Costa Rica Institute of Technology

Electronic Engineering Department

Improvement of flight performance for an UAV system

Graduation project report to qualify for the title of Electronics Engineer with a Licenciatura degree

José Ángel Pacheco Chaverri

Prague, Spring Semester 2016

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado Improvement of flight performance for an UAV system, realizado por el señor José Ángel Pacheco Chaverri y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

Ing. Faustino Montes de Oca Murillo
Profesor lector

Ing. Sergio Morales Hernández
Profesor lector

TEC | Tecnológico de Costa Rica
Ingeniería Electrónica

Ing. Gabriela Ortiz León
Profesor asesor

Cartago, 29 de agosto, 2016

Declaration of Authenticity

I declare that this Graduation Project has been made, entirely, by me, using and applying literature on the subject and introducing my own knowledge.

In cases where I have used bibliographic material, I have proceeded to indicate the sources by quoting them.

Consequently, I take full responsibility for the graduation work done and the contents of the final report.

Prague, Czech Republic

José Ángel Pacheco Chaverri

ID: 1-1483-0811

Abstract

In the recent years UAV´s have become very popular. Due to the acquire fame of these devices, a wide number of applications have been developed in order to satisfy a demanding market that seeks new ways to improve the flight capability. Because of this the Department of measurement- Aeronautical Instrumentation of Czech Technical University in Prague (CVUT), has decided to control a Drone using Open-Source software in order to enhance its performance by implementing a control algorithm that upgrades the stability in the vertical position.

The system design allows the software to control the Drone with your PC whether it is remotely (Joystick) or by interpreting instructions from generated code, also it is flexible due to the fact that enables the possibility to modify its vertical control loop algorithm

In order to accomplish the system mentioned above, the technology incorporated inside the UAV was used. For example, the ultrasonic sensor and the bottom camera were used to increase the stability of the vertical position.

A GPS module was attached, so the system could use it as a reference to flight through selected waypoints programed by the software.

Keywords: UAV, software, control algorithms, waypoints, ultrasonic sensor, bottom camera.

Resumen

En los últimos años los sistemas UAV se han popularizado. Debido a la fama que han adquirido estos dispositivos, un gran número de aplicaciones se han desarrollado para poder satisfacer un mercado demandante que busca nuevas formas de mejorar la capacidad de vuelo de estos dispositivos. Debido a esto el Departamento de medición-Instrumentación Aeronáutica del Instituto Técnico Checo en Praga (CVUT), ha decidido controlar una unidad Drone usando un software Open-Source para poder mejorar su rendimiento al implementar un algoritmo de control que permita mejorar la estabilidad en la posición vertical.

El sistema a diseñar debe de permitir controlar el Drone con el computador ya sea de vía remota (Palanca de mando) o interpretando instrucciones de código generado a través del software. Debe de ser también flexible permitiendo modificar sus archivos, para que de esta forma se pueda incorporar el nuevo algoritmo de posición vertical.

Para poder alcanzar el sistema propuesto, se utilizó la tecnología incorporada dentro del UAV. Por ejemplo para estabilizar la posición vertical se utilizó el sensor ultrasónico y la cámara inferior.

Un módulo GPS fue incorporado, de forma que se pudiera utilizar como referencia para permitir el sistema volar por puntos de navegación definidos programados en el software.

Palabras clave: UAV, software, algoritmo de control, puntos de navegación, sensor ultrasónico, cámara inferior.

Dedication

To my family and friends thanks for all the love and support…

Acknowledgments

First of all, I want to thank God for all the opportunities that has provided me and for giving me the strength to conclude my studies.

To my family and friends, that has always given me their unconditional support and has always been there when I need them most.

To my professors thanks for all the teachings and for always pushing us to take that extra mile. I will take many lessons not only academically, but also personally.

To my advisors Ing. Gabriela Ortiz, Ing. Martin Sipos and Ing. Mushfiqul Alam, thanks for all the support and advise throughout the project and for guiding me to the conclusion of it.

Thanks to all who participated in this stage of my live that now concludes, but prepares me for the next.

José Ángel Pacheco Chaverri

Prague, June 2016.

General Index

Index of Figures

Index of Tables

List of Abbreviations

UAV	Unmanned Aerial Vehicle
CVUT	Czech Technical University
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
GCS	Ground Control Station
GPS	Global Positioning System

1 Introduction

1.1. Context of the problem

The first system of this kind was created by the army for military the purposes. Over the years these systems have improve, because of the extended research and investigation, so the areas of applications have grown.

Currently these devices are being used for multiple utilities, such as: Precision farming where they can extract data with their sensors that would be used later for analyzing whether the land is fertile or not, for aerial photography and videos, for monitoring wind turbines, for inspecting pipelines, etc.

One of the great challenges facing the UAV industry is security, this due to the number of utilities that these devices are given that involve interaction with civilians. Another aspect to consider is that these systems consist of expensive electronic equipment, so their safety is paramount. Also it is important to consider that the laws that revolve around this industry are becoming more severe and are demanding high-quality devices. For this reasons there is a need to incorporate control algorithms to ensure optimal and safe operation.

Considering the aspects listed above the problem for this project can be defined as to use an Open-Source software that allows to implement a control algorithm in order to stabilize the vertical position.

1.2. Description of the system to innovate

Since the invention of UAV systems, there has always been a need to improve their performance. Therefore, this project seeks to implement a control algorithm that would enable to stabilize the vertical position. This was achieved by using a software that allows to implement Open-Source code and combined it with the hardware inside the device.

1.3. Stakeholder analysis

The main involved in this project is the *Department of Measurement - Aeronautical Instrumentation Laboratory* from the Czech Technical University in Prague (CVUT), currently in charge by Ing. Martin Sipos Ph.D.

This department shows interest in this project due to commercial growth that has experience this industry, so they are looking for new ways to enhanced an UAV performance in order to ensure a better quality device.

Obviously the users will be the last ones to use this sort of devices they can use it for all kind of purposes such as: Flight tests or adding new improvements.

1.4. Description of the problem

Drones or unmanned aerial vehicles (UAVs) were created by the army for reconnaissance operations. Like many military inventions, as the years went by civilian applications started to appear such as: search, rescue, monitoring, mapping various areas, flights hobby, etc.

The popularity of UAVs has increase due to the development of new applications in different areas, so innovative ways to control them must be found. This project aims to enhance the drone's performance in stabilizing the vertical position by taking advantage of the technology incorporated in it.

It is important to stand out that the data recorded in the experimental flights will be analyzed and evaluated to confirm proper operation.

1.5. Synthesis problem

Implement open-source software code that incorporates an autopilot system and ground station software for rotorcrafts, in order to flight through selected waypoints and stabilize the vertical position.

1.6. Solution approach

To solve this problem first a software must be picked up that allows to control the drone from the computer, then algorithms must be developed that will enable autonomous flights. It is important to notice that in this project autonomous flight is understand as the capability of the device to move in the air without any remotely tool, so all the tasks assign are done by interpreting code from the software.

After investigating and understanding the software, a GPS module will be attached. The necessary code must be developed so the drone can fly through selected waypoints.

Once the UAV can flight autonomously through different selected waypoints, a control algorithm was implemented to enhanced the stabilization of the vertical position. This was accomplished by seizing the electronics inside the drone such as: HD camera, ultrasound sensors, brushless in runner motors, among others.

The next step consists in developing several experimental flights, so that the performance of the drone can be tested. In figure 1.1 it can be seen a block diagram of the expected electronic system.

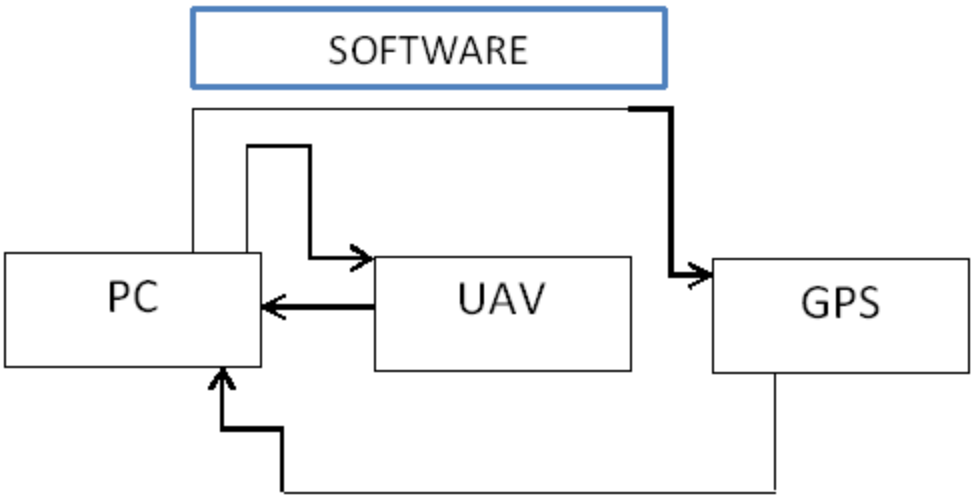

Figure 1.1 Electronic system diagram

2 Goal and Objectives

2.1. Goal

Improve the flight performance of the ARDrone 2 by combining the software with a GPS module and technology incorporate in the Drone.

2.2. General objective

Improve the estimation of altitude from the ARDrone 2 by implementing a control system capable of enhancing the stabilization of the vertical position.

2.3. Specific objectives

1. Investigate for software that allows creating algorithms for autonomous flights.

2. Attach a GPS model that enables the drone to flight through several selected waypoints.

3. Implement a control algorithm that allows the drone to estimate the correct altitude for the flight.

4. Develop experimental flights to test the system functionality.

3 Theoretical framework

In this section all necessary theoretical foundations that were involved in the design process will be detailed in order to reach the appropriate solution using the engineering designs criteria.

3.1. Description of the UAV system

Due to the complexity of this system it is mentioned only the relevant technical aspects to solve the problem posed above.

The UAV to use is the Parrot ARDrone 2, its structure is composed by: Carbon fiber tubes with a total weight of 380g with outdoor hull and 420g with indoor hull, foam to isolate engine vibrations and it is full reparable. Also it is designed to handle aerobatic maneuvers.

It has an onboard technology gives control and automatic stabilization features. The electronic characteristics worth mentioning are: 1GHz 32 ARM Cortex A8 processor, Linux 2.6.32, Wi-Fi, 3 axis gyroscope 2000 deg/s precision, 3 axis accelerometer ±50mg precision, 3 axis magnetometer 6° precision, ultrasound sensor and four brushless in runner motors 14.5 W 28500RPM.

Figure 3.1 ARDrone 2 dimensions and illustrations [16].

3.1.1. GPS module

The module to use is the Parrot Flight Recorder its technical characteristics can be seen in Table 3.1.

Table 3.1 Technical specifications for the Flight Recorder

Dimensions	77.7x38.3x12.5 mm
Weight	31g
Accuracy	±2m
Frequency	5Hz
Voltage	5V
Flash Memory	4GB

Figure 3.2. Flight Recorder [16].

3.2. Fundamental Concepts

In this section important concepts are explained in order to reach a suitable solution for the proposed problem above.

3.2.1. Accelerometer

An accelerometer works by measuring the inertial force generated when a suspended proof mass accelerates. It has three main components a proof mass, a suspension to hold the mass and a picoff for the output signal, see Figure 3.3.

Generally, accelerometers contain capacitive plates internally that are attached to miniscule springs, they move internally as acceleration forces act upon the sensor. As these plates move the capacitance changes and from these changes the acceleration can be determined.

The output is called specific force and is given by:

$$\boldsymbol{f = a - g} \tag{3. 1}$$

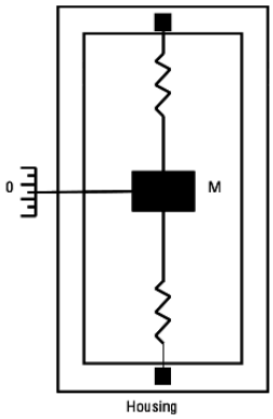

Figure 3.3. Basic accelerometer [1].

The Parrot company has really strict confidential rules, due to this no datasheet was acquire. Although in [6] there is a thorough explanation of the technical specifications for the ARDrone, where it indicates that a BMA150 was used with a precision of ±2g. Compare to the ARDrone 2 with ±50mg, it can be observed that this unit is more precise than the older.

3.2.2. Gyroscopes

Gyroscopes, or gyros, are devices that measures rotational motion. Almost all gyroscopes use vibrating mechanical elements, which are used on the principle of energy transfer between two vibration modes caused by the Coriolis force, see Figure 3.4.

Due to the effect of Coriolis force a body moving from A to B will have an acceleration given by:

$$\overrightarrow{a_c} = -2\vec{w}\, x\, \vec{v} \qquad (3.\ 2)$$

Through this Coriolis effect is how gyroscopes measure angular rates.

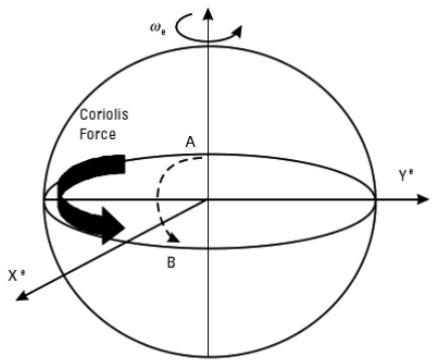

Figure 3.4. Coriolis force [1].

In the ARDrone for the gyroscope there is a 2 axis gyro IDG500 with rotation rates up to 500 deg/s and a precision gyro Epson XV3700 for the vertical axis. Comparing with ARDrone 2 that has a 3 axis gyro with 2000 deg/s it can be observed that it is more precise, but lacks accuracy in the vertical axis.

3.2.3. Magnetometer

Is a device that measures magnetic fields at a point in space. Usually inside this mechanism there are tiny coils and a beam, for each axis there is a beam. That beam has a pressure sensor on each side, and a coil wrapped around it passing current through it. Due to Lorentz force, the stronger the magnetic field, the more the bar with the current passing through it bends. The pressure sensors measure this bending; and do the correct calculation for the magnetic field.

There is no way to compare with the old unit, due to the fact that the ARDrone has no magnetometer, so these proves that the ARDrone 2 has more available data for a better performance.

3.2.4. Inertial Measurement Unit

The Inertial Measurement Unit (IMU) is an electronic device, that usually comprises an accelerometer, a gyroscope and sometimes a magnetometer; it outputs the given position, velocity and altitude.

The angular rate measure by the gyros is used to maintain its altitude and by integrating the acceleration produces the velocity solution and by integrating the velocity it gives the position solution.

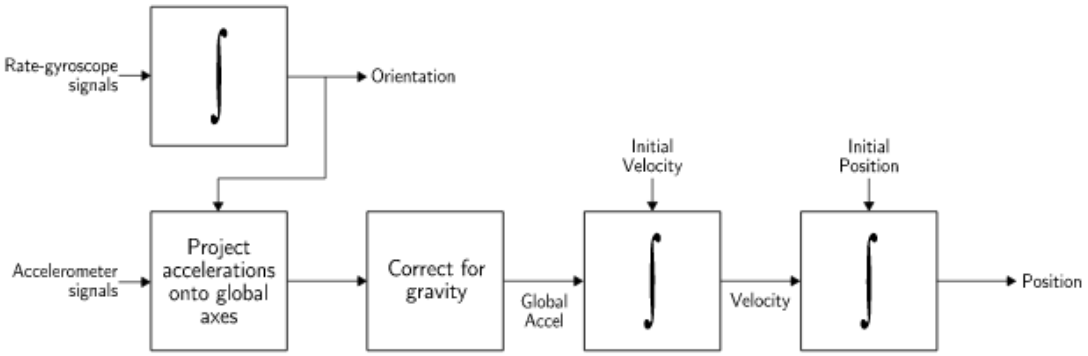

Figure 3.5. IMU algorithm [21].

