circuit scheme	Dibuixat:	Alejandro Boadella



CLIENT:	UPC EETAC	Planell:	2024_02
CONJUNT:	TFG	Data:	29-07-24
DENOMINACIÓ:	Circuit scheme	Dibuixat:	Alejandro Boadella