3.2.5. Inertial Navigation System

Inertial Navigation System (INS) is very similar to the IMU. It uses the information given by the accelerometers and gyros in order to compute the current velocity and position. The main difference lies that it uses initial conditions in order to have a reference to realize its algorithms. These algorithms consist on comparing the data given from the sensors with the reference, so a proper navigation of the system can be done. They may change depending on the systems frame reference, see Figure 3.6.

Due to the latest advances in microelectromechanical systems the range of possible areas have increase in recent years. Typically, these systems are used in aircrafts, submarines and spacecraft.

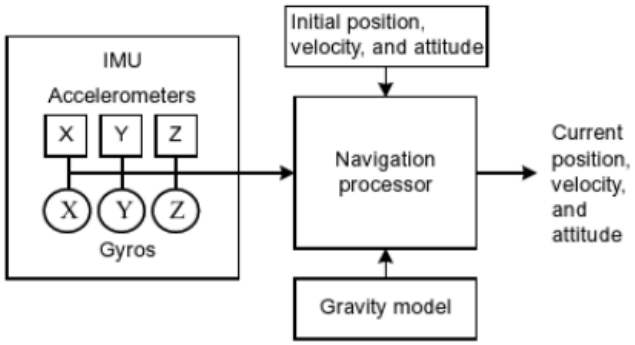

Figure 3.6. Schematic of an INS [1].

3.2.6. Global Positioning System

The Global Positioning System (GPS) is a system that consists on a network of 24 satellites. These satellites were placed in orbit by the army for military purposes, but now the system is available to anyone around the world.

The GPS satellites are circling the Earth gathering information, in order to determine how far the receiver is. In order to determine the receiver distance, it uses a time difference between the time the signal was transmitted and the time it was received. Once the user has this information it uses trilateration to calculate the exact position.

There are several global reference systems use by the GPS, the one used in this project is the WGS84 were the Earths center of mass is the origin for the reference system and the geodetic datum [9].

3.3. Optical Flow

Optical flow is a computer vision analysis method which computes the pixel displacements between two frames from a camera, this displacement is due to the horizontal and vertical movements of the Drone. Once known the pixel displacement, velocity can be calculated. Usually this method is used to derive the groundspeed of the Drone, but there are other applications such as obstacle avoidance and automatic landing where the optical flow estimation evaluates the time it takes to make contact with an obstacle [11].

The optical flow is modelled by equation (3.3) were: I_2 is the new image captured by the camera, I_1 is the previous image and I is the Intensity between I_2 and I_1.

$$\frac{\partial I_{2(x,y)}}{\partial x} u + \frac{\partial I_{2(x,y)}}{\partial y} v = -\frac{\partial I}{\partial t} \tag{3.3}$$

With u been the optic flow pixel in the X direction and v in the Y direction, this equation can be written in a more simplified way:

$$\nabla I_2 \begin{pmatrix} u \\ v \end{pmatrix} = -\frac{\partial I}{\partial t} \tag{3.4}$$

By analyzing equation 3.3 there are two unknown variables, for this reason a constraint must be added and this can change depending on the resolution method.

- Horn-Schunk method: Is a smooth constraint based on the assumption that one pixel doesn't change a lot between its neighbors. The computation is the result of finding the minimum in the next expression:

$$\iint \left(\nabla I \begin{pmatrix} u \\ v \end{pmatrix} + \frac{\partial I}{\partial t} \right) + \alpha^2 \left(\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 \right) dx dy \tag{3.5}$$

- Lucas-Kanade method: The optical flow is computed in a small window where the pixel of interest is surrounded, each pixel gives one equation so in order to get to a solution, the following equation must be minimized:

$$\sum_{\boldsymbol{p} \in \Omega} \left(\nabla \boldsymbol{I}(\boldsymbol{p}) \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} + \frac{\partial \boldsymbol{I}(\boldsymbol{p})}{\partial \boldsymbol{t}} \right) \tag{3.6}$$

These two are two main methods, further work has been developed throughout the years with methods such as: Block matching, features tracking method, pyramidal Lucas-Kanade and many others.

In order to calculate the speed from the optical flow, as mentioned above optical flow computes pixel displacements, this depends on six maneuvers on the rotation in the three axis (w_x, w_y, w_z), the horizontal translation (T_X, T_y) and the vertical translation (T_Z).

$$\boldsymbol{v_x} = \frac{\boldsymbol{T_z x - T_x f}}{\boldsymbol{Z}} - \boldsymbol{w_y f} - \boldsymbol{w_z x} + \frac{\boldsymbol{w_x y^2 - w_y xy}}{\boldsymbol{f}} \tag{3.7}$$

$$\boldsymbol{v_y} = \frac{\boldsymbol{T_z y - T_y f}}{\boldsymbol{Z}} + \boldsymbol{w_x f} + \boldsymbol{w_z y} + \frac{\boldsymbol{w_x xy - w_y x^2}}{\boldsymbol{f}} \tag{3.8}$$

Where f stands for focal length of the camera and Z for the distance between the camera and the filmed plane.

The algorithm used in this project is a combination of features tracking and Lucas-Kanade. First the feature tracking algorithm is in charge of extracting corners from the picture. This extraction is done by analyzing the 16 neighbor pixels, if at least 12 of these are brighter or darker than the pixel been analyzed, then it is classified as a corner. The Lucas-Kanade algorithm is in charge of detecting points, after detecting these points the

displacement of the pixels is calculated in order to determine the average optic flow. Once the optic flow is determined it is translated in order to obtain the ground speed.

4 Methodological Procedure

4.1. Recognition and definition of the problem

To define the problem a visit was made to CVUT, where the lab manager Ing. Martin Sipos Ph.D expose the need for a UAV system that would fly autonomously and also to improve the stabilization of the vertical position.

Given the information above, a software capable of autonomous flight and also with flexibility to modify the automatic control system files was found. Once the software was defined, objectives were elaborate in order to allow solutions to the different stages that are required.

4.2. Collection and analysis of information

Once known the needs of the system it took a literature search to complete knowledge and see that by combining the software with the GPS module, autonomous flights could be achieved. The stabilization of the vertical position was done by implementing Open-Source code and by taking advantage of technology incorporated inside the Drone.

The analysis of data was achieved by using the knowledge acquired during all the years of study. In addition, the advice of the advisors professors was crucial in order to encompass any doubt regarding the project.

4.3. Evaluation of alternatives

The system requirements are an ARDrone 2 that enables the possibility to flight through waypoints. This was achieved by attaching a GPS module and programing it with Open-Source code. The software used also is flexible by allowing to add a controller that stabilizes the vertical position.

For each specific objective an evaluation was made, in order to adjust them for the necessities of the system. The first objective was to selected a software, different were considered, but it was chosen the one that adjust to the system necessities. In the second objective different GPS were considered, the Parrot Flight Recorder was selected due to the capacity of storing flight data. For third objective different control algorithms designs were considered, finally it was selected the one that allowed a vertical stabilization using the resources in hand. In the last objective different test were done, this are discussed and evaluated in this project.

4.4. Solution implementation

For the development of the final system a division of work was done so that it was possible to attack the different problems according to the specific objectives. The first problem encounter was to connect the software with ARDrone 2, it was required a thorough investigation and understanding of the software. Once the software was connected to the Drone, the interface was programed in order to fulfill the system requirements. Then the GPS module was attached, it had to be configured manually due to the fact that is not the default GPS that the software uses. After attaching the GPS, code was generated in order to flight the Drone through waypoints, the results are evaluated in this project. Then a control algorithm was implemented using the ultrasonic sensor and the bottom camera incorporate in the Drone in order to stabilize the vertical position, results are discussed in this project.

4.5. Re-evaluation and design

In the design process different tests where elaborate in order to check the correct operation of the system.

The data recollect can be used to improve the system and to develop in the future new applications.

5 Design Process

In this chapter the solution that was implemented to solve the problems encountered in each specific objective would be addressed.

5.1. Selection of the software

The selection of the software was made taking into account several factors, the most relevant are that the software was Open-Source, also that it could allow autonomous flights. It needed to be flexible enough to modify the files, so that a new controller for stabilizing the vertical position could be added.

By investigating two software where encountered that fulfill the requirements state above for the ARDrone 2. The first one Paparazzi UAV, highly recommended for applications outside, besides this software has been used on previous projects in CVUT. The second one is QgroundControl used by several Parrot users for autonomous flights.

After an analysis of the two software selected, Paparazzi UAV was the one which adapted better to the necessities of the project. In Table 5.1 there is an evaluation of the characteristics evaluated in order to choose the program mentioned above.

Table 5.1. Comparison of the software selected

	Paparazzi UAV	**QGroundControl**
Autonomous flight	Yes	Yes
Allows to modify files	Yes	Yes
Open-Source	Yes	Yes
Feedback information	Yes	Not too much.

5.1.1. Overview of the software

Paparazzi UAV is a complete system of open-source hardware and software that contains autopilot systems for multicopters, fixedwing, hybrid aircraft, etc. Also it includes the complete ground station for planning and monitoring by using a bi-directional datalink for telemetry and control.

Due to its open characteristic it allows users to add new features and improve the system. Another feature worth mentioning is that this software was mainly created for autonomous flight as the main goal and manual flying as the second.

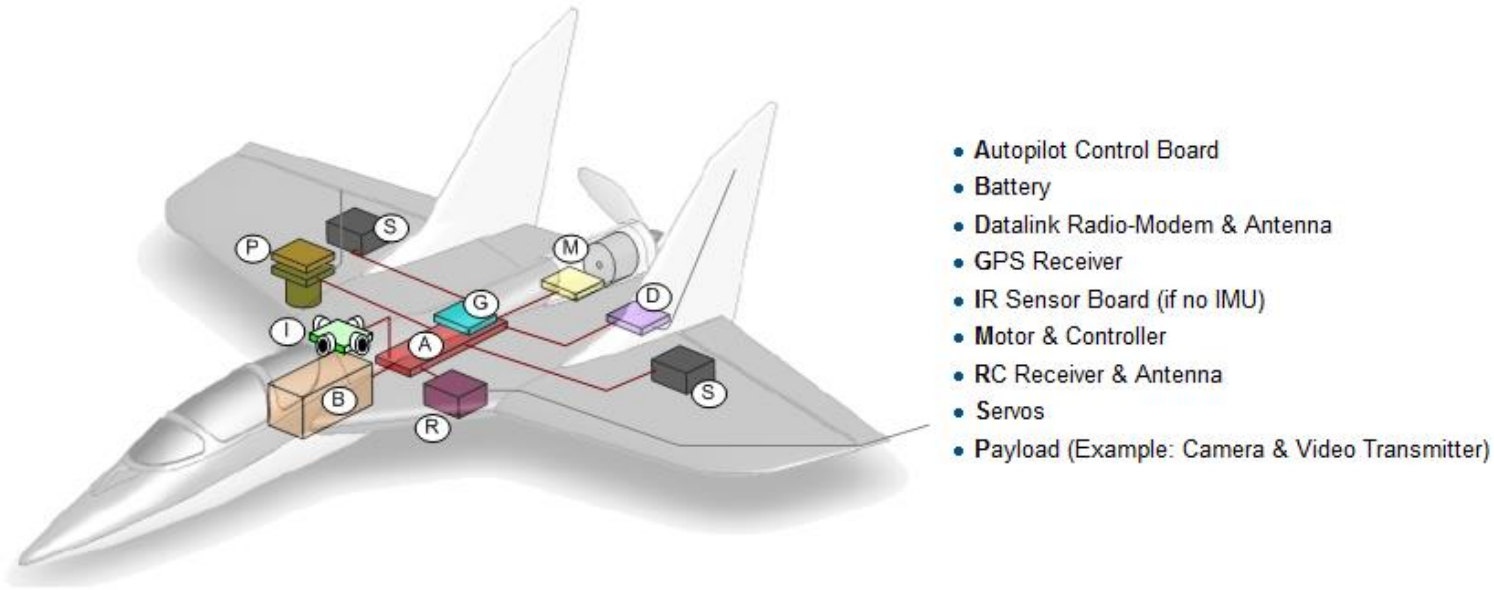

Figure 5.1. Key components of Paparazzi UAV [18].

The key components can be observed on the figure above, it's important to notice that this is an example of fixedwing in this project a rotorcraft was used, but the components also applied for this kind of aircraft.

5.1.2. How the software works

The code runs on the main autopilot control board, this code is written in C and it features a level of hardware abstraction. Paparazzi UAV uses XML files in order to allow easy configuration, this process can be seen in Figure 5.2.

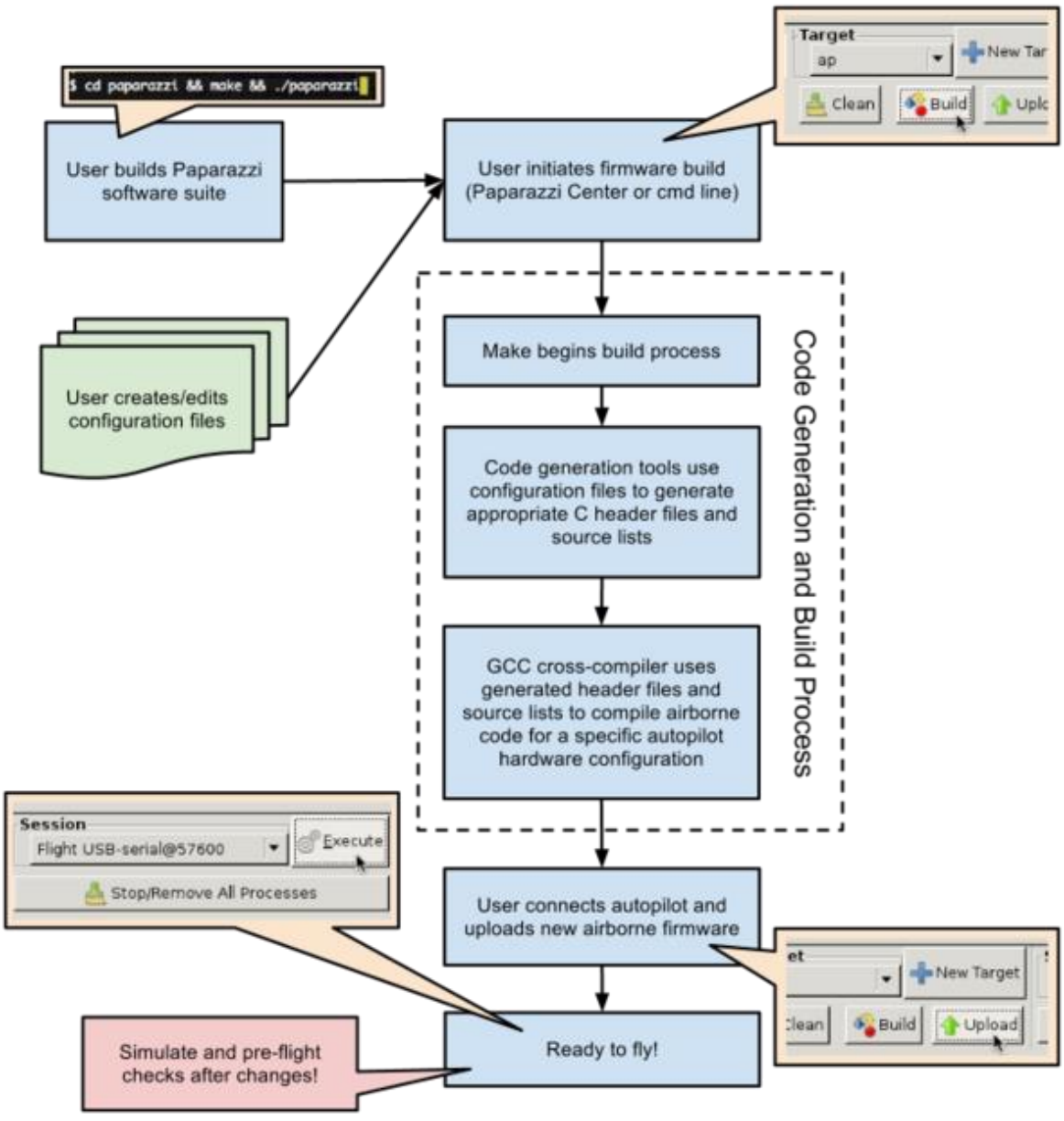

Figure 5.2. Paparazzi UAV software built process [18].

This onboard code divides in two: periodic tasks and event tasks. The periodic tasks are the time sensitive tasks, for example: control loops and telemetry messages. The event tasks are a response to something for example new GPS data.

5.1.3. Understanding and configuring the software

5.1.3.1. Paparazzi Center

Paparazzi Center is the initial window of the software; it is used to configure an aircraft as the union of the XML files explain in Table 5.2 Also it allows building the code and running session whether it is to flight, a simulation or to replay a flight, see Figure 5.4.

As mention above the aircraft is the union of the XML files, these files were configured in order to be able to compile correctly the code inside the autopilot of the ARDrone 2.

Table 5.2. Aircraft configuration files

XML File	**Task**
Airframe	Defines the autopilot parameters: Which modules will be used, configure control loops and hardware configurations.
Flight Plan	Autonomous flight behavior.
Radio	Radio transmitter under manual control.
Telemetry	Describes what messages are sent and at what rate.
Settings	Describes the available and the valid settings for the autopilot parameters.

In Figure 5.3 it is shown in more detail how this configuration works. It is important to highlight that in this diagram there are only mention the general configurations. In the

airframe file the sensors and the hardware that were configure where the IMU, ultrasound sensor, the bottom camera and the pressure sensor. Also control loops were configured first for the navigation with the GPS reference, then with new controller in order to stabilize the vertical position. In the flight plan code was generated in order to make the Drone flight through waypoints. In the telemetry file all data that was needed in order to analyze the behavior of the Drone was configured, such as: The roll angle, the pitch angle, the yaw angle, velocity, altitude, pressure, acceleration and others. In the settings file modifications were made in order to use several interactive buttons and to display data in the Ground Control Station interface (GCS), this is explained with more detail in the next section. In the radio file the communication used between the Drone and software is bidirectional through Wi-Fi.

The code of the autopilot is built from the configuration files, during the building all the details are reported in the console, also all the compiling errors. Then the code is uploaded through Wi-Fi to the Drone.

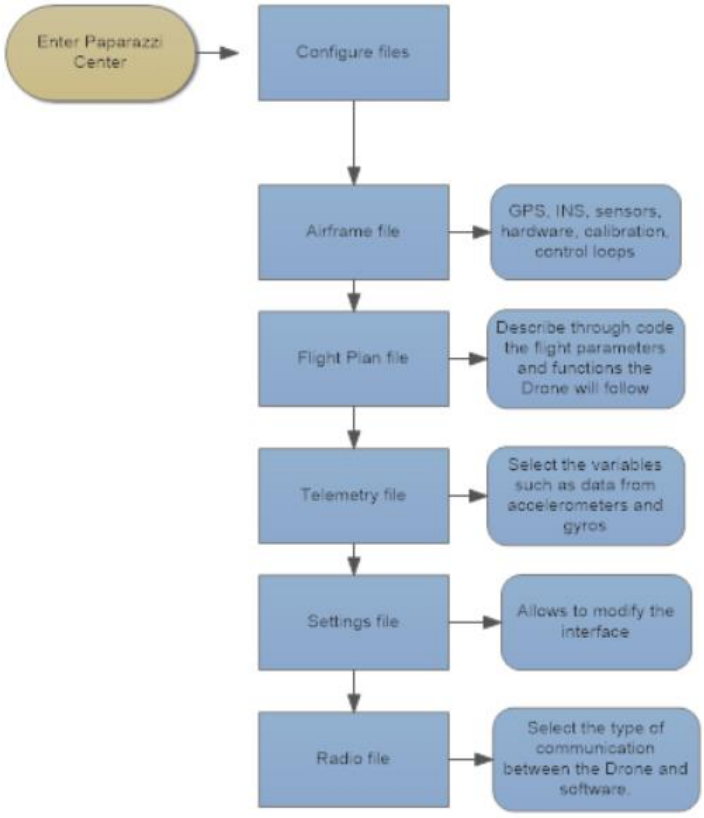

Figure 5.3. Configuration files diagram

Execution of the built code is done with a set of communication agents; these running agents can be stopped and restarted at any minute. These agents can be accessed through the Paparazzi Center by using the Tools menu. In this document only the Tools that were used will be specify:

- Messages: It allows seeing the numerical data of the respective telemetry.
- The Real Time Plotter: It allows to graphically plotting data from the telemetry messages.
- Data Link: Enables connection to modems, in it you can specify type of device that will be used and the baud rate.
- Log File Player: Allows recreating previous flights.
- Joystick: Allows to use joystick to send commands to the aircraft.

These tools helped to analyzed the behavior of the aircraft in real time, for example by combining the Messages with Real Time Plotter data was displayed in a graphical way. Also with Joystick the Drone was controlled manually.

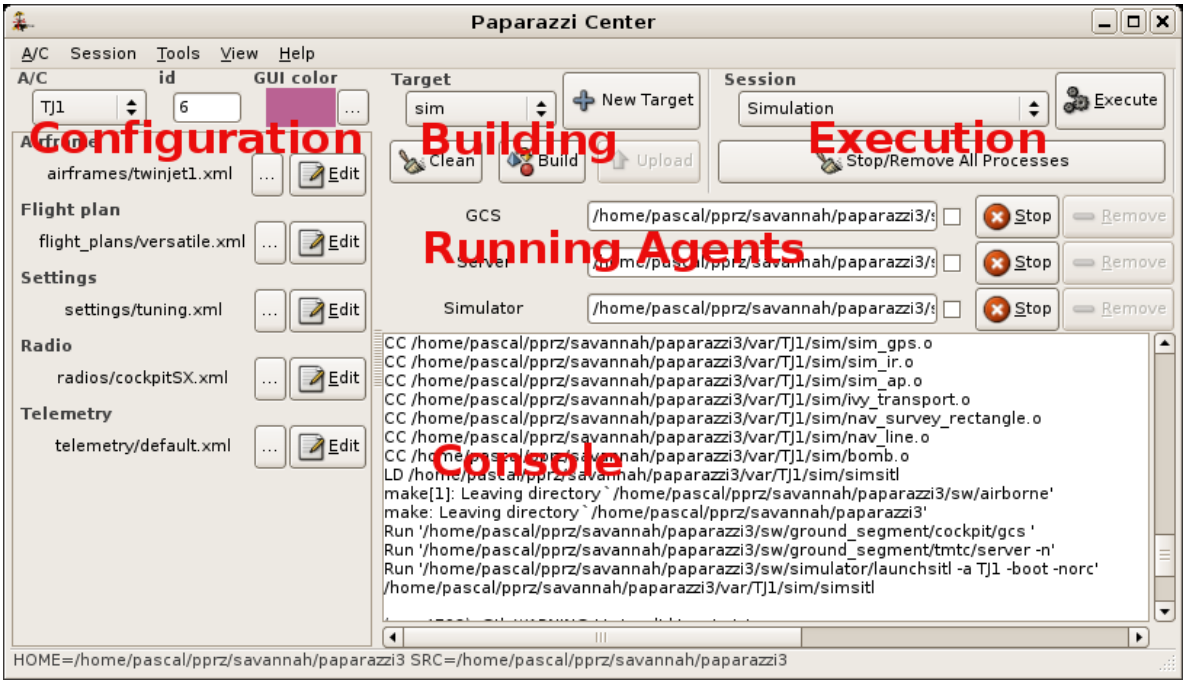

Figure 5.4. Paparazzi Center [18].

5.1.3.2. Ground Control Station (GCS)

It is the main interface of the software; it displays information during the flight or simulation. It also allows the user to interact with the UAV by moving waypoints or changing parameters of navigation and settings. In Figure 5.5 the GCS interface is showed, all functions and configurations are explained below.

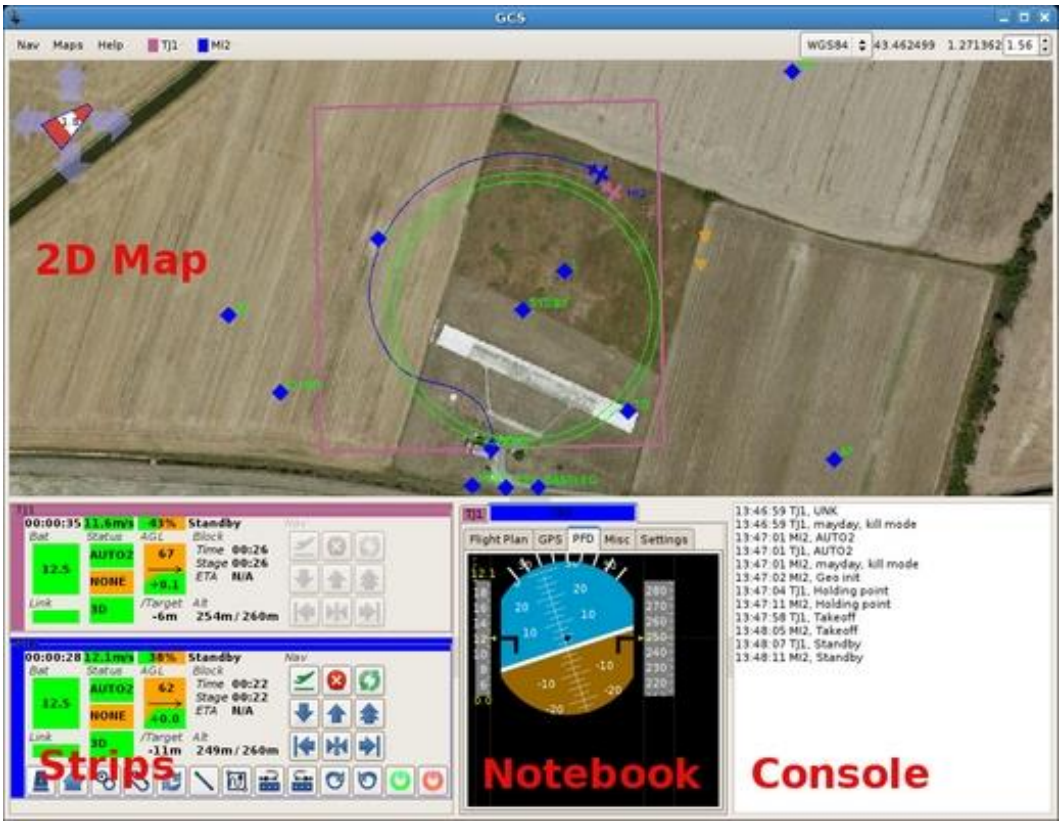

Figure 5.5. Ground Control Station [18]

Each aircraft has an associate Strip, this is the name of the window that allows for the available information to be display and also there are some interactive buttons for common commands. This window was configured through the settings file, although the software displays this command buttons, this had to be configured in order for them to execute the function showed in Figure 5.6. The buttons configured are the ones located in the bottom such as: Takeoff, standby, line and others. Also the altitude and lateral shift were enabled.

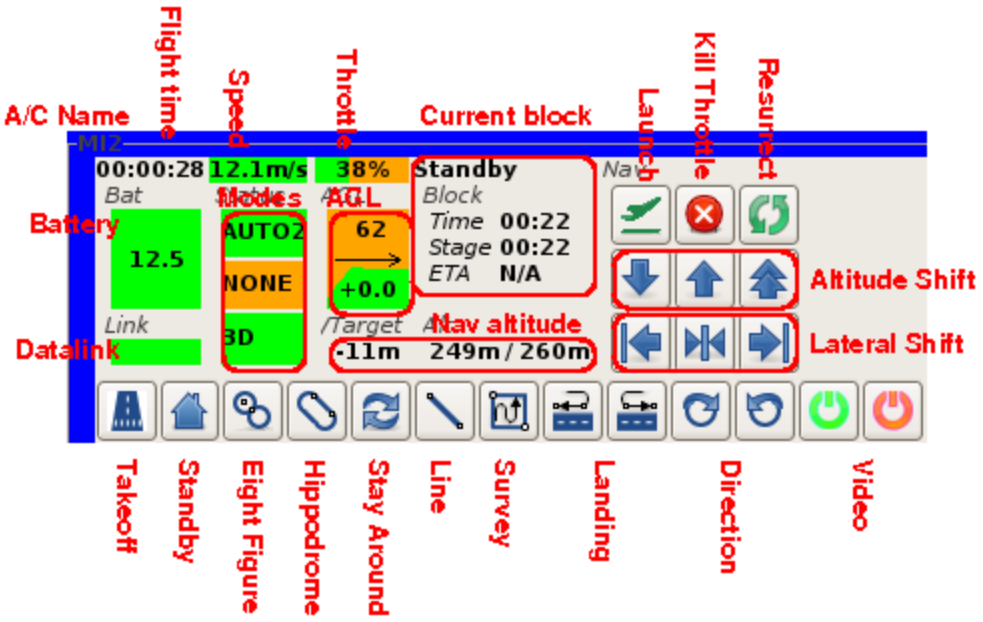

Figure 5.6. GCS Strip [18]

The 2D map in Figure 5.5 displays the following information:

- The waypoints defined in the Flight Plan (Blue diamonds in Figure 5.5).
- The default background is black, but it can be automatically filled with Google calibrated images from there servers.
- The intended trajectory of the aircraft is marked in green.
- The GPS coordinates are displayed at the top right hand corner, the WGS84 system reference is used.

This map follows the code generated in the Flight Plan in order for the aircraft to move through waypoints, it is important to stand out that the waypoints can be placed by programing them in the Flight Plan. Another important option in the map is the possibility to edit the waypoints through the waypoint editor; it allows you to change the coordinates and also the height, see Figure 5.7

It is important to highlight that during navigation it is possible to zoom in and out in the map, also there is an option that allows to center and fit the map in order to see where all the waypoints are located.

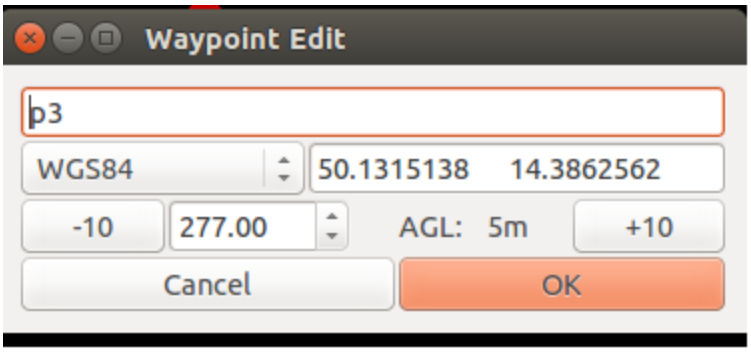

Figure 5.7. Waypoint Editor

The Notebook itself is divided in subpages displaying different kinds of data such as telemetry data or autopilot tuning parameters in Table 5.2 there is summary of what can be found in this part of the GCS. In the Notebook section the Flight Plan was configured as stated in the last section. The settings section was also configured, so the different variables configured in the telemetry file can be displayed, also it is presented the stabilization loops this were configured in order that the PID values could be change through the GCS interface. Also other modules were configured such as calibration, logger file to store data, the camera and the Optic Flow control algorithm, this are going to be discussed later.

Table 5.3. Notebook Subpages

Subpages	Function
Flight Plan	Gives a detail view at the functions in the flight plan, every active block is highlighted and to switch to another a double click is necessary.
GPS	It gives the level of tracked satellites and estimate position accuracy.
PFD	Displays the estimate altitude of the aircraft and the ground speed.
Settings	The Settings subpage is the more complex due to the amount of data that can be handled by it. First the telemetry can be access through here so different messages such as Euler Angles or measurements from the IMU can be display. Then there is the option of selecting the different navigation modes and also tuning parameters can be done through here. At last modules such as loggers can be controlled in here.

At last in the GCS the Console in Figure 5.5 is in charge of indicating any block change or alarm, this meaning that it will alert when a function on the Flight Plan is starting or if it is over. Also if the Drone is running out of battery or if it is flying too low. This alarms were configured in the settings file.

In Figure 5.8 there is a block diagram that summarizes the interaction between the software and the ARDrone 2.

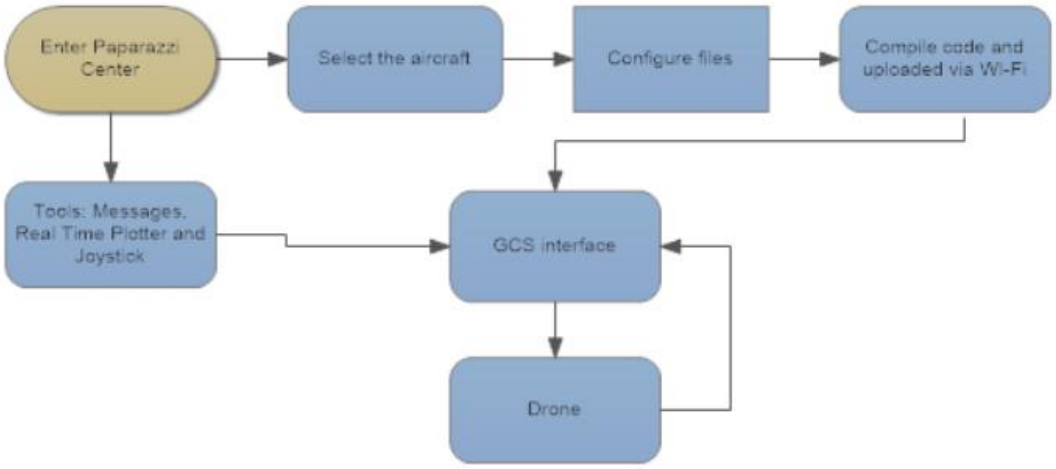

Figure 5.8. Software interaction with Drone

5.2. Attach a GPS module

In order to attach a GPS module several changes were made due to the fact that Paparazzi UAV works by default with uBlox GPS systems. The first thing was to identified which order type of GPS were available for ARDrone 2, by checking the Airframe file it was notice that for the Parrot Flight Recorder it was necessary to use the Sirf Binary subsystem. (Appendix 1)

The second thing that was notice is that inside Paparazzi the GPS was configure automatically, due to this the Flight Recorder had to be configure manually. Also the GPS module created by Parrot works with NMEA and Sirf Binary protocols, so SIRF Demo was used in order to configure the module in Sirf Binary protocol. In Figure 5.9 shows a block diagram explaining the configuration done to change the Flight Recorder protocol from NMEA to Sirf Binary.

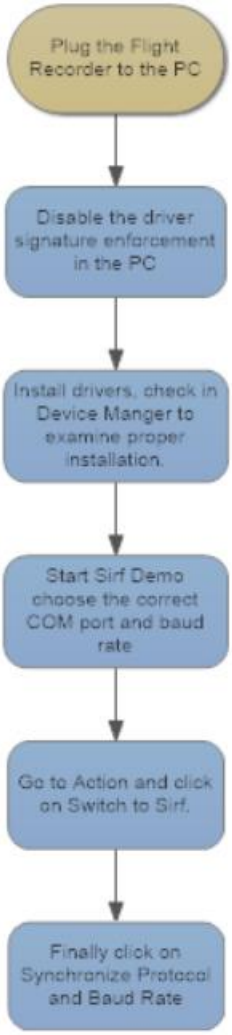

Figure 5.9. GPS configuration

After doing the configuration in the block diagram above, Sirf Demo should display a window like Figure 5.10. It can be notice that in the Map View, shows the coordinates and altitude where the configuration was made and in the Radar View it is shown how many satellites the GPS can have linked.

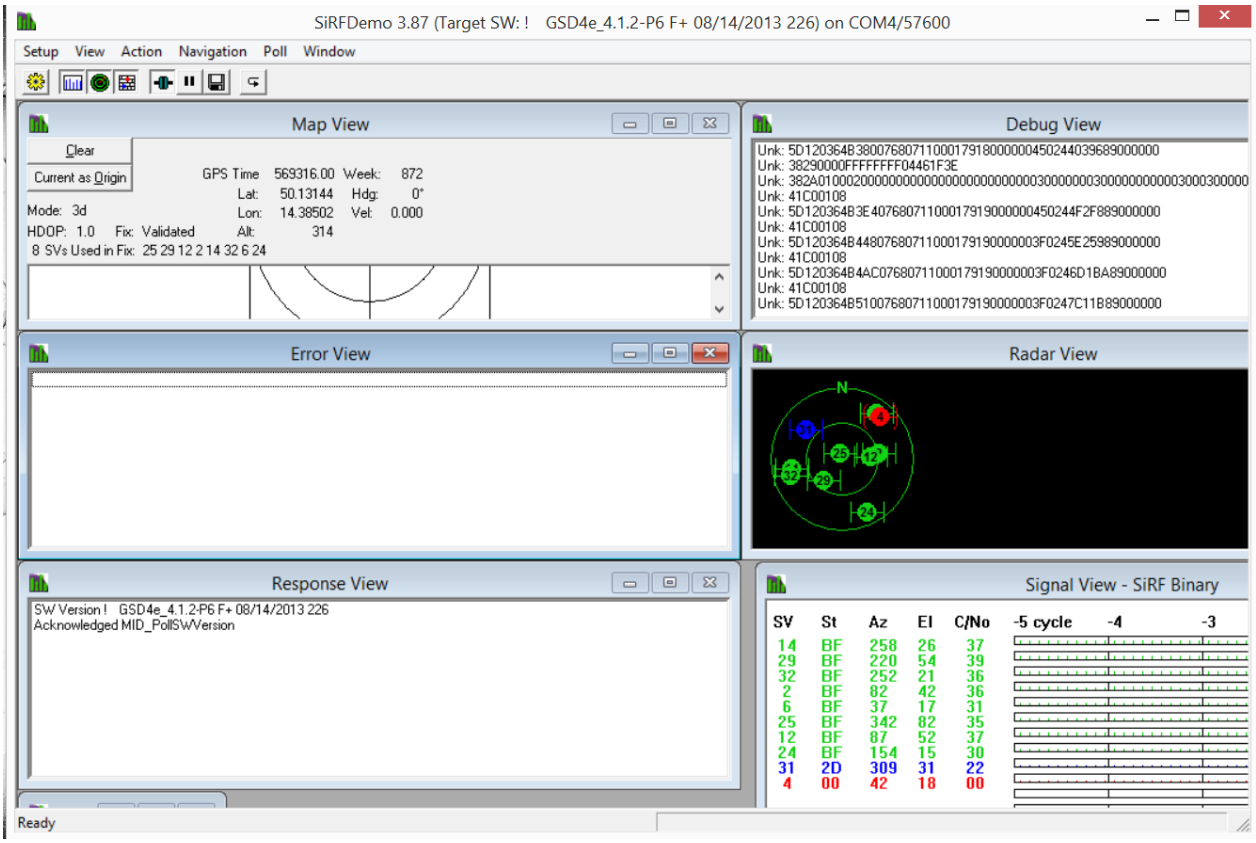

Figure 5.10. SIRF Demos Software

After configuring the Flight Recorder with the SIRF Demo software and making the necessary changes in the Airframe file, the module was successfully attached.

5.2.1. Fly through selected waypoints

In order to ensure a successful flight a correct calibration for the accelerometer and the magnetometer was necessary, this was done because these sensors are critical for the INS execution. It is important to highlight that this calibration should be done always before starting the flight.

With Paparazzi UAV this can be done, due to fact that it uses a preloaded algorithm to calculate the scaling (sf) and neutral (n) factors for the calibration, see Equation 5.1 and Equation 5.2.

$$\boldsymbol{value} = (\boldsymbol{sf}) * (\boldsymbol{sensor} - \boldsymbol{n}) \quad (5.1)$$

$$\boldsymbol{9.81^2} = (\boldsymbol{sf_x}(\boldsymbol{sensor} - \boldsymbol{n_x})^2 + (\boldsymbol{sf_y}(\boldsymbol{sensor} - \boldsymbol{n_y})^2 + (\boldsymbol{sf_z}(\boldsymbol{sensor} - \boldsymbol{n_z})^2 \quad (5.2)$$

Equation 5.2 is derived from the fact that at a static position the drone will measure a constant vector of gravity. The way the algorithm works is that first an initial guess is calculated, whose values are close to the average gravity value, then a data fitting algorithm is used to optimized the first guess.

For the accelerometer it is not necessary a calibration, if the components of the acceleration in x and y axis does not exceed the ±4 m/s^2, this was done by using the Real Time Plotter and Messages in the Paparazzi Center Tools men, see Figure 5.11

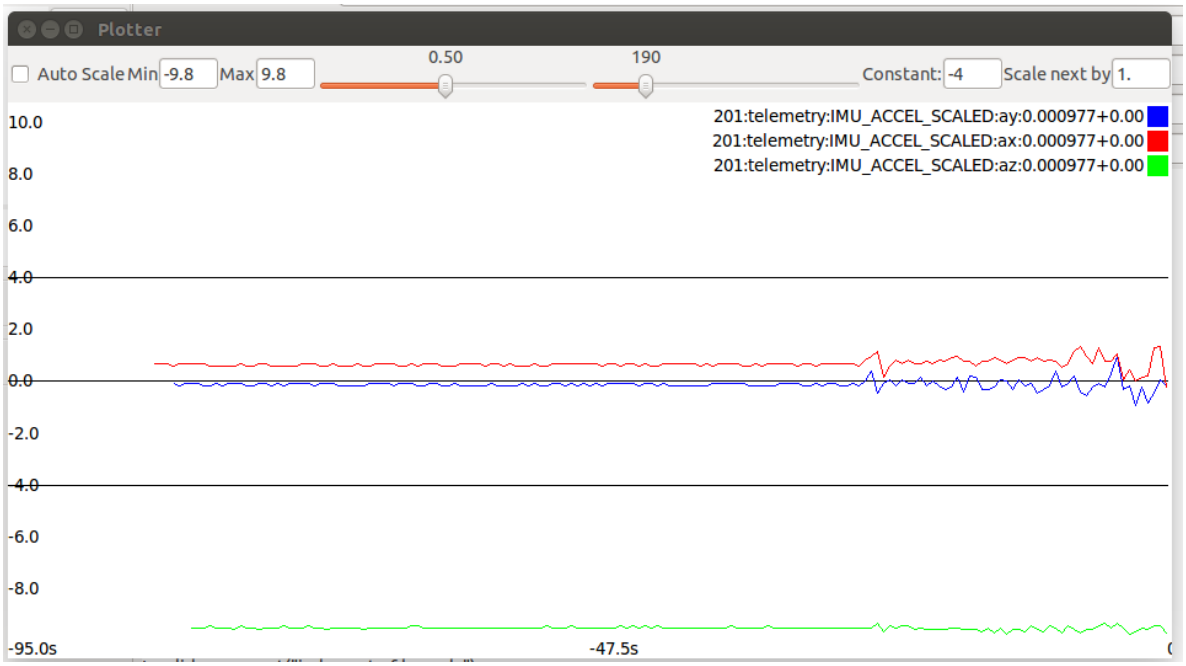

Figure 5.11. Calibration of accelerometer

The calibration is necessary for the magnetometer, this was done by using also the Real Time Plotter and Messages tools, once the three axes are plotted the Drone must be moved around all its axes, in order to checked if the collected data was done appropriately, run the command: *sw/tools/calibration/calibrate.py -s MAG var/logs/YY_MM_DD__hh_mm_ss.data -vp*, in the Linux console. An ellipsoid will be generated, if the blue points are well distributed all around the ellipsoid, then the calibration was successful. In Figure 5.12 two ellipsoid are shown showing the distribution of measurements taken with preloaded algorithm, the difference between the figures is that the one on the right is normalized.

Another way to checked if the magnetometer is well adjusted is by checking the heading of the Drone in the GCS map, once this is done the Drone is ready to flight.

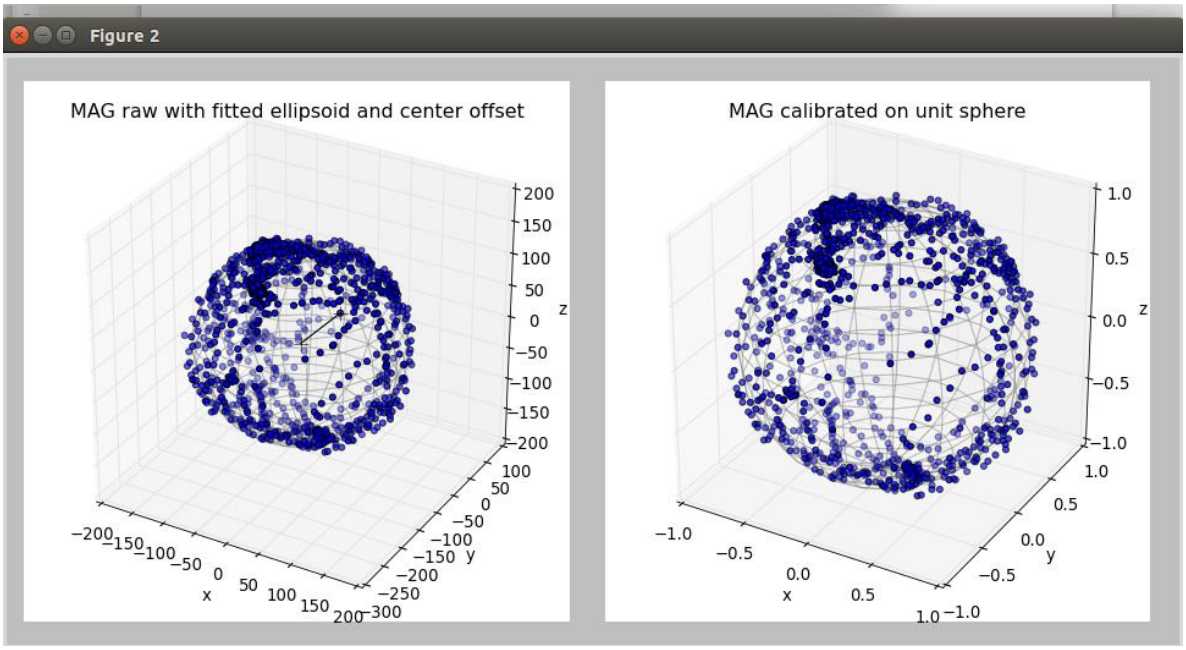

Figure 5.12. Magnetometer Calibration

5.3. Improve the altitude stabilization with a new controller

5.3.1. Recorded Data

In order to implement a new controller, it was necessary to use the memory in the Flight Recorder in order to store the necessary data for further analysis. First it was necessary to use a logger module this can be seen in Appendix 1, called logger file which was modify in order to record the data that was of interest. In Table 5.3 the variables that were recorded are presented. The angles phi, psi and theta represent the roll, the yaw and the pitch. For the accelerometer, the gyros and magnetometers the three axis were recorded, in order to verify proper functionality. The other variables such as: Longitude, latitude altitude, the measurement from the ultrasound sensor and the velocity were recorded in order to implement properly the stabilization for the vertical position.

Table 5.4. Data Recorded

Data	**Scaling factor**	**Units**
Phi (Roll)	No scale	Rad
Psi (Yaw)	No scale	Rad
Theta (Pitch)	No scale	Rad
Gyro (In the 3 axis)	0.0139882	deg/s
Accelerometer(In the 3 axis)	0.0009766	m/s^2
Magnetometer(In the 3 axis)	No scale	T
Altitude	No scale	M
Latitude	10^7	Deg
Longitude	10^7	Deg
Ultrasound	(measuredata-880)*0.00047	Cm
Velocity (In the 3 axis)	No scale	mm/s

5.3.2. Implement a new controller

The new controller was added with the help of the modules created by the Paparazzi developers. The selected module was the Optic Flow that uses the bottom camera and ultrasonic sensors to accomplish stability. The Optic Flow control algorithm was implemented due to the results obtain with the Flight Recorder (See Section 6.2).

As mentioned above a new controller independent from the GPS was necessary, this presented a challenge because Paparazzi UAV is program to use all its control loops with the reference from the GPS coordinates.

In Figure 5.13 the architecture of the algorithm is explained, from the video camera with the Optic-Flow algorithm the groundspeed is used to calculate the horizontal velocity. Combine with the gyros, accelerometers and ultrasound sensors the altitude can be estimated. Once known the altitude an angular corrector and a velocity corrector is used to stabilize the Drone, this is done through two symmetrical PI loops in phi and theta.

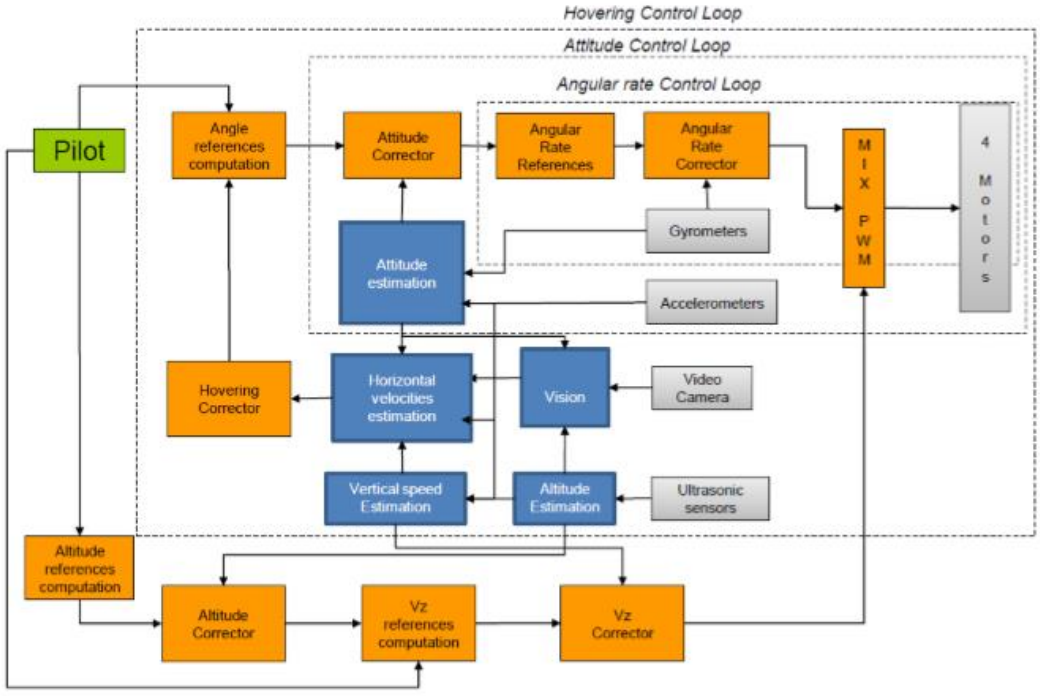

Figure 5.13. Optic Flow Loops [23]

In order to execute this controller some changes were needed in the Airframe file, first the camera was configured in order to obtain an image, after this the Optic Flow algorithm is implemented the vision loop constants can be seen (Appendix 4). Once the changes were made the camera was tested in order to see if the software capture the image from the bottom camera. Then some variables had to be added in order to ensure a proper functionality of the algorithm, see Figure 5.14.

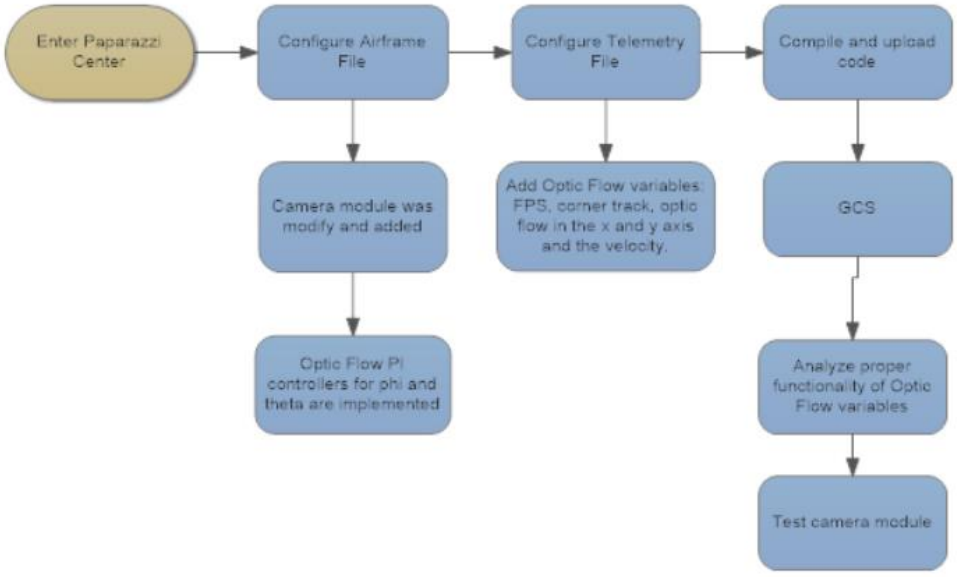

Figure 5.14. Configurations to add new controller

5.4. Flight Tests

5.4.1. With GPS

The flight tests were done by editing the flight plan, the first aspect to consider are the main attributes of a flight:

- Latitude 0 and Longitude 0: It defines the reference point {0,0} in the WGS84 coordinates.
- Ground Altitude: The ground altitude in meters above the sea level.
- Altitude: It defines the altitude of the waypoints.
- Security Altitude: The lowest altitude the aircraft can flight.
- Maximum distance from home: A representative radius of how far the aircraft can get away from the HOME waypoint.

Another relevant aspect is the waypoints which are geographic locations to specify trajectories, these are specified by a name and relative coordinates: <waypoint name x y [alt]/>. This x y are coordinates from point {0,0}, also another way edit is as explain in section 5.1.3.2.

The blocks are the main part of the plan they are in charge to describe each unit of the mission. They are made of stages and exceptions, when a stage or block ends it go to next one. Through the blocks the Navigation Modes are used this are explained in Table 5.5.

Table 5.5. Navigation Modes

Navigation Modes	**Function**
Attitude	Keeps a fixed attitude in the waypoint.
Heading	Keeps a heading course.
Go	Goes to a given waypoint.
Path	List of waypoints linked by the navigation mode go.
Circle	Circles around a waypoint.
Oval	Two and half circles between two waypoints.
Eight	Fly a figure of eight through several waypoints.
Stay	Hold position in a desired waypoint.
Follow	Follows another aircraft.

In Figure 5.15 and Figure 5.16 are examples of how the block programming works, Figure 5.17 shows a block diagram of how the Flight Plan code was designed.

```
<block name="circle CAM" pre_call="nav_set_heading_towards_waypoint(WP_CAM)">
  <circle radius="10" wp="CAM"/>
</block>
```

Figure 5.15. Circle Navigation Mode Code

```
<block name="line_p1_p2">
  <go from="p1" hmode="route" wp="p2"/>
  <stay wp="p2" until="stage_time>10"/>
  <go from="p2" hmode="route" wp="p1"/>
  <deroute block="stay_p1"/>
</block>
```

Figure 5.16. Line Navigation Code

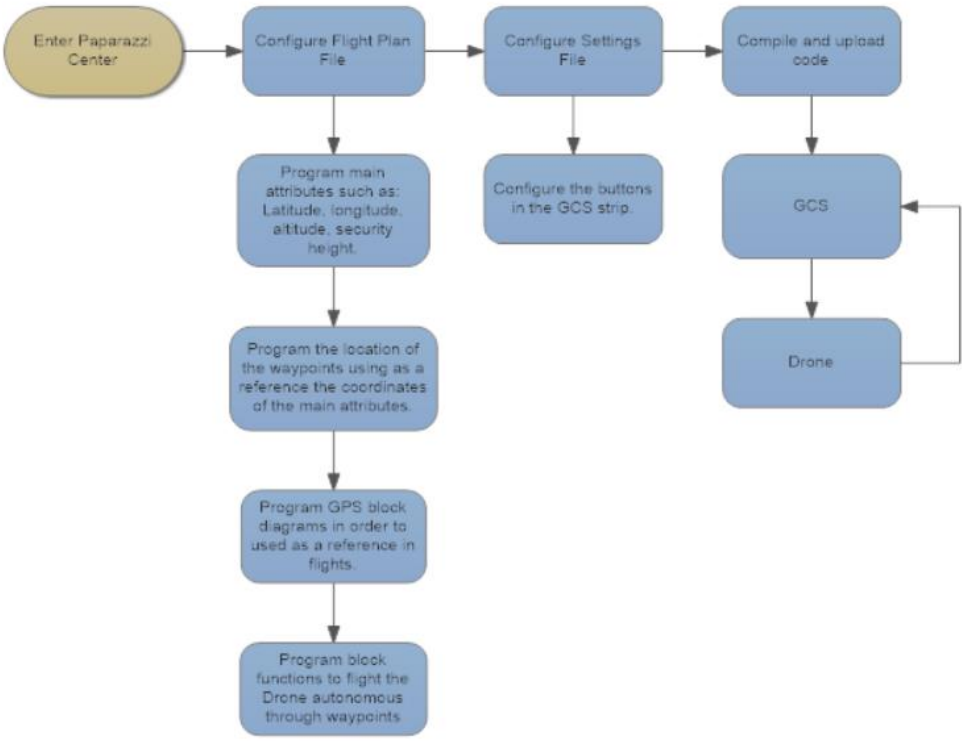

Figure 5.17 Flight Plan Configuration

5.4.2. With Optic Flow Module

In order to realize this tests a Joystick was needed, due to the fact that the flight plan couldn't be used, because of the lack of GPS. Inside the Paparazzi UAV software, it includes several manual flight modes which can be selected in Settings through the GCS:

- Attitude Direct (ATT): Full control of the Drone with the Joystick.
- Attitude Z Hold (A_ZH): Drone holds the altitude, still full control with Joystick.
- Hover Z Hold (H_ZH): The Drone hovers in the 3D position when entering this mode.

There are more manual flight modes which can be used for different application, in this case only these three modes where used. First with ATT the Drone was flown until the wanted altitude was achieve so A_ZH was activated, then H_ZH was used in order to record data for further analysis, see Figure 5.18.

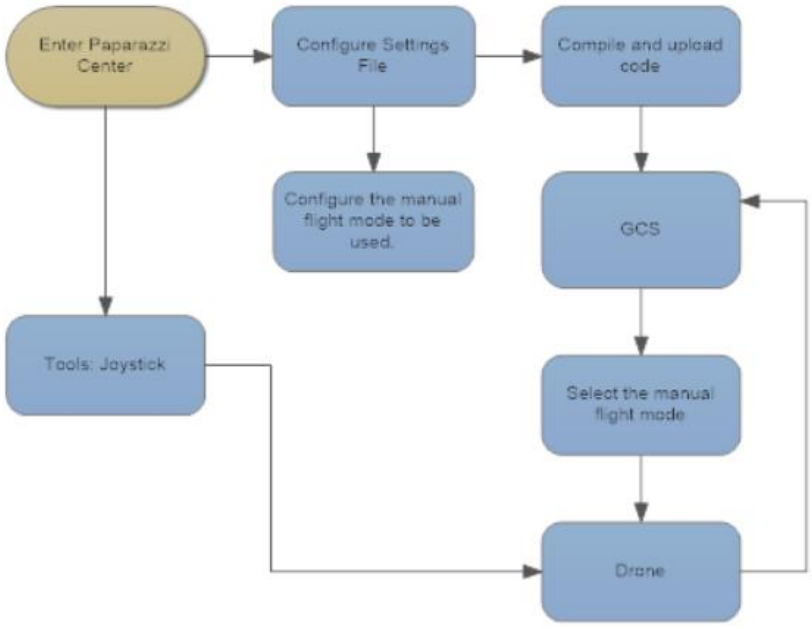

Figure 5.18. Manual Flight Mode

6 Experimental Results and Analysis

This section presents and discusses the results obtained, this in order to justify the operation obtained from the proposed solutions to each specific objective.

6.1. Software

6.1.1. Data visualized from the software

Once the Paparazzi UAV software was understand collecting data from it was very straight forward, just by changing the telemetry different data could be observed with the Messages Tool and could be graphic with the Real Time Plotter Tool, see Figure 6.1 and Figure 6.2.

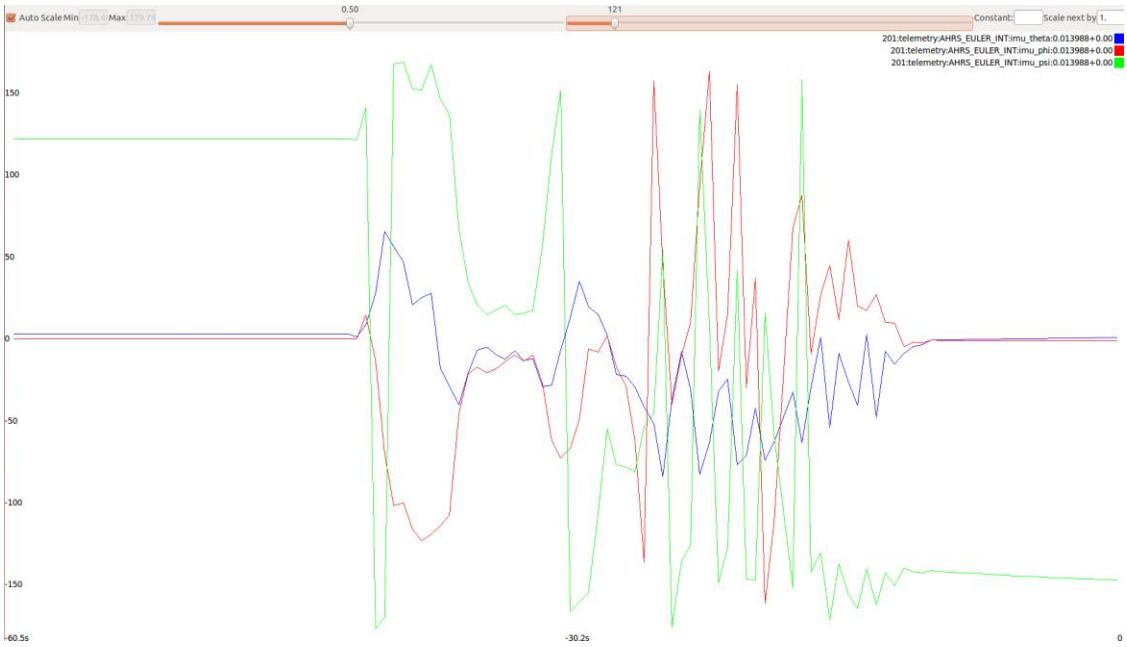

Figure 6.1. Euler Angles from Paparazzi UAV

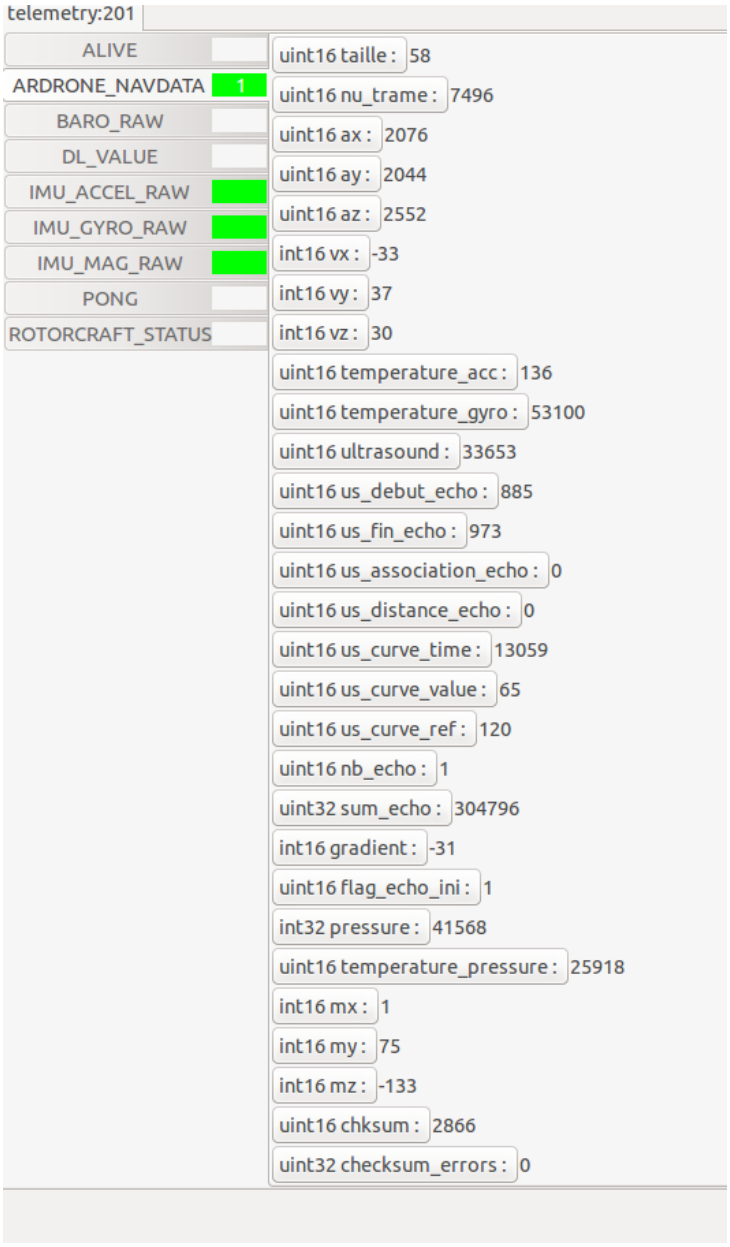

Figure 6.2. Navigation data from Paparazzi UAV

In Figure 6.1 there is a plot of the Euler Angles in green psi, in blue theta and in red phi the variations are really high due to fact that this data was recorded while the Drone was rotated through all its axes. Figure 6.2 displays the navigation data from the ARDrone 2 driver while it was hold about 20 cm from the ground (See Table 5.4 for ultrasound scaling) this values were configured in telemetry file in order to be shown, the high values are due to a scaling factor.

Figure 6.1 and 6.2 are just examples to show the correct functionality of the configuration done in telemetry file and the use of the Tool menu in order to display these data.

6.1.2. Flight the Drone Control by the software

Flying the Drone was easily done with the Tool Joystick from the Paparazzi Center where the software has different Joystick files that can be selected and just by choosing the correct file the Drone can be flown with it.

In Figure 6.3 there is a plot where the Joystick commands were tested: The throttle in orange, the roll in red, the pitch in blue and the yaw in green.

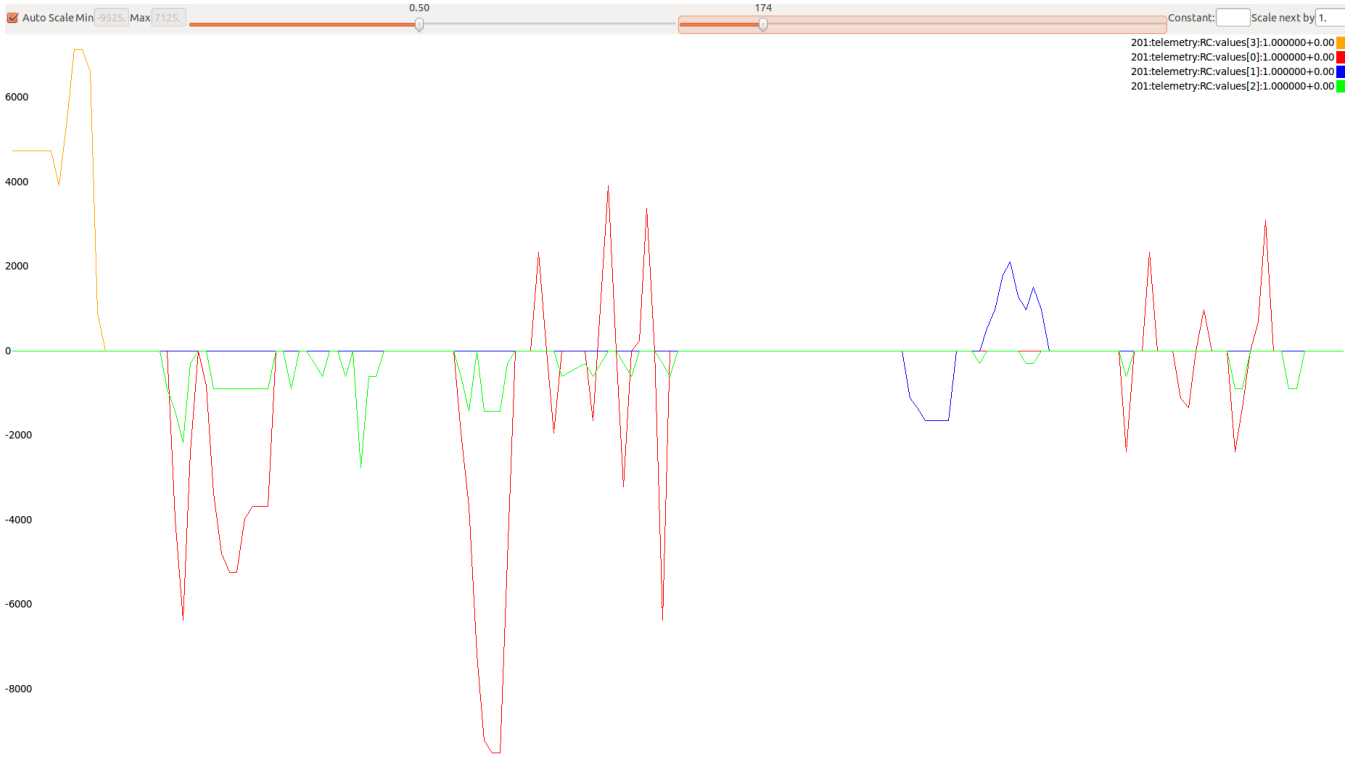

Figure 6.3. Commands from Joystick Tool

The Joystick commands agree with the movements of the Drone, which result in a successful flight by controlling with Paparazzi UAV.

6.2. GPS module attached

6.2.1. Overview of the GCS

Once the GPS module was configured and attached a Google Map was downloaded in order to move around the waypoints and a better view of where the aircraft was heading during the flight, see Figure 6.4.

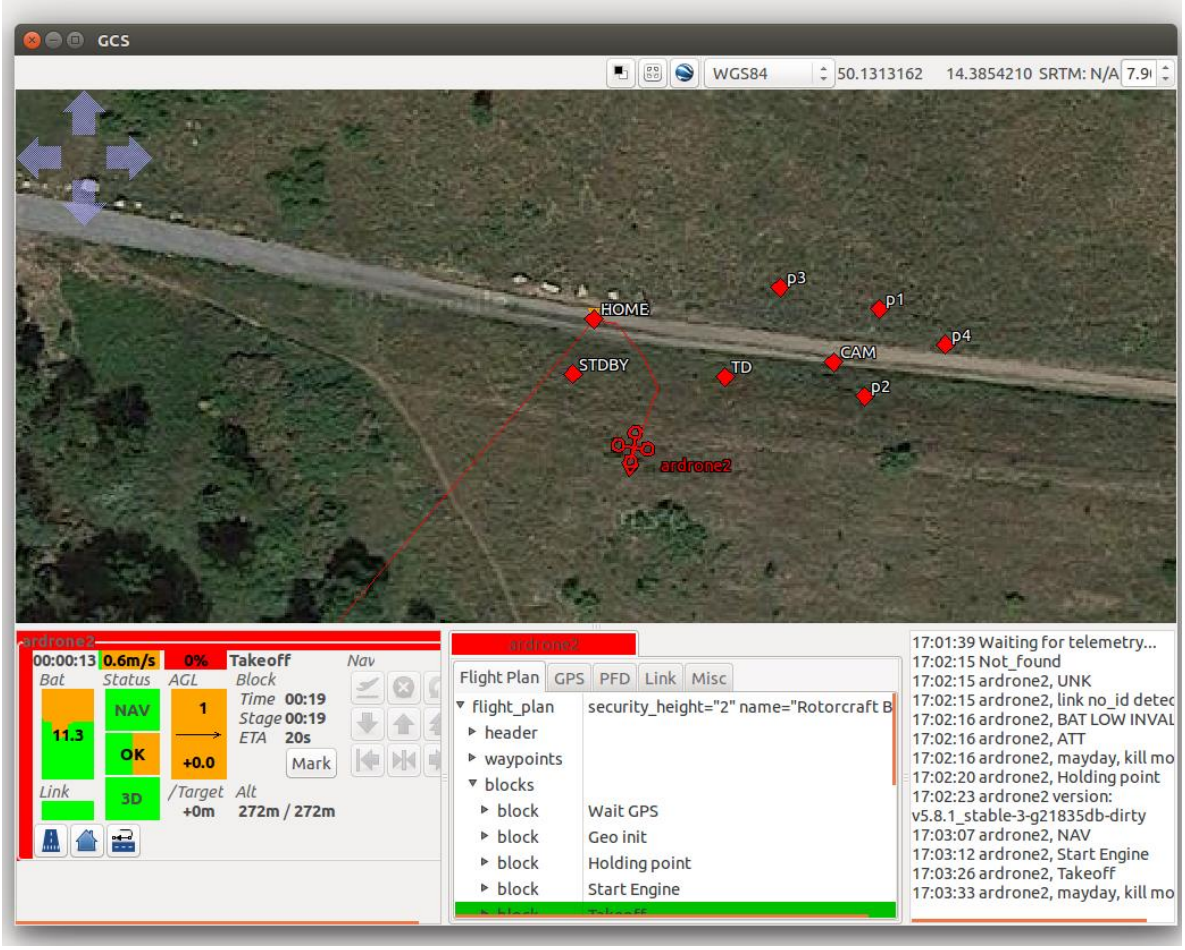

Figure 6.4. GCS with GPS Fix

This Figure 6.4 has taken in a flight test where the map was downloaded and the first blocks of the Flight Plan worked appropriately when the GPS signal was detected. Also GPS accuracy was recorded during other flight tests, see Table 6.1.

Table 6.1. GPS accuracy

Flight Test Number	**Accuracy (m)**
1	7.2
2	7.3
3	6.8
4	6.7
5	7.1
Average	7

From Table 6.1 it can be analyzed that there is a discrepancy from the theoretical accuracy of the Flight Recorder which is ±2m. The measures took in Table 6.1 show that the error is higher than the technical specifications dictated.

6.2.2. Flight Results

A number of approximately five flights were done in order to test the autonomous flight, unfortunately a successful one wasn't achieved. The quadrotor understands the commands written in the Flight Plan, for example the GPS is recognized by the software and the engine can be started, but when it takes off in order to start the flight instead of staying in its position (hovering), it rolls down to the ground.

This behavior was review and analyzed, thanks to the expertise of [10] a forum of Paparazzi UAV developers, it was understood that the Flight Recorder is not a reliable GPS and its autonomous flights have a poor performance. Another issue encounter for

the ARDrone 2 is their sensitive magnetometer, this must be well calibrated if not heading problems can be presented.

Once known the main problems presented in the quadrotor a straightforward experiment was done to test the GPS, a simple route was done where the initial point matches the ending point, the plots for latitude and longitude can be seen below, see Figure 6.5 and Figure 6.6.

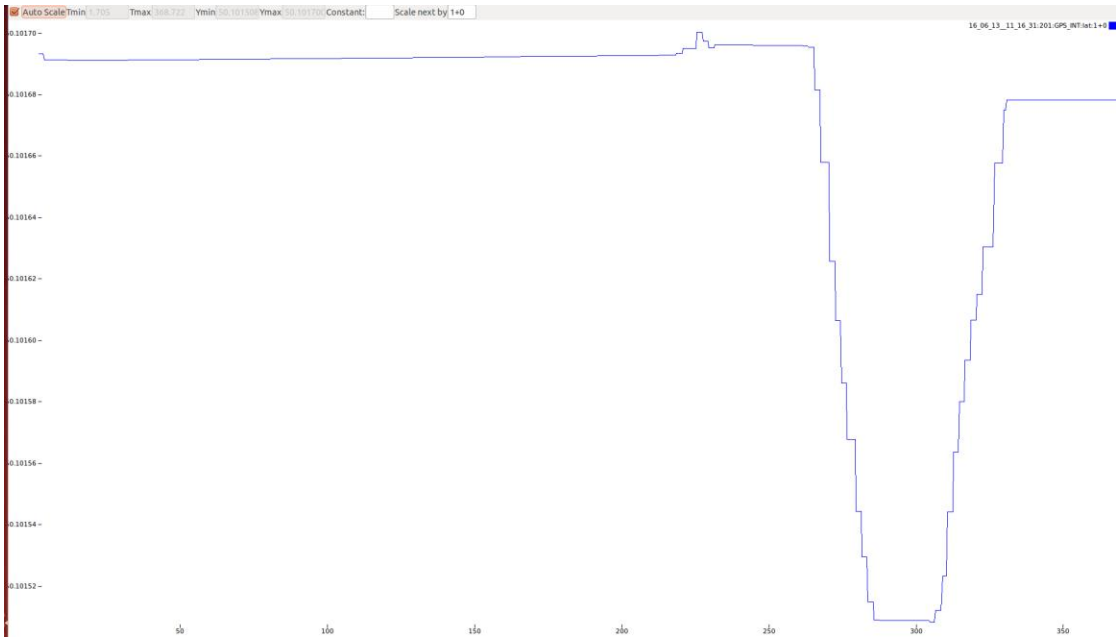

Figure 6.5. Latitude Plot

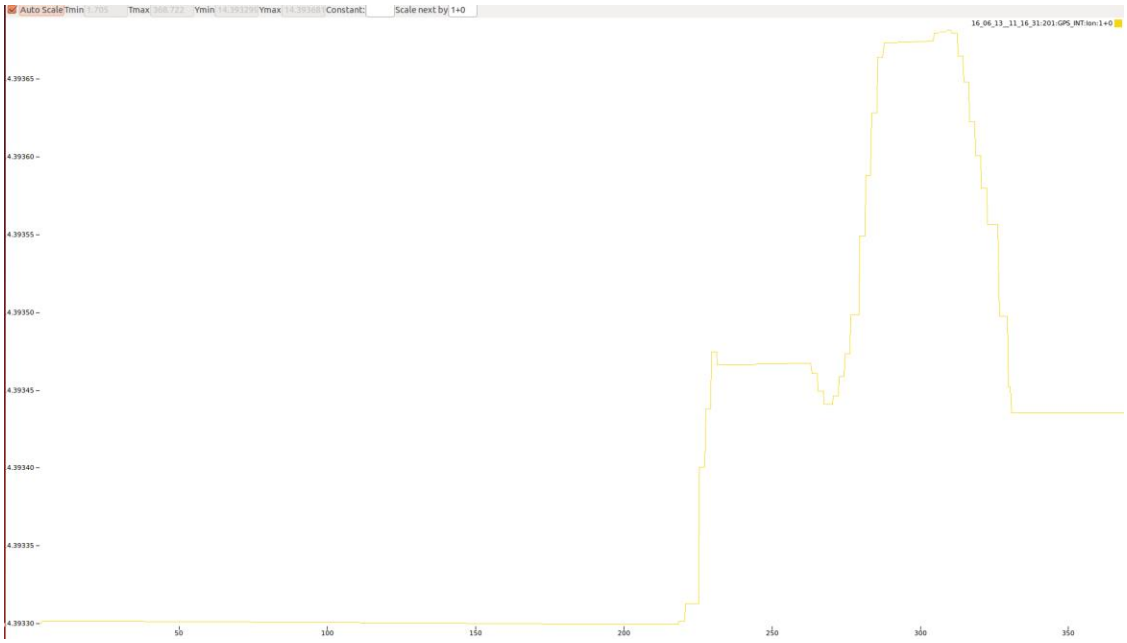

Figure 6.6. Longitude Plot

As it can be seen in both plots the initial point doesn't match the final point, for this in Google Maps the distance between this two points was measure, see Figure 6.7.

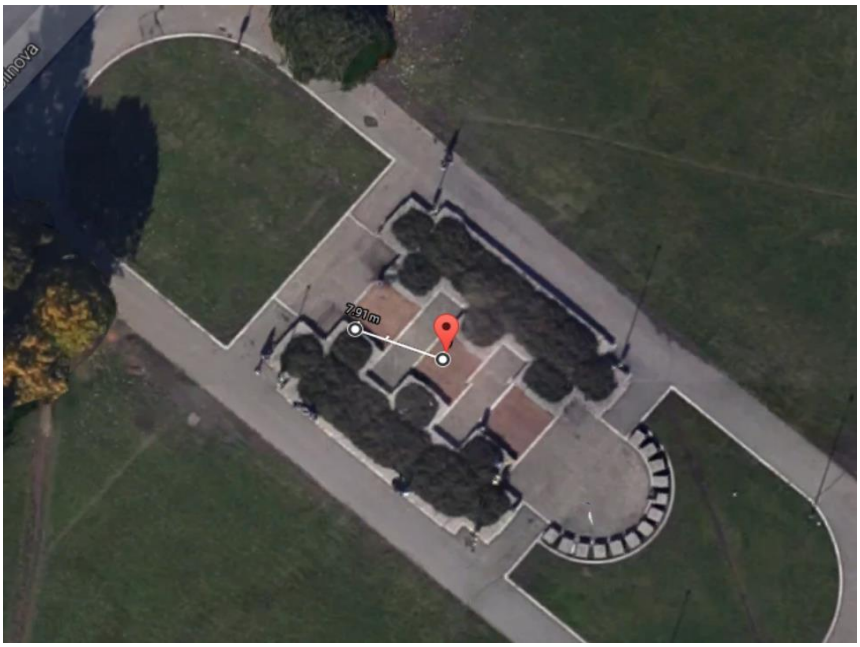

Figure 6.7. GPS Error

An error of almost 8 m is measure which confirms the data recorded in Table 6.1, this proves the GPS is not very reliable, due to the fact that the error is greater than the one specified in the technical specifications.

Then another measure that was acquire was the local magnetic field which is obtain also by the GPS, so this was compare with the theoretical magnetic field from [16], see Table 6.2 and Table 6.3.

Table 6.2. Theoretical Magnetic Field

Values	**North Component (Hx)**	**East Component (Hy)**	**Vertical Component (Hz)**
Without normalize	19, 861.4 nT	1,292.5 nT	44,800.7 nT
Normalize	0.405145 nT	0.02636 nT	0.913872 nT

Table 6.3. Measure Magnetic Field Normalize

Flight Test	Magnetic Components	Values (nT)	Percentage Error (%)
1	Hx	0.393066	3
	Hy	0.00293	88
	Hz	0.918945	0.55
2	Hx	0.381836	6
	Hy	0.00295	89
	Hz	0.923828	1.1
3	Hx	0.39555	2
	Hy	0.00293	88
	Hz	0.918945	0.55

In Table 6.3 there is a big error in the East Component from the acquire data, this can result in heading errors.

6.3. Adding a new controller

6.3.1. Recorded data

Once the data was recorded it was analyzed with the help of Matlab, this was done by storing the data in a logger, then a CSV file was generated which incorporated the data of interest. This data was imported to Matlab, see Figure 6.8.

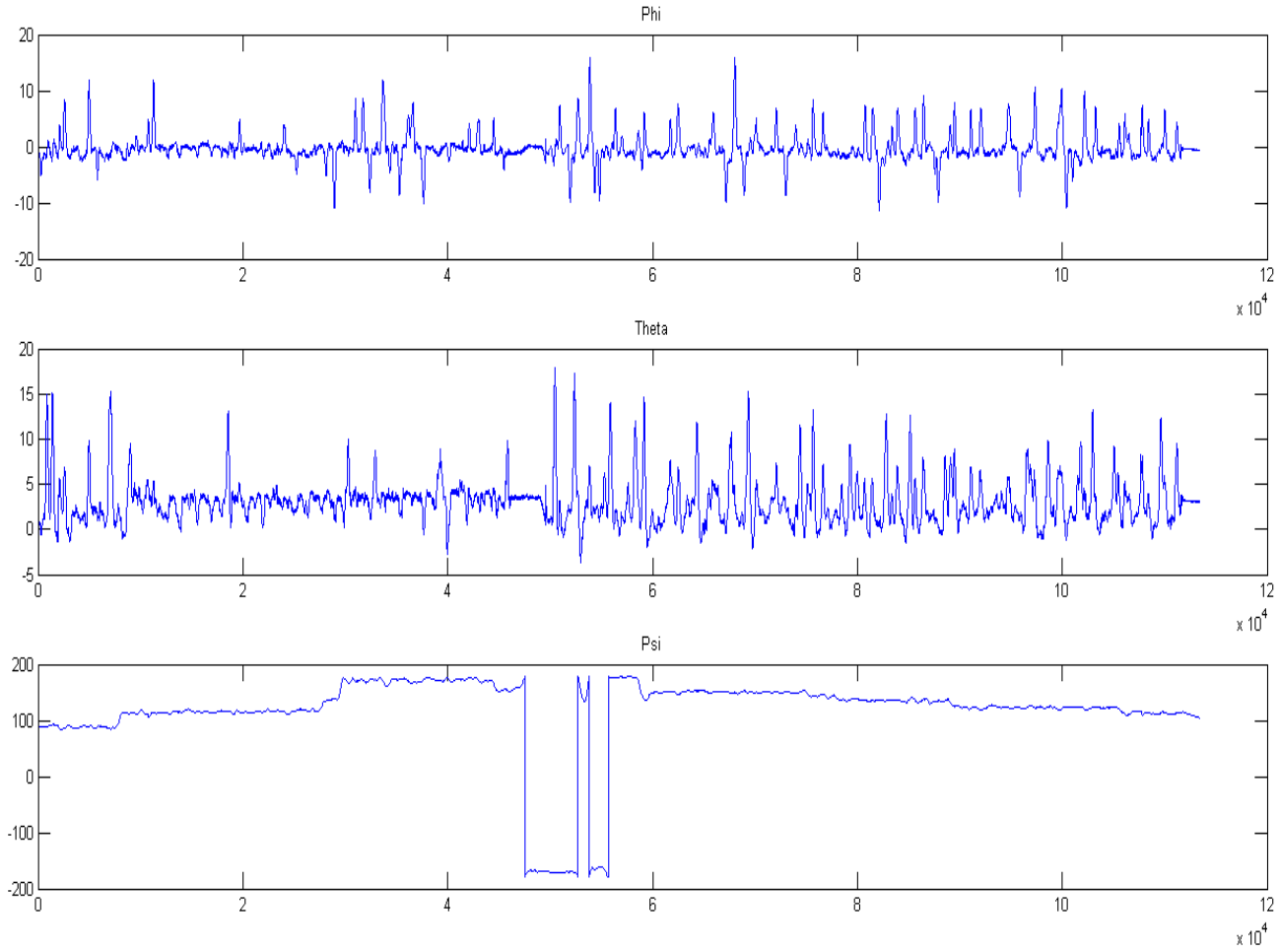

Figure 6.8. Data Recorded from Log File

The data shown on Figure 6.8 are the phi, psi and theta from an indoor flight in order to test the log file, this Euler Angles agree with the flight movements that were realized.

6.3.2. Flight Tests with the new controller

First the camera was tested in order to verify if the configuration done in the airframe file was done correctly, see Figure 6.9. Then the Messages was used in order to check if the telemetry variables were configured correctly, see Figure 6.10.

Figure 6.9. Verify camera functionality

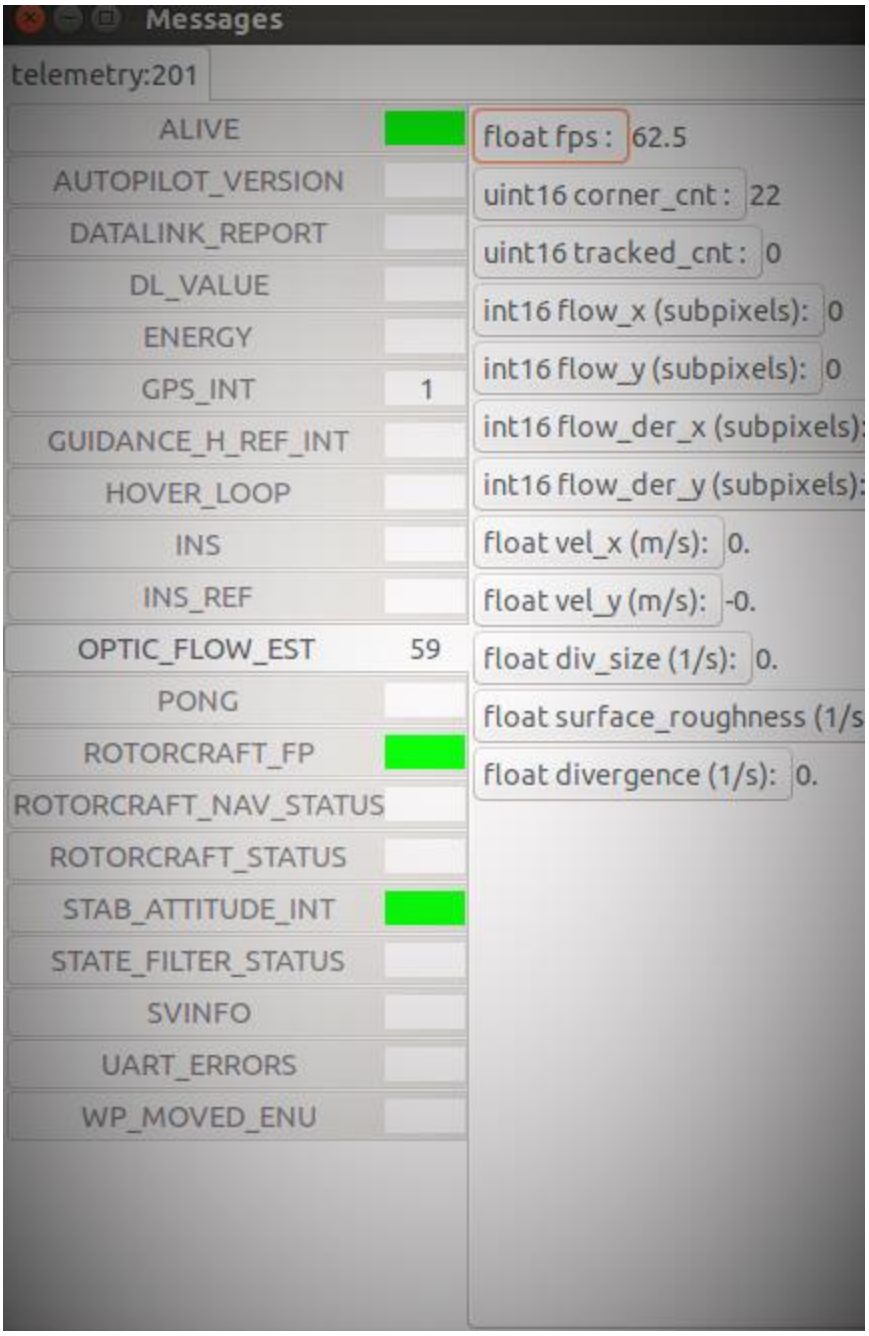

Figure 6.10. Optic Flow variables

After checking the correct functionality of the Optic Flow algorithm, its PI control constants for phi and theta were tuned using the heuristic method. In order to achieved the best possible outcome, see Table 6.4.

Table 6.4. PI control Values for Optic Flow Controller

P constant	**I constant**	**Average altitude (m)**
1500	800	0.8
1250	800	0.6
1000	800	0.6
1500	700	0.8
1250	600	0.6
1000	600	0.7

In Table 6.4 different values of PI control are presented which were used to tuned the Optic Flow controller, also the average measure altitude is given, in all the tests the altitude was kept at 1 m.

Analyzing the altitude values there seems to be a slight change which can be neglected due to the low altitude that the Drone is flying.

In Figure 6.11 and Figure 6.12 the phi and theta values for the different PI constants are plotted in order to analyze which constants give a better flight performance.

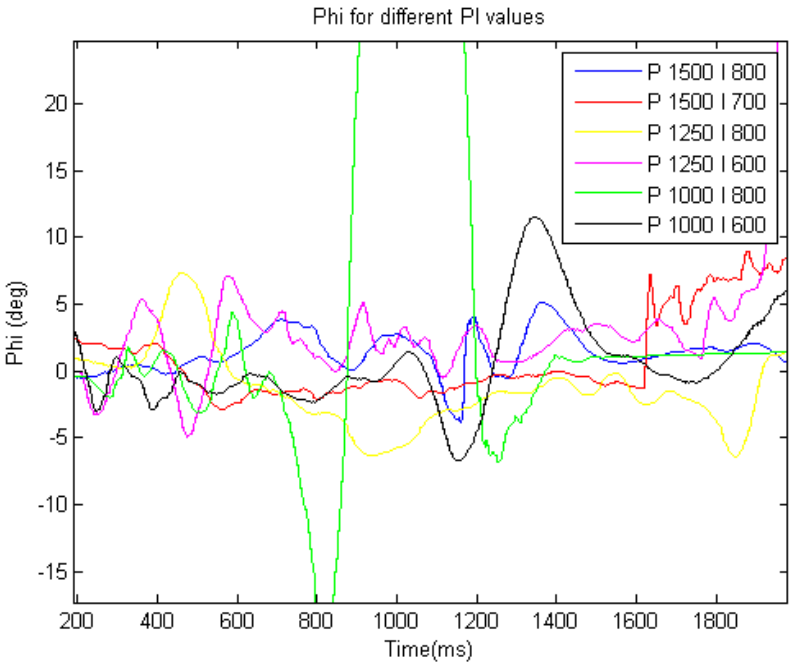

Figure 6.11. Phi for different PI values

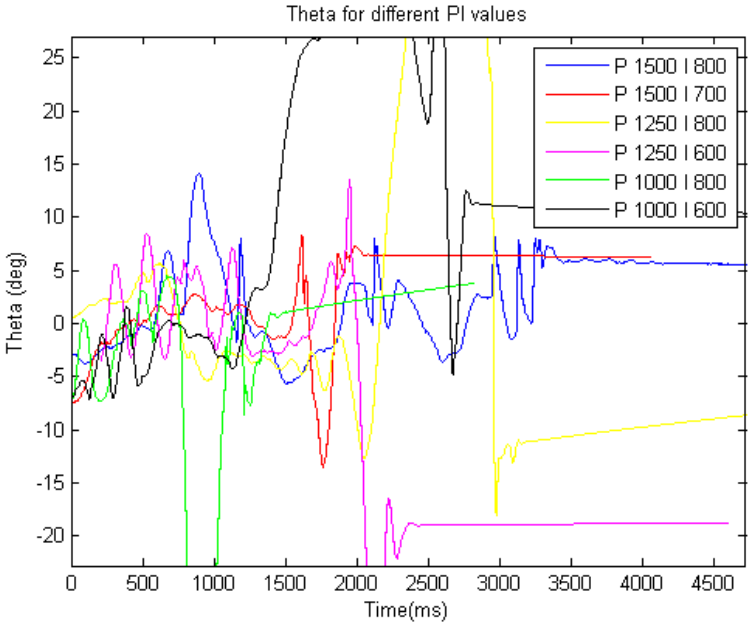

Figure 6.12. Theta for different PI values

For the plots above is important to point that the default values are $K_P = 1500$ and $K_I = 800$.

Analyzing the trending of the phi graph it can be seen that when the constants lowers its value, the Drone rolls more drastically. The same behavior can be observed in the theta plot where a decrease in the constants results in a more drastic pitch.

It is important to remember that the goal of this controller is to stabilize the vertical position of the Drone at a certain height. During the tests, stabilization was achieved only for a few seconds, then the UAV change its course and kept heading straight with no control. The reasons for this behavior are due to the errors presented with the magnetometer and the altitude calculation from the ultrasound which occasionally registered erratic values of the vertical height. This erratic behavior is due to a false interpretation of data provided by the Drone to Paparazzi messages, for example during a flight test at low altitudes the ultrasound sensors detected altitudes of fifty meters or more.

The errors commented above are frequently known for users who use the ARDrone 2 with Paparazzi UAV, this are usually solved by buying another autopilot unit.

7 Conclusions and Recommendations

7.1. Conclusions

Paparazzi UAV is an excellent tool for generating rotorcrafts application due to its open-source characteristic that allows you to improve its performance.

A great accuracy is necessary in the GPS modules, due to the unbound error growth from the IMU and the external sensors from the Drone.

ARDrone 2 is ideal for developing different applications, due to the simplicity of uploading new code into its autopilot and the technology incorporated in it.

Optic Flow stabilization loops are suitable for indoor applications, because they don't depend on GPS references, but instead they rely on DSP algorithms to compute its position.

Tuning is imperative in order to enchased a controller performance, so that the error can be quickly eliminated without causing the process variable to fluctuate excessively.

7.2. Recommendations

Before every flight it is recommend to calibrate the sensors, in order to avoid unexpected errors during a flight test.

Use a different GPS module due to the fact that it is not very accurate as its technical specifications specifies.

Use a uBlox system for the GPS is advise, because Paparazzi UAV works with these type of GPS by default and it has better accuracy than the Parrot Flight Recorder.

There are several ARDrone 2 units who have issues communicating with the software Paparazzi UAV that result in false interpretation of data leading to errors. By adding a reliable GPS this errors should be solved, otherwise it is advised to change the autopilot unit.

8 References

[1] Aggarwal, P., Syed, Z., & El-Sheimy, N. (2010). *MEMS-Based Integrated Navigation*. Artech House Books.

[2] Balas, C. (2007). *Modelling and Linear Control of Quadroptor* . Bedfordshire: Cranfield University.

[3] Beard, R. W., & Mclain, T. W. (2012). *Small Unmanned Aircraft: Theory and Practices*. Princeton Universtiy Press.

[4] Biever, E. (2015). *Controlling of an Single Drone*. Eindhoven: Department of Mechanical Engineering .

[5] Brisset, P., Drouin, A., Gorraz, M., Huard, P.-S., & Tyler, J. (2006). *The Paparazzi Solution, presented at ENAC, France.* Toulouse.

[6] Bristeau, P.-J., Callou, F., Vissiere, D., & Petit, N. (2011). *The Navigation and Control inside the AR.Drone micro UAV, presented in IFAC World Congress.* Milano.

[7] De Lellis Costa de Oliveira, M. (2011). *Modeling, Identification and Control of a Quadroptor Aircraft*. Prague: Department of Control Engineering .

[8] Elruby, A., El-khatib, M., El-Amary, N., & Hashad, A. (n.d.). *Dynamic Modelling and Control of Quadrotor Vehicle.*

[9] Galati, S. (2006). *Geographic Information Systems Demystified.* Artech House Books.

[10] *Gitter*. (30 de May de 2016). Obtenido de Paparazzi Discuss: https://gitter.im/

[11] Horn, B. K., & Schunck, B. G. (s.f.). Obtenido de Determining Optical Flow: http://image.diku.dk/imagecanon/material/HornSchunckOptical_Flow.pdf

[12] Krajnik, T., Vonasek, V., Fiser, D., & Faigl, J. (s.f.). *AR-Drone as a Plataform for Robotic Research and Education*. Prague: The Gerstner Laboratory for Intelligent Decision Making and Control.

[13] Li, Q. (2014). *Grey-Box System Identification of a Quadroptor Unmanned Aerial Vehicle*. South Holland: Delf University of Technology.

[14] Luukkonen, T. (2011). *Modelling and control of quadcopter*. Aalto University.

[15] Martinez, A. M. (s.f.). Obtenido de Image Processing: Optical Flow: http://www2.ece.ohio-state.edu/~aleix/OpticalFlow.pdf

[16] *National Centers for Environmental Information*. (31 de May de 2016). Obtenido de Magnetic Field Calculators: http://www.ngdc.noaa.gov/geomag-web/#igrfwmm

[17] *Paparazzi UAS*. (n.d.). Retrieved March 15, 2016, from http://docs.paparazziuav.org/latest/index.html

[18] *Paparazzi UAV*. (s.f.). Recuperado el 1 de March de 2016, de http://wiki.paparazziuav.org/wiki/Main_Page

[19] Parrot SA. (n.d.). *Parrot*. Retrieved February 28, 2016, from http://www.parrot.com/usa/products/ardrone-2/

[20] Pestan, J., Sanchez-Lopez, J. L., Mellado-Bataller, I., Changhong, F., & Campoy, P. (s.f.). *AR Drone Indentification and Navigation Control at CVG-UPM*. Centre for Automation and Robotics.

[21] Remes, B., Hensen, D., Van Tienen, F., De Wagter, C., Van der Host, E., & De Croon, G. (2013). *Paparazzi: how to make swarm of Parot AR Drones fly autonomously based on GPS*. Toulouse: International Micro Air Vehicle Conference and Flight Competition .

[22] Woodman, O. J. (2007). *An introduction to inertial navigation*. Cambridge: University of Cambridge.

[23] Yifang, S. (2013). *Vision-Based Control for Autonomous Quadrotor UAVs.* New Jersey: Department of Mechanical and Aerospace Engineering Princeton University.

9 Appendix

Paparazzi UAV is a complete open-source software, meaning that code is already developed, so in this section only the modifications to this code are presented.

Appendix 1. Airframe file modifications to use the flight recorder

```
    <!-- Subsystem section -->
  <subsystem name="telemetry" type="transparent_udp"/>
  <subsystem name="radio_control" type="datalink"/>
  <subsystem name="motor_mixing"/>
  <subsystem name="actuators" type="ardrone2"/>
  <subsystem name="imu" type="ardrone2"/>
  <!-- gps: "ublox" or change to "sirf" for usage with parrot flight recorder -->
  <subsystem name="gps" type="sirf"/>
  <subsystem name="stabilization" type="int_quat"/>
   <subsystem name="ahrs" type="int_cmpl_quat"/>
  <subsystem name="ins" type="extended"/>
 <modules main_freq="512">
  <load name="bat_voltage_ardrone2.xml"/>
  <!-- remove the gps_ubx_ucenter module if you use the sirf gps (flight recorder) -->
  <!-- load name="gps_ubx_ucenter.xml"/-->
  <load name="send_imu_mag_current.xml"/>
  <load name="air_data.xml"/>
  <load name="geo_mag.xml"/>
  <load name="logger_file.xml"/>
```

```
  </modules>
```

Appendix 2. Airframe file modifications for testing the camera

```
 <modules main_freq="512">
   <load name="bat_voltage_ardrone2.xml"/>
   <!-- remove the gps_ubx_ucenter module if you use the sirf gps (flight recorder) -->
   < !--load name="gps_ubx_ucenter.xml"/-->
   <load name="send_imu_mag_current.xml"/>
   <load name="air_data.xml"/>
   <load name="geo_mag.xml"/>
   <load name="logger_file.xml"/>
   <load name="video_thread.xml">
<define name="VIDEO_THREAD_FPS" value="4"/>
<define name="VIDEO_THREAD_CAMERA" value="bottom_camera"/>
<define name="VIDEO_THREAD_SHOT_PATH"
value="/data/ftp/internal_000/images"/>
</load>
<load name="video_rtp_stream.xml">
<define name="VIEWVIDEO_DOWNSIZE_FACTOR" value="1"/>
<define name="VIEWVIDEO_QUALITY_FACTOR" value="70"/>
</load>
   </modules>
```

Appendix 3. Airframe file modifications for the optic flow controller

```
 <modules main_freq="512">
   <module name="bat_voltage_ardrone2.xml"/>
   <module name="gps_ubx_ucenter.xml"/>
   <module name="send_imu_mag_current.xml"/>
   <module name="logger_file.xml"/>
   <module name="video_thread">
```

```
    <define name="VIDEO_THREAD_FPS" value="4"/>
    <define name="VIDEO_THREAD_CAMERA" value="bottom_camera"/>
   <define                                    name="VIDEO_THREAD_SHOT_PATH"
value="/data/ftp/internal_000/images"/>
    </module>

    <module name="cv_opticflow">
     <define name="OPTICFLOW_CAMERA" value="bottom_camera"/>
    </module>
  </modules>

   <section name="VISION" prefix="VISION_">
    <define name="HOVER" value="FALSE"/>
    <define name="PHI_PGAIN" value="1500"/>
    <define name="PHI_IGAIN" value="800"/>
    <define name="THETA_PGAIN" value="1500"/>
    <define name="THETA_IGAIN" value="800"/>
    <define name="DESIRED_VX" value="0"/>
    <define name="DESIRED_VY" value="0"/>
   </section>
```

Appendix 4. Flight Plan file modifications with line and circle commands

```
<!DOCTYPE flight_plan SYSTEM "flight_plan.dtd">

<flight_plan alt="152" ground_alt="147" lat0="50.1316888" lon0="14.3860203"
max_dist_from_home="150" name="Rotorcraft Basic (Enac)" security_height="2">
  <header>
#include "autopilot.h"
```

```
  </header>
  <waypoints>
    <waypoint name="HOME" x="0.0" y="0.0"/>
    <waypoint name="CLIMB" x="0.0" y="5.0"/>
    <waypoint name="STDBY" x="-2.0" y="-5.0"/>
    <waypoint name="p1" x="3.6" y="-13.9"/>
    <waypoint name="p2" x="27.5" y="-48.2"/>
    <waypoint name="p3" x="16.7" y="-19.6"/>
    <waypoint name="p4" x="13.7" y="-40.7"/>
    <waypoint name="CAM" x="-20" y="-50" height="2.0"/>
    <waypoint name="TD" x="5.6" y="-10.9"/>
  </waypoints>
  <blocks>
<!--Don't change GPS blocks if not flight plan would not work-->
    <block name="Wait GPS">
      <call fun="NavKillThrottle()"/>
      <while cond="!GpsFixValid()"/>
    </block>
    <block name="Geo init">
      <while cond="LessThan(NavBlockTime(), 10)"/>
      <call fun="NavSetGroundReferenceHere()"/>
      <!-- call fun="NavSetAltitudeReferenceHere()"/-->
      <call fun="nav_set_heading_current()"/>
    </block>
    <block name="Holding point">
      <call fun="NavKillThrottle()"/>
      <attitude pitch="0" roll="0" throttle="0" vmode="throttle" until="FALSE"/>
    </block>
    <block name="Start Engine">
      <call fun="NavResurrect()"/>
      <attitude pitch="0" roll="0" throttle="0" vmode="throttle" until="FALSE"/>
```

```
    </block>
    <block name="Takeoff" strip_button="Takeoff" strip_icon="takeoff.png">
      <exception cond="stateGetPositionEnu_f()->z > 2.0" deroute="Standby"/>
      <call fun="NavSetWaypointHere(WP_CLIMB)"/>
      <stay vmode="climb" climb="nav_climb_vspeed" wp="CLIMB"/>
    </block>
    <block name="Standby" strip_button="Standby" strip_icon="home.png">
      <stay wp="STDBY"/>
    </block>
    <block name="stay_p1">
      <stay wp="p1"/>
    </block>
    <block name="go_p2">
      <call fun="nav_set_heading_deg(90)"/>
      <go wp="p2"/>
      <deroute block="stay_p1"/>
    </block>
    <block name="line_p1_p2">
      <go from="p1" hmode="route" wp="p2"/>
      <stay wp="p2" until="stage_time>10"/>
      <go from="p2" hmode="route" wp="p1"/>
      <deroute block="stay_p1"/>
    </block>
    <block name="circle CAM"
pre_call="nav_set_heading_towards_waypoint(WP_CAM)">
      <circle radius="10" wp="CAM"/>
    </block>  </blocks>
</flight_plan>
```

Appendix 5. Modifications in logger file to record data

```
#include "file_logger.h"
#include "subsystems/ins/vf_extended_float.h"
#include "subsystems/ahrs/ahrs_int_cmpl_quat.h"
#include <stdio.h>
#include "std.h"
#include "modules/sonar/sonar_adc.h"
#include "navdata.h"
#include "mcu_periph/adc.h"
#include "subsystems/abi.h"
#include "subsystems/gps.h"
#include "subsystems/imu.h"
#include "firmwares/rotorcraft/stabilization.h"
#include "state.h"
#include "math/pprz_algebra_int.h"
#include "math/pprz_orientation_conversion.h"
```

```
/** Set the default File logger path to the USB drive */
#ifndef FILE_LOGGER_PATH
#define FILE_LOGGER_PATH /data/video/usb
#endif

/** The file pointer */
static FILE *file_logger = NULL;

/** Start the file logger and open a new file */
void file_logger_start(void)
```

```
{
  uint32_t counter = 0;
  char filename[512];

  // Check for available files
  sprintf(filename, "%s/%05d.csv", STRINGIFY(FILE_LOGGER_PATH), counter);
  while ((file_logger = fopen(filename, "r"))) {
    fclose(file_logger);

    counter++;
    sprintf(filename, "%s/%05d.csv", STRINGIFY(FILE_LOGGER_PATH), counter);
  }

  file_logger = fopen(filename, "w");

  if (file_logger != NULL) {
    fprintf(
      file_logger,

"counter,phi,psi,theta,gyro_p,gyro_q,gyro_r,gyro_bias_p,gryo_bias_q,gyro_bias_r,accel_b
ias,z_meas,accel_x,accel_y,accel_z,COMMAND_THRUST,COMMAND_ROLL,COMMAN
D_PITCH,COMMAND_YAW,altitude,lat,lon,ultrasound,vx,vy,vz,mx,my,mz,qi,qx,qy,qz\n"
    );
  }
}

/** Stop the logger an nicely close the file */
void file_logger_stop(void)
{
  if (file_logger != NULL) {
    fclose(file_logger);
```

```
    file_logger = NULL;
  }
}

/** Log the values to a csv file */
void file_logger_periodic(void)
{
  if (file_logger == NULL) {
    return;
  }
  static uint32_t counter;

  struct Int32Quat *quat = stateGetNedToBodyQuat_i();

*In here you add the variables that need to be recorded, check if it is %d or %f

  fprintf(file_logger,
"%d,%f,%f,%f,%d,%d,%d,%d,%d,%d,%f,%f,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%
d,%d,%d,%d,%d,%d,%d,%d,%d,%d\n",
        counter,
        stateGetNedToBodyEulers_f()->phi,
        stateGetNedToBodyEulers_f()->psi,
        stateGetNedToBodyEulers_f()->theta,
        imu.gyro.p,
        imu.gyro.q,
        imu.gyro.r,
        ahrs_icq.gyro_bias.p,
        ahrs_icq.gyro_bias.q,
        ahrs_icq.gyro_bias.r,
        vff.bias,
        vff.z,
```

```
        imu.accel.x,
        imu.accel.y,
        imu.accel.z,
        stabilization_cmd[COMMAND_THRUST],
        stabilization_cmd[COMMAND_ROLL],
        stabilization_cmd[COMMAND_PITCH],
        stabilization_cmd[COMMAND_YAW],
        gps.lla_pos.alt,
        gps.lla_pos.lat,
        gps.lla_pos.lon,
        navdata.measure.ultrasound,
        navdata.measure.vx,
        navdata.measure.vy,
        navdata.measure.vz,
        imu.mag.x,
        imu.mag.y,
        imu.mag.z,
        quat->qi,
        quat->qx,
        quat->qy,
        quat->qz
       );
  counter++;
}
